
PyLith

Release 5.0.1

Brad T. Aagaard, Matthew G. Knepley, Charles A. Williams

Jun 24, 2026

TABLE OF CONTENTS

1	Description	1
2	Other sources of documentation	3
3	PyLith development team	5
4	License	7
4.1	Introduction	7
4.2	User Guide	41
4.3	Developer Guide	637
4.4	References	700
	Bibliography	701

DESCRIPTION

PyLith is a finite-element code with a primary focus on modeling interseismic and coseismic deformation of Earth's crust and upper mantle. PyLith supports 2D and 3D static, quasistatic (neglecting inertia), and dynamic (including inertia) formulations of the governing equations, which can be elasticity, incompressible elasticity, or poroelasticity. A variety of elastic and viscoelastic bulk rheologies are supported. Boundary conditions include Dirichlet, Neumann, and absorbing boundaries. Faults are treated as interior interfaces. Currently, only the slip must be prescribed (kinematic rupture). We plan to reimplement spontaneous rupture (fault friction) in an upcoming release; see `ref{sec-development-plan}` for more information.

The code is written in C++ and Python and uses MPI for parallel processing. We use the Pyre Python framework to setup the simulation. We leverage [PETSc](#) for finite-element data structures and operations as well as linear and nonlinear solvers.

OTHER SOURCES OF DOCUMENTATION

- [This manual in other formats epub pdf](#)
- [PyLith tutorials](#)
- [SpatialData documentation](#)
- [PyLith installer documentation](#) (for installing from source)

PYLITH DEVELOPMENT TEAM

The primary developers are:

- Brad Aagaard (U.S. Geological Survey)
- Matthew Knepley (University at Buffalo)
- Charles Williams (Earth Sciences New Zealand)

LICENSE

Copyright (c) 2010-2026 University of California, Davis

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4.1 Introduction

4.1.1 Preface

This documentation targets two categories of users: (1) scientists who prefer to use prepackaged and specialized analysis tools, and (2) experienced computational Earth scientists. If you want to modify the source code, it is helpful to be familiar with object-oriented programming, Python and C++, and finite-element analysis.

Citation

The Computational Infrastructure for Geodynamics (CIG) (geodynamics.org) makes this software and source code available to you at no cost in hopes that the software will enhance your research in geophysics. A number of individuals have contributed a significant portion of their careers toward the development of this software. It is essential that you recognize these individuals in the normal scientific practice by citing the appropriate software release and peer-reviewed papers as well as making appropriate acknowledgments in talks and publications. The preferred way to generate the list of publications (in BibTeX format) to cite is to run your simulations with the `--include-citations` command line argument, or equivalently, the `--petsc.citations` command line argument. The `--version` command line argument will generate the BibTeX entries for the references mentioned below.

The following peer-reviewed paper discusses the development of PyLith:

- Aagaard, B. T., M. G. Knepley, and C. A. Williams (2013). A domain decomposition approach to implementing fault slip in finite-element models of quasistatic and dynamic crustal deformation, *Journal of Geophysical Research: Solid Earth*, 118, doi: 10.1002/jgrb.50217.

To cite the software and manual, use:

- Aagaard, B., M. Knepley, C. Williams (2026a), *PyLith v5.0.1*. Davis, CA: Computational Infrastructure of Geodynamics. DOI: 10.5281/zenodo.20557504.
- Aagaard, B., M. Knepley, C. Williams (2026b), *PyLith Manual, Version 5.0.1*. Davis, CA: Computational Infrastructure of Geodynamics. <https://pylith.readthedocs.io/en/v5.0.1>

Publishing Models

Open research statements are now a common requirement when publishing research. These support reuse, validation, and citation and often take the form of *Data availability*, *Data access*, *Code availability*, *Open Research*, and *Software availability* statements. We recommend including all input files to allow your published research to be reproduced and output model data to support your research outcomes and figures. In addition, consider including model files that may be useful to others. Files should be deposited in an approved repository.

Remember to cite software and data in your text as well as in your *Data availability* or similar statement. Additional information on *Publishing* and repositories is available on the CIG website.

Files

Common input files include `.cfg`, `.spatialdb`, and `.exo`, `.msh`, or `mesh` files. Common output files include the solution fields and state variables in `.h5`, `.xmf`, and `vtu` files.

Example Statement

The configuration files, parameters of the simulation, and solution field for the models in this study are available at DOI (Authors X, Y, Z) under CC BY-NC-SA 4.0.

PyLith version 5.0.1 (Aagaard et al., 2013; Aagaard et al., 2026a; Aagaard et al., 2026b) used in these models is freely available under the MIT license for download through its software landing page <https://geodynamics.org/resources/pylith> and Zenodo (10.5281/zenodo.20557504). The project is being actively developed on GitHub and can be accessed via <https://github.com/geodynamics/pylith>.

Support

Current PyLith development is supported by the CIG, and internal Earth Sciences New Zealand <https://earthsciences.nz/> and U.S. Geological Survey <https://www.usgs.gov/> funding. Pyre development was funded by the Department of Energy's <https://www.energy.gov/energygov> Advanced Simulation and Computing program and the National Science Foundation's Information Technology Research (ITR) program.

This material is based upon work supported by the National Science Foundation under Grants No. 0313238, 0745391, and EAR-1550901. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Acknowledgments

Many members of the community contribute to PyLith through reporting bugs, suggesting new features and improvements, running benchmarks, and asking questions about the software. See the contributors list for each release for specific contributions.

Request for Comments

Your suggestions and corrections can only improve this documentation. Please report any errors, inaccuracies, or typos to the PyLith section of the CIG Community Forum <https://community.geodynamics.org/c/pylith> or create a GitHub pull request.

4.1.2 Conventions

Warning

This is a warning.

Important

This is something important.

Tip

This is a tip, helpful hint, or suggestion.

For features recently added to PyLith, we show the version number when they were added. *New in v3.0.0.*

Command Line Arguments

Example of a command line argument: `--help`.

Filenames and Directories

Example of filenames and directories: `pylith`, `/usr/local`.

Unix Shell Commands

Commands entered into a Unix shell (i.e., terminal) are shown in a box. Comments are delimited by the `#` character. We use `$` to indicate the bash shell prompt.

```
#This is a comment.
$ ls -l
```

Excerpts of cfg Files

Example of an excerpt from a `.cfg` file:

```
# This is a comment.
[pylithapp.problem]
timestep = 2.0*s
bc = [x_pos, x_neg]
```

4.1.3 Quickstart

1. Download the binary package appropriate for your system.
See *Installation* for instructions.
2. Examine and run relevant examples.
See *Examples*.

4.1.4 Release Notes

See <https://github.com/geodynamics/pylith/commits/main> for the complete log of changes made to PyLith.

Note

Starting with v3.0.0, we strictly follow the [semantic versioning guidelines](#). The version numbers are in the form MAJOR.MINOR.PATCH, where major releases indicate changes to the public API (parameters), minor releases indicate new functionality that is backward compatible, and patch releases indicate backward compatible bug fixes.

Version 5.0.2 (2026-XX-XX)

- **Changed**
 - Update default solver settings for incompressible elasticity for Schur factorization from `full` to `lower`.
 - Set the default length scale to 1 km instead of 100 km for consistency with most quasi-static problem with faults. For simulations with faults, the length scale should be set to the discretization size on the fault; this is related to the preconditioner which links the length scale to the discretization size.
- **Added**
 - `examples/subduction-3d`: Add expected Gmsh output for geometric operations in documentation.
- **Fixed**
 - `examples/subduction-3d`: Increase tolerance for slab surface creation. Fix setting mesh filename in parameter files using Cubit mesh.
- **Developer Changes**
 - Remove flag (`--output-sync=target`) in top-level `Makefile.am` that is not backwards compatible.

Version 5.0.1 (2026-06-07)

- **Changed**
 - Updated PETSc to 3.25.2.
- **Fixed**
 - Fixed PETSc bug in output of processor rank for materials when running in parallel.
 - Updated `examples/troubleshooting-2d` to match behavior after latest bugfixes.
 - Add missing files for `examples/subduction-3d` to source distribution.
 - Switch from `numpy.atan2()` to `numpy.arctan2()` in `examples/subduction-3d/generate_gmsh.py` for compatibility with numpy 1.x.
 - Additional clean up of parameter files for `reverse-2d` and `subduction-2d` examples.

Version 5.0.0 (2026-06-02)

- **Changed**
 - Updated the nondimensionalization code, moving it from `SpatialData` to `PyLith`.
 - * Renamed `Nondimensional` to `Scales`.
 - * Added displacement scale and improve the names of the other scales.
 - * The time scale for poroelasticity is derived from the other scales and nominal material properties.

- * Added discussion of nondimensionalization for each of the governing equations.
- * Updated default solver tolerances consistent with the new nondimensionalization scales.
- Added option to use adaptive time stepping and update examples `reverse-2d` Steps 7 and 8 and `magma-2d` Step 1 to demonstrate its use.
- Updated multi-grid preconditioner solver settings for elasticity for better scalability with problem size.
- Updated `examples/subduction-3d` to include generation of the finite-element mesh using Gmsh or Cubit using Python scripts. The Gmsh model includes topography and bathymetry.
- Mesh formats
 - * Added support for use of Cubit side sets and Gmsh physical groups with edges (2D) and surfaces (3D). Use of Cubit vertex sets and Gmsh physical groups with vertices are deprecated and support for them will be removed in version 6.0.0.
 - Use of Cubit side sets and Gmsh physical groups with curves (2D) and surfaces (3D) are using
 - * ASCII mesh format
 - Changed `group` to `vertex-group`; remove `group type`.
 - * Renamed `VertexGroup` to `BoundaryGroup` in `meshio.gmsh_utils` and changed default behavior to not be recursive (generate “face” groups, not “vertex” groups).
 - * Added `improve_quality()` method to `meshio.gmsh_utils.GenerateMesh` for improving quality in 3D meshes. Used in `examples/crustal-strikeslip-3d`.
- Replaced hardcoded steps for initializing meshes (reading mesh, reordering a mesh, inserting cohesive cells, and refining a mesh) with a modular mesh initialization approach.
 - * The default mesh initialization implementation reads a mesh in serial, reorders it, distributes it (if necessary), inserts cohesive cells (if necessary), and refines it (if requested).
 - * A parallel mesh initiation implementation reads a mesh in parallel, redistributes it (if necessary), inserts cohesive cells (if necessary), and refines it (if requested).
- Updated error handling to use Pyre journal channels and PyLith exceptions.
 - * Add PyLith exceptions.
 - * Add C++ backtrace to PyLith exception messages.
- Reorganized use of Pyre journal channels with names based on use rather than components.
 - * The journal info channel ‘application-flow’ is enabled by default in the PyLith application.
 - * The journal device is set to color-console in the examples and full-scale tests.
- Use the VTU (XML) format for VTK files instead of the legacy ASCII format.
- Material `description` property is no longer used; a deprecation warning is printed to stdout if it is specified. This feature will be removed in v6.0.
- Internal labels for meshes and fields are now derived from the Pyre component name and identifier.
- Updated PETSc to 3.25.1.
- **Added**
 - Improved documentation for `pylith_eqinfo` and illustrate use in `examples/strikeslip-2d`, `examples/crustal-strikeslip-2d`, and `examples/crustal-strikeslip-3d`.
 - Added `pylith_convertmesh` for converting Cubit and Gmsh files to the PETSc HDF5 mesh format.
 - Added support for specifying boundary conditions using marked boundaries rather than vertices.

- * **MeshIOAscii**: Use `face-group` for specifying boundary conditions via cell and face vertices.
- * **MeshIOCubit**: Read sidesets from ExodusII files (generated by Cubit).
- * **MeshIOPetsc**
 - **Gmsh**: Create physical groups only at the dimension of the boundary.
 - Added compatibility for reading PETSc HDF5 format.
- Improved the organization of the governing equations section and added documentation for poroelasticity with prescribed fault slip, including governing equations and default PETSc solver settings. Also added a full-scale test.
- Added performance benchmarks for linear elasticity with prescribed slip in `benchmarks/performance`.
- **Fixed**
 - Account for processes without cells in initial condition patch when verifying configuration.
 - Avoid divide by zero for `KinSrcRamp` when final slip is zero.
 - Removed `solid_bulk_modulus` as a spatial database field for poroelasticity; compute it from the other fields.
 - Fixed typos in setting up gravity with poroelasticity.
 - Fixed pythia import in `pylith_eqinfo`.
- **Developer Changes**
 - Replace `err = PetscFoo(args); PYLITH_CHECK_ERROR(err);` with `PyolithCallPetsc(PetscFoo(args));`
 - Replace `assert(arg)` in test code with `REQUIRE(arg);`

Known issues

- The default PETSc options provide a computationally expensive preconditioner when solving incompressible elasticity problems. We expect to have a more optimal preconditioner in the next few months.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735
- Charles Williams  0000-0001-7435-9196
- Ritwik Patil  0009-0008-3305-6728

Version 4.2.0 (2025-01-15)

- **Changed**
 - Updated `examples/strikeslip-2d` Steps 4-7 to use a more realistic slip distribution and mesh refinement in output.
 - Updated to PETSc 3.22.2
 - Switch CI from Azure Pipelines to GitHub Actions.
- **Added**

- Default filenames for progress monitor and parameters file are set from the simulation name like the other output files.
- Consistency check for material properties and scales used in nondimensionalization (currently just the shear modulus).
- Added section to User Guide on troubleshooting solver issues.
- Added section to User Guide on how to run PyLith binary offline.
- Allow output on a finer resolution mesh than used in the simulation to facilitate accurate representation of fields with a basis order of 2 or greater.

- **Fixed**

- Fixed inconsistency in normal direction on fault surfaces. Orientation was correct but direction was flipped at some locations. This affected local slip direction and the resulting deformation close to the fault. This bug fix in a PETSc branch was not in version 4.1.3.
- Updated `TimeDependentAuxiliaryFactory::updateAuxiliaryField()` to account for some processes potentially not having points in a boundary condition (`auxiliaryFieldArray == NULL_PTR`).
- Updated autoconf numpy macros for compatibility with location of include files in numpy version 2.x.

Version 4.1.3 (2024-07-31)

- **Added**

- Expanded the section of the manual discussing mesh generation with Cubit and Gmsh. Added a list of useful functions in the Cubit and Gmsh Python interfaces.
- Added a test to verify Gmsh files with entities that are not embedded generate errors when read via `MeshIO Petsc`.
- Expanded section of manual discussing installation on Windows Subsystem for Linux, including components needed to run GUI applications (Gmsh and `pylith_viz`).
- Add Gmsh Python script and corresponding mesh in `examples/magma-2d`.

- **Fixed**

- Added output of fault traction change when computing static Green's functions.
- Turn on PETSc HDF5 collective input and output by default. This resolves errors when using HDF5 1.14 on NFS filesystems.
- Include all necessary files from `examples/subduction-3d` in source and binary packages.
- Fix typos in documentation and update Components section of User Guide (sync with code).

Version 4.1.2 (2024-06-12)

- **Fixed**

- Fix inconsistency in normal direction on fault surfaces. Orientation was correct but direction was flipped at some locations. This affected local slip direction and the resulting deformation close to the fault.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735
- Charles Williams  0000-0001-7435-9196

- Grant Block 0009-0003-3402-0923

Version 4.1.1 (2024-06-09)

- **Fixed**
 - Improved setup of variable block Jacobian preconditioner used for elasticity with faults to reduce runtime.
 - Fix several typos and update a few diagrams in the documentation.

Version 4.1.0 (2024-06-05)

- **Changed**
 - Improved the default preconditioners for poroelasticity for improved scalability.
 - Improved the default preconditioners for elasticity with faults for improved scalability.
 - Replaced use of Cubit journal files with Cubit Python scripts for several examples, and use the skeleton sizing function to increase cell size with distance from the fault.
 - Removed ParaView Python scripts (replaced by `pylith_viz`)
- **Added**
 - New 2D and 3D crustal strike-slip faults examples based on the 2019 Ridgecrest earthquake.
 - New 2D subduction zone outer-rise faulting example examining poroelastic response to bending stresses.
 - New `pylith_viz` utility for plotting PyLith results.
 - Updated `examples/strikeslip-2d` and `examples/reverse-2d` to demonstrate use of uniform refinement and higher order discretizations to improve resolution of solution.
 - Documentation
 - * Developer Guide: Added description of the process for adding event logging and evaluating performance, including performance logging.
 - * Developer Guide: Added checklist for what is needed when contributing examples.
- **Fixed**
 - Updated `examples/subduction-3d` Steps 6, 7, and 8 to work with PyLith v3+.
 - Fix performance bottleneck when reading Gmsh files.
 - Remove extra (wrong) kernels for poroelasticity when using state variables.
 - Update Python unit tests setup for compatibility with the current unittest API (use `load_tests()`).

Contributors

- Brad Aagaard 0000-0002-8795-9833
- Matthew Knepley 0000-0002-2292-0735
- Charles Williams 0000-0001-7435-9196
- Daniel Douglas 0000-0002-7871-018X
- Evan Marschall 0009-0003-0916-6656
- Zechao Zhuo 0000-0002-8163-5731

Version 4.0.0 (2023-12-14)

- **User parameters**
 - Changed name of fault Lagrange multiplier field for solution component in Python from `lagrange_fault` to `lagrange_multiplier_fault` to match name of solution field in C++.
 - Removed support for importing meshes from LaGriT.
- **Added**
 - Change in fault tractions are now included in the `fault_data_fields` for prescribed slip.
 - Fault and boundary orientation directions are now included in the `info_fields` for simulation output.
 - State variables are now included in the default `data_fields` for simulation output.
 - The default solver settings use the PETSc proper orthogonal decomposition (POD) methodology for initial guess of solutions to improve convergence.
 - Add demonstration of `pylith_powerlaw_gendb` in Step 8 of `examples/reverse-2d`.
 - Add demonstration of using poroelasticity with porosity as a state variable to `examples/magma-2d`.
 - Improve integration with VSCode for testing and debugging (see Developer Guide)
- **Changed**
 - Switched from CppUnit to Catch2 for the C++ testing framework.
 - Update PETSc to version 3.20.2
 - Update Python requirement to version 3.8 or later.
 - Update Pyre requirement to version 1.1.0 or later.
 - Update SpatialData requirement to version 3.1.0 or later.
- **Fixed**
 - Fix errors in `KinSrcTimeHistory.py`
 - Fix creation of PETSc label for edges when importing Gmsh files. This fixes creation of faults with buried edges for 3D meshes imported from Gmsh.
 - Add containers for solution fields for poroelasticity with faults.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735
- Charles Williams  0000-0001-7435-9196
- Grant Block  0009-0003-3402-0923
- Daniel Douglas  0000-0002-7871-018X
- Lorraine Hwang  0000-0002-1021-3101
- Rezgar Shakeri  0000-0003-4790-7016
- Robert Walker  0000-0001-7856-1949

Version 3.0.3 (2022-10-14)

This is a bug fix release with no new features or changes to the user interface.

- **Fixed**

- Fixed duplicate integration of fault terms if a fault had one material on one side and multiple materials on the other side.
- Fixed bugs related to running in parallel.
 - * Creating constraints on buried fault edges failed for some mesh distribution cases.
 - * Green's function problems did not manage fault impulses on multiple processes.
 - * Creating a point mesh for `OutputSolnPoints` failed when running in parallel.
 - * PetscSF inconsistencies generated errors at various times when running in parallel.

- **Changed**

- Update to PETSc 3.18.0.

Note: We now use PETSc routines to write the HDF5 files. As a result, there is one change to the layout: `topology/cells` is now `viz/topology/cells`. The corresponding Xdmf files reflect this change.

Binary packages

- Update to Python 3.10.6.
- Use `gmforker` process manager with MPICH to avoid localhost name issues.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735
- Charles Williams  0000-0001-7435-9196

Version 3.0.2 (2022-08-24)

This is a bug fix release with no new features or changes to the user interface.

- **Added**

- Add check of PyLith version against version requirements specified in metadata of parameter files.
- Documentation
 - * Add discussion of translating boundary value problem information to parameter settings. Add more code blocks to manual.
 - * Add discussion of `examples/troubleshooting-2d` to manual.

- **Changed**

- Update defaults to better match most use cases.
 - * Use nonlinear solver.
 - * Basis order is 1 for solution fields.
 - * Basis order is 0 for Cauchy stress and strain.

- * Use ML algebraic multigrid preconditioner (from Trilinos) instead of GAMG preconditioner for more robust solves. This is a temporary change until we find better GAMG settings.
- Update PETSc to v3.17.3.
- **Fixed**
 - Add viz directory missing from `examples/subduction-2d` in source distribution.
 - Project output fields using correct PETSc routine (`DMPProjectFieldLabel()`). Fixes memory access bugs in both serial and parallel.
 - Fix build warnings.
 - Fix reordering that causes errors when importing Gmsh files.
- **Removed**
 - Remove obsolete LaTeX documentation.

Binary packages

- Added PyQT5 Python module for interactive plotting with matplotlib.
- Update PyLith Parameter Viewer to v2.0.1 (fix errors in packaging).

Known issues

- The default PETSc options provide a computationally expensive preconditioner when solving incompressible elasticity problems in parallel. We expect to have a more optimal preconditioner in the next release.
- You may still encounter a few bugs when running in parallel; they appear to cases with specific partitioning of the mesh in relation to one or more faults.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735
- Charles Williams  0000-0001-7435-9196

Version 3.0.1 (2022-06-19)

This is a bug fix release with no new features or changes to the user interface.

- **Fixed**
 - Fix lots of small bugs related to running in parallel
 - Fix several discrepancies among the code, examples, and manual
- **Added**
 - Examples
 - * Added `examples/subduction-3d` steps 1-4 (included in the manual)
 - * Added `examples/troubleshooting-2d` (included in the PyLith v3.0 tutorials but not yet added to the manual)
 - Documentation
 - * Added instructions for how to remove Apple quarantine attributes

- * Fix LaTeX build of documentation (now available at <https://pylith.readthedocs.io>)
- * Improved instructions on how to run ParaView Python scripts when starting ParaView from a shortcut
- * Added notes indicating steps of examples are not yet updated for v3.0
- * Fix lots of typos

Binary packages

- Updated PyLith Parameter Viewer (v2.0.0) for Python 3.

Known issues

- The default PETSc options provide a computationally expensive preconditioner when solving incompressible elasticity problems in parallel. We expect to have a more optimal preconditioner in the next release.
- You may still encounter a few bugs when running in parallel; they appear to cases with specific partitioning of the mesh in relation to one or more faults.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735
- Charles Williams  0000-0001-7435-9196

Version 3.0.0 (2022-06-08)

Version 3.0.0 includes major changes to the underlying finite-element formulation and implementation in order to support a more flexible specification of the governing equations and higher order basis functions. These changes affect how simulations are defined. Parameter files for previous versions will need to be updated; the changes are too complex for a simple translation table. Some features present in v2.2.2, such as spontaneous rupture and finite strain, have not yet been implemented in the new formulation.

- **Added**
 - Multiphysics
 - * Elasticity for linear isotropic materials and linear Maxwell, generalized Maxwell, and power law viscoelastic models
 - * Incompressible elasticity for linear isotropic materials
 - * Prescribed slip for quasistatic and dynamic simulations
 - Higher order basis functions Allow user to select order of basis functions independent of the mesh (which defines the geometry). This permits higher resolution for a given mesh.
 - Added ability to import finite-element meshes from Gmsh
 - Switch to using PETSc time-stepping (TS) algorithms Replace simple Python-based time-stepping implementations with PETSc time-stepping algorithms that provide support for higher order discretization in time and real adaptive time stepping.
 - Modular approach for initial conditions
 - Output of subfields with user-defined basis order
 - Simulation metadata with command line utility for searching metadata

- Convert to Python 3
- Convert LaTeX documentation to Sphinx + MyST
- Testing with the Method of Manufactured Solutions
- Automatically assign label value for fault cohesive cells (id setting is obsolete).

- **Changed**

- Use `description` for descriptive labels and `label` and `label_value` for tagging entities. PyLith's use of `label` and `label_value` now corresponds to PETSc labels and label values.

- **Deprecated**

- We plan to discontinue support for reading LaGriT mesh files in version 3.2. Gmsh provides an open-source alternative with a graphical user interface.

SpatialData settings

```
db = spatialdata.spatialdb.UniformDB

# Old
db.label = Slip spatial database

# New
db.description = Slip spatial database
```

Material settings

```
material = pylith.materials.Elasticity

# Old
material.label = Elastic material
material.id = 2

# New
material.description = Elastic material
material.label_value = 2
```

Known issues

- Running in parallel has a few minor bugs due to communication mismatches and over-aggressive error checking. We will be fixing these in the next week.
- We will be updating the 3D subduction zone example (`examples/subduction-3d`) to v3.0.0 in the next week, including providing the input mesh file; in the meantime do not attempt to run this example.
- We have included Gmsh in the binary packages. For Linux there are additional libraries that must be installed for Gmsh to run; these are associated with the graphical user interface and included in most default installations.

Contributors

- Brad Aagaard  0000-0002-8795-9833
- Matthew Knepley  0000-0002-2292-0735

- Charles Williams  0000-0001-7435-9196
- Kali Allison  0000-0002-0561-8681
- Alex Berne  0000-0001-7857-8513
- Jed Brown  0000-0002-9945-0639
- Lorraine Hwang  0000-0002-1021-3101
- Rey Koki  0009-0007-6551-743X
- Shengduo Liu
- Chris Mills
- Thea Ragon  0000-0002-1276-1910
- Robert Walker  0000-0001-7856-1949

Version 2.2.2 (2019-07-02)

- Bug fixes
 - Fix several typos in the manual.
 - Fix order of deallocation of data members in ElasticMaterial to prevent a segmentation fault, thereby allowing error messages to be shown when throwing an exception.
 - Fix tests for MPI and PETSc version info for more use cases.
 - Ensure all Python script are executable and use nemesi instead of python for correct paths to modules on Darwin.
- Added ability to write residual to an HDF5 file during solves. This is intended for debugging and is enabled at runtime using `--journal.debug.formulation=1`. The residual will be written to `residual.h5`. To generate the associated `.xdmf` file run `pylith_genxdmd -f residual.h5`.
- Updated to PETSc 3.10.2

Version 2.2.1 (2017/09/06)

- Added new examples.
 - **examples/3d/subduction**: New suite of examples for a 3-D subduction zone. This intermediate level suite of examples illustrates a wide range of PyLith features for quasi-static simulations.
 - **examples/2d/subduction**: Added quasi-static spontaneous rupture earthquake cycle examples (Steps 5 and 6) for slip-weakening and rate- and state-friction.
 - These new examples make use of ParaView Python scripts to facilitate using ParaView with PyLith.
- Improved the PyLith manual
 - Added diagram to guide users on which installation method best meets their needs.
 - Added instructions for how to use the Windows Subsystem for Linux to install the PyLith Linux binary on systems running Windows 10.
- Fixed bug in generating Xdmf files for 2-D vector output. Converted Xdmf generator from C++ to Python for more robust generation of Xdmf files from Python scripts.
- Updated spatialdata to v1.9.10. Improved error messages when reading SimpleDB and SimpleGridDB files.

- Updated PyLith parameter viewer to v1.1.0. Application and documentation are now available on line at https://geodynamics.github.io/pylith_parameters. Small fix to insure hierarchy path listed matches the one for PyLith.
- Updated PETSc to v3.7.6. See the PETSc documentation for a summary of all of the changes.
- Switched to using CentOS 6.9 for Linux binary builds to insure compatibility with glibc 2.12 and later.

Version 2.2.0 (2017-03-28)

- Added a browser-based parameter viewer for interactive viewing of all PyLith parameters and version information. See Section 4.10 PyLith Parameter Viewer of the PyLith user manual.
- Adjusted packaging of the binary distributions so that they can be used to extend PyLith and/or integrate other code with PyLith.
- Converted the user manual from Lyx to LaTeX and added syntax highlighting of parameter and spatial database files. Fixed several typos.
- Fixed bug that sometimes resulted in an inconsistent fault orientation when running in parallel. The bug appears to have been introduced in v2.0.
- Fixed two bugs in output of solution at points that sometimes happened in parallel simulations. The errors include:
 - The order of the station names does not match the order of the points. The point data is written in parallel by process order, so the points for process 0 are written first, then those for process 1, etc. This often results in reordering of the points. The station names were written in the original order.
 - The output values for some points are incorrect. The wrong cells were being used in the interpolation.
- Updated PETSc to v3.7.5.

Version 2.1.4 (2016-10-21)

- Added `--version` command line argument to display version information for PyLith and its dependencies.
- Improved information displayed with the `--help` command line argument.
- Added `--include-citations` command line argument to display publications to cite when publishing results from computations using PyLith. General PyLith references are also displayed with the `--version` command line argument.
- Allow use of NetCDF versions greater than 4.1.3. Switch from using C++ API to C API.
- Fixed bug in Pythia associated with validation of parameters being done before help could be displayed.
- Fixed typos in manual for gravity and point forces.
- Added integration with Travis for automated testing.

Version 2.1.3 (2016-07-20)

- Add `generate_statedb.py` and `postseismic.pvsm` files missing from `examples/2d/gravity`.
- Update handling of fault intersection when creating boundary condition nodesets in `examples/meshing/surface_nurbs/subduction`.
- Fixes to Darwin binary package.
 - Fix linking of netCDF4 Python module.
 - Fix linking and executable mode permissions for Python scripts in binary by using `nemesis` so relative links are valid.

Version 2.1.2 (2016-06-16)

- Bugfixes for finite-strain formulation.
 - Added output of the Cauchy stresses (`cauchy_stress`). The second Piola-Kirchoff stresses are output via the stress field.
 - Material properties and state variables were not retrieved properly when updating state variables.
- Bugfixes for setting initial stress and state variables for viscoelastic materials. The deviatoric stress state is carried forward using the state variables, so the initial deviatoric stress should not be considered when computing the stresses.
- Created new examples showing how to use gravity, initial stress, and finite-strain in 2-D (`examples/2d/gravity`).
- Reintroduced check (that had been inadvertently removed in v2.x) for ambiguous description of fault surface based on groups of vertices defining faces of cells.
- Flush the output of the progress monitor so progress reports are updated promptly.
- Updates to the user manual.
 - Added section on the debugging examples covered in recent tutorials.
 - Added tables describing the spatial database values for each material.
 - Included a more complete discussion of the finite-strain formulation.
- PETSc
 - Updated to PETSc v3.7.2 (`knepley/pylith` branch).
 - Fixed Trilinos/ML configuration and code so that it can be built without a Fortran compiler.

Version 2.1.0 (2015-02-19)

- Station names are required for output at arbitrary points (`OutputSolnPoints`) and are included in a `/stations` dataset in HDF5 files.
- A progress monitor will update a text file with the progress of a simulation (time in the time stepping loop or the number of impulses completed) and given an estimate of when the simulation will be completed.

Bug fixes

- A few bugs related to creating cohesive cells for fault intersections have been fixed. Faults can now meet at T intersections provided the buried edges of the faults are clamped. In other words, the fault ending at the T intersection has a clamped edge along the intersection. The fault ending at the intersection must also come AFTER the through-going fault in the list of fault interfaces.
- There have been two major bug fixes for Drucker-Prager plasticity, for both `DruckerPrager3D` and `DruckerPragerPlaneStrain`. The first fix was a missing initial pressure term for the plastic multiplier in the Drucker-Prager formulation. This affects plasticity calculations when initial stresses are used. The error has been corrected in the code, the manual, and the unit tests. The second bug was an incorrect test for tensile yield that could cause PyLith to exit with an error when plastic yield had not actually occurred. The error would only occur when the `allow_tensile_yield` flag was set to `False`. This bug has been fixed in the code, and the new test is also described in the manual. This should prevent problems that previously existed when `allow_tensile_yield` was set to `False` (as it should be for most quasi-static problems).
- Fixed bug in `DataWriterHDF5Ext` associated with multiple processes writing information to the HDF5 file. With external datasets the HDF5 file is limited to metadata and is maintained by process 0.

- A two-dimensional gravity example has been added, based on the tutorial from the June, 2014 workshop at Stanford University. The tutorial itself is in `examples/2d/gravity`, and a new section has also been added to the manual describing the example.
- Fixed inconsistent fault orientation when running in parallel for 2-D domains.

Migrating from v2.0.x to v2.1.x

The points file for `OutputSolnPoints` must now contain station names as the first column.

Version 2.0.3 (2014-08-19)

Bug fixes

- Updated autotools files (`Makefile.am`, `configure.ac`) for compatibility with recent versions of automake (up to and including v1.14.1).

Version 2.0.2

Bug fixes

- Fixed linking issue in Darwin binary distribution, primarily affecting systems with OS X 10.7 and 10.8.
- Improved example journal files for Cubit/Trelis to improve compatibility (`examples/meshing/surface_nurbs/dem`).
- Updated more journal in examples so that APREPRO lines have a leading '\$' instead of a '#' to differentiate from comments.
- Added `examples/debugging` files from Crustal Deformation Modeling workshop debugging tutorial.

Version 2.0.1 (2014-06-22)

Bug fixes

- Improved example journal files for Cubit/Trelis to improve compatibility. All journal files should work with Cubit 14.1 and Trelis 15.0.
- Created examples of IDless journal files in `examples/2d/greensfns`. These files should work with all recent versions of Cubit and Trelis.
- Switched journal APREPRO lines to have leading '\$' instead of '#' to differentiate from comments.

Version 2.0.0 (2014-06-05)

- Replaced C++ Sieve implementation of finite-element data structures with C DMPlex implementation.
DMPlex provides a simpler, more efficient implementation of the finite-element data structures that conforms to the PETSc data management (DM) interface. This provides tighter integration with the rest of PETSc. Additionally, this rewrite of the data structures results in a more efficient memory layout, resulting in better performance.
- Improved treatment of buried fault edges, so that the slip naturally tapers to zero along the buried edges.
An additional `nodeset/pset` is used to designate the buried edges of a fault. This allows the cohesive cells to be inserted up to the edge of the fault without splitting the mesh at the fault edge. The slip will naturally taper to zero at along the buried edges as a result of how the cohesive cells are created.
- Switched from using Subversion to Git for version control.

The source code repository changed from a CIG maintained Subversion repository to a Git repository at GitHub.com. The URL for the Git repository is <https://github.com/geodynamics/pylith>. The installer has been updated accordingly.

- Added ability to recursively refine a mesh.

Global uniform refinement can now be done recursively. Each refinement reduces the vertex spacing by a factor of two. Using more than one level of refinement should be done carefully as the mesh quality will generally deteriorate with more levels of refinement.

- Directories for output are created as necessary.

Directories where output files are written will be created if necessary. Previously, the directories would not be created, so that opening the output files in a nonexistent directory would generate an error.

- Improved error messages.

Error messages originating in PETSc will include a stack trace that includes both PyLith and PETSc code. Previously, only the PETSc code was included. This provides significantly more information for debugging.

- Improved Cubit example for mesh sizing functions.

Based on experimentation with Cubit 14.0, 14.1, and Trelis 15.0, we have improved the Cubit mesh sizing examples (`examples/meshing/cubit_cellsize`). We were able to simplify the journal files and use fewer Cubit commands. The new procedure also eliminates some Cubit warnings.

- Several small improvements to various sections of the manual based on feedback and questions from users.
 - Added more information about the workflow involved in using PyLith.
 - Added a discussion of how to set scales for nondimensionalization.
 - Added a discussion of how the stable time step is computed for the various materials.
 - Updated and expanded the discussion of using initial state variables.

Bug fixes

- Fixed two MPI related bugs in computing Green's functions in parallel. The number of impulses corresponded to only those on process 0.
- Corrected computation of fault tractions (Lagrange multipliers) on process boundaries for prescribed slip with explicit time stepping.
- Fixed bug when reading in list of output points with just one point.
- Adjusted autoconf Python setup macro to remove temporary `sysconfig.pyc` file.
- Added check to make sure degree of freedom specified in Dirichlet BC is consistent with spatial dimension of problem.
- Corrected two typos in the manual related to fault opening and tractions in `examples/3d/hex8/step20` and updating to the use of `cell.dimension` for the quadrature scheme with tets.
- Fixed stable time step computed for power-law viscoelastic rheology to match manual.

Migrating from v1.9.x to 2.0.x

Changes to various C++ objects permitted simplifying the specification of a number of components. The map below indicates the name changes.


```

CellFilterAvgMesh -> CellFilterAvg
CellFilterAvgSubMesh -> CellFilterAvg
DataWriterVTKMesh -> DataWriterVTK
DataWriterVTKSubMesh -> DataWriterVTK
DataWriterVTKSubSubMesh -> DataWriterVTK
DataWriterHDF5Mesh -> DataWriterHDF5
DataWriterHDF5SubMesh -> DataWriterHDF5
DataWriterHDF5SubSubMesh -> DataWriterHDF5
DataWriterHDF5ExtMesh -> DataWriterHDF5Ext
DataWriterHDF5ExtSubMesh -> DataWriterHDF5Ext
DataWriterHDF5ExtSubSubMesh -> DataWriterHDF5Ext

```

Running the script:

```
bash $PYLITH_DIR/doc/developer/update_1.9to2.0.sh
```

will update all .cfg files in the current directory and all subdirectories with the new names (you will need to replace \$PYLITH_DIR with the directory containing the PyLith source code).

PyLith allows use of the Chaco and ParMetis/Metis partitioners. The name of the ParMetis/Metis partitioner was changed from “parmetis” to “metis”.

```

[pylithapp.mesh_generator]
distributor.partitioner = metis

```

Buried edges of faults are handled differently in v2.0. A separate nodeset/pset should be created and contain the vertices on the buried edges of the fault. See the Section 6.4.2 of the PyLith manual for more information.

Version 1.9.0 (2014-05-05)

New features

- Added Newton-Raphson algorithm for spontaneous rupture simulations with explicit-stepping.
- Enforcing the friction criterion in a spontaneous rupture simulation with explicit time-stepping now uses a Newton-Raphson algorithm to find the correct traction increment. This provides a more stable numerical solution and eliminates oscillatory behavior when using rate-state friction.
- Added SCEC spontaneous rupture benchmark TPV102 to the benchmark repository. PyLith produces results very similar to several other finite-element codes.

Bug fixes

- Fixed two MPI related bugs in computing Green’s functions in parallel. The number of impulses corresponded to only those on process 0 and the output of the impulses for vertices on processor boundaries was inconsistent.
- Corrected computation of fault tractions (Lagrange multipliers) on process boundaries for prescribed slip with explicit time stepping.
- Fixed bug when reading in list of output points with just one point.
- Adjusted autoconf Python setup macro to remove temporary sysconfig.pyc file.
- Added check to make sure degree of freedom specified in Dirichlet BC is consistent with spatial dimension of problem.
- Corrected two typos in the manual related to fault opening and tractions in examples/3d/hex8/step20 and updating to the use of cell.dimension for the quadrature scheme with tetrahedral cells.

Migrating from v1.8.x to v1.9.x

No changes are needed in .cfg files to switch from v1.8.0 to v1.9.0. Version 1.9.0 does include some changes to the friction and material model interfaces, so extensions do require changes. See the templates for details.

Version 1.8.0

New features

- Additional flexibility in PETSc nonlinear solver parameters

The default line search type for the PETSc nonlinear (SNES) solver is a customized backtrace method included in PyLith. The user may now select alternative line search types (basic, bt, l2, cp) available in PETSc.

- Post-processing utility `pylith_eqinfo` to compute slip information.

This post-processing utility computes the moment magnitude, seismic moment, seismic potency, and average slip at user-specified snapshots in time from PyLith HDF5 output. Information is given for each fault and across all faults. See the Post-processing section in the Running PyLith chapter of the manual for more information.

- Computation of the stable time step for explicit time-stepping.

The stable time step for explicit time-stepping is computed based on the CFL condition and minimum edge lengths. For triangular and tetrahedral cells we also account for a reduction in the stable time step due to distorted cells (e.g., slivers and needles). See the Stable time step section in the Materials chapter of the manual for more information.

- Output the stable time step for each cell in a material.

Output `cell_info_fields` “stable_dt_implicit” and “stable_dt_explicit” can be included in material output.

- Added netCDF Python module to binary distribution to provide Python interface to NetCDF files, including Exodus-II files. This is used in a new meshing example for setting the discretization size using an Exodus-II vertex field. Note that this required updating the NetCDF library.

Bug fixes

- Fixed omission of synchronization of stable time step computation among processors. Minimum time step among all processors rather than local value should be used.
- Fixed density scale not being set in `NondimElasticQuasistatic`. Density scale should be set based on shear modulus, length scale, and relaxation time.
- Added warning when initial state for a fault constitutive model is not set. If an initial state value is not given, for rate-state friction using a default value of $L / \text{reference slip rate}$. Other fault constitutive models use a default value of 0.0 for initial state variables.
- Separated tensor components in Xdmf files to avoid confusion. The corresponding HDF5 files remain unchanged.
- Removed explicit time-stepping formulation with non-lumped Jacobian. This formulation was not setup properly for spontaneous rupture models and is too computationally expensive for practical problems. The `ExplicitLumped` formulations are now simply `Explicit`.
- Fixed parallel bug that resulting in inconsistent orientation of fault slip directions. Flipping the fault orientation was not synchronized across processors. This bug would only appear when running in parallel with faults that change from dipping in one direction to dipping in the opposite direction.
- Fixed bug in setting name of field in `OutputSolnPoints` when output multiple fields. This bug caused the name of the first output field to be used and output data to overwrite each other.

Migrating from v1.7.x to v1.8.x

Explicit time stepping with a non-lumped Jacobian has been eliminated and ExplicitLumped is now Explicit.

Old setting

```
-----
formulation = pylith.problems.ExplicitLumped
formulation = pylith.problems.ExplicitLumpedTri3
formulation = pylith.problems.ExplicitLumpedTet4
```

New setting

```
-----
formulation = pylith.problems.Explicit
formulation = pylith.problems.ExplicitTri3
formulation = pylith.problems.ExplicitTet4
```

Version 1.7.1

Bug fixes

- Fixed a couple of bugs in the spontaneous earthquake rupture for quasi-static problems when running in parallel. These prevented the nonlinear solve from converging and erroneously generated fault-opening in some cases.
- Minor updates to the documentation and manual. Added Green's function examples to the manual.

Version 1.7.0

New features

- User-friendly interface for Green's functions
A new problem type provides a user-friendly interface for computing Green's functions associated with fault slip for complex spatial variation in elastic properties. See `examples/2d/greensfns` in the tutorials for examples.
- Output of solution field at user-specified locations
Added a new output manager for interpolation of the solution field to user-specified point locations. This feature is useful for comparison of the solution with observations and in computing Green's functions. See `examples/3d/hex8/step19` and `examples/2d/greensfns` in the tutorials for examples.
- Plane strain version of Drucker-Prager elastoplastic model
Added a plane strain version of the Drucker-Prager elastoplastic model. Additionally, the user can now select whether to use an inscribed, intermediate, or circumscribed fit to the Mohr Coulomb criterion.
- Spatial and temporal variation in tractions for spontaneous earthquake rupture
Switched from a simple constant spatial variation in initial fault tractions to the more flexible spatial and temporal variation consistent with the Dirichlet, Neumann, and point force boundary conditions. Also added a switch to turn on/off applying prescribed fault tractions when the fault opens; the default behavior is to stop applying prescribed fault tractions when the fault opens, but turning this off allows simulation of dike intrusions via prescribed fault tractions. See `examples/3d/hex8/step20` in the tutorials for an example of how to specify fault tractions with the new implementation.
- Ability to use PETSc GPU solvers
Added ability to build PyLith with either double (default) or single precision floating point values to facilitate use of GPUs. In order to use PETSc GPU solvers, CUDA and cusp must be installed and PETSc must be configured to use CUDA. See the PyLith manual and PETSc documentation for details.

- User-specified start time for simulations.

Users can set the simulation start time to any desired value. This facilitates combining simulations to model the earthquake cycle.

- Elastic prestep in quasi-static simulations is optional.

The elastic prestep in quasi-static simulations can be skipped (the default is to include the elastic prestep). This facilitates combining simulations to model the earthquake cycle.

Bug fixes

- Fixed bug in the spontaneous earthquake rupture for quasi-static problems when running in parallel.

Migrating from v1.6.x to v1.7.x

Two changes are required when migrating from version 1.6 to 1.7.

1. The FIATSimplex object now has the same parameters as the FIATLagrange object.

Old setting	New setting
-----	-----
<code>cell.shape = line</code>	<code>cell.dimension = 1</code>
<code>cell.shape = triangle</code>	<code>cell.dimension = 2</code>
<code>cell.shape = tetrahedron</code>	<code>cell.dimension = 3</code>

2. Prescribed fault tractions for spontaneous earthquake rupture use a new, more flexible implementation that follows the same functional form for spatial and temporal variation as that used in the Dirichlet and Neumann boundary conditions. Consequently, the output info fields are also different and follow the naming scheme used in the other time-dependent boundary conditions.

Old settings

<code>[pylithapp.timedependent.interfaces.fault]</code> <code>db_initial_tractions = spatialdata.spatialdb.SimpleDB</code> <code>db_initial_tractions.iohandler.filename = tractions.spatialdb</code> <code>db_initial_tractions.label = Initial fault tractions</code>
New settings

<code>traction_perturbation = pylith.faults.TractPerturbation</code> <code>[pylithapp.timedependent.interfaces.fault.traction_perturbation]</code> <code>db_initial = spatialdata.spatialdb.SimpleDB</code> <code>db_initial.iohandler.filename = tractions.spatialdb</code> <code>db_initial.label = Initial fault tractions</code>

Version 1.6.3

Bug fixes

- Improved error messages for problems encountered during processing of parameters. A backtrace of the object hierarchy is now included to pinpoint in which object the error occurred.
- Added a line search to the inner friction solve in quasi-static simulations to increase the robustness of the nonlinear solve. Simulations using rate and state friction now converge under a much wider range of circumstances.

- Fixed bug in updating slip state variable in slip-weakening friction. This caused slight errors in the cumulative slip. We also added a parameter that forces healing to occur in a single time step. This is used to confine slip to a single time step in quasi-static simulations. See `examples/3d/hex8/step13.cfg` for an example.
- Tuned parameters in the slip-weakening friction and rate and state friction examples (`step13.cfg` and `step14.cfg`, respectively) in `examples/3d/hex8` to give stick-slip behavior.
- Fixed communication issue associated with writing boundary condition information output in parallel.
- Changed info in Xdmf file for fields that are not scalars, vectors, or tensors so that the each component is extracted, facilitating visualization in ParaView. The corresponding HDF5 file remains the same.
- Added the ability to specify non-derived units (e.g., degree and radian). This is useful in specifying parameters for the Drucker-Prager elastoplastic rheologies. If no units are specified, radians are assumed.

Internal changes

- Rate and state friction with ageing law

The implementation of rate and state friction with ageing law was modified to work better with the iterative solver. We switched to the conventional, unregularized formulation but added a minimum cutoff for the slip rate. Below this cutoff friction has a linear rather than logarithmic dependence on slip rate. As long as this cutoff is close to the SNES solver tolerance, the difference in behavior is negligible while improving the ability of the solver to converge for very small deformations.

Known Issues

- The rate and state friction with ageing law has not been tested for dynamic rupture simulations. We plan to run the SCEC Dynamic Rupture benchmarks for rate and state friction as soon as we add a spatial-temporal specification of initial fault tractions, which are required for the benchmark problems.
- Running simulations with more than a million cells and large faults in parallel can result in severe memory imbalances among processors. Some processors around the fault may use 10x more memory than processors away from the fault. We expect this problem to disappear in v1.7 when we switch to new, more efficient Sieve implementation.

Version 1.6.2

Bug fixes

- Fixed bug in writing tensor data for Xdmf files. Switched Tensor to Tensor6 to account for symmetry.
- Fixed bug in writing HDF5 files in parallel when one processor does not write any information (e.g., faults and boundary conditions).
- Added dimensioning of time dataset in HDF5 files. The units are now seconds rather than nondimensional time.
- Fixed memory allocation error (`std::bad_alloc`) when a processor did not contain cells for a boundary condition or output. This bug did not show up on all architectures.
- Increased robustness of spontaneous rupture (fault friction) implementation to broaden the range of conditions it can handle. The implementation now properly handles cases with fault opening and cases with zero shear or normal tractions.

Internal changes

- Fault implementation

Several changes have been made to the fault implementation, but none of these affect the user interface. The run-time performance is nearly identical with improved accuracy for spontaneous rupture (fault friction) simulations.

These changes involved switching to using tractions (non-integrated quantities) for the Lagrange multipliers in the global coordinate system rather than integrated quantities in the fault coordinate system. Additionally, initial fault tractions are associated with the fault vertices and their interpolation uses the finite-element basis functions.

- Distribution of mesh among processors

The data structures used to distribute the mesh among processors have been improved. This reduces memory use and runtime for this stage of the simulations.

Known Issues

The custom line search used with the PETSc nonlinear solver (SNES) has difficulty handling some loading cases. In cases where the direction of the line search tends to be nearly orthogonal to the residual, the rate of convergence in the SNES iterations is extremely slow. In other cases the nonlinear solver gets stuck in a local minimum. We plan to improve the line search algorithm in a future release in order to resolve this issue and improve the rate of convergence in spontaneous rupture simulations.

Version 1.6.1

- Validation of user input

Added stricter requirements for descriptive labels of various objects, including spatial databases and friction models. The default labels are empty strings which do not result in useful error messages; the user is now required to specify a non-empty string for the labels. This makes errors related to spatial databases much easier to diagnose.

- Updates to manual
 - Updated description of `cell_info_fields` for Neumann boundary condition. The description had not been updated to reflect the time-dependence introduced in version 1.4.
 - Added steps 18 and 19 that discuss time-dependent Neumann boundary conditions to `examples/3d/hex8`.

Bug fixes

- Fixed bug in writing rupture information to VTK and HDF5 files when using multiple earthquake sources. Field names did not include name of rupture. This caused loss of information in VTK output and a corrupted Xdmf metadata file for HDF5 output.
- Fixed error in use of initial stress tensor with generalized Maxwell models. The initial stress tensor was added to the current stress tensor twice.
- Fixed two bugs in the fault friction implementation. One bug pertained to accounting for roundoff errors and convergence tolerances in computing the slip rate. Slip rates less than $1.0e-12$ (nondimensionalized) are set to zero. The friction implementation for quasi-static problems contained a bug that resulted in slip extending over all of the fault rather than the appropriate isolated patch.
- Cleaned up Green's function example (`examples/greensfns/hex8`) so that it runs without errors. Eliminated extraneous processing.
- Cleaned up meshing examples (`examples/meshing`), including elimination of superfluous pre-processing.
- Adjusted absolute tolerances for PETSc solves in `examples/3d/hex8` so that solver terminates with desired convergence criterion.
- Updated `examples/2d/subduction/geometry.jou` to use APREPRO functions and variables to store id values.

Version 1.6.0

New features

- Parallel binary output via HDF5

Provides much faster output by writing HDF5 files in parallel, which can be accessed directly from Matlab or indirectly from ParaView or Visit via automatically created Xdmf files. Temporal data is stored in 3-D arrays, permitting slicing in time and/or space. See examples/3d/hex8 Steps 6-9 and examples/2d/subduction in the tutorials for examples.

- 2-D generalized Maxwell viscoelastic bulk rheology

Added a 2-D generalized Maxwell viscoelastic bulk rheology corresponding to the plane strain version of the 3-D generalized Maxwell viscoelastic model.

- Time-weakening fault constitutive model

Added a linear time-weakening fault constitutive model. Some spontaneous rupture modelers prefer this model over linear slip-weakening because it is easier to maintain resolution of the cohesive zone.

- Global uniform parallel mesh refinement

Permits running larger problems through uniform global refinement of the mesh by a factor of 2 (reduces the node spacing by a factor of 2) after the mesh is distributed among processors. This allows running problems that are 4x larger in 2-D and 8x larger in 3-D. See examples/3d/tet4 Steps 2 and 4 for examples.

- Custom algebraic multigrid preconditioner

Adds a custom preconditioner for Lagrange multiplier degrees of freedom associated with fault slip via prescribed slip or spontaneous ruptures with algebraic multigrid preconditioning for quasi-static solutions. In most cases, this results in fewer iterations in the linear solve and the number of iterations increases much less with problem size. See examples/3d/tet4 Steps 2 and 4 for examples.

- PyLith installer utility

This utility provides a much more robust method for building PyLith and all of its dependencies from source, including dependency checking, installation to a central location, and creation of a shell script to set environment variables.

Bug fixes

- Fixed the fault friction implementation to correctly update Lagrange multiplier values when the slip is overestimated in an iteration. This primarily fixes problems encountered with the use of the Dieterich-Ruina rate and state fault constitutive model.
- Corrected viscoelastic rheologies to properly account for a nonzero initial strain tensor.

Migrating from v1.5.x to v1.6.x

No changes in parameters are required. Version 1.6.1 does require users to specify descriptive labels for spatial databases and friction models.

Version 1.5.2

- PyLith 1.5.2 requires FIAT version 0.9.9 or later and an updated PETSc development version. It also requires users to update to the latest spatialdata version for compatibility of the SWIG generated files. These are included in the binary distribution, but users building PyLith from source will need to update FIAT, PETSc, and spatialdata.

Bug fixes

- Fixed setting of elastic constants in DruckerPrager3D and computation of the yield function. Some off-diagonal elasticity constants were off by a factor of 2.0 and the yield function was missing a factor of 0.5 and `sqrt()`.
- Fixed computation of stable time step when using initial stresses with PowerLaw3D. If effective stress is zero, then stable time step is infinite.
- Re-enabled check for compatibility of quadrature scheme and cells for bulk rheologies.
- Added check to configure for compatible version of FIAT.
- Fixed bug where buffer for output of initial stresses for dynamic (spontaneous) rupture.

Version 1.5.1

Bug fixes

- Fixed dimensioning of velocity and acceleration fields in output. The scales were set to just the length scale rather than the length scale divided by the time scale and length scale divided by the time scale squared.
- Fixed partitioning of cohesive cells. Cohesive cells were ignored during partitioning of the mesh, so they were randomly distributed among processors.

Version 1.5.0

- Fault constitutive models
Added fault friction interface conditions with static friction, linear slip-weakening friction, and rate- and state-friction with the ageing law. The implementation can be used in static, quasi-static, and dynamic problems.
- Drucker-Prager elastoplastic bulk rheology
Added a Drucker-Prager elastoplastic bulk rheology. This is a perfect plasticity implementation (no hardening). This is a nonlinear constitutive model, so the nonlinear solver is required when this rheology is used. Refer to the 'Material Models' section of the manual.
- Plane strain Maxwell viscoelastic bulk rheology
Linear Maxwell viscoelastic rheology for plane strain problems.
- Finite-deformation formulation
Added a finite-deformation (rigid body motion and small strains) implementation of elasticity with stress calculated using the Second Piola Kirchhoff stress tensor and strains calculated using the Green-Lagrange strain tensor.
- Lumped Jacobian for explicit-time stepping
Added the option to lump cell Jacobian matrices to form a diagonal system Jacobian matrix for explicit time stepping. This decouples all degrees of freedom and permits use of a fast, trivial, direct solver.
- Optimized elasticity objects
Added optimized elasticity objects for the most popular cell types and basis functions (linear polynomials). For tri3 and tet4 cells with one quadrature point, the optimized implementations do not use reference (mapped) cells in order to reduce the number of operations.
- Scientific notation for ASCII VTK files
Data values in ASCII data files are written in scientific notation with user-specified precision.
- Nodeset names in Cubit Exodus files

Use of nodeset names in Cubit Exodus files for boundary conditions and faults. Users can specify to use nodeset names (default behavior) or ids.

- Velocity and slip rate as output fields

Velocity (domain and subdomain) and slip rate (fault) fields are can be requested as output fields. The fields are computed using the time-stepping algorithm and alleviates the need to compute them via post-processing.

- Dimensionless values in spatial databases no longer need artificial dimensions. Values without dimensions are understood by the parser as dimensionless quantities.
- Bug fixes
- Updating state variables did not retrieve physical properties for cell. Last physical properties retrieved were used. Physical properties are now retrieved when updating state variables.
- Fixed incorrect dimensioning of physical properties and state variables for the power-law rheology in output.
- Fixed memory bug for a fault in a 1-D mesh when constructing the cohesive cells.

Contributors

- Surendra Somala - fault friction implementation.

Migrating from v1.4.x to 1.5.x

Three changes to the code require updating old parameters settings for use with version 1.5.

1. Recent releases of Cubit include nodeset names in the Exodus file and PyLith now uses them to associate vertices with boundary conditions and faults. Use the NetCDF utility `ncdump` to examine the contents of the Exodus (.exo) file to see if it includes the variable `ns_names`. If it does, then use nodeset names rather than nodeset ids for boundary condition label properties. If your Exodus file does not contain nodeset names, then set the MeshIOcubit property `use_nodeset_names` to False to continue to use nodeset id values for boundary condition labels.
2. The power-law constitutive parameters have been changed so that the parameter units are no longer dependent on the power-law exponent. This is a more logical implementation and allows (among other things) users to vary power-law parameters using a spatial database. Previously, it was not possible to vary power-law parameters unless everything used the same power-law exponent. The new implementation uses reference-strain-rate, reference-stress, and power-law-exponent to describe the material. This is described in the ‘Material Models’ section of the manual.
3. The fault property `normal_dir` is obsolete. Only the property `up_dir` is required to enforce that positive slip is left-lateral, reverse, and fault-opening for dipping faults in 2-D and horizontal fault surfaces in 3-D. Previously, in 2-D positive slip was always left-lateral, but now the up-direction is used to enforce positive slip corresponds to reverse motion for dipping faults. For horizontal fault surfaces in 3-D a normal of (0,0,1) is assumed in determining the up-dip direction.

Migrating from v1.3.x to 1.4.x

A number of changes to the code require updating old parameter settings for use with version 1.4.

1. The mesh “importer” is now called “reader”.
2. The spatial database facility for a material, `db`, is separated into a `db_properties` and a `db_initial_state`. The initial stress and strain tensors are specified using the `db_initial_stress` and `db_initial_strain` facilities. The names of some of the spatial database values for physical properties for viscoelastic properties have changed.

3. The code is now intelligent enough to determine the dimensions of the quadrature required (e.g., Quadrature2D and Quadrature2Din3D, etc). Setting the quadrature to the object for a given spatial dimension and cell dimension is no longer allowed because it is done automatically.
4. The names of the output filters have changed and include suffixes Mesh or SubMesh to indicate that they operate on a mesh or submesh (e.g., CellFilterAvg is now CellFilterAvgMesh or CellFilterAvgSubMesh). This is related to the use of C++ templates.
5. The DirichletPoints boundary condition has been renamed to DirichletBC.
6. The procedure for enabling certain features no longer involves setting a “use” property to True. Instead, the features are enabled when the user sets the component to a facility. This applies to gravity, initial stresses, initial strains, and initial state variables, and time-dependent boundary conditions (Dirichlet, Neumann, and point force).
7. Nondimensionalization of the problem eliminates the need to condition the fault constraints. The “mat_db” facility was removed.
8. The Dirichlet and Neumann boundary conditions now follow a more general time dependence. The names of the facilities and the names of the values in the spatial databases are, in most cases, different.
9. The FixedDOFDB has been renamed to ZeroDispDB in order to better reflect the type of spatial database.

Version 1.3.1

- Added stages to PETSc logging (`--petsc.log_summary`) to collect event logging into groups.

Bug fixes

- Fixed partitioning options. Partitioning options were ignored in the 1.3.0 release.
- Fixed assembling of Jacobian, residual, and fault sections across processors. This bug caused errors in the computation of the change in tractions over the fault surface.

Version 1.3.0

- New time stepping options

In addition to a uniform, user-specified time step, which is the default, there are two new time-stepping options. The user may supply a file with nonuniform time steps or, for quasi-static simulations, the user can request the code to compute the time step automatically. For the current bulk constitutive models, the automatically determined time step is independent of the deformation rate, so it is uniform.

- Initial stresses

Users may optionally supply an initial stress state for each material via a spatial database. The initial stress state can balance the gravitational body forces so that the model is in equilibrium without any deformation. This implementation of an initial stress state is a prelude to specifying an initial state for each material, which will be available in a future release.

Bug fixes

- Fixed labeling of physical properties in output for the Maxwell viscoelastic and generalized Maxwell viscoelastic materials (μ and λ were switched).

Migrating from v1.2.x to v1.3.x

The implementation of different options for controlling the time step requires adjusting input parameters from those used with PyLith 1.2. The time stepping is specified under the time-stepping formulation rather than the problem (i.e., one level deeper).

Version 1.2.0

- New Sieve implementation

The previous implementation of Sieve provided a very generalized implementation of data structures and operations for finite-element meshes. Switching to a more rigid implementation in the new implementation streamlined the data structures, resulting in a significant reduction in the memory use for storing the mesh. This leads to an overall reduction in memory use of 25-30% in many cases.

- Multiple kinematic ruptures

A single kinematic rupture on a fault has been replaced by a dynamic array of kinematic ruptures. This allows creation of an arbitrary number of kinematic ruptures on each fault surface. By using spatial databases to control the spatial and temporal extent of slip in each rupture independently, slip from different earthquake ruptures can overlap in space and/or time. Additionally, the rupture time at each location is specified with respect to the origin time of the corresponding earthquake rupture.

- New slip time functions

- Step slip time function (now the default)

This slip time function simplifies specifying instantaneous slip in a quasi-static simulation compared with using the Brune slip time function.

- Constant slip rate slip time function

This slip time function permits prescribing a constant slip rate on the fault surface.

- Gravitational body forces

Gravitational body forces are implemented (they are turned off by default). The direction and acceleration of gravity may be specified.

- Fixed Makefile.am files to not delete source files during “make clean” when building in the source tree.

Migrating from v1.1.x to 1.2.x

There are two new features in PyLith version 1.2 that require users to adjust input parameters from those used with PyLith 1.1. A dynamic array of kinematic rupture replaces a single kinematic rupture on a fault. Additionally, the default slip time function is now a step-function. This eliminates the need to specify a peak slip rate for quasi-static simulations. When using PyLith version 1.2 with a problem previously setup for PyLith 1.1, look for warnings about unknown components and settings in the screen output at the beginning of a run.

Version 1.1.2

- Fixed bug in output of solution over sub-domain boundary surfaces in parallel.
- Fixed Makefile.am files to include documentation files in source distribution.

Version 1.1.1

- Fixed Makefile.am files to include files missing from the source distribution.

Version 1.1.0

- New boundary conditions
 - Neumann (traction) boundary conditions
 - Absorbing boundary conditions via simple, tuned dampers
 - Dirichlet boundary conditions with displacement and/or velocity values
- New bulk constitutive models
 - Generalized Maxwell viscoelastic model
- New output implementation

The output to VTK files has been completely rewritten. This new implementation includes output of physical properties and state variables associated with the bulk constitutive models, as well as output of fault information (earthquake rupture parameters and slip and traction time histories). Additionally, the VTK file with the solution no longer includes fault related values- it contains just the displacement field over the domain as one would expect. A user can now also request output of the solution over an arbitrary number of sub-domains of the domain boundary, e.g., the ground surface. For each of these different kinds of output, the frequency of output and the values included can be customized by the user. The names of the VTK files and the variable names have also been adjusted to permit animation of solutions within most VTK visualization tools.

- New spatial database implementations

Spatialdata includes two new spatial database implementations. The SCECCVMHDB provides a seamless interface to the SCEC CVM-H seismic velocity model for elastic material properties. The UniformDB permits creating a spatial database for uniform values using only .cfg files or the command line; this eliminates the need to create a SimpleDB database file with one location.

- Dynamic arrays of components in Pyre

Pyre now contains dynamic arrays of components, eliminating the need for containers for materials, boundary conditions, and faults.

- Better consistency checking of input parameters
 - Uniqueness of material identifiers for materials and faults is enforced.
 - The material identifier of each cell in the mesh is checked to make sure it matches a material model.
 - Each boundary, interface condition, and output group is checked to make sure it exists in the mesh.

Bug fixes

- Fixed bug causing segmentation fault with multiple, non-overlapping Dirichlet boundary conditions applied to vertices.
- Fixed numerous bugs related to explicit time integration for dynamic problems.
- Eliminated several small memory errors.
- Fixed several bugs associated with writing VTK files in parallel.

Known issues

- PyLith still uses much more memory than PyLith 0.8 due to the current general Sieve implementation. A much more efficient, albeit less general Sieve implementation is under development. Additionally, distribution of the mesh will also be improved in a future release.

- The preconditioner for explicit time stepping provides relatively poor overall performance compared to a direct solve with traditional mass lumping. An appropriate preconditioner and traditional mass lumping will be supported in a future release.

Migrating from v1.0.x to 1.1.x

There are two new features in PyLith version 1.1 that require users to adjust input parameters from those used with PyLith 1.0. The elimination of containers in favor of the dynamic arrays of components present in the latest version of Pyre requires switching from setting the container to specifying the array of components on the command line or .cfg file. Additionally, the new implementation of output requires a completely new set of parameters. When using PyLith version 1.1 with a problem previously setup for PyLith 1.0, look for warnings about unknown components and settings in the output at the beginning of a run.

Version 1.0.2

- Performance optimizations have significantly reduced runtime and memory use relative to version 1.0.1. The default quadrature order for tetrahedral cells is now 1, which is appropriate for the default basis functions.
- Added checks to verify the compatibility of quadrature scheme for solid and cohesive cells.

Bug fixes

- In some cases, cohesive cells were not inserted into the finite-element mesh properly. The cells mixed together vertices from the different sides of the fault. A more efficient procedure for creating cohesive cells fixed this problem.
- Cell adjacency graph was created incorrectly which resulted in a poor quality of partitioning among processors.
- VTK output for meshes with N faults included cohesive cells for N-1 faults. Since VTK output does not understand cohesive cells, we now remove all cohesive cells from the VTK output.
- Using the SimpleDB in Spatialdata from Python limited interpolation to the “linear” scheme instead of allowing use of the “nearest” scheme. Setting the SimpleDB property to “nearest” and “linear” now works as expected.
- The reader for Spatialdata coordinate systems information did not correctly putback characters in the input stream, resulting in reading errors. The putback routines were fixed.
- Fault “up” and “normal” directions remained as string arrays when passed to the module, instead of being converted to float arrays.

Version 1.0.1

Bug fixes

- Cohesive cells lacked consistent orientation (inconsistent normals) in cases where cells were not ordered one side of the fault and then the other.
- Final slip of zero resulted in fault slip and slip increments of Nan.
- Parallel importing of meshes from LaGrit and Cubit lacked guards against all processors reading the files.

Version 1.0.0

- Code now includes both dynamic and quasi-static solutions.
- Completely rewritten in Python and C++, with bindings provided by Pyrex/Pyrexembed.
- Easier specification of simulations:
 - Parameters are all set using .cfg files (or .pml/command-line).

- Mesh may be directly imported from Cubit, LaGriT, or using PyLith mesh ASCII format.
- Material properties, fault dislocations, and BC are all given using spatial databases, which are independent of mesh discretization.
- Faults are now implemented using cohesive elements:
 - Easy specification of kinematic fault slip using a spatial database.
 - Cohesive elements generate offsets in the mesh corresponding to fault slip, which increase the accuracy of displacement fields near faults and facilitate visualization of fault slip.
 - Usage of cohesive elements will facilitate the upcoming addition of fault constitutive relations, where fault slip occurs in response to prescribed physics.
- Improved implicit time-stepping eliminates need to perform more than one iteration for linear rheologies.
- Code is now completely modular and object-oriented, which allows much easier addition of new features. Modules may be added without having to recompile the code.
- Features present in 0.8 that are not present in 1.0 that will be added in the near future.
 - Traction boundary conditions
 - Generalized Maxwell and Power-law Maxwell viscoelastic models

4.1.5 Development Plan

Future implementation of features is guided by several target applications, including

- Earthquake cycle modeling with quasi-static simulation of interseismic deformation and dynamic simulation of coseismic deformation.
- Inversion of geodetic data for slow slip events, fault creep, and long-term fault slip rates.
- Quasistatic and dynamic modeling of fluids and faulting.

We also strive to find the correct balance between adding new features and improving the code internals, performance, and maintainability. Short-term priorities focus on reimplementing features in version 2.2 that have not yet been migrated to the current formulations for governing equations and discretization.

Anticipated Releases

Note

Because we strictly follow the [semantic versioning guidelines](#), a minor release may get promoted to a major releases if we make changes to the public API (parameters). This can happen if realize that we should modify the parameters to improve maintainability or prepare for future changes.

Version 5.1 (November 2026)

- Reorganize finite-element kernels  [10%]
- Additional code clean up  [0%]

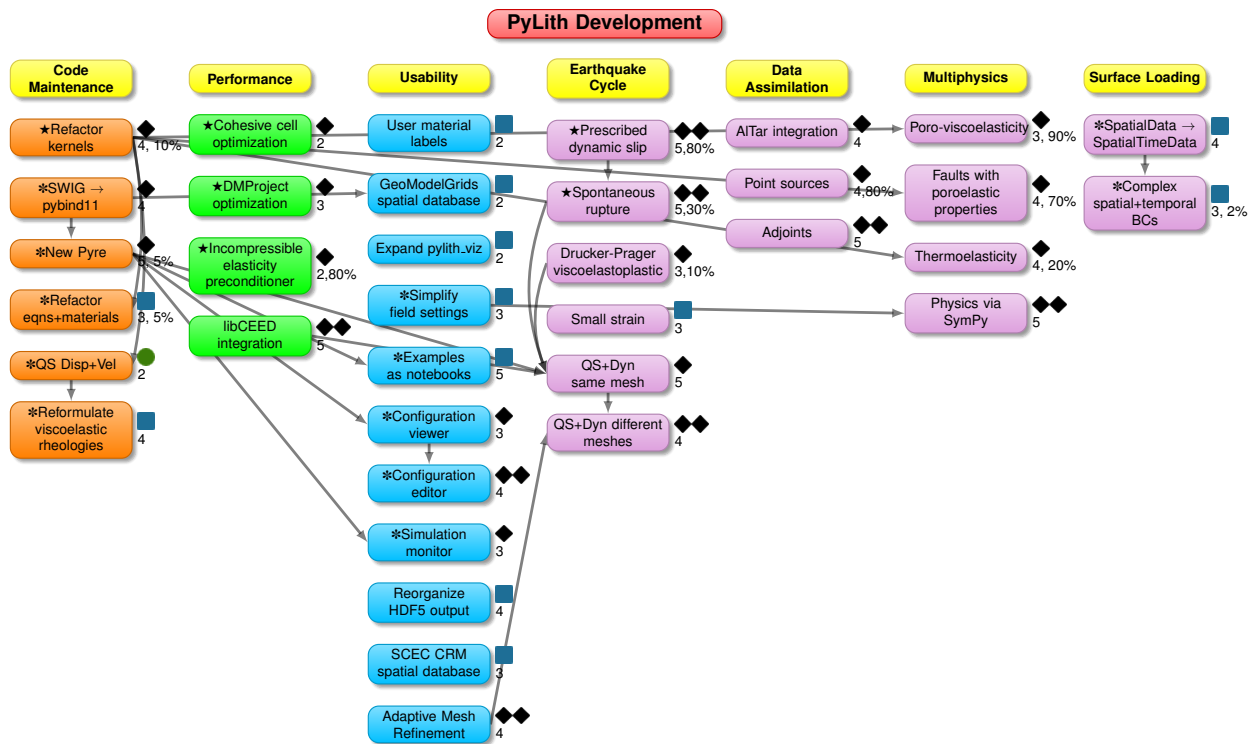


Fig. 4.1: PyLith development priorities by category. Information to the right of each feature indicates difficulty (colored icon), the relative amount of effort (lower left), and an estimate of how much is completed (lower right percentage). The arrows indicate the order in which some features must be implemented. Star symbols indicate high priority for implementation, and asterisk symbols indicate features that are part of the major code restructuring associated with use of the new Pyre implementation.

Version 5.2 (February 2027)

- Dynamic prescribed slip with diagonal Jacobian for explicit part of IMEX formulation   [80%]
- Better preconditioner for incompressible elasticity   [50%]

Version 5.3 (July 2027)

- Spontaneous rupture for quasistatic and dynamic simulations   [30%]

Version 6.0 (June 2028)

This version will include major changes to the public API associated with the switch to the current version of the Pyre Python framework.

- Convert from SWIG to pybind11  [0%]
- Add support for GeoModelGrids implementation of spatial databases for 3D seismic velocity models.  [0%]
- Update to current version of Pyre 
- Migrate examples to notebooks 
- Improve robustness of HDF5 output by opening/closing at each time step  [0%]
- Improve creation of auxiliary, diagnostic, and derived fields.
- More flexible specification of time-dependent boundary conditions.  [0%]
- Reimplement Drucker-Prager elastoplastic bulk rheology  [0%]

Version 7.0 (TBD)

- Output of fault rupture auxiliary subfields  [0%]
- Dirichlet boundary conditions with constraints on normal and tangential components.  [0%]
- Integration with libCEED for fast high order residual evaluation   \
- Add ability to output residual field during nonlinear solve for debugging  [0%]

Features for Future Releases

These features are not yet assigned to a release. They may be added to the planned releases based upon further discussion and contributions from the community.

- Adaptive mesh refinement  
- Faults with poroelastic properties   [80%]
- Elasticity with self-gravitation  [0%]
- Coupling of problems with compatible meshes  [10%]
Implement “injectors” for solution and state variables.

- Reimplementation of small strain formulation for elasticity  [20%]
- Moment tensor point sources [80%]
Moment tensor point sources provide a mesh independent deformation source that is better suited for Green's function calculations than slip on a fault surface via cohesive cells.
- Line/point fluid sources in poroelasticity   [20%]
- Consolidate HDF5 output into a single file 
- Drucker-Prager bulk rheology with relaxation to yield surface 
- Drucker-Prager bulk rheology with strain hardening/softening 
- Adjoint for data assimilation 
- Fault with both prescribed slip and spontaneous rupture 
Use fault constitutive model to control slip on fault except during episodes of prescribed slip. Need some way to describe when to turn on/off prescribed slip.

4.2 User Guide

4.2.1 Introduction

PyLith is portable, scalable software for simulation of crustal deformation across spatial scales ranging from meters to hundreds of kilometers and temporal scales ranging from milliseconds to thousands of years. Its primary applications are quasistatic and dynamic modeling of earthquake faulting.

Significant Recent Changes

See [Release Notes](#) for a summary of features and bug fixes for each release.

Version 5.0

- Use the VTU (XML) format for VTK files instead of the legacy ASCII format.
- Renamed `VertexGroup` to `BoundaryGroup` in `meshio.gmsh_utils` and changed default behavior to not be recursive (generate “face” groups, not “vertex” groups).
- Material description property is no longer used; a deprecation warning is printed to stdout if it is specified. This feature will be removed in v6.0.

Version 4.2

- Default filenames for progress monitor and parameters file are set from the simulation name like the other output files.
- Allow output on a finer resolution mesh than used in the simulation to facilitate accurate representation of fields with a basis order of 2 or greater.
- Fixed inconsistency in normal direction on fault surfaces. Orientation was correct but direction was flipped at some locations. This affected local slip direction and the resulting deformation close to the fault. This bug fix in a PETSc branch was not in version 4.1.3.

Version 4.1

- Improved runtime performance, including better preconditioners for elasticity with faults and poroelasticity
- 2D and 3D crustal strike-slip faults examples based on the 2019 Ridgecrest earthquake.
- 2D subduction zone outer-rise faulting example examining poroelastic response to bending stresses.
- `pylith_viz` utility for plotting PyLith results.
- Updated `examples/strikeslip-2d` and `examples/reverse-2d` to demonstrate use of uniform refinement and higher order discretizations to improve resolution of solution.

Version 4.0

- Changed name of fault Lagrange multiplier field for solution component in Python from `lagrange_fault` to `lagrange_multiplier_fault` to match name of solution field in C++.
- Removed support for importing meshes from LaGriT.
- Change in fault tractions are now included in the `fault_data_fields` for prescribed slip.
- Fault and boundary orientation directions are now included in the `info_fields` for simulation output.
- State variables are now included in the default `data_fields` for simulation output.
- The default solver settings use the PETSc proper orthogonal decomposition (POD) methodology for initial guess of solutions to improve convergence.

Version 3.0

- Major rewrite of the finite-element implementation to support higher order discretizations and flexible specification of the governing equations.
 - Use of pointwise functions to implement governing equations;
 - Higher order discretizations;
 - Problem specification independent of cell shape (quadrilateral vs triangle, hexahedron vs tetrahedron);
 - Incompressible elasticity;
 - Poroelasticity; and
 - Use of PETSc time-stepping algorithms.
- Simulations now require metadata, such as description, command line arguments, and PyLith version compatibility.
- New utilities
 - `pyre_doc.py` Display facilities and components available for a Pyre component;
 - `pylith_cfgsearch` Find files matching criteria for metadata; and
 - `pylith_runner` Run all simulations in a specified path.
- New examples
 - Simple 2-D and 3-D examples of Dirichlet and Neumann boundary conditions without faults;
 - Prescribed slip on a 2-D through-going strike-slip fault;
 - Gravitational body forces with elasticity and incompressible elasticity;
 - Distributed surface loads using Neumann boundary conditions; and

- Prescribed slip on a reverse fault with a splay fault.
- Documentation is now available online at <https://pylith.readthedocs.io>.
- Import finite-element meshes from Gmsh in addition to Cubit (Exodus II) and LaGriT.
- Updated to Python 3.
 - Pythia/Pyre, spatialdata, and PyLith have all been migrated to Python 3; and
 - The nemesis package has been merged into Pyre/Pyre.

History

PyLith 1.0 was the first version to allow the solution of both implicit (quasistatic) and explicit (dynamic) problems and was a complete rewrite of the original PyLith (version 0.8). PyLith 1.0 combines the functionality of EqSim [Aagaard *et al.*, 2001], [Aagaard *et al.*, 2001] and PyLith 0.8. PyLith 0.8 was a direct descendant of LithoMop and was the first version that ran in parallel, as well as providing several other improvements over LithoMop. LithoMop was the product of major re-engineering of Tecton, a finite-element code for simulating static and quasistatic crustal deformation. The major new features present in LithoMop included dynamic memory allocation and the use of the Pyre simulation framework and PETSc solvers. EqSim was written by Brad Aagaard to solve problems in earthquake dynamics, including rupture propagation and seismic wave propagation.

The release of PyLith 1.0 has been followed by additional releases that expand the number of features as well as improve performance. The PyLith 1.x series of releases allows the solution of both quasistatic and dynamic problems in one, two, or three dimensions. The code runs in either serial or parallel, and the design allows for relatively easy scripting using the Python programming language. Material properties and values for boundary and fault conditions are specified using spatial databases, which permit easy prescription of complex spatial variations of properties and parameters. Simulation parameters are generally specified through the use of simple ASCII files or the command line. At present, mesh information may be provided using a simple ASCII file (PyLith mesh ASCII format) or imported from Cubit or Gmsh. The elements currently available include a linear bar in 1D, linear triangles and quadrilaterals in 2D, and linear tetrahedra and hexahedra in 3D. Materials presently available include isotropic elastic, linear Maxwell viscoelastic, generalized Maxwell viscoelastic, power-law viscoelastic, and Drucker-Prager elastoplastic. Boundary conditions include Dirichlet (prescribed displacements and velocities), Neumann (traction), point forces, and absorbing boundaries. Cohesive elements are used to implement slip across interior surfaces (faults) with both kinematically-specified fault slip and slip governed by fault constitutive models. PyLith also includes an interface for computing static Green's functions for fault slip.

PyLith 2.0 replaced the finite-element data structures provided by the C++ Sieve implementation with those provided by the C DMPlex implementation. The newly developed DMPlex implementation by the PETSc developers conforms to the PETSc data manager (DM) interface, thereby providing tighter integration with other PETSc data structures, such as vectors and matrices. Other improvements include significantly reduced memory use and memory balancing.

PyLith 3.0 involves restructuring the code to permit more flexible specifications of governing equations and discretization and use of PETSc time-stepping algorithms. The finite-element integrations, constraints, and transformations are done through a suite of point-wise functions. The PETSc DMPlex interface calls these functions to perform the finite-element integrations.

PyLith is under active development and we expect a number of additions and improvements in the near future. Likely enhancements will include additional bulk and fault constitutive models, coupled quasistatic and dynamic simulations for earthquake cycle modeling, and coupling between elasticity, heat flow, and/or fluid flow.

PyLith Workflow

PyLith is one component in the process of investigating problems in tectonics (Fig. 4.2). Given a geological problem of interest, you must first provide a geometrical representation of the desired domain. Once you define the geometry, you need to discretize it as a finite-element mesh. PyLith presently provides three mesh importing options: Cubit Exodus format, Gmsh ASCII and binary files, and PyLith mesh ASCII format. The modeling of the physical processes of

interest is performed by a code such as PyLith. Present output consists of VTK or HDF5/Xdmf files which can be used by a number of visualization codes (e.g., ParaView, Visit, and Matlab).

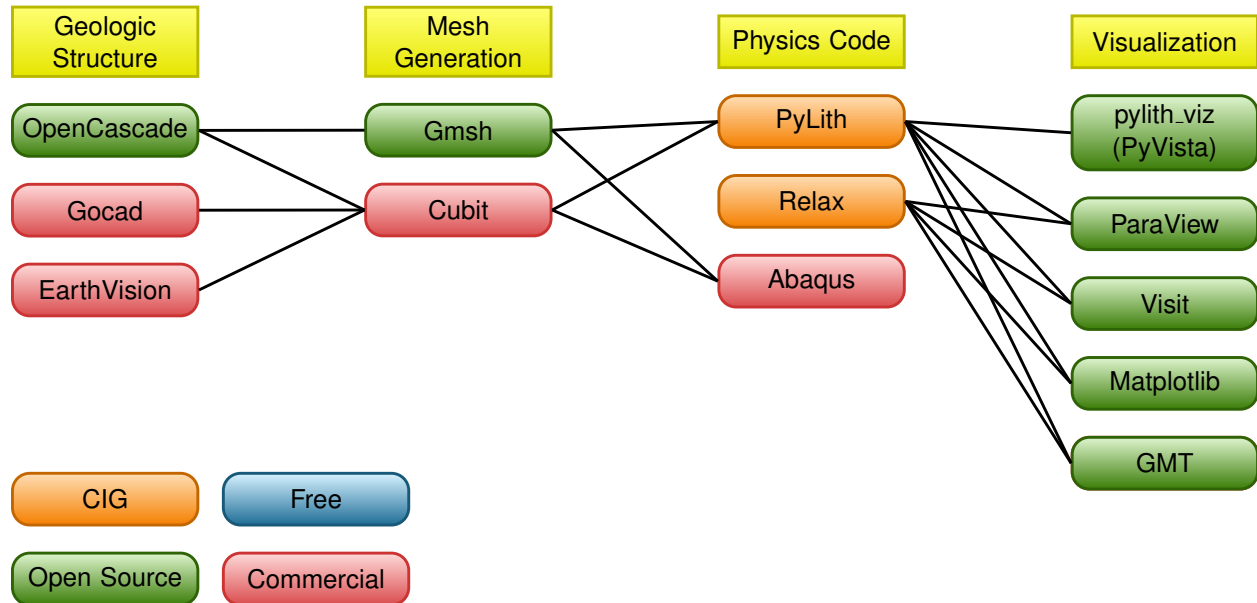


Fig. 4.2: Workflow involved in going from geologic structure to problem analysis.

Architecture

PyLith is separated into modules to encapsulate behavior and facilitate use across multiple applications. This allows expert users to replace functionality of a wide variety of components without recompiling or polluting the main code. PyLith employs external packages (see Fig. 4.3) to reduce development time and enhance computational efficiency.

PyLith is written in two programming languages. High-level code is written in Python; this rich, expressive interpreted language with dynamic typing reduces development time and permits flexible addition of user-contributed modules. This high-level code makes use of Pyre, a science-neutral simulation framework developed at Caltech by Michael Aivazis, to link the modules together at runtime and gather user-input. Low-level code is written in C++, providing fast execution while still allowing an object-oriented implementation. This low-level code relies on PETSc for finite-element data structures, time-stepping algorithms, and solvers. We use SWIG to create Python bindings for the C++ objects.

In writing PyLith 1.0, the code was designed to be object-oriented and modular. Each type of module is accessed through a specified interface (set of functions). This permits adding, replacing, and rewriting modules without affecting other parts of the code. This code structure simplifies code maintenance and development. Extending the set of code features is also easier, since developers can create new modules derived from the existing ones.

The current code design leverages Pyre and PETSc extensively. We use Pyre to define the simulation as a hierarchy of components and specify the parameters. Most of the PyLith source code pertains to implementing the geodynamics, such as the governing equations, bulk rheology, boundary conditions, and earthquake rupture via slip on faults.

Nemesis (Pyre subpackage) allows PyLith to run Python using the Message Passing Interface (MPI) for parallel processing. Additional, indirect dependencies (see Figure \ref{fig:pylith:dependencies}) include numpy (efficient operations on numerical arrays in Python), Proj.4 (geographic projections).

During development we implement three levels of testing: (1) unit testing, which occurs at the class/function level, (2) testing via the Method of Manufactured Solutions to test the finite-element implementation of the governing equations, and (3) full-scale testing, which involves complete PyLith simulations. We run these tests throughout the development cycle to expose bugs and isolate their origin. As additional changes are made to the code, the tests are rerun to help prevent introduction of new bugs.

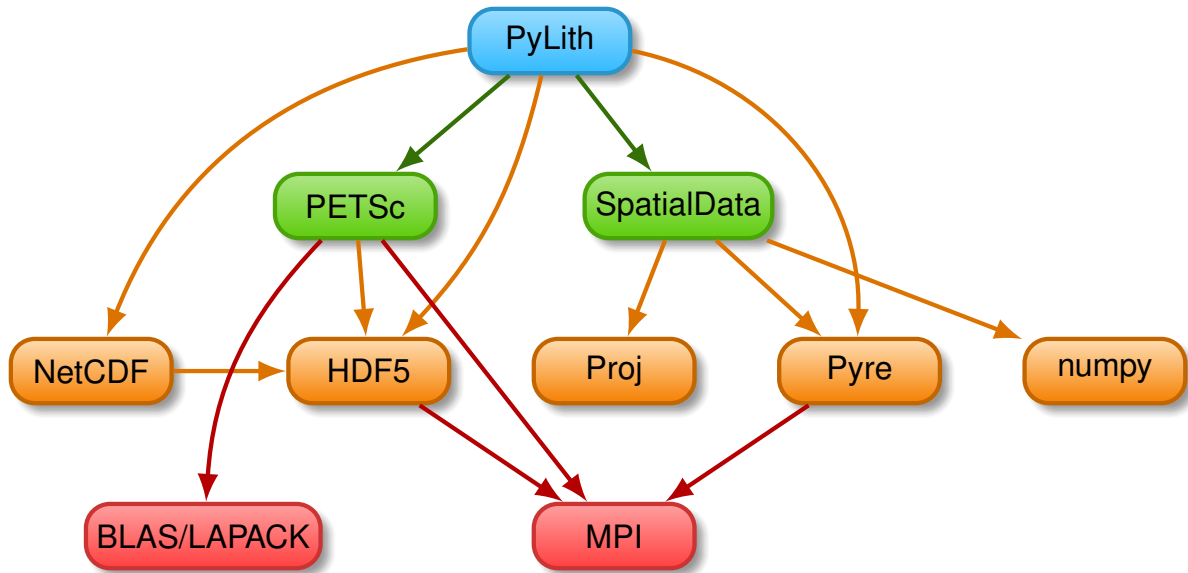


Fig. 4.3: Diagram of PyLith dependencies. PyLith makes direct use of several other packages, some of which have their own dependencies.

Additionally, we use community benchmarks, such as developed through the Southern California Earthquake Center for crustal deformation and dynamic rupture to determine the relative local and global error (see Chapter \vref{sec:benchmarks}).

Pyre

Pyre is an object-oriented environment capable of specifying and launching numerical simulations on multiple platforms, including Beowulf-class parallel computers and grid computing systems. Pyre allows the binding of multiple components such as solid and fluid models used in Earth science simulations, and different meshers. The Pyre framework enables the elegant setup, modification and launching of massively parallel solver applications.

Pyre as a framework is a combination of software and design philosophy that promotes the reuse of code. Applications based on Pyre will look similar to those written in any other object-oriented language.

The Pyre framework incorporates features aimed at enabling the scientific non-expert to perform tasks easily without hindering the expert. Target features for end users allow complete and intuitive simulation specification, reasonable defaults, consistency checks of input, good diagnostics, easy access to remote facilities, and status monitoring. Target features for developers include easy access to user input, a shorter development cycle, and good debugging support.

PETSc

PyLith 2.x and later make use of a set of data structures and routines in PETSc called `DMPlex`, which is still under active development. `DMPlex` provides data structures and routines for representing and manipulating computational meshes, and it greatly simplifies finite-element computations. `DMPlex` represents the topology of the domain. Zero volume elements, called cohesive cells, are inserted along all fault surfaces to implement kinematic (prescribed) or dynamic (constitutive model) implementations of fault slip. Material properties and other parameters are represented as scalar and vector fields over the mesh using vectors to store the values and sections to map vertices, edges, faces, and cells to indices in the vector. For each problem, functions are provided to calculate the residual and its Jacobian. All numerical integration is done in these functions, and parallel assembly is accomplished using the get/set closure paradigm of the `DMPlex` framework.

`PETSc`, the Portable, Extensible Toolkit for Scientific computation, provides a suite of routines for parallel, numerical

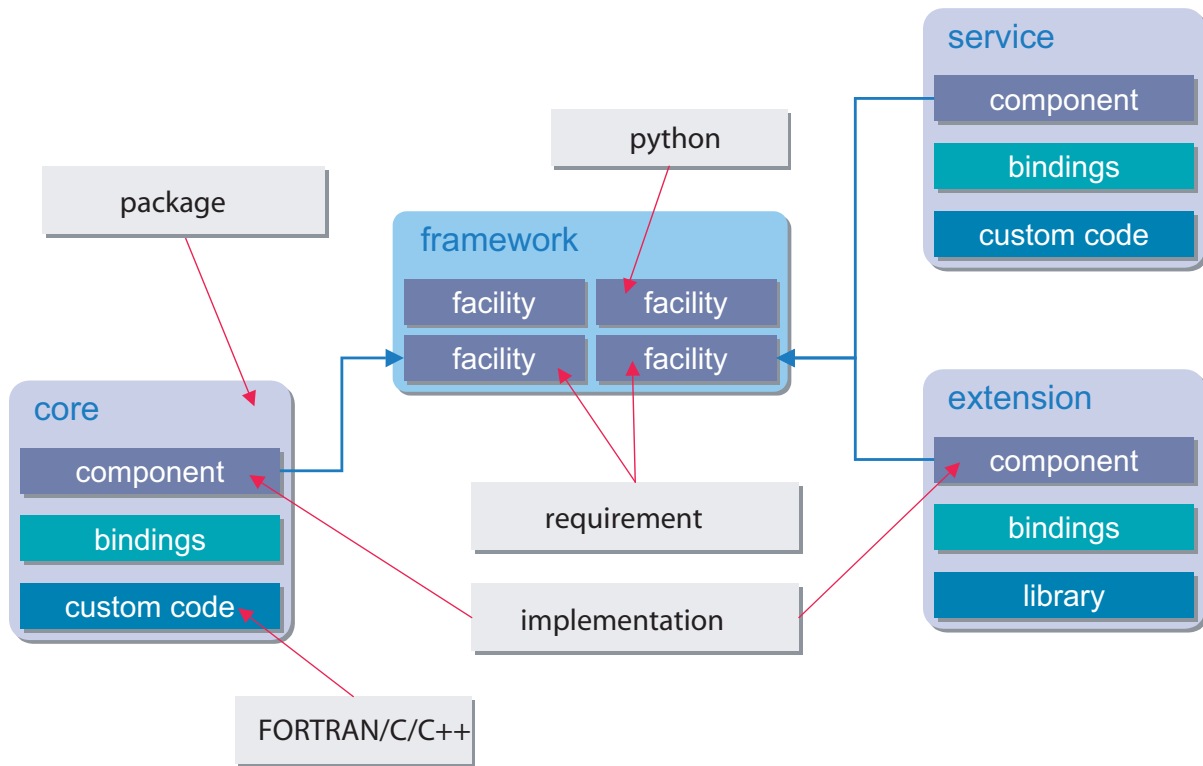


Fig. 4.4: Diagram of Pyre architecture. The integration framework is a set of cooperating abstract services.

solution of partial differential equations for linear and nonlinear systems with large, sparse systems of equations. PETSc includes time-stepping algorithms and solvers that implement a variety of Newton and Krylov subspace methods. It can also interface with many external packages, including ESSL, MUMPS, Matlab, ParMETIS, PVODE, and Hypr, thereby providing additional solvers and interaction with other software packages.

PETSc includes interfaces for FORTRAN 77/90, C, C++, and Python for nearly all of the routines, and PETSc can be installed on most Unix systems. Users can use PETSc parallel matrices, vectors, and other data structures for most parallel operations, eliminating the need for explicit calls to Message Passing Interface (MPI) routines. Many settings and options can be controlled with PETSc-specific command-line arguments, including selection of preconditioners, solvers, and generation of performance logs.

Getting Help and Reporting Bugs

The CIG forum has a category dedicated to CIG issues associated with PyLith. You can discuss PyLith, get help with installation, and more at <https://community.geodynamics.org/c/pylith/29>.

CIG uses *GitHub* for source control and bug tracking. If you find a bug in PyLith, please submit a bug report to the GitHub issue tracking system for PyLith <https://github.com/geodynamics/pylith/issues>. Of course, it is helpful to first check to see if someone else already submitted a report related to the issue; one of the CIG developers may have posted a work around to the problem. You can reply to a current issue by clicking on the issue title. To submit a new issue, click on the *New Issue* button.

When submitting a question about running a simulation, be sure to include the info:

1. Indicate which PyLith version you are using along with whether you are using the binary, container, or building from source.
2. Describe what you are trying to do
 - Overview of the problem and boundary conditions (diagrams are very helpful)

- 2-D or 3-D
 - Cell type (tri, quad, hex, or tet)
 - Type of fault: prescribed slip or spontaneous rupture
3. Attach the PyLith parameters .json file generated by running `pylith` or `pylith_dumpparameters`.
 4. Send the *entire* error message, not just what you think is important (entire log is best).

4.2.2 Installation

Fig. 4.5 provides a guide to select the appropriate method for installing PyLith. Installation of PyLith on a desktop or laptop machine is, in most cases, very easy. Binary packages have been created for Linux and macOS platforms. For Windows users, we recommend installing the Windows Subsystem for Linux and using the Linux binary (see instructions in Section *Windows Subsystem for Linux Setup*). You can also run PyLith inside a Docker container, which provides a virtual Linux environment on any platform that Docker supports, including Linux, macOS, and Windows. Installation of PyLith on other operating systems - or installation on a cluster - requires building the software from the source code, which can be difficult for inexperienced users. We have created a small utility called PyLith Installer that makes installing PyLith and all of its dependencies from source much easier.

Help for installing and using PyLith is available from both a CIG forum and the GitHub issue tracking system <https://github.com/geodynamics/pylith/issues>. See *Getting Help and Reporting Bugs* for more information.

Binary Package

The binaries are intended for users running on laptops or desktop computers (as opposed to clusters). The binaries contain the compilers and header files, so users wishing to extend the code can still use the binary and do not need to build PyLith and its dependencies from source. See *Contributing to PyLith* for more information on extending PyLith.

Binary executables are available for Linux (glibc 2.12 and later) and macOS (Intel 10.15 and later and arm64 11.0 and later) from the PyLith web page <https://geodynamics.org/resources/pylith/>. Users running Windows 10 build 14316 and later can install a Linux bash environment and use the PyLith binary for Linux (see Section *Windows Subsystem for Linux Setup* for more information).

Tip

On Linux systems you can check which version of glibc you have by running `ldd-version`

On macOS systems you can check the operating system version by clicking on the Apple icon and *About This Mac*.

Tip

When running PyLith, we recommend starting with a clean environment to avoid conflicts from any other software you may have installed.

```
PATH=/usr/bin:/bin:/sbin:/usr/sbin
unset PYTHONPATH LD_LIBRARY_PATH
```

You should not have a Python virtual environment activated as PyLith uses its own Python virtual environment.

Linux and macOS

1. Open a terminal window and change to the directory where you want to place the distribution.

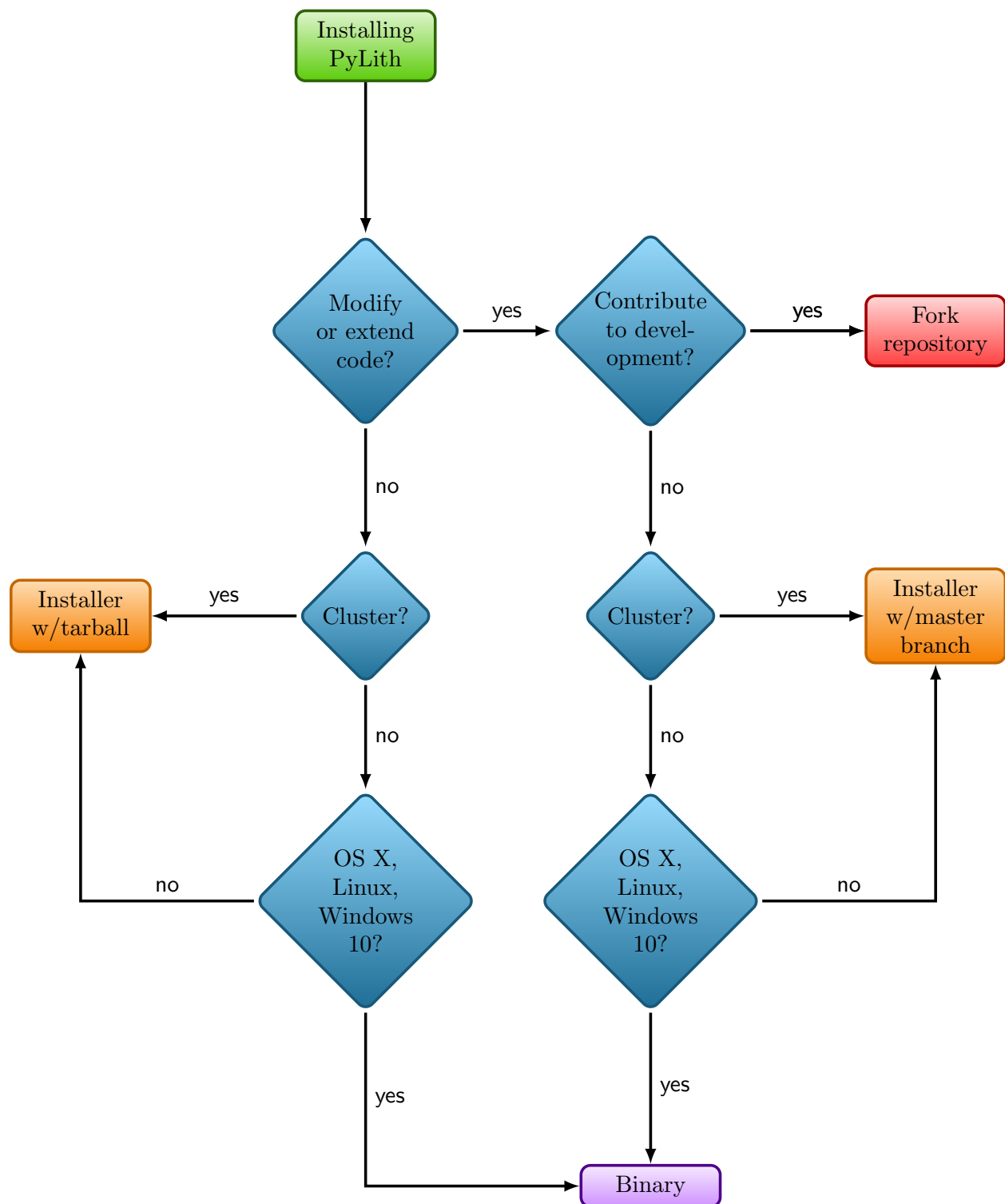


Fig. 4.5: Guide for selecting the appropriate installation choice based on a hardware and intended use. The installation options are discussed in more detail in the following sections.


```
cd $HOME
mkdir pylith
cd pylith
```

2. Download the Linux or macOS tarball from the PyLith web page <https://geodynamics.org/resources/pylith/supportingdocs/>, and save it to the desired location, e.g., \$HOME/pylith.
3. Unpack the tarball.

```
# Linux 64-bit
tar -xzf pylith-5.0.1-linux-x86_64.tar.gz

# macOS
tar -xzf pylith-5.0.1-macOS-10.15-x86_64.tar.gz
```

4. Set environment variables. The provided `setup.sh` script only works if you are using a bash compatible shell. If you are using a different shell, you will need to alter how the environment variables are set in `setup.sh`. Move to the newly extracted directory and run the following.

```
$ source setup.sh
Ready to run PyLith.
```

Tip

To bypass macOS quarantine restrictions, simply use command line program `curl` to download the tarball from within a terminal rather than using a web browser.

```
curl -L -O https://github.com/geodynamics/pylith/releases/download/v5.0.1/pylith-5.0.1-macOS-10.15-x86_64.tar.gz
```

Alternatively, if you do download the tarball using a web browser, after you unpack the tarball you can remove the macOS quarantine flags using the following commands (requires Administrator access):

```
# Show extended attributes
xattr ./pylith-5.0.1-macOS-10.15-x86_64

# Output should be
com.apple.quarantine

# Remove quarantine attributes
sudo xattr -r -d com.apple.quarantine ./pylith-5.0.1-macOS-10.15-x86_64
```

Warning

The binary distribution contains PyLith and all of its dependencies. If you have any of this software already installed on your system, you need to be careful in setting up your environment so that preexisting software does not conflict with the PyLith binary. By default the `setup.sh` script will prepend to the `PATH` and `PYTHONPATH` (for macOS and Linux) and `LD_LIBRARY_PATH` (for Linux) environment variables. This will prevent most conflicts.

Windows Subsystem for Linux Setup

PyLith is developed within the Unix/Linux framework, and we do not provide a native PyLith binary distribution for Windows. The preferred approach for installing PyLith on a computer running Windows is to enable use of a Linux subsystem. This permits use of the PyLith Linux x86_64 binary within the bash environment.

To enable the Linux subsystem on Windows 10 build 14316 and later (users running an earlier Windows build should use the PyLith Docker container):

1. Go to *Settings* → *Security* .
2. Under *For developers* select *Developer mode*. This step should not be required for Windows build 16215 and later.
3. Go to *Control Panel* → *Programs* → *Turn Windows Features On or Off*.
4. Enable *Windows Subsystem for Linux* and click *OK* .
5. Restart the computer.
6. Go to *Start* → *bash*. You will be prompted to download “Bash on Ubuntu on Windows” from the Windows Store. Create a user account and password for the bash environment.
7. Install the PyLith Linux x86 binary within the bash environment following the instructions for installing the PyLith binary for Linux. You will run PyLith within the bash environment just like you would for a Linux operating system.

WSL Setup for Gmsh and PyVista

The PyLith Linux binary package includes Gmsh and PyVista, which rely on several Linux libraries that are not included with PyLith. In this section, we outline how to install these libraries based on user experiences; we do not have access to a Windows Subsystem for Linux for testing.

Listing 4.1: Installation of extra libraries needed for Gmsh and PyVista.
You only need to run these commands once per WSL installation.

```
# Update Linux
sudo apt-get update

# Install X libraries
sudo apt-get install libgl1-mesa-dri libglu1-mesa x11-apps gnome-terminal -y

# Make drivers visible to locations where Gmsh and PyVista expect them to be (/usr/lib/
↳dri)
sudo ln -s /usr/lib/x86_64-linux-gnu/dri /usr/lib/dri

# Install libstdc++6
sudo apt-get install libstdc++6
```

Important

Instead of running `source setup.sh` you will need to run `source setup.sh add-wsl-libs`.

Tip

If you are able to run Gmsh or `pylith_viz`, but do not see any graphics, then you likely need to turn on software rendering by setting the environment variable `LIBGL_ALWAYS_SOFTWARE=1` in your shell. You can set this environment variable or have the `setup.sh` script set it for you with the `enable-software-rendering` argument.

```
# Add WSL libraries and turn on libGL software rendering.
source setup.sh add-wsl-libs enable-software-rendering
```

See also

Refer to [Microsoft Tutorial: Run Linux GUI apps on the WSL](#) for additional information about running graphic user interface applications within the Windows Subsystem for Linux.

Extending PyLith or Integrating Other Software Into PyLith

New in v3.0.0.

We strongly recommend using the [PyLith development environment Docker container](#) if you want to extend PyLith or integrate PyLith into other software.

Source Installation

PyLith depends on a number of other packages (see [Fig. 4.3](#)). This complicates building the software from the source code. In many cases some of the packages required by PyLith are available as binary packages. On the one hand, using the binary packages for the dependencies removes the burden of configuring, building, and installing these dependencies, but that can come with its own host of complications if consistent compiler and configuration settings are not used across all of the packages on which PyLith depends. This is usually not an issue with Linux distributions, such as Fedora, Ubuntu, and Debian that have good quality control; it can be an issue with macOS package managers, such as Fink, MacPorts, and Homebrew, where there is limited enforcement of consistency across packages. Nevertheless, PyLith can be built on most systems provided the instructions are followed carefully. PyLith is developed and tested on Linux and macOS.

A small utility, PyLith Installer, removes most of the obstacles in building PyLith and its dependencies from source. For each package this utility downloads the source code, configures it, builds it, and installs it. This insures that the versions of the dependencies are consistent with PyLith and that the proper configure arguments are used. The minimum requirements for using the PyLith installer are a C compiler, `tar`, and `wget` or `curl`. Detailed instructions for how to install PyLith using the installer are included in the installer distribution, which is available from the PyLith web page <https://geodynamics.org/resources/pylith/supportingdocs/>.

Verifying PyLith Installation

The easiest way to verify that PyLith has been installed correctly is to run one or more of the examples supplied with the binary and source code. In the binary distribution, the examples are located in `src/pylith-5.0.1/examples` while in the source distribution, they are located in `pylith-5.0.1/examples`. [Examples](#) discusses how to run and visualize the results for the examples. To run the example discussed in Section [Axial and Shear Deformation \(2D Box\)](#):

```
$ cd examples/box-2d
$ pylith step01_axialdisp.cfg

# A bunch of stuff will be written to stdout. The last few lines should be:
>> .../lib/python3.12/site-packages/pylith/problems/Problem.py:201:finalize
-- info (application-flow)
-- Finalizing problem.
```

If you run PyLith in a directory without any input, you will get the error message:

```
$ pylith

# Output
>> {default}::
-- pyre.inventory(error)
-- metadata.description <- ''
-- Nonempty string required.
>> {default}::
-- pyre.inventory(error)
-- metadata.arguments <- '[]'
-- List of command line arguments required.
>> {default}::
-- pyre.inventory(error)
-- metadata.pylith_version <- '[]'
-- List of PyLith version constraints required.
>> {default}::
-- pyre.inventory(error)
-- meshimporter.meshioascii.filename <- ''
-- Filename for ASCII input mesh not specified. To test PyLith, run an example as_
discussed in the manual.
>> {default}::
-- pyre.inventory(error)
-- timedependent.problem_defaults.name <- ''
-- Missing required property 'name' in default options for problem.
pylithapp: configuration error(s)
```

This indicates that at a very minimum metadata and the finite-element mesh file must be specified in order to run PyLith.

Configuration on a Cluster

If you are installing PyLith on a cluster with a batch system, you can configure Pyre such that the `pylith` command automatically submits jobs to the batch queue. Pyre contains support for the LSF, PBS, SGE, and Globus batch systems. Properly configured, Pyre can handle job submissions automatically, insulating users from the details of the batch system and the site configuration.

See *Running in Parallel on a Cluster* for more information on how to use PyLith on a cluster with a job scheduling system.

4.2.3 Governing Equations

This chapter presents the solution schemes we use for solving variations of the elasticity equation using the finite-element method. In all of our derivations, we use the notation described in *Mathematical notation*.

Table 4.1: Mathematical notation

Symbol	Description
$\vec{a}$	Vector field a
$\mathbf{a}$	Second order tensor field a
$\vec{u}$	Displacement vector field
$\vec{v}$	Velocity vector field
$\vec{d}$	Fault slip vector field
$\vec{f}$	Body force vector field
$\vec{\tau}$	Traction vector field
σ	Stress tensor field
$\vec{n}$	Normal vector field
ρ	Mass density scalar field

Finite-Element Formulation with PETSc

Within the PETSc solver framework, we want to solve a system of partial differential equations in which the weak form can be expressed as $F(t, s, \dot{s}) = G(t, s)$, $s(t_0) = s_0$, where F and G are vector functions, t is time, and s is the solution vector.

Using the finite-element method¹ we manipulate the weak form of the system of equations involving a vector field $\vec{u}$ into integrals over the domain Ω matching the form,

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{f}_0(t, s, \dot{s}) + \nabla \vec{\psi}_{trial}^u : \mathbf{f}_1(t, s, \dot{s}) d\Omega = \int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{g}_0(t, s) + \nabla \vec{\psi}_{trial}^u : \mathbf{g}_1(t, s) d\Omega, \quad (4.1)$$

where $\vec{\psi}_{trial}^u$ is the trial function for field $\vec{u}$, $\vec{f}_0$ and $\vec{g}_0$ are vectors, and $\mathbf{f}_1$ and $\mathbf{g}_1$ are tensors. With multiple partial differential equations we will have multiple equations of this form, and the solution vector s , which we usually write as $\vec{s}$, will be composed of several different fields, such as displacement $\vec{u}$, velocity $\vec{v}$, pressure p , and temperature T . Boundary conditions will also contribute similar terms with integrals over the corresponding boundaries.

For consistency with the PETSc time stepping formulation, we call $G(t, s)$ the RHS function and call $F(t, s, \dot{s})$ the LHS (or I) function. Likewise, the Jacobian of $G(t, s)$ is the RHS Jacobian and the Jacobian of $F(t, s, \dot{s})$ is the LHS Jacobian. Using a finite-element discretization we break up the domain and boundary integrals into sums over the cells and boundary faces/edges, respectively. Using numerical quadrature those sums in turn involve sums over the values at the quadrature points with appropriate weights. Thus, computation of the RHS function boils down to pointwise evaluation of $\vec{g}_0(t, s)$ and $\mathbf{g}_1(t, s)$, and computation of the LHS function boils down to pointwise evaluation of $\vec{f}_0(t, s, \dot{s})$ and $\mathbf{f}_1(t, s, \dot{s})$.

Jacobian

The LHS Jacobian $J_F = \frac{\partial F}{\partial s} + s_{tshift} \frac{\partial F}{\partial \dot{s}}$ and the RHS Jacobian $J_G = \frac{\partial G}{\partial s}$, where s_{tshift} arises from the temporal discretization. We put the Jacobians for each equation into the form:

$$\begin{aligned} J_F &= \int_{\Omega} \vec{\psi}_{trial} \cdot \mathbf{J}_{f0}(t, s, \dot{s}) \cdot \vec{\psi}_{basis} + \vec{\psi}_{trial} \cdot \mathbf{J}_{f1}(t, s, \dot{s}) : \nabla \vec{\psi}_{basis} + \nabla \vec{\psi}_{trial} : \mathbf{J}_{f2}(t, s, \dot{s}) \cdot \vec{\psi}_{basis} + \nabla \vec{\psi}_{trial} : \mathbf{J}_{f3}(t, s, \dot{s}) : \nabla \vec{\psi}_{basis} \\ J_G &= \int_{\Omega} \vec{\psi}_{trial} \cdot \mathbf{J}_{g0}(t, s) \cdot \vec{\psi}_{basis} + \vec{\psi}_{trial} \cdot \mathbf{J}_{g1}(t, s) : \nabla \vec{\psi}_{basis} + \nabla \vec{\psi}_{trial} : \mathbf{J}_{g2}(t, s) \cdot \vec{\psi}_{basis} + \nabla \vec{\psi}_{trial} : \mathbf{J}_{g3}(t, s) : \nabla \vec{\psi}_{basis} d\Omega, \end{aligned} \quad (4.2)$$

¹ Some resources for learning about the finite-element method include Zienkiewicz and Taylor [2000], Taylor [2003], Farrell [2021], and Cotter and Ham [2023].

where $\vec{\psi}_{basis}$ is a basis function. Expressed in index notation the Jacobian coupling solution field components s_i and s_j is

$$J^{s_i s_j} = \int_{\Omega} \psi_{trial_i} J_0^{s_i s_j} \psi_{basis_j} + \psi_{trial_i} J_1^{s_j s_j l} \psi_{basis_{j,l}} + \psi_{trial_{i,k}} J_2^{s_i s_j k} \psi_{basis_j} + \psi_{trial_{i,k}} J_3^{s_i s_j k l} \psi_{basis_{j,l}} d\Omega, \quad (4.3)$$

It is clear that the tensors J_0 , J_1 , J_2 , and J_3 have various sizes: $J_0(n_i, n_j)$, $J_1(n_i, n_j, d)$, $J_2(n_i, n_j, d)$, $J_3(n_i, n_j, d, d)$, where n_i is the number of components in solution field s_i , n_j is the number of components in solution field s_j , and d is the spatial dimension. Alternatively, expressed in discrete form, the Jacobian for the coupling between solution fields s_i and s_j is

$$J^{s_i s_j} = J_0^{s_i s_j} + J_1^{s_i s_j} B + B^T J_2^{s_i s_j} + B^T J_3^{s_i s_j} B, \quad (4.4)$$

where B is a matrix of the derivatives of the basis functions and B^T is a matrix of the derivatives of the trial functions.

Important

See <https://www.mcs.anl.gov/petsc/petsc-master/docs/manualpages/FE/PetscFEIntegrateJacobian.html> for the ordering of indices in the Jacobian pointwise functions.

PETSc TS Notes

Explicit Time Stepping

Explicit time stepping with the PETSc TS requires $F(t, s, \dot{s}) = \dot{s}$.

- We do not specify the functions $\vec{f}_0(t, s, \dot{s})$ and $\mathbf{f}_1(t, s, \dot{s})$ because the PETSc TS will assume $F(t, s, \dot{s}) = \dot{s}$ if no LHS (or I) function is given.
- The PETSc TS will verify that the LHS (or I) function is null.
- We also do not specify J_F or J_G .
- This leaves us with only needing to specify $\vec{g}_0(t, s)$ and $\mathbf{g}_1(t, s)$.

For explicit time stepping with the PETSc TS, we need $F(t, s, \dot{s}) = \dot{s}$. Using a finite-element formulation for elastodynamics, $F(t, s, \dot{s})$ generally involves integrals of the inertia over the domain. It is tempting to simply move these terms to the RHS, but that introduces inertial terms into the boundary conditions, which makes them less intuitive. Instead, we transform our equation into the form $\dot{s} = G^*(t, s)$ where $G^*(t, s) = M^{-1}G(t, s)$. We take M to be a lumped (diagonal) matrix, so that M^{-1} is trivial to compute. In computing the RHS function, $G^*(t, s)$, we compute $G(t, s)$, then compute M and M^{-1} , and then $M^{-1}G(t, s)$. For the elasticity equation with inertial terms, M contains the mass matrix.

Implicit Time Stepping

The LHS (or I) function is associated with implicit time-stepping. When using implicit time-stepping, we place all of the terms on the LHS. Even though placing all of the terms on the LHS sometimes requires different pointwise functions for implicit and explicit time stepping, it minimizes the number of pointwise functions needed for implicit time stepping. If no RHS function is given, then the PETSc TS assumes $G(t, s) = 0$, so we only need to specify $F(t, s, \dot{s})$ and J_f .

Implicit-Explicit Time Stepping

For implicit-explicit time stepping algorithms, the equations integrated with explicit time stepping have $\dot{s}$ as the LHS function, and the equations integrated with implicit time stepping have 0 as the RHS function.

Elasticity

Derivation of Elasticity Equation

For completeness we start our discussion of the governing equations with a derivation of the elasticity equation. Consider domain Ω bounded by boundary Γ . Applying a Lagrangian description of the conservation of momentum gives

$$\frac{\partial}{\partial t} \int_{\Omega} \rho(\vec{x}) \frac{\partial \vec{u}}{\partial t} d\Omega = \int_{\Omega} \vec{f}(\vec{x}, t) d\Omega + \int_{\Gamma} \vec{\tau}(\vec{x}, t) d\Gamma. \quad (4.5)$$

The traction vector field is related to the stress tensor through

$$\vec{\tau}(\vec{x}, t) = \boldsymbol{\sigma}(\vec{u}) \cdot \vec{n}, \quad (4.6)$$

where $\vec{n}$ is the outward normal vector to Γ . Substituting into equation (4.5) yields

$$\frac{\partial}{\partial t} \int_{\Omega} \rho(\vec{x}) \frac{\partial \vec{u}}{\partial t} d\Omega = \int_{\Omega} \vec{f}(\vec{x}, t) d\Omega + \int_{\Gamma} \boldsymbol{\sigma}(\vec{u}) \cdot \vec{n} d\Gamma. \quad (4.7)$$

Applying the divergence theorem,

$$\int_{\Omega} \nabla \cdot \vec{a} d\Omega = \int_{\Gamma} \vec{a} \cdot \vec{n} d\Gamma, \quad (4.8)$$

to the boundary integral results in

$$\frac{\partial}{\partial t} \int_{\Omega} \rho(\vec{x}) \frac{\partial \vec{u}}{\partial t} d\Omega = \int_{\Omega} \vec{f}(\vec{x}, t) d\Omega + \int_{\Omega} \nabla \cdot \boldsymbol{\sigma}(\vec{u}) d\Omega, \quad (4.9)$$

which we can rewrite as

$$\int_{\Omega} \left(\rho(\vec{x}) \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(\vec{x}, t) - \nabla \cdot \boldsymbol{\sigma}(\vec{u}) \right) d\Omega = \vec{0}. \quad (4.10)$$

Because the domain Ω is arbitrary, the integrand must be the zero vector at every location in the domain, so that we end up with

$$\rho(\vec{x}) \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(\vec{x}, t) - \nabla \cdot \boldsymbol{\sigma} = \vec{0} \text{ in } \Omega, \quad (4.11)$$

$$\boldsymbol{\sigma}(\vec{u}) \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_{\tau}, \quad (4.12)$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u, \text{ and} \quad (4.13)$$

$$\vec{u}^+ - \vec{u}^- = \vec{d} \text{ on } \Gamma_f. \quad (4.14)$$

We specify tractions, $\vec{\tau}$, on boundary Γ_f , displacements, $\vec{u}_0$, on boundary Γ_u , and slip, $\vec{d}$, on fault interface Γ_f .

Nondimensionalization

Starting with the elasticity equation,

$$\rho(\vec{x}) \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(\vec{x}, t) - \nabla \cdot \boldsymbol{\sigma} = \vec{0} \text{ in } \Omega, \quad (4.15)$$

$$\boldsymbol{\sigma}(\vec{u}) \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_{\tau}, \quad (4.16)$$

$$\vec{u}^+ - \vec{u}^- = \vec{d} \text{ on } \Gamma_f. \quad (4.17)$$

we define nondimensional values:

$$\vec{x}^* = \frac{\vec{x}}{x_o}, \quad (4.18)$$

$$\vec{u}^* = \frac{\vec{u}}{u_o}, \quad (4.19)$$

$$\rho^* = \frac{\rho}{\rho_o}, \quad (4.20)$$

$$\vec{f}^* = \frac{\vec{f}}{f_o}, \quad (4.21)$$

$$\boldsymbol{\sigma}^* = \frac{\boldsymbol{\sigma}}{\sigma_o}, \quad (4.22)$$

$$\vec{\tau}^* = \frac{\vec{\tau}}{\sigma_o}. \quad (4.23)$$

We also recognize that

$$\nabla^* = x_o \nabla. \quad (4.24)$$

Substituting into the equations, we have

$$\rho_o \rho^*(\vec{x}^*) \frac{u_o}{t_o^2} \frac{\partial^2 \vec{u}^*}{\partial t^{*2}} - f_o \vec{f}^*(\vec{x}^*, t^*) - \frac{\sigma_o}{x_o} \nabla^* \cdot \boldsymbol{\sigma}^* = \vec{0} \text{ in } \Omega, \quad (4.25)$$

$$\sigma_o \boldsymbol{\sigma}^*(\vec{u}^*) \cdot \vec{n} = \sigma_o \vec{\tau}^*(\vec{x}^*, t^*) \text{ on } \Gamma_\tau, \quad (4.26)$$

$$u_o (\vec{u}^{*+} - \vec{u}^{*-}) = u_o \vec{d}^* \text{ on } \Gamma_f. \quad (4.27)$$

For the second two equations, the nondimensional scales in for the terms in each equation are consistent, so we will limit our discussion to the first equation. Grouping terms and multiplying by $\frac{x_o}{\sigma_o}$, we have

$$\left(\rho_o \frac{u_o x_o}{t_o^2 \sigma_o} \right) \rho^*(\vec{x}^*) \frac{\partial^2 \vec{u}^*}{\partial t^{*2}} - \left(f_o \frac{x_o}{\sigma_o} \right) \vec{f}^*(\vec{x}^*, t^*) - \nabla^* \cdot \boldsymbol{\sigma}^* = \vec{0} \text{ in } \Omega. \quad (4.28)$$

All terms should be nondimensional, which implies

$$f_o = \frac{\sigma_o}{x_o}, \quad (4.29)$$

$$\rho_o = \frac{t_o^2 \sigma_o}{u_o x_o}. \quad (4.30)$$

We want to determine the stress scale, σ_o . Considering isotropic, linear elasticity we have

$$\boldsymbol{\sigma} = \boldsymbol{C} : \boldsymbol{\epsilon} = \boldsymbol{C} : \frac{1}{2} (\nabla + \nabla^T) \vec{u}. \quad (4.31)$$

Substituting in our nondimensional values yields

$$\sigma_o \boldsymbol{\sigma}^* = \mu_o \boldsymbol{C}^* : \frac{u_o}{x_o} \frac{1}{2} (\nabla^* + \nabla^{*T}) \vec{u}^*. \quad (4.32)$$

We recognize that for the equation to be nondimensional,

$$\sigma_o = \mu_o \frac{u_o}{x_o}. \quad (4.33)$$

Returning to the expression for ρ_o and substituting in the expression for σ_o , we have

$$\rho_o = \frac{t_o^2 \sigma_o}{u_o x_o}, \quad (4.34)$$

$$\rho_o = \mu_o \frac{t_o^2}{x_o^2}. \quad (4.35)$$

Inertia

The scale of the inertial term is $\Pi_{inertia} = \frac{\rho_o x_o^2}{\mu_o t_o^2}$. If this term is $O(1)$, then we should include inertia and solve the dynamic form of the elasticity equation. If this term is very small, then we can neglect inertia and solve the quasistatic form of the elasticity equation.

If the time scale equals the time it takes the shear wave ($v_o^2 = \frac{\mu_o}{\rho_o}$) to propagate over the length scale, then

$$t_o = \frac{x_o}{v_o}, \quad (4.36)$$

$$t_o^2 = \frac{x_o^2}{v_o^2}, \quad (4.37)$$

$$t_o^2 = \frac{x_o^2 \rho_o}{\mu_o}. \quad (4.38)$$

Substituting into the expression for $\Pi_{inertia}$, we have

$$\Pi_{inertia} = \frac{\rho_o x_o^2}{\mu_o t_o^2}, \quad (4.39)$$

$$\Pi_{inertia} = \frac{\rho_o x_o^2}{\mu_o} \frac{\mu_o}{\rho_o x_o^2}, \quad (4.40)$$

$$\Pi_{inertia} = 1. \quad (4.41)$$

In this case, the inertial term is important, and we have the dynamic case with propagating seismic waves.

If the time scale is much larger than the time it takes the shear wave to propagate over the length scale, then

$$t_o \gg \frac{x_o}{v_o}, \quad (4.42)$$

$$t_o^2 \gg \frac{x_o^2 \rho_o}{\mu_o}. \quad (4.43)$$

Substituting into the expression for $\Pi_{inertia}$, we have

$$\Pi_{inertia} = \frac{\rho_o x_o^2}{\mu_o t_o^2}, \quad (4.44)$$

$$\Pi_{inertia} \ll \frac{\rho_o x_o^2}{\mu_o} \frac{\mu_o}{\rho_o x_o^2}, \quad (4.45)$$

$$\Pi_{inertia} \ll 1. \quad (4.46)$$

In this case, the inertial term is negligible, and we have quasistatic elasticity.

Note

In PyLith v4 and earlier, we used a displacement scale equal to the length scale. Using separate length and displacement scales facilitates accurately solving for displacements many orders of magnitude smaller than the length scale. It also separates the stress scale used to nondimensionalize stress, tractions, and fluid pressure from the rigidity scale.

Table 4.2: Nondimensionalization of elasticity.

Quantity	Scale
x_o	Length scale
u_o	Displacement scale
μ_o	Rigidity scale
t_o	Time scale
$\sigma_o = \mu_o \frac{u_o}{x_o}$	Stress scale
$f_o = \frac{\sigma_o}{x_o}$	Body force scale
$\rho_o = \mu_o \frac{t_o^2}{x_o^2}$	Density scale

Quasistatic Elasticity

There is no inherent length scale in quasistatic elasticity. Consequently, the solver is relatively insensitive to the length scale for elasticity problems without a fault; however, the solver will be most robust when the length scale roughly matches the size of the domain.

Important

Our default preconditioner for elasticity with prescribed slip requires that the length scale match the discretization size. If your discretization size differs substantially from the default length scale (1.0 km), then you will need to set the length scale.

Infinitesimal Strain and No Faults

We begin with the elasticity equation including the inertial term,

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(\vec{x}, t) - \nabla \cdot \boldsymbol{\sigma}(\vec{u}) = \vec{0} \text{ in } \Omega, \quad (4.47)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau, \quad (4.48)$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u, \quad (4.49)$$

where $\vec{u}$ is the displacement vector, ρ is the mass density, $\vec{f}$ is the body force vector, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, $\vec{x}$ is the spatial coordinate, and t is time. We specify tractions $\vec{\tau}$ on boundary Γ_τ , and displacements $\vec{u}_0$ on boundary Γ_u . Because both $\vec{\tau}$ and $\vec{u}$ are vector quantities, there can be some spatial overlap of boundaries Γ_τ and Γ_u ; however, a degree of freedom at any location cannot be associated with both prescribed displacements (Dirichlet) and traction (Neumann) boundary conditions simultaneously.

Table 4.3: Mathematical notation for elasticity equation with infinitesimal strain.

Category	Symbol	Description
Unknowns	$\vec{u}$	Displacement field
	$\vec{v}$	Velocity field
Derived quantities	σ	Cauchy stress tensor
	ϵ	Cauchy strain tensor
Common constitutive parameters	ρ	Density
	μ	Shear modulus
	K	Bulk modulus
Source terms	$\vec{f}$	Body force per unit volume, for example $\rho\vec{g}$

Quastistatic

If we neglect the inertial term ($\rho \frac{\partial \vec{v}}{\partial t} \approx \vec{0}$), then time dependence only arises from history-dependent constitutive equations and boundary conditions. Our solution vector is the displacement vector and the elasticity equation reduces to

$$\begin{aligned} \vec{f}(\vec{x}, t) + \nabla \cdot \sigma(\vec{u}) &= \vec{0} \text{ in } \Omega, \\ \sigma \cdot \vec{n} &= \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau, \\ \vec{u} &= \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u. \end{aligned} \quad (4.50)$$

Because we will use implicit time stepping, we place all of the terms in the elasticity equation on the LHS. We create the weak form by taking the dot product with the trial function $\vec{\psi}_{trial}^u$ and integrating over the domain:

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot (\vec{f}(t) + \nabla \cdot \sigma(\vec{u})) d\Omega = 0. \quad (4.51)$$

Using the divergence theorem and incorporating the Neumann boundary conditions, we have

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^v : -\sigma(\vec{u}) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma = 0 \quad (4.52)$$

Residual Pointwise Functions

Identifying $F(t, s, \dot{s})$ and $G(t, s)$, we have

$$F^u(t, s, \dot{s}) = \int_{\Omega} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{f}(\vec{x}, t)}_{\vec{f}_0^u} + \nabla \vec{\psi}_{trial}^u : \underbrace{-\sigma(\vec{u})}_{\vec{f}_1^u} d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{\tau}(\vec{x}, t)}_{\vec{f}_0^u} d\Gamma, \quad (4.53)$$

$$G^u(t, s) = 0 \quad (4.54)$$

Note that we have multiple $\vec{f}_0$ functions, each associated with a trial function and an integral over a different domain or boundary. Each material and boundary condition (except Dirichlet) contribute pointwise functions. The integral over the domain Ω is subdivided into integrals over the materials and the integral over the boundary Γ_τ is subdivided into integrals over the Neumann boundaries. Each bulk constitutive model provides a different pointwise function for the stress tensor $\sigma(\vec{u})$. With $G = 0$ it is clear that we have a formulation that will use implicit time stepping algorithms.

Jacobian Pointwise Functions

We only have a Jacobian for the LHS:

$$J_F^{uu} = \frac{\partial F^u}{\partial u} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial u} (-\sigma) d\Omega = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : -C : \frac{1}{2} (\nabla + \nabla^T) \vec{\psi}_{basis}^u d\Omega = \int_{\Omega} \psi_{trial,i,k}^u \underbrace{(-C_{ikjl})}_{J_{f_3}^{uu}} \psi_{basis,j,l}^u d\Omega. \quad (4.55)$$

Dynamic

For compatibility with PETSc TS algorithms, we want to turn the second order equation (4.47) into two first order equations. We introduce the velocity as a unknown, $\vec{v} = \frac{\partial \vec{u}}{\partial t}$, which leads to

$$\begin{aligned}\frac{\partial \vec{u}}{\partial t} &= \vec{v} \text{ in } \Omega, \\ \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} &= \vec{f}(\vec{x}, t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}) \text{ in } \Omega, \\ \boldsymbol{\sigma} \cdot \vec{n} &= \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau, \\ \vec{u} &= \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u.\end{aligned}\tag{4.56}$$

We create the weak form by taking the dot product with the trial function $\vec{\psi}_{trial}^u$ or $\vec{\psi}_{trial}^v$ and integrating over the domain:

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \frac{\partial \vec{u}}{\partial t} d\Omega = \int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{v} d\Omega, \tag{4.57}$$

$$\int_{\Omega} \vec{\psi}_{trial}^v \cdot \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Omega} \vec{\psi}_{trial}^v \cdot \left(\vec{f}(t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}) \right) d\Omega. \tag{4.58}$$

Using the divergence theorem and incorporating the Neumann boundaries, we can rewrite the second equation as

$$\int_{\Omega} \vec{\psi}_{trial}^v \cdot \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Omega} \vec{\psi}_{trial}^v \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^v : -\boldsymbol{\sigma}(\vec{u}) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma. \tag{4.59}$$

For explicit time stepping, we want $F(t, s, \dot{s}) = \dot{s}$, so we solve an augmented system in which we multiply the RHS residual function by the inversion of the lumped LHS Jacobian,

$$F^*(t, s, \dot{s}) = G^*(t, s), \text{ where} \tag{4.60}$$

$$F^*(t, s, \dot{s}) = \dot{s} \text{ and} \tag{4.61}$$

$$G^*(t, s) = J_F^{-1} G(t, s). \tag{4.62}$$

With the augmented system, we have

$$\frac{\partial \vec{u}}{\partial t} = M_u^{-1} \int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{v} d\Omega, \tag{4.63}$$

$$\frac{\partial \vec{v}}{\partial t} = M_v^{-1} \left(\int_{\Omega} \vec{\psi}_{trial}^v \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^v : -\boldsymbol{\sigma}(\vec{u}) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma \right), \tag{4.64}$$

$$M_u = \text{Lump} \left(\int_{\Omega} \psi_{trial i}^u \delta_{ij} \psi_{basis j}^u d\Omega \right), \tag{4.65}$$

$$M_v = \text{Lump} \left(\int_{\Omega} \psi_{trial i}^v \rho(\vec{x}) \delta_{ij} \psi_{basis j}^v d\Omega \right). \tag{4.66}$$

Residual Pointwise Functions

With explicit time stepping the PETSc TS assumes the LHS is $\dot{s}$, so we only need the RHS residual functions:

$$\begin{aligned}G^u(t, s) &= \int_{\Omega} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{v}}_{\vec{g}_0^u} d\Omega, \\ G^v(t, s) &= \int_{\Omega} \vec{\psi}_{trial}^v \cdot \underbrace{\vec{f}(\vec{x}, t)}_{\vec{g}_0^v} + \nabla \vec{\psi}_{trial}^v : \underbrace{-\boldsymbol{\sigma}(\vec{u})}_{\vec{g}_1^v} d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \underbrace{\vec{\tau}(\vec{x}, t)}_{\vec{g}_0^v} d\Gamma,\end{aligned}\tag{4.67}$$

In the second equation these are the same pointwise functions as in the quasistatic case, only now they are on the RHS instead of the LHS.

Jacobian Pointwise Functions

These are the pointwise functions associated with M_u and M_v for computing the lumped LHS Jacobian. We premultiply the RHS residual function by the inverse of the lumped LHS Jacobian while s_{tshift} remains on the LHS with $\dot{s}$. As a result, we use LHS Jacobian pointwise functions, but set $s_{tshift} = 1$. The LHS Jacobians are:

$$\begin{aligned} M_u &= J_F^{uu} = \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} = \int_{\Omega} \psi_{trial i}^u \underbrace{s_{tshift} \delta_{ij} \psi_{basis j}^u}_{J_{f0}^{uu}} d\Omega, \\ M_v &= J_F^{vv} = \frac{\partial F^v}{\partial v} + s_{tshift} \frac{\partial F^v}{\partial \dot{v}} = \int_{\Omega} \psi_{trial i}^v \underbrace{\rho(\vec{x}) s_{tshift} \delta_{ij} \psi_{basis j}^v}_{J_{f0}^{vv}} d\Omega \end{aligned} \quad (4.68)$$

Absorbing Boundary

The absorbing boundary is a special case of a Neumann boundary condition, in which the traction depends on the deformation. Consider a plane wave propagating at velocity c . We can write the displacement field as

$$\vec{u}(\vec{x}, t) = \vec{u}^t(t - \frac{\vec{x}}{c}), \quad (4.69)$$

where $\vec{x}$ is position, t is time, and $\vec{u}^t$ is the shape of the propagating wave. For an absorbing boundary we want the traction on the boundary to be equal to the traction associated with the wave propagating out of the domain. Starting with the expression for the traction on a boundary, $T_i = \sigma_{ij} n_j$, and using the local coordinate system for the boundary $s_h s_v n$, where $\vec{n}$ is the direction normal to the boundary, $\vec{s}_h$ is the horizontal direction tangent to the boundary, and $\vec{s}_v$ is the vertical direction tangent to the boundary, the tractions on the boundary are

$$\tau_{s_h} = \sigma_{s_h n} \quad (4.70)$$

$$\tau_{s_v} = \sigma_{s_v n} \quad (4.71)$$

$$\tau_n = \sigma_{nn}. \quad (4.72)$$

In the case of a horizontal boundary, we define a reference directions in order to assign unique tangential directions. For a linear elastic isotropic material, $\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij}$, and we can write the tractions as

$$\tau_{s_h} = 2\mu \epsilon_{s_h n} \quad (4.73)$$

$$\tau_{s_v} = 2\mu \epsilon_{s_v n} \quad (4.74)$$

$$\tau_n = (\lambda + 2\mu) \epsilon_{nn} + \lambda (\epsilon_{s_h s_h} + \epsilon_{s_v s_v}). \quad (4.75)$$

For infinitesimal strains, $\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$ and we have

$$\epsilon_{s_h n} = \frac{1}{2}(u_{s_h, n} + u_{n, s_h}) \quad (4.76)$$

$$\epsilon_{s_v n} = \frac{1}{2}(u_{s_v, n} + u_{n, s_v}) \quad (4.77)$$

$$\epsilon_{nn} = u_{n, n}. \quad (4.78)$$

For our propagating plane wave, we recognize that

$$\frac{\partial \vec{u}^t(t - \frac{\vec{x}}{c})}{\partial x_i} = -\frac{1}{c} \frac{\partial \vec{u}^t(t - \frac{\vec{x}}{c})}{\partial t} = -\frac{1}{c} \vec{v}^t(t - \frac{\vec{x}}{c}), \quad (4.79)$$

so that our expressions for the tractions become

$$\tau_{s_h} = -\frac{\mu}{c} \left(v_{s_h}^t(t - \frac{\vec{x}}{c}) + v_n^t(t - \frac{\vec{x}}{c}) \right), \quad (4.80)$$

$$\tau_{s_v} = -\frac{\mu}{c} \left(v_{s_v}^t(t - \frac{\vec{x}}{c}) + v_n^t(t - \frac{\vec{x}}{c}) \right). \quad (4.81)$$

For the normal traction consider a dilatational wave propagating normal to the boundary at speed v_p ; in this case $u_{s_h} = u_{s_v} = 0$ and $c = v_p$. For the shear tractions, consider a shear wave propagating normal to the boundary at speed v_s ; we can decompose this into one case where $u_n = u_{s_v} = 0$ and another case where $u_n = u_{s_h} = 0$, with $c = v_s$ in both cases. We also recognize that $\mu = \rho v_s^2$ and $\lambda + 2\mu = \rho v_p^2$. This leads to the following expressions for the tractions:

$$\tau_{s_h} = -\rho v_s v_{s_h}^t \left(t - \frac{\vec{x}}{c} \right) \quad (4.82)$$

$$\tau_{s_v} = -\rho v_s v_{s_v}^t \left(t - \frac{\vec{x}}{c} \right) \quad (4.83)$$

$$\tau_n = -\rho v_p v_n^t \left(t - \frac{\vec{x}}{c} \right) \quad (4.84)$$

Substituting the tractions into the weak form for the Neumann boundary, we have

$$\int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma = \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot (-\rho c_i v_i(t)) d\Gamma, \quad (4.85)$$

where c_i equals v_p for the normal traction and v_s for the shear tractions.

Residual Pointwise Function

$$G^v(t, s) = \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \underbrace{(-\rho c_i v_i(t))}_{\vec{g}_0^v} d\Gamma, \quad (4.86)$$

Infinitesimal Strain and Prescribed Fault Slip

For each fault, which is an internal interface, we add a boundary condition to the elasticity equation prescribing the jump in the displacement field across the fault,

$$-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t) = \vec{0} \text{ on } \Gamma_f, \quad (4.87)$$

where $\vec{u}^+$ is the displacement vector on the “positive” side of the fault, $\vec{u}^-$ is the displacement vector on the “negative” side of the fault, $\vec{d}$ is the slip vector on the fault, and $\vec{n}$ is the fault normal which points from the negative side of the fault to the positive side of the fault. We enforce the jump in displacements across the fault using a Lagrange multiplier corresponding to equal and opposite tractions on the two sides of the fault.

We apply conservation of momentum,

$$\int_{\Omega} \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Omega} \vec{f}(\vec{x}, t) d\Omega + \int_{\Gamma} \vec{\tau}(\vec{x}, t) d\Gamma, \quad (4.88)$$

to a fault interface Ω_f with boundaries Γ_{f+} and Γ_{f-} . For a fault interface, the body force is zero, $\vec{f}(\vec{x}, t) = \vec{0}$. The tractions on the positive and negative fault faces are

$$\tau^+(\vec{x}, t) = \boldsymbol{\sigma}^+ \cdot \vec{n} - \vec{\lambda} \quad (4.89)$$

$$\tau^-(\vec{x}, t) = \boldsymbol{\sigma}^- \cdot \vec{n} + \vec{\lambda}, \quad (4.90)$$

where $\vec{\lambda}$ is the Lagrange multiplier that corresponds to the fault traction generating the prescribed slip and $\boldsymbol{\sigma}^+$ and $\boldsymbol{\sigma}^-$ are the stresses in the domain at the positive and negative sides of the fault. Thus, for a fault interface, we have

$$\int_{\Omega_f} \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Gamma_{f+}} \boldsymbol{\sigma} \cdot \vec{n} - \vec{\lambda} d\Gamma + \int_{\Gamma_{f-}} \boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda} d\Gamma. \quad (4.91)$$

Table 4.4: Mathematical notation for elasticity equation with infinitesimal strain and prescribed slip on faults.

Category	Sym- bol	Description
Unknowns	$\vec{u}$	Displacement field
	$\vec{v}$	Velocity field
	$\vec{\lambda}$	Lagrange multiplier field
Derived quantities	$\boldsymbol{\sigma}$	Cauchy stress tensor
	$\boldsymbol{\epsilon}$	Cauchy strain tensor
Common constitutive pa- rameters	ρ	Density
	μ	Shear modulus
	K	Bulk modulus
Source terms	$\vec{f}$	Body force per unit volume, for example $\rho\vec{g}$
	$\vec{d}$	Slip vector field on the fault corresponding to a jump in the displacement field across the fault

Quasistatic

As in the case of elasticity without faults, we first consider the quasistatic case in which we neglect the inertial term ($\rho \frac{\partial \vec{v}}{\partial t} \approx \vec{0}$). We place all of the terms in the elasticity equation on the LHS, consistent with implicit time stepping. Our equation of the conservation of momentum on the fault interface reduces to

$$\int_{\Gamma_{f+}} \boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda} d\Gamma + \int_{\Gamma_{f-}} \boldsymbol{\sigma} \cdot \vec{n} - \vec{\lambda} d\Gamma = 0. \quad (4.92)$$

We enforce this equation on each portion of the fault interface along with our prescribed slip constraint, which leads to

$$\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda} = \vec{0} \text{ on } \Gamma_{f+}, \quad (4.93)$$

$$\boldsymbol{\sigma} \cdot \vec{n} - \vec{\lambda} = \vec{0} \text{ on } \Gamma_{f-}, \quad (4.94)$$

$$\vec{u}^+ - \vec{u}^- - \vec{d}(\vec{x}, t) = \vec{0}, \quad (4.95)$$

Our solution vector consists of both displacements and Lagrange multipliers, and the strong form for the system of equations is

$$\vec{s}^T = \begin{pmatrix} \vec{u} & \vec{\lambda} \end{pmatrix}^T \quad (4.96)$$

$$\vec{f}(\vec{x}, t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}) = \vec{0} \text{ in } \Omega, \quad (4.97)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_{\tau}, \quad (4.98)$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u, \quad (4.99)$$

$$-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t) = \vec{0} \text{ on } \Gamma_f, \quad (4.100)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = -\vec{\lambda}(\vec{x}, t) \text{ on } \Gamma_{f+}, \text{ and} \quad (4.101)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = +\vec{\lambda}(\vec{x}, t) \text{ on } \Gamma_{f-}. \quad (4.102)$$

We create the weak form by taking the dot product with the trial function $\vec{\psi}_{trial}^u$ or $\vec{\psi}_{trial}^{\lambda}$ and integrating over the domain. After using the divergence theorem and incorporating the Neumann boundary and fault interface conditions,

we have

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^u : -\boldsymbol{\sigma}(\vec{u}) d\Omega + \int_{\Gamma_{\tau}} \vec{\psi}_{trial}^u \cdot \vec{\tau}(\vec{x}, t) d\Gamma + \int_{\Gamma_f} \vec{\psi}_{trial}^{u+} \cdot (-\vec{\lambda}(\vec{x}, t)) + \vec{\psi}_{trial}^{u-} \cdot (+\vec{\lambda}(\vec{x}, t)) d\Gamma = 0 \quad (4.103)$$

$$\int_{\Gamma_f} \vec{\psi}_{trial}^{\lambda} \cdot (-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t)) d\Gamma = 0. \quad (4.104)$$

We have chosen the signs in the last equation so that the Jacobian will be symmetric with respect to the Lagrange multiplier. We solve the system of equations using implicit time stepping, making use of residuals functions and Jacobians for the LHS.

Residual Pointwise Functions

Identifying $F(t, s, \dot{s})$ and $G(t, s)$, we have

$$\begin{aligned} F^u(t, s, \dot{s}) &= \int_{\Omega} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{f}(\vec{x}, t)}_{\vec{f}_0^u} + \nabla \vec{\psi}_{trial}^u : \underbrace{-\boldsymbol{\sigma}(\vec{u})}_{\vec{f}_1^u} d\Omega + \int_{\Gamma_{\tau}} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{\tau}(\vec{x}, t)}_{\vec{f}_0^u} d\Gamma + \int_{\Gamma_f} \vec{\psi}_{trial}^{u+} \cdot \underbrace{(-\vec{\lambda}(\vec{x}, t))}_{\vec{f}_0^u} + \vec{\psi}_{trial}^{u-} \cdot \underbrace{(+\vec{\lambda}(\vec{x}, t))}_{\vec{f}_0^u} d\Gamma \\ F^{\lambda}(t, s, \dot{s}) &= \int_{\Gamma_f} \vec{\psi}_{trial}^{\lambda} \cdot \underbrace{(-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t))}_{\vec{f}_0^{\lambda}} d\Gamma, \\ G^u(t, s) &= 0 \\ G^{\lambda}(t, s) &= 0 \end{aligned} \quad (4.105)$$

Compared to the quasistatic elasticity case without a fault, we have simply added additional pointwise functions associated with the fault. Our fault implementation does not change the formulation for the materials or external Dirichlet or Neumann boundary conditions.

Jacobian Pointwise Functions

The LHS Jacobians are:

$$\begin{aligned} J_F^{uu} &= \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : -\boldsymbol{C} : \frac{1}{2}(\nabla + \nabla^T) \vec{\psi}_{basis}^u d\Omega = \int_{\Omega} \psi_{trial, k}^u \underbrace{(-C_{ikjl})}_{J_{f3}^{uu}} \psi_{basis, j, l}^u d\Omega \\ J_F^{u\lambda} &= \frac{\partial F^u}{\partial \lambda} + s_{tshift} \frac{\partial F^u}{\partial \dot{\lambda}} = \int_{\Gamma_f} \psi_{trial, i}^{u+} \underbrace{(-\delta_{ij})}_{J_{f0}^{u\lambda}} \psi_{basis, j}^{\lambda} + \psi_{trial, i}^{u-} \underbrace{(+\delta_{ij})}_{J_{f0}^{u\lambda}} \psi_{basis, j}^{\lambda} d\Gamma, \\ J_F^{\lambda u} &= \frac{\partial F^{\lambda}}{\partial u} + s_{tshift} \frac{\partial F^{\lambda}}{\partial \dot{u}} = \int_{\Gamma_f} \psi_{trial, i}^{\lambda} \underbrace{(-\delta_{ij})}_{J_{f0}^{\lambda u}} \psi_{basis, j}^{u+} + \psi_{trial, i}^{\lambda} \underbrace{(+\delta_{ij})}_{J_{f0}^{\lambda u}} \psi_{basis, j}^{u-} d\Gamma, \\ J_F^{\lambda\lambda} &= 0 \end{aligned} \quad (4.106)$$

This LHS Jacobian has the structure

$$J_F = \begin{pmatrix} J_F^{uu} & J_F^{u\lambda} \\ J_F^{\lambda u} & 0 \end{pmatrix} = \begin{pmatrix} J_F^{uu} & C^T \\ C & 0 \end{pmatrix}, \quad (4.107)$$

where C contains entries of ± 1 for degrees of freedom on the two sides of the fault. The Schur complement of J with respect to J_F^{uu} is $-C (J_F^{uu})^{-1} C^T$.

Dynamic

The equation prescribing fault slip is independent of the Lagrange multiplier, so we do not have a system of equations that we can put in the form $\dot{s} = G^*(t, s)$. Instead, we have a differential-algebraic set of equations (DAEs), which we solve using an implicit-explicit (IMEX) time integration scheme. As in the case of dynamic elasticity without faults, we introduce the velocity ($\vec{v}$) as an unknown to turn the elasticity equation into two first order equations. Our constraint for prescribed slip is

$$\vec{u}^+ - \vec{u}^- - \vec{d} = \vec{0}, \quad (4.108)$$

where $\vec{u}$ is the displacement vector and $\vec{d}$ is the slip vector. In order to match the order of the time derivative in the elasticity equation, we take the second derivative of the prescribed slip equation with respect to time,

$$\frac{\partial \vec{v}^+}{\partial t} - \frac{\partial \vec{v}^-}{\partial t} - \frac{\partial^2 \vec{d}}{\partial t^2} = \vec{0}. \quad (4.109)$$

This means that our differential algebraic equations has a differentiation index of 2.

The strong form for our system of equations is:

$$\vec{s}^T = \begin{pmatrix} \vec{u} & \vec{v} & \vec{\lambda} \end{pmatrix}^T \quad (4.110)$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{v}, \quad (4.111)$$

$$\rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} = \vec{f}(\vec{x}, t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}), \quad (4.112)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau} \text{ on } \Gamma_\tau. \quad (4.113)$$

$$\vec{u} = \vec{u}_0 \text{ on } \Gamma_u, \quad (4.114)$$

$$\frac{\partial \vec{v}^+}{\partial t} - \frac{\partial \vec{v}^-}{\partial t} - \frac{\partial^2 \vec{d}(\vec{x}, t)}{\partial t^2} = \vec{0}, \quad (4.115)$$

$$\int_{\Omega_f} \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Gamma_{f+}} \boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda} d\Gamma + \int_{\Gamma_{f-}} \boldsymbol{\sigma} \cdot \vec{n} - \vec{\lambda} d\Gamma. \quad (4.116)$$

We generate the weak form in the usual way,

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \frac{\partial \vec{u}}{\partial t} d\Omega = \int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{v} d\Omega, \quad (4.117)$$

$$\int_{\Omega} \vec{\psi}_{trial}^v \cdot \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Omega} \vec{\psi}_{trial}^v \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^v : -\boldsymbol{\sigma}(\vec{u}) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma \quad (4.118)$$

$$+ \int_{\Gamma_f} \vec{\psi}_{trial}^{v+} \cdot \left(-\vec{\lambda}(\vec{x}, t) \right) + \vec{\psi}_{trial}^{v-} \cdot \left(+\vec{\lambda}(\vec{x}, t) \right) d\Gamma, \quad (4.119)$$

$$\int_{\Gamma_f} \vec{\psi}_{trial}^\lambda \cdot \left(\frac{\partial \vec{v}^+}{\partial t} - \frac{\partial \vec{v}^-}{\partial t} - \frac{\partial^2 \vec{d}(\vec{x}, t)}{\partial t^2} \right) d\Gamma = 0. \quad (4.120)$$

$$\int_{\Omega_f} \vec{\psi}_{trial}^\lambda \cdot \rho(\vec{x}) \frac{\partial \vec{v}}{\partial t} d\Omega = \int_{\Gamma_{f+}} \vec{\psi}_{trial}^\lambda \cdot \left(\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda} \right) d\Gamma + \int_{\Gamma_{f-}} \vec{\psi}_{trial}^\lambda \cdot \left(\boldsymbol{\sigma} \cdot \vec{n} - \vec{\lambda} \right) d\Gamma. \quad (4.121)$$

For compatibility with PETSc TS IMEX implementations, we need $\dot{\vec{s}}$ on the LHS for the explicit part (displacement-velocity and elasticity equations) and we need $\vec{\lambda}$ in the equation for the implicit part (prescribed slip equation). We first

focus on the explicit part and select numerical quadrature that yields a lumped mass matrix, M , so that we have

$$\begin{aligned}
 \frac{\partial \vec{u}}{\partial t} &= M_u^{-1} \int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{v} d\Omega, \\
 \frac{\partial \vec{v}}{\partial t} &= M_v^{-1} \int_{\Omega} \vec{\psi}_{trial}^v \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^v : -\boldsymbol{\sigma}(\vec{u}) d\Omega + M_v^{-1} \int_{\Gamma_{\tau}} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma \\
 &\quad + M_{v^+}^{-1} \int_{\Gamma_f^+} \vec{\psi}_{trial}^{v^+} \cdot (-\vec{\lambda}(\vec{x}, t)) d\Gamma + M_{v^-}^{-1} \int_{\Gamma_f^-} \vec{\psi}_{trial}^{v^-} \cdot (+\vec{\lambda}(\vec{x}, t)) d\Gamma, \\
 M_u &= Lump \left(\int_{\Omega} \psi_{trial_i}^u \delta_{ij} \psi_{basis_j}^u d\Omega \right), \\
 M_v &= Lump \left(\int_{\Omega} \psi_{trial_i}^v \rho(\vec{x}) \delta_{ij} \psi_{basis_j}^v d\Omega \right).
 \end{aligned} \tag{4.122}$$

For the implicit part, we can separate the integration of the weak form for negative and positive sides of the fault interface, which yields

$$M_{v^+} \frac{\partial \vec{v}^+}{\partial t} = \int_{\Gamma_{f^+}} \vec{\psi}_{trial}^{\lambda} \cdot (\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda}) d\Gamma, \tag{4.123}$$

$$M_{v^-} \frac{\partial \vec{v}^-}{\partial t} = \int_{\Gamma_{f^-}} \vec{\psi}_{trial}^{\lambda} \cdot (\boldsymbol{\sigma} \cdot \vec{n} - \vec{\lambda}) d\Gamma. \tag{4.124}$$

Using these equations to substitute in the expressions for the time derivative of the velocity on the negative and positive sides of the fault into the prescribed slip constraint equation yields

$$M_{v^+}^{-1} \int_{\Gamma_f^+} \vec{\psi}_{trial}^{\lambda} \cdot (\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda}) d\Gamma + M_{v^-}^{-1} \int_{\Gamma_f^-} \vec{\psi}_{trial}^{\lambda} \cdot (-\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda}) d\Gamma + \int_{\Gamma_f} \vec{\psi}_{trial}^{\lambda} \cdot \left(-\frac{\partial^2 \vec{d}}{\partial t^2} \right) d\Gamma = \vec{0}. \tag{4.125}$$

Residual Pointwise Functions

Combining the explicit parts of the weak form in equation (4.122) with the implicit part of the weak form in equation (4.125) and identifying $F(t, s, \dot{s})$ and $G(t, s)$, we have

$$F^u(t, s, \dot{s}) = \frac{\partial \vec{u}}{\partial t} \tag{4.126}$$

$$F^v(t, s, \dot{s}) = \frac{\partial \vec{v}}{\partial t} \tag{4.127}$$

$$F^{\lambda}(t, s, \dot{s}) = \underbrace{M_{v^+}^{-1}}_{c^+} \int_{\Gamma_f^+} \vec{\psi}_{trial}^{\lambda} \cdot \underbrace{(\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda})}_{f_0^{\lambda}} d\Gamma + \underbrace{M_{v^-}^{-1}}_{c^-} \int_{\Gamma_f^-} \vec{\psi}_{trial}^{\lambda} \cdot \underbrace{(-\boldsymbol{\sigma} \cdot \vec{n} + \vec{\lambda})}_{f_0^{\lambda}} d\Gamma + \int_{\Gamma_f} \vec{\psi}_{trial}^{\lambda} \cdot \underbrace{-\frac{\partial^2 \vec{d}}{\partial t^2}}_{f_0^{\lambda}} d\Gamma \tag{4.128}$$

$$G^u(t, s) = \underbrace{M_u^{-1}}_c \int_{\Omega} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{v}}_{\vec{g}_0^u} d\Omega, \tag{4.129}$$

$$G^v(t, s) = \underbrace{M_v^{-1}}_c \left(\int_{\Omega} \vec{\psi}_{trial}^v \cdot \underbrace{\vec{f}(\vec{x}, t)}_{\vec{g}_0^v} + \nabla \vec{\psi}_{trial}^v : \underbrace{-\boldsymbol{\sigma}(\vec{u})}_{\vec{g}_1^v} d\Omega + \int_{\Gamma_{\tau}} \vec{\psi}_{trial}^v \cdot \underbrace{\vec{\tau}(\vec{x}, t)}_{\vec{g}_0^v} d\Gamma + \int_{\Gamma_f} \vec{\psi}_{trial}^{v^+} \cdot \underbrace{(-\vec{\lambda}(\vec{x}, t))}_{\vec{g}_0^v} + \vec{\psi}_{trial}^{v^-} \cdot \underbrace{(+\vec{\lambda}(\vec{x}, t))}_{\vec{g}_0^v} \right) \tag{4.130}$$

$$G^{\lambda}(t, s) = 0 \tag{4.131}$$

The integrals for the explicit part are all weighted by the inverse of the lumped mass matrix. For the implicit part, only the integrals over the positive and negative sides of the fault are weighted by the inverse of the lumped mass matrix.

Jacobian Pointwise Functions

For the explicit part we have pointwise functions for computing the lumped LHS Jacobian. These are exactly the same pointwise functions as in the dynamic case without a fault,

$$J_F^{uu} = \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} = \int_{\Omega} \psi_{trial_i^u} \underbrace{s_{tshift} \delta_{ij}}_{J_{f0}^{uu}} \psi_{basis_j^u} d\Omega, \quad (4.132)$$

$$J_F^{vv} = \frac{\partial F^v}{\partial v} + s_{tshift} \frac{\partial F^v}{\partial \dot{v}} = \int_{\Omega} \psi_{trial_i^v} \underbrace{\rho(\vec{x}) s_{tshift} \delta_{ij}}_{J_{f0}^{vv}} \psi_{basis_j^v} d\Omega \quad (4.133)$$

For the implicit part, we have pointwise functions for the LHS Jacobians associated with the prescribed slip,

$$J_F^{\lambda u} = \frac{\partial F^{\lambda}}{\partial u} + s_{tshift} \frac{\partial F^{\lambda}}{\partial \dot{u}} = \underbrace{M_{v+}^{-1}}_{c^+} \int_{\Gamma_{f+}} \psi_{trial_i}^{\lambda} \underbrace{C_{kijl} n_k}_{J_{f1}^{\lambda u}} \psi_{basis_{j,l}}^u d\Gamma + \underbrace{M_{v-}^{-1}}_{c^-} \int_{\Gamma_{f-}} \psi_{trial_i}^{\lambda} \underbrace{-C_{kijl} n_k}_{J_{f1}^{\lambda u}} \psi_{basis_{j,l}}^u d\Gamma, \quad (4.134)$$

$$J_F^{\lambda \lambda} = \frac{\partial F^{\lambda}}{\partial \lambda} + s_{tshift} \frac{\partial F^{\lambda}}{\partial \dot{\lambda}} = \underbrace{M_{v+}^{-1}}_{c^+} \int_{\Gamma_{f+}} \psi_{trial_i}^{\lambda} \underbrace{\delta_{ij}}_{J_{f0}^{\lambda \lambda}} \psi_{basis_j}^{\lambda} d\Gamma + \underbrace{M_{v-}^{-1}}_{c^-} \int_{\Gamma_{f-}} \psi_{trial_i}^{\lambda} \underbrace{\delta_{ij}}_{J_{f0}^{\lambda \lambda}} \psi_{basis_j}^{\lambda} d\Gamma \quad (4.135)$$

Bulk Rheologies

In this section we describe the mathematic formulations of the bulk rheologies for elasticity. The bulk rheologies include

- Isotropic linear elasticity,
- Isotropic linear (Maxwell) viscoelasticity,
- Isotropic linear generalized Maxwell viscoelasticity, and
- Isotropic power-law viscoelasticity.

For the viscoelastic rheologies, we assume the viscous deformation is incompressible; consequently, we separate the stress and strain fields into deviatoric and volumetric components.

Table 4.5: Mathematical notation for viscoelastic formulations.

Variable	Symbol	Definition
Mean stress	P	$\frac{\text{Tr}(\boldsymbol{\sigma})}{3}$
Mean strain	θ	$\frac{\text{Tr}(\boldsymbol{\epsilon})}{3}$
Deviatoric stress	$\boldsymbol{\sigma}^{dev}$	$\boldsymbol{\sigma} - P \mathbf{I}$
Deviatoric strain	$\boldsymbol{\epsilon}^{dev}$	$\boldsymbol{\epsilon} - \theta \mathbf{I}$
2nd deviatoric stress invariant	J_2'	$\frac{1}{2} \boldsymbol{\sigma}^{dev} : \boldsymbol{\sigma}^{dev}$
2nd deviatoric strain invariant	L_2'	$\frac{1}{2} \boldsymbol{\epsilon}^{dev} : \boldsymbol{\epsilon}^{dev}$

Elasticity Constitutive Models

The Jacobian for the elasticity equation (4.55) is

$$J_{f3}^{uu} = \frac{\partial F^{u_i}}{\partial u_j}. \quad (4.136)$$

In computing the derivative, we consider the linearized form:

$$\begin{aligned}
 \sigma_{ik} &= C_{ikjl} \epsilon_{jl} \\
 \sigma_{ik} &= C_{ikjl} \frac{1}{2} (u_{j,l} + u_{l,j}) \\
 \sigma_{ik} &= \frac{1}{2} (C_{ikjl} + C_{iklj}) u_{j,l} \\
 \sigma_{ik} &= C_{ikjl} u_{j,l}.
 \end{aligned} \tag{4.137}$$

In computing the Jacobian, we take the derivative of the stress tensor with respect to the displacement field,

$$\frac{\partial}{\partial u_j} \sigma_{ik} = C_{ikjl} \psi_{basis\ j,l}^u, \tag{4.138}$$

so we have

$$J_{f3}^{uu}(i, j, k, l) = -C_{ikjl}. \tag{4.139}$$

For many elasticity constitutive models we prefer to separate the stress into the mean stress and deviatoric stress:

$$\begin{aligned}
 \boldsymbol{\sigma} &= \sigma^{mean} \mathbf{I} + \boldsymbol{\sigma}^{dev}, \text{ where} \\
 \sigma^{mean} &= \frac{1}{3} \text{Tr}(\boldsymbol{\sigma}) = \frac{1}{3} (\sigma_{11} + \sigma_{22} + \sigma_{33}).
 \end{aligned} \tag{4.140}$$

Sometimes it is convenient to use pressure (positive pressure corresponds to compression) instead of the mean stress:

$$\begin{aligned}
 \boldsymbol{\sigma} &= -p \mathbf{I} + \boldsymbol{\sigma}^{dev}, \text{ where} \\
 p &= -\frac{1}{3} \text{Tr}(\boldsymbol{\sigma}).
 \end{aligned} \tag{4.141}$$

The Jacobian with respect to the deviatoric stress is

$$\begin{aligned}
 \frac{\partial \sigma_{ik}^{dev}}{\partial u_j} &= \frac{\partial}{\partial u_j} \left(\sigma_{ik} - \frac{1}{3} \sigma_{mm} \delta_{ik} \right) \\
 \frac{\partial \sigma_{ik}^{dev}}{\partial u_j} &= C_{ikjl} \psi_{basis\ j,l}^u - \frac{1}{3} C_{mmjl} \delta_{ik} \psi_{basis\ j,l}^u.
 \end{aligned} \tag{4.142}$$

We call these modified elastic constants C_{ijkl}^{dev} , so that we have

$$C_{ijkl}^{dev} = C_{ikjl} - \frac{1}{3} C_{mmjl} \delta_{ik}. \tag{4.143}$$

Linear Isotropic Elastic Models

We implement isotropic linear elasticity both with and without a reference stress and strain state. With a linear elastic material we often compute the deformation relative to an unknown initial stress and strain state. An initial undeformed configuration with zero stress and strain corresponds to reference stress and strain of zero.

Without a reference stress and strain state, we have

$$\sigma_{ij} = \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij} \tag{4.144}$$

and with a reference stress and strain state, we have

$$\sigma_{ij} = \sigma_{ij}^{ref} + \lambda (\epsilon_{kk} - \epsilon_{kk}^{ref}) \delta_{ij} + 2\mu (\epsilon_{ij} - \epsilon_{ij}^{ref}). \tag{4.145}$$

The mean stress is

$$\begin{aligned}\sigma^{mean} &= \frac{1}{3}\sigma_{kk}, \\ \sigma^{mean} &= \frac{1}{3}\sigma_{kk}^{ref} + K\left(\epsilon_{kk} - \epsilon_{kk}^{ref}\right),\end{aligned}\tag{4.146}$$

where $K = \lambda + 2\mu/3$ is the bulk modulus. If the reference stress and reference strain are both zero, then this reduces to

$$\sigma^{mean} = K\epsilon_{kk}.\tag{4.147}$$

The deviatoric stress is

$$\begin{aligned}\sigma_{ij}^{dev} &= \sigma_{ij} - \sigma^{mean}\delta_{ij}, \\ \sigma_{ij}^{dev} &= \sigma_{ij}^{ref} + \lambda\left(\epsilon_{kk} - \epsilon_{kk}^{ref}\right)\delta_{ij} + 2\mu\left(\epsilon_{ij} - \epsilon_{ij}^{ref}\right) - \left[\frac{1}{3}\sigma_{kk}^{ref} + \left(\lambda + \frac{2}{3}\mu\right)\left(\epsilon_{kk} - \epsilon_{kk}^{ref}\right)\right]\delta_{ij}, \\ \sigma_{ij}^{dev} &= \sigma_{ij}^{ref} - \frac{1}{3}\sigma_{kk}^{ref}\delta_{ij} + 2\mu\left(\epsilon_{ij} - \epsilon_{ij}^{ref}\right) - \frac{2}{3}\mu\left(\epsilon_{kk} - \epsilon_{kk}^{ref}\right)\delta_{ij}.\end{aligned}$$

For isotropic linear elasticity, all components of C_{ijkl} are zero except for:

$$\begin{aligned}C_{1111} &= C_{2222} = C_{3333} = \lambda + 2\mu, \\ C_{1122} &= C_{1133} = C_{2233} = \lambda, \\ C_{1212} &= C_{2323} = C_{1313} = \mu.\end{aligned}\tag{4.148}$$

The deviatoric elastic constants are:

$$\begin{aligned}C_{1111}^{dev} &= C_{2222}^{dev} = C_{3333}^{dev} = \frac{4}{3}\mu, \\ C_{1122}^{dev} &= C_{1133}^{dev} = C_{2233}^{dev} = -\frac{2}{3}\mu, \\ C_{1212}^{dev} &= C_{2323}^{dev} = C_{1313}^{dev} = \mu.\end{aligned}\tag{4.149}$$

Linear Viscoelastic Models

Fig. 4.6 shows schematic representations of the viscoelastic models. The linear Maxwell model can be represented by a spring in series with a linear dashpot.

For a one-dimensional model, the response is given by

$$\frac{d\epsilon_{Total}}{dt} = \frac{d\epsilon_D}{dt} + \frac{d\epsilon_S}{dt} = \frac{\sigma}{\eta} + \frac{1}{E}\frac{d\sigma}{dt},\tag{4.150}$$

where ϵ_{Total} is the total strain, ϵ_D is the strain in the dashpot, ϵ_S is the strain in the spring, σ is the stress, η is the viscosity of the dashpot, and E is the spring constant. When a Maxwell material is subjected to constant strain, the stresses relax exponentially with time. When a Maxwell material is subjected to a constant stress, there is an immediate elastic strain, corresponding to the response of the spring, and a viscous strain that increases linearly with time. Because the strain response is unbounded, the Maxwell model actually represents a fluid.

Another simple model is the Kelvin-Voigt model, which consists of a spring in parallel with a dashpot. In this case, the one-dimensional response is given by

$$\sigma(t) = E\epsilon(t) + \eta\frac{d\epsilon(t)}{dt}.\tag{4.151}$$

As opposed to the Maxwell model, which represents a fluid, the Kelvin-Voigt model represents a solid undergoing reversible, viscoelastic strain. If the material is subjected to a constant stress, it deforms at a decreasing rate, gradually approaching the strain that would occur for a purely elastic material. When the stress is released, the material gradually relaxes back to its undeformed state.

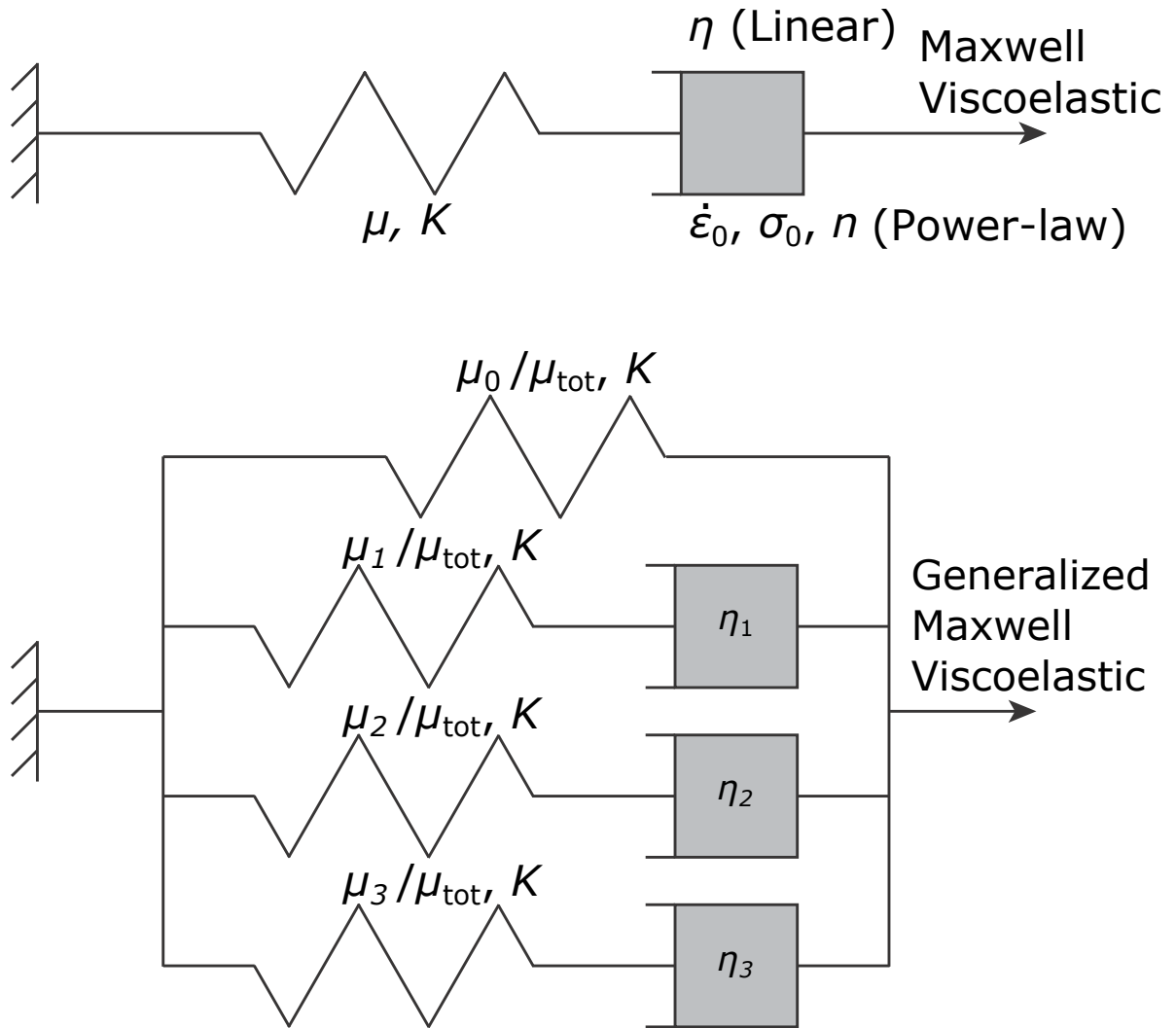


Fig. 4.6: Schematic representations of viscoelastic bulk rheologies available in PyLith. Note that the models with linear (Newtonian) and power-law viscous behavior are treated as separate rheologies.

Generalized Maxwell Viscoelastic Models

The most general form of linear viscoelastic model is the generalized Maxwell model, which can be represented by a spring in parallel with a number of Maxwell models (see Fig. 4.6). With this model it is possible to represent a number of simpler viscoelastic models. For example, a simple Maxwell model corresponds to setting the elastic constants of all springs to zero, with the exception of the spring contained in the first Maxwell model (μ_1). Similarly, the Kelvin-Voigt model corresponds to setting the elastic constants $\mu_2 = \mu_3 = 0$, and setting $\mu_1 = \infty$ (or a very large number).

TODO

Add more information about using the generalized Maxwell model to implement various viscoelastic models (Burgers material).

We follow formulations similar to those used by [Zienkiewicz and Taylor, 2000] and [Taylor, 2003]. In this formulation, we specify the total shear modulus of the model (μ_{tot}) and the bulk modulus (K). We then provide the fractional shear modulus for each Maxwell element spring in the model. It is not necessary to specify the fractional modulus for μ_0 , because we can find it by subtracting the sum of the other ratios from 1. Note that the sum of all these fractions must equal 1. We use a similar formulation for our linear Maxwell viscoelastic model, but in that case μ_0 is always zero and we only use a single Maxwell model. The parameters defining the both materials are listed in Table 4.24.

As for all our viscoelastic models, the volumetric strain is completely elastic, and the viscoelastic deformation may be expressed purely in terms of the deviatoric components:

$$\boldsymbol{\sigma}^{dev} = 2\mu_{tot} \left[\mu_0 \boldsymbol{\epsilon}^{dev} + \sum_{i=1}^N \mu_i \mathbf{q}^i - \boldsymbol{\epsilon}^{ref dev} \right] + \boldsymbol{\sigma}^{ref dev}; P = 3K(\theta - \theta^{ref}) + P^{ref}, \quad (4.152)$$

where K is the bulk modulus, N is the number of Maxwell models, the terms with *ref* superscripts refer to reference states, and the variable q_i follows the evolution equations

$$\dot{\mathbf{q}}^i + \frac{1}{\tau_i} \mathbf{q}^i = \dot{\boldsymbol{\epsilon}}^{dev}. \quad (4.153)$$

The τ_i are the relaxation times for each Maxwell model:

$$\tau_i = \frac{\eta_i}{\mu_{tot}\mu_i}. \quad (4.154)$$

An alternative to the differential equation form above is an integral equation form expressed in terms of the relaxation modulus function. This function is defined in terms of an idealized experiment in which, at time zero ($t = 0$), a specimen is subjected to a constant strain, $\boldsymbol{\epsilon}_0^{dev}$, and the stress response, $\boldsymbol{\sigma}^{dev}(t)$, is measured. For a linear material we obtain:

$$\boldsymbol{\sigma}(t) = 2\mu(t)(\boldsymbol{\epsilon}_0^{dev} - \boldsymbol{\epsilon}^{ref dev}) + \boldsymbol{\sigma}^{ref dev}, \quad (4.155)$$

where $\mu(t)$ is the shear relaxation modulus function. Using linearity and superposition for an arbitrary state of strain yields an integral equation:

$$\boldsymbol{\sigma}^{dev}(t) = \int_{-\infty}^t \mu(t-T) \dot{\boldsymbol{\epsilon}}^{dev} dT. \quad (4.156)$$

Writing the modulus function in Prony series form we obtain

$$\mu(t) = \mu_{tot} \left(\mu_0 + \sum_{i=1}^N \mu_i \exp \frac{-t}{\tau_i} \right), \quad (4.157)$$

where

$$\mu_0 + \sum_{i=1}^N \mu_i = 1. \quad (4.158)$$

With the form in (4.157), the integral equation form is identical to the differential equation form.

If we assume the material is undisturbed until a strain is suddenly applied at time zero, we can divide the integral into

$$\int_{-\infty}^t (\cdot) dT = \int_{-\infty}^{0^-} (\cdot) dT + \int_{0^-}^{0^+} (\cdot) dT + \int_{0^+}^t (\cdot) dT. \quad (4.159)$$

The first term is zero, the second term includes a jump term associated with ϵ_0^{dev} at time zero, and the last term covers the subsequent history of strain. Applying this separation to (4.156),

$$\sigma^{dev}(t) = 2\mu(t)(\epsilon_0^{dev} - \epsilon^{refdev}) + \sigma^{refdev} + 2 \int_0^t \mu(t-T) \dot{\epsilon}^{dev}(T) dT, \quad (4.160)$$

where we have left the sign off of the lower limit on the integral.

Substituting (4.157) into (4.160), we obtain

$$\sigma^{dev}(t) = 2\mu_{tot} \left\{ \mu_0 \epsilon^{dev}(t) + \sum_{i=1}^N \left[\mu_i \exp \frac{-t}{\tau_i} \left(\epsilon_0^{dev} + \int_0^t \exp \frac{T}{\tau_i} \dot{\epsilon}^{dev}(T) dT \right) \right] - \epsilon^{refdev} \right\} + \sigma^{refdev}. \quad (4.161)$$

We then split each integral into two ranges: from 0 to t_n , and from t_n to t , and define each integral as

$$i_i^1(t) = \int_0^t \exp \frac{T}{\tau_i} \dot{\epsilon}^{dev}(T) dT. \quad (4.162)$$

The integral then becomes

$$i_i^1(t) = i_i^1(t_n) + \int_{t_n}^t \exp \frac{T}{\tau_i} \dot{\epsilon}^{dev}(T) dT. \quad (4.163)$$

Including the negative exponential multiplier, we have

$$h_i^1(t) = \exp \frac{-t}{\tau_i} i_i^1. \quad (4.164)$$

Then

$$h_i^1(t) = \exp \frac{-\Delta t}{\tau_i} h_i^1(t_n) + \Delta h_i. \quad (4.165)$$

where

$$\Delta h_i = \exp \frac{-t}{\tau_i} \int_{t_n}^t \exp \frac{T}{\tau_i} \dot{\epsilon}^{dev}(T) dT. \quad (4.166)$$

Approximating the strain rate as constant over each time step, the solution may be found as

$$\Delta h_i = \frac{\tau_i}{\Delta t} \left(1 - \exp \frac{-\Delta t}{\tau_i} \right) (\epsilon^{dev} - \epsilon_n^{dev}) = \Delta h_i (\epsilon^{dev} - \epsilon_n^{dev}). \quad (4.167)$$

The approximation is singular for zero time steps, but a series expansion may be used for small time-step sizes:

$$\Delta h_i \approx 1 - \frac{1}{2} \left(\frac{\Delta t}{\tau_i} \right) + \frac{1}{3!} \left(\frac{\Delta t}{\tau_i} \right)^2 - \frac{1}{4!} \left(\frac{\Delta t}{\tau_i} \right)^3 + \dots \quad (4.168)$$

This converges with only a few terms. With this formulation, the constitutive relation now has the simple form:

$$\boldsymbol{\sigma}^{dev}(t) = 2\mu_{tot} \left(\mu_0 \boldsymbol{\epsilon}^{dev}(t) + \sum_{i=1}^N \mu_i \mathbf{h}_i^1(t) - \boldsymbol{\epsilon}^{refdev} \right) + \boldsymbol{\sigma}^{refdev}. \quad (4.169)$$

We need to compute the tangent constitutive matrix when forming the Jacobian matrix. In addition to the volumetric contribution to the tangent constitutive matrix, we require the deviatoric part:

$$\frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{kl}} = \frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{mn}^{dev}} \frac{\partial \epsilon_{mn}^{dev}}{\partial \epsilon_{kl}}, \quad (4.170)$$

where the second derivative on the right may be easily deduced from the definition of deviatoric strain in [Mathematical notation for viscoelastic formulations](#).. The other derivative is given by

$$\frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{mn}^{dev}} = 2\mu_{tot} \left[\mu_0 \delta_{im} \delta_{jn} + \sum_{l=1}^N \mu_l \frac{\partial h_{lij}^1}{\partial \epsilon_{mn}^{dev}} \right]. \quad (4.171)$$

From (4.165) and (4.167), the derivative inside the brackets is

$$\frac{\partial h_{lij}^1}{\partial \epsilon_{mn}^{dev}} = \Delta h_l(\Delta t) \delta_{im} \delta_{jn}. \quad (4.172)$$

The complete deviatoric tangent relation is then

$$\frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{mn}} = 2\mu_{tot} \left[\mu_0 + \sum_{l=1}^N \mu_l \Delta h_l(\Delta t) \right] \frac{\partial \epsilon_{ij}^{dev}}{\partial \epsilon_{mn}}. \quad (4.173)$$

We use this formulation for both our Maxwell and generalized Maxwell viscoelastic models. For the Maxwell model, $\mu_0 = 0$ and $N = 1$. For the generalized Maxwell model, $N = 3$. The stable time step is equal to 1/5 of the minimum relaxation time for all of the Maxwell models (4.154).

Effective Stress Formulation for Viscoelastic Materials

An alternative technique is used for power-law viscoelastic materials, as well as for Drucker-Prager elasto-plastic materials. This approach, known as the effective stress function formulation [Kojic and Bathe, 1987] is more suitable for such nonlinear behavior, as it simplifies the equations that must be solved. As for our linear viscoelastic models, the viscous volumetric strains are zero (incompressible flow), and we separate the general stress-strain relationship at time $t + \Delta t$ into deviatoric and volumetric parts:

$$\begin{aligned} \boldsymbol{\sigma}^{dev}(t + \Delta t) &= 2\mu \left[\boldsymbol{\epsilon}^{dev}(t + \Delta t) - \boldsymbol{\epsilon}^{creepdev}(t + \Delta t) - \boldsymbol{\epsilon}^{refdev} \right] + \boldsymbol{\sigma}^{refdev} \\ \boldsymbol{\sigma}^{dev}(t + \Delta t) &= \frac{1}{a_E} \left[\boldsymbol{\epsilon}^{dev}(t + \Delta t) - \boldsymbol{\epsilon}^{creepdev}(t + \Delta t) - \boldsymbol{\epsilon}^{refdev} \right] + \boldsymbol{\sigma}^{refdev} \\ P(t + \Delta t) &= 3K \left[\theta(t + \Delta t) - \theta^{ref} \right] + P^{ref} = \frac{1}{a_m} \left[\theta(t + \Delta t) - \theta^{ref} \right] + P^{ref}, \end{aligned} \quad (4.174)$$

where $\boldsymbol{\epsilon}^{dev}(t + \Delta t)$ is the total deviatoric strain, $\boldsymbol{\epsilon}^{creepdev}(t + \Delta t)$ is the total viscous strain, $\boldsymbol{\epsilon}^{refdev}$ is the reference deviatoric strain, $P(t + \Delta t)$ is the total pressure, $\theta(t + \Delta t)$ is the mean strain evaluated at time $t + \Delta t$ and θ^{ref} is the reference mean strain. The reference deviatoric stress and reference pressure are given by $\boldsymbol{\sigma}^{refdev}$ and P^{ref} , respectively. The middle equation in (4.174) may also be written as

$$\boldsymbol{\sigma}^{dev}(t + \Delta t) = \frac{1}{a_E} \left[\boldsymbol{\epsilon}^{dev}(t + \Delta t) - \Delta \boldsymbol{\epsilon}^{creepdev} \right] + \boldsymbol{\sigma}^{refdev}, \quad (4.175)$$

where

$$\begin{aligned}\epsilon^{dev}(t + \Delta t) &= \epsilon^{dev}(t + \Delta t) - \epsilon^{creepdev}(t) - \epsilon^{refdev}, \\ \Delta \epsilon^{creepdev} &= \epsilon^{creepdev}(t + \Delta t) - \epsilon^{creepdev}(t).\end{aligned}\quad (4.176)$$

The creep strain increment is approximated using

$$\Delta \epsilon^{creepdev} = \Delta t \gamma(\tau) \sigma^{dev}(\tau), \quad (4.177)$$

where, using the α -method of time integration,

$$\sigma^{dev}(\tau) = (1 - \alpha) \sigma^{dev}(ref, t) + \alpha \sigma^{dev}(ref, t + \Delta t) + \sigma^{refdev} = (1 - \alpha) \sigma^{dev}(t) + \alpha \sigma^{dev}(t + \Delta t), \quad (4.178)$$

and

$$\gamma(\tau) = \frac{3 \Delta \bar{\epsilon}^{creepdev}}{2 \Delta t \bar{\sigma}(\tau)}, \quad (4.179)$$

where

$$\Delta \bar{\epsilon}^{creepdev} = \sqrt{\frac{2}{3} \Delta \epsilon^{creepdev} : \Delta \epsilon^{creepdev}} \quad (4.180)$$

and

$$\bar{\sigma}(\tau) = (1 - \alpha) \bar{\sigma}(ref, t) + \alpha \bar{\sigma}(ref, t + \Delta t) + \bar{\sigma}^{ref} = \sqrt{3 J_2'(\tau)}. \quad (4.181)$$

Power-law Viscoelastic Models

Laboratory results on rock rheology are typically performed using a triaxial experiment, and the creep data are fit to a power-law equation of the form (e.g., [Kirby and Kronenberg, 1987]):

$$\dot{\epsilon}_{11}^{creep} = A_E \exp\left(\frac{-Q}{RT}\right) (\sigma_1 - \sigma_3)^n = A_E \exp\left(\frac{-Q}{RT}\right) \sigma_d^n, \quad (4.182)$$

where $\dot{\epsilon}_{11}^{creep}$ is the strain rate in the direction of the maximum principal stress (σ_1), A_E is the experimentally-derived pre-exponential constant, Q is the activation enthalpy, R is the universal gas constant, T is the absolute temperature, n is the power-law exponent, $\sigma_3 (= \sigma_2)$ is equal to the confining pressure, and σ_d is the differential stress. To properly formulate the flow law, it must be generalized so that the results are not influenced by the experiment type or the choice of coordinate systems (e.g., [Paterson, 1994]). The flow law may then be generalized in terms of the deviatoric stress and strain rate invariants:

$$\sqrt{\dot{L}_2^{creep}} = A_M \exp\left(\frac{-Q}{RT}\right) \sqrt{J_2'}^n, \quad (4.183)$$

where A_M is now a pre-exponential constant used in the formulation for modeling. In practice, it is necessary to compute each strain rate component using the flow law. This is accomplished using:

$$\dot{\epsilon}_{ij}^{creep} = A_M \exp\left(\frac{-Q}{RT}\right) \sqrt{J_2'}^{n-1} \sigma_{ij}^{dev}. \quad (4.184)$$

Note that (4.183) and (4.184) are consistent, since (4.183) may be obtained from (4.184) by taking the scalar inner product of both sides, multiplying by 1/2, and taking the square root.

In a triaxial experiment with confining pressure P_c , we have

$$\begin{aligned}\sigma_2 &= \sigma_3 = P_c \\ \sigma_1 &= \sigma_1^{app} \\ P &= \frac{\sigma_1 + 2P_c}{3},\end{aligned}\quad (4.185)$$

where σ_1^{app} is the applied load. The deviatoric stresses are then:

$$\begin{aligned}\sigma_1^{dev} &= \frac{2}{3}(\sigma_1 - P_c) \\ \sigma_2^{dev} = \sigma_3^{dev} &= -\frac{1}{3}(\sigma_1 - P_c).\end{aligned}\tag{4.186}$$

This gives

$$\begin{aligned}\sigma_1^{dev} &= \frac{2}{3}(\sigma_1 - \sigma_3) = \frac{2}{3}\sigma_d \\ \sigma_2^{dev} = \sigma_3^{dev} &= -\frac{1}{3}(\sigma_1 - \sigma_3) = -\frac{1}{3}\sigma_d.\end{aligned}\tag{4.187}$$

In terms of the second deviatoric stress invariant, we then have

$$\sqrt{J_2'} = \frac{\sigma_d}{\sqrt{3}}.\tag{4.188}$$

Under the assumption that the creep measured in the laboratory experiments is incompressible, we have

$$\begin{aligned}\dot{\epsilon}_{11}^{creepdev} &= \dot{\epsilon}_{11} \\ \dot{\epsilon}_{22}^{creepdev} = \dot{\epsilon}_{33} &= -\frac{1}{2}\dot{\epsilon}_{11}.\end{aligned}\tag{4.189}$$

In terms of the second deviatoric strain rate invariant we then have

$$\sqrt{\dot{L}_2^{creep}} = \frac{\sqrt{3}}{2}\dot{\epsilon}_{11}.\tag{4.190}$$

Substituting (4.188) and (4.190) into (4.182), we obtain

$$\sqrt{\dot{L}_2^{creep}} = A_E \frac{\sqrt{3}^{n+1}}{2} \exp\left(\frac{-Q}{RT}\right) \sqrt{J_2'}^n,\tag{4.191}$$

and therefore,

$$A_M = \frac{\sqrt{3}^{n+1}}{2} A_E.\tag{4.192}$$

When the exponential factor is included, we define a new parameter:

$$A_T = A_M \exp\left(\frac{-Q}{RT}\right) = \frac{\sqrt{3}^{n+1}}{2} A_E \exp\left(\frac{-Q}{RT}\right).\tag{4.193}$$

There is a problem with the usage of parameters A_E , A_M , and A_T . The dimensions of these parameters depend on the value of the power-law exponent; they are not really constants. In addition to being logically inconsistent, this presents problems when specifying parameters for PyLith, since the power-law exponent must be known before the units can be determined. An alternative way of writing the flow rule is (e.g., [Prentice, 1968]):

$$\frac{\sqrt{\dot{L}_2^{creep}}}{\dot{\epsilon}_0^{dev}} = \left(\frac{\sqrt{J_2'}}{\sigma_0^{dev}}\right)^n,\tag{4.194}$$

where $\dot{\epsilon}_0^{dev}$ and σ_0^{dev} are reference values for the deviatoric strain rate and stress. This means that

$$\frac{\dot{\epsilon}_0^{dev}}{\sigma_0^{dev}} = A_T.\tag{4.195}$$

Users must therefore specify three parameters for a power-law material. The properties `power_law_reference_strain_rate`, `power_law_reference_stress` and `power_law_exponent` in *Properties defining elasticity bulk rheologies* refer to $\dot{\epsilon}_0^{dev}$, σ_0^{dev} , and n , respectively. To specify the power-law properties for PyLith using laboratory results, the user must first compute A_T using (4.193). Then, values for $\dot{\epsilon}_0^{dev}$ and σ_0^{dev} must be provided. The simplest method is probably to assume a reasonable value for the reference strain rate, and then compute σ_0^{dev} as

$$\sigma_0^{dev} = \left(\frac{\dot{\epsilon}_0^{dev}}{A_T} \right)^{\frac{1}{n}}. \quad (4.196)$$

We provide `pylith_powerlaw_gendb` (see *pylith_powerlaw_gendb*) to convert laboratory results to the properties used by PyLith. To use the code you must specify the spatial variation of A_E , Q , n , and T . An additional parameter is given to define the units of A_E . You must also specify either a reference stress or a reference strain rate.

The flow law in component form is

$$\dot{\epsilon}_{ij}^{creepdev} = \frac{\dot{\epsilon}_0^{dev} \sqrt{J_2'}^{n-1} \sigma_{ij}^{dev}}{(\sigma_0^{dev})^n}, \quad (4.197)$$

and the creep strain increment is approximated as

$$\Delta \epsilon^{creepdev} \approx \frac{\Delta t \dot{\epsilon}_0^{creepdev} \sqrt{J_2'(\tau)}^{n-1} \sigma^{dev}(\tau)}{(\sigma_0^{dev})^n} = \frac{\Delta t \dot{\epsilon}_0^{creepdev} \sqrt{3} \bar{\sigma}(\tau)^{n-1} \sigma^{dev}(\tau)}{(\sqrt{3} \sigma_0^{dev})^n}. \quad (4.198)$$

Therefore,

$$\Delta \bar{\epsilon}^{creepdev} \approx \frac{2 \Delta t \dot{\epsilon}_0^{creepdev} \sqrt{J_2'(\tau)}^n}{\sqrt{3} (\sigma_0^{dev})^n} = \frac{2 \Delta t \dot{\epsilon}_0^{creepdev} \bar{\sigma}(\tau)^n}{\sqrt{3}^{n+1} (\sigma_0^{dev})^n}, \quad (4.199)$$

and

$$\gamma(\tau) = \frac{\dot{\epsilon}_0^{creepdev} \sqrt{J_2'(\tau)}^{n-1}}{(\sigma_0^{dev})^n}. \quad (4.200)$$

Combining (4.178), (4.198), (4.199), (4.200), and (4.175), we obtain:

$$\sigma^{dev}(t + \Delta t) = \frac{1}{a_E} \{ \epsilon'^{dev}(t + \Delta t) - \Delta t \gamma(\tau) [(1 - \alpha) \sigma^{dev}(t) + \alpha \sigma^{dev}(t + \Delta t)] \} + \sigma^{refdev}, \quad (4.201)$$

which we can rewrite as

$$\sigma^{dev}(t + \Delta t) [a_E + \alpha \Delta t \gamma(\tau)] = \epsilon'^{dev}(t + \Delta t) - \Delta t \gamma(\tau) (1 - \alpha) \sigma^{dev}(t) + a_E \sigma^{refdev}. \quad (4.202)$$

Taking the scalar inner product of both sides we obtain:

$$a^2 J_2'(t + \Delta t) - b + c \gamma(\tau) - d^2 \gamma(\tau)^2 = F = 0, \quad (4.203)$$

where

$$\begin{aligned} a &= a_E + \alpha \Delta t \gamma(\tau), \\ b &= \frac{1}{2} \epsilon'^{dev}(t + \Delta t) : \epsilon'^{dev}(t + \Delta t) + a_E \epsilon'^{dev}(t + \Delta t) : \sigma^{refdev} + a_E^2 J_2'^{ref}, \\ c &= \Delta t (1 - \alpha) \epsilon'^{dev}(t + \Delta t) : \sigma^{dev}(t) + \Delta t (1 - \alpha) a_E \sigma^{dev}(t) : \sigma^{refdev}, \\ d &= \Delta t (1 - \alpha) \sqrt{J_2'(t)}. \end{aligned} \quad (4.204)$$

Equation (4.203) is a function of a single unknown – the square root of the second deviatoric stress invariant at time $t + \Delta t$ – which we can solve by bisection or by Newton's method. Then we compute the deviatoric stresses for the current

time step from (4.181), (4.200), and (4.201). To compute the total stress we combine the deviatoric and volumetric components from (4.174).

To compute the tangent stress-strain relation, we first rewrite (4.202) as

$$\begin{aligned}\sigma_{ij}^{dev}(t + \Delta t) [a_E + \alpha \Delta t \gamma(\tau)] - \epsilon_{ij}'^{dev}(t + \Delta t) + \sigma_{ij}^{dev}(t) \Delta t \gamma(\tau) (1 - \alpha) - a_E \sigma_{ij}^{ref dev} &= 0, \\ \sigma_{ij}^{dev}(t + \Delta t) G - \epsilon_{ij}'^{dev}(t + \Delta t) + \sigma_{ij}^{dev}(t) H - a_E \sigma_{ij}^{ref dev} &= 0.\end{aligned}\quad (4.205)$$

Taking derivatives with respect to ϵ_{kl} , we obtain

$$G \frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{kl}} + \sigma_{ij}^{dev} \frac{\partial G}{\partial \sigma_{ij}^{dev}} \frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{kl}} - \frac{\partial \epsilon_{ij}'^{dev}}{\partial \epsilon_{kl}} + \sigma_{ij}^{dev}(t) \frac{\partial H}{\partial \sigma_{ij}^{dev}} \frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{kl}} = 0, \quad (4.206)$$

where we have removed the time notation for quantities evaluated at $t + \Delta t$. Rearranging, we compute the Jacobian for the deviatoric stress components using

$$\frac{\partial \sigma_{ij}^{dev}}{\partial \epsilon_{kl}} = \frac{\frac{\partial \epsilon_{ij}'^{dev}}{\partial \epsilon_{kl}}}{G + \sigma_{ij}^{dev} \frac{\partial G}{\partial \sigma_{ij}^{dev}} + \sigma_{ij}^{dev}(t) \frac{\partial H}{\partial \sigma_{ij}^{dev}}}.$$

Incompressible Elasticity

Infinitesimal Strain (Bathe) and No Faults

In this section we apply a similar approach to the one we use for the elasticity equation to the case of an incompressible material. As the bulk modulus (K) approaches infinity, the volumetric strain ($\text{Tr}(\epsilon)$) approaches zero and the pressure remains finite, $p = -K \text{Tr}(\epsilon)$. We consider pressure p as an independent variable and decompose the stress into the pressure and deviatoric components. As a result, we write the stress tensor in terms of both the displacement and pressure fields,

$$\boldsymbol{\sigma}(\vec{u}, p) = \boldsymbol{\sigma}^{dev}(\vec{u}) - p \mathbf{I}. \quad (4.208)$$

The strong form is

$$\vec{s}^T = (\vec{u} \quad p)^T, \quad (4.209)$$

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(\vec{x}, t) - (\boldsymbol{\sigma}^{dev}(\vec{u}) - p \mathbf{I}) = \vec{0} \text{ in } \Omega, \quad (4.210)$$

$$\vec{\nabla} \cdot \vec{u} + \frac{p}{K} = 0 \text{ in } \Omega, \quad (4.211)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau} \text{ on } \Gamma_\tau, \quad (4.212)$$

$$\vec{u} = \vec{u}_0 \text{ on } \Gamma_u, \quad (4.213)$$

$$p = p_0 \text{ on } \Gamma_p. \quad (4.214)$$

Table 4.6: Mathematical notation for incompressible elasticity with infinitesimal strain

Category	Symbol	Description
Unknowns	$\vec{u}$	Displacement field
	p	Pressure field ($p > 0$ corresponds to negative mean stress)
Derived quantities	$\boldsymbol{\sigma}$	Cauchy stress tensor
	$\boldsymbol{\sigma}^{dev}$	Cauchy deviatoric stress tensor
	ϵ	Cauchy strain tensor
Common constitutive parameters	ρ	Density
	μ	Shear modulus
	K	Bulk modulus
Source terms	$\vec{f}$	Body force per unit volume, for example $\rho \vec{g}$

Quastistatic

If we neglect the inertial term ($\rho \frac{\partial \vec{u}}{\partial t} \approx \vec{0}$), then time dependence only arises from history-dependent constitutive equations and boundary conditions. Our solution vector is the displacement vector and the elasticity equation reduces to

$$\vec{s}^T = (\vec{u} \quad p)^T, \quad (4.215)$$

$$\vec{f}(t) + \nabla \cdot (\boldsymbol{\sigma}^{dev}(\vec{u}) - p\mathbf{I}) = \vec{0} \text{ in } \Omega, \quad (4.216)$$

$$\vec{\nabla} \cdot \vec{u} + \frac{p}{K} = 0 \text{ in } \Omega, \quad (4.217)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau} \text{ on } \Gamma_\tau, \quad (4.218)$$

$$\vec{u} = \vec{u}_0 \text{ on } \Gamma_u, \quad (4.219)$$

$$p = p_0 \text{ on } \Gamma_p. \quad (4.220)$$

Because we will use implicit time stepping, we place all of the terms in the elasticity equation on the LHS. Using trial functions $\vec{\psi}_{trial}^u$ and ψ_{trial}^p and incorporating the Neumann boundary conditions, we write the weak form as

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{f}(t) + \nabla \vec{\psi}_{trial}^u : (-\boldsymbol{\sigma}^{dev}(\vec{u}) + p\mathbf{I}) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^u \cdot \vec{\tau}(t) d\Gamma = 0 \quad (4.221)$$

$$\int_{\Omega} \psi_{trial}^p \cdot \left(\vec{\nabla} \cdot \vec{u} + \frac{p}{K} \right) d\Omega = 0. \quad (4.222)$$

Residual Pointwise Functions

Identifying $F(t, s, \dot{s})$, we have

$$F^u(t, s, \dot{s}) = \int_{\Omega} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{f}(t)}_{f_0^u} + \nabla \vec{\psi}_{trial}^u : \underbrace{(-\boldsymbol{\sigma}^{dev}(\vec{u}) + p\mathbf{I})}_{f_1^u} d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{\tau}(t)}_{f_0^u} d\Gamma, \quad (4.223)$$

$$F^p(t, s, \dot{s}) = \int_{\Omega} \psi_{trial}^p \cdot \underbrace{\left(\vec{\nabla} \cdot \vec{u} + \frac{p}{K} \right)}_{f_0^p} d\Omega. \quad (4.224)$$

Jacobians Pointwise Functions

With two fields we have four Jacobian pointwise functions for the LHS:

$$J_F^{uu} = \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial u} (-\boldsymbol{\sigma}^{dev}) d\Omega = \int_{\Omega} \psi_{trial,i,k}^u \underbrace{(-C_{ikjl}^{dev})}_{J_{f3}^{uu}} \psi_{basis,j,l}^u d\Omega \quad (4.225)$$

$$J_F^{up} = \frac{\partial F^u}{\partial p} + s_{tshift} \frac{\partial F^u}{\partial \dot{p}} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \mathbf{I} \psi_{basis}^p d\Omega = \int_{\Omega} \psi_{trial,i,k}^u \underbrace{\delta_{ik}}_{J_{f2}^{up}} \psi_{basis}^p d\Omega \quad (4.226)$$

$$J_F^{pu} = \frac{\partial F^p}{\partial u} + s_{tshift} \frac{\partial F^p}{\partial \dot{u}} = \int_{\Omega} \psi_{trial}^p \left(\vec{\nabla} \cdot \vec{\psi}_{basis}^u \right) d\Omega = \int_{\Omega} \psi_{trial}^p \underbrace{\delta_{jl}}_{J_{f1}^{pu}} \psi_{basis,j,l}^u d\Omega \quad (4.227)$$

$$J_F^{pp} = \frac{\partial F^p}{\partial p} + s_{tshift} \frac{\partial F^p}{\partial \dot{p}} = \int_{\Omega} \psi_{trial}^p \underbrace{\frac{1}{K}}_{J_{f0}^{pp}} \psi_{basis}^p d\Omega \quad (4.228)$$

Nondimensionalization

Starting with the incompressible elasticity equation,

$$\rho(\vec{x}) \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(\vec{x}, t) - \nabla \cdot \boldsymbol{\sigma} = \vec{0} \text{ in } \Omega, \quad (4.229)$$

$$\vec{\nabla} \cdot \vec{u} + \frac{p}{K} = 0 \text{ in } \Omega, \text{ where} \quad (4.230)$$

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{dev} - p\mathbf{I}. \quad (4.231)$$

we define nondimensional values:

$$\vec{x}^* = \frac{\vec{x}}{x_o}, \quad (4.232)$$

$$\vec{u}^* = \frac{\vec{u}}{u_o}, \quad (4.233)$$

$$p^* = \frac{p}{p_o}, \quad (4.234)$$

$$\rho^* = \frac{\rho}{\rho_o}, \quad (4.235)$$

$$\vec{f}^* = \frac{\vec{f}}{f_o}, \quad (4.236)$$

$$\boldsymbol{\sigma}^* = \frac{\boldsymbol{\sigma}}{\sigma_o}, \quad (4.237)$$

$$\vec{\tau}^* = \frac{\vec{\tau}}{\sigma_o}. \quad (4.238)$$

As in the case of the elasticity equation, we also recognize that

$$\nabla^* = x_o \nabla. \quad (4.239)$$

Substituting into the equations, we have

$$\rho_o \rho^*(\vec{x}^*) \frac{u_o}{t_o^2} \frac{\partial^2 \vec{u}^*}{\partial t^{*2}} - f_o \vec{f}^*(\vec{x}^*, t^*) - \frac{\sigma_o}{x_o} \nabla^* \cdot \boldsymbol{\sigma}^* = \vec{0} \text{ in } \Omega, \quad (4.240)$$

$$\frac{u_o}{x_o} \left(\vec{\nabla}^* \cdot \vec{u}^* \right) + \frac{p_o}{\mu_o} \frac{p^*}{K^*} = 0 \text{ in } \Omega. \quad (4.241)$$

Grouping terms we have

$$\left(\rho_o \frac{u_o}{t_o^2} \frac{x_o}{\sigma_o} \right) \rho^*(\vec{x}^*) \frac{\partial^2 \vec{u}^*}{\partial t^{*2}} - \left(f_o \frac{x_o}{\sigma_o} \right) \vec{f}^*(\vec{x}^*, t^*) - \nabla^* \cdot \boldsymbol{\sigma}^* = \vec{0} \text{ in } \Omega, \quad (4.242)$$

$$\vec{\nabla}^* \cdot \vec{u}^* + \frac{p_o u_o}{\mu_o x_o} \frac{p^*}{K^*} = 0 \text{ in } \Omega. \quad (4.243)$$

All terms should be nondimensional, which implies

$$f_o = \frac{\sigma_o}{x_o}, \quad (4.244)$$

$$\rho_o = \frac{t_o^2 \sigma_o}{u_o x_o}, \quad (4.245)$$

$$p_o = \mu_o \frac{x_o}{u_o}. \quad (4.246)$$

Note

In PyLith v4 and earlier, we used a displacement scale equal to the length scale. Using separate length and displacement scales facilitates accurately solving for displacements many orders of magnitude smaller than the length scale. It also separates the stress scale used to nondimensionalize stress, tractions, and pressure from the rigidity scale.

Table 4.7: Nondimensionalization of incompressible elasticity.

Quantity	Scale
x_o	Length scale
u_o	Displacement scale
μ_o	Rigidity scale
t_o	Time scale
$\sigma_o = p_o = \mu_o \frac{u_o}{x_o}$	Stress scale
$f_o = \frac{\sigma_o}{x_o}$	Body force scale
$\rho_o = \mu_o \frac{t_o^2}{x_o^2}$	Density scale

Bulk Rheologies

In this section we describe the mathematic formulations of the bulk rheologies for incompressible elasticity. The bulk rheologies include

- Isotropic linear elasticity,

Linear Isotropic Incompressible Elastic Models

For isotropic, linear incompressible elasticity, the deviatoric elastic constants are

$$C_{1111}^{dev} = C_{2222}^{dev} = C_{3333}^{dev} = +\frac{4}{3}\mu \quad (4.247)$$

$$C_{1122}^{dev} = C_{1133}^{dev} = C_{2233}^{dev} = -\frac{2}{3}\mu \quad (4.248)$$

$$C_{1212}^{dev} = C_{1313}^{dev} = C_{2323}^{dev} = \mu \quad (4.249)$$

Porolasticity**Infinitesimal Strain and No Faults**

We base this formulation for poroelasticity on Detournay and Cheng [1993] and Cheng [2016]. We assume a slightly compressible fluid that completely saturates a porous solid, undergoing infinitesimal strain.

We begin with the conservation of linear momentum, including inertia, borrowed from linear elasticity:

$$\rho_s \frac{\partial^2 \vec{u}}{\partial t^2} = \vec{f}(t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}, p). \quad (4.250)$$

Enforcing mass balance of the fluid gives

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} + \nabla \cdot \vec{q}(p) = \gamma(\vec{x}, t) \text{ in } \Omega, \quad (4.251)$$

$$\vec{q} \cdot \vec{n} = q_0(\vec{x}, t) \text{ on } \Gamma_q, \quad (4.252)$$

$$p = p_0(\vec{x}, t) \text{ on } \Gamma_p, \quad (4.253)$$

where ζ is the variation in fluid content, $\vec{q}$ is the rate of fluid volume crossing a unit area of the porous solid, γ is the rate of injected fluid per unit volume of the porous solid, q_0 is the outward fluid velocity normal to the boundary Γ_q , and p_0 is the fluid pressure on boundary Γ_p .

We require the fluid flow to follow Darcy's law (Navier-Stokes equation neglecting inertial effects),

$$\vec{q}(p) = -\frac{\mathbf{k}}{\mu_f}(\nabla p - \vec{f}_f), \quad (4.254)$$

where $\mathbf{k}$ is the intrinsic permeability, μ_f is the viscosity of the fluid, p is the fluid pressure, and $\vec{f}_f$ is the body force in the fluid. If gravity is included in a problem, then usually $\vec{f}_f = \rho_f \vec{g}$, where ρ_f is the density of the fluid and $\vec{g}$ is the gravitational acceleration vector.

Constitutive Behavior

We assume linear elasticity for the solid phase, so the constitutive behavior can be expressed as

$$\boldsymbol{\sigma}(\vec{u}, p) = \mathbf{C} : \boldsymbol{\epsilon} - \alpha p \mathbf{I}, \quad (4.255)$$

where $\boldsymbol{\sigma}$ is the stress tensor, $\mathbf{C}$ is the tensor of elasticity constants, α is the Biot coefficient (effective stress coefficient), $\boldsymbol{\epsilon}$ is the strain tensor, and $\mathbf{I}$ is the identity tensor.

For the constitutive behavior of the fluid, we use the volumetric strain to couple the fluid-solid behavior,

$$\zeta(\vec{u}, p) = \alpha \text{Tr}(\boldsymbol{\epsilon}) + \frac{p}{M}, \quad (4.256)$$

$$\frac{1}{M} = \frac{\alpha - \phi}{K_s} + \frac{\phi}{K_f}, \quad (4.257)$$

where $1/M$ is the specific storage coefficient at constant strain, K_s is the bulk modulus of the solid, and K_f is the bulk modulus of the fluid. We can write the trace of the strain tensor as the dot product of the gradient and displacement field, so we have

$$\zeta(\vec{u}, p) = \alpha(\nabla \cdot \vec{u}) + \frac{p}{M}. \quad (4.258)$$

Table 4.8: Mathematical notation for poroelasticity with infinitesimal strain.

Category	Sym- bol	Description
Unknowns	$\vec{u}$	Displacement field
	$\vec{v}$	Velocity field
	p	Pressure field (corresponds to pore fluid pressure)
	ϵ_v	Volumetric (trace) strain
Derived quantities	$\boldsymbol{\sigma}$	Cauchy stress tensor
	$\boldsymbol{\epsilon}$	Cauchy strain tensor
	ζ	Variation of fluid content (variation of fluid vol. per unit vol. of PM), $\alpha\epsilon_v + \frac{p}{M}$
	ρ_b	Bulk density, $(1 - \phi)\rho_s + \phi\rho_f$
	$\vec{q}$	Darcy flux, $-\frac{\mathbf{k}}{\mu_f} \cdot (\nabla p - \vec{f}_f)$
Common constitutive parameters	ρ_s	Solid (matrix) density
	ρ_f	Fluid density
	μ_f	Fluid viscosity
	ϕ	Porosity
	μ	Shear modulus
	K_d	Drained bulk modulus
	α	Biot coefficient, $1 - \frac{K_d}{K_s}$
	M	Biot modulus
	$\mathbf{k}$	Permeability
	$\vec{f}$	Body force per unit volume, for example: $\rho_b\vec{g}$
Source terms	$\vec{f}_f$	Fluid body force, for example: $\rho_f\vec{g}$
	γ	Source density; rate of injected fluid per unit volume of the porous solid

Quasistatic

For ease of solution in the quasistatic case, we introduce a third variable in the form of volumetric strain (ϵ_v). The strong form of the problem may be expressed as

$$\vec{s}^T = (\vec{u} \quad p \quad \epsilon_v), \quad (4.259)$$

$$\vec{f}(t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) = \vec{0} \text{ in } \Omega, \quad (4.260)$$

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} - \gamma(\vec{x}, t) + \nabla \cdot \vec{q}(p) = 0 \text{ in } \Omega, \quad (4.261)$$

$$\nabla \cdot \vec{u} - \epsilon_v = 0 \text{ in } \Omega, \quad (4.262)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau, \quad (4.263)$$

$$\vec{q} \cdot \vec{n} = q_0(\vec{x}, t) \text{ on } \Gamma_q, \quad (4.264)$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u, \text{ and} \quad (4.265)$$

$$p = p_0(\vec{x}, t) \text{ on } \Gamma_p. \quad (4.266)$$

We place all terms for the elasticity, pressure, and volumetric strain equations on the left-hand-side, consistent with PETSc TS implicit time stepping.

We create the weak form by taking the dot product with the trial functions ψ_{trial}^u , ψ_{trial}^p , and $\psi_{trial}^{\epsilon_v}$ and integrating over

the domain:

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \left(\vec{f}(\vec{x}, t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) \right) d\Omega = 0, \quad (4.267)$$

$$\int_{\Omega} \psi_{trial}^p \left(\frac{\partial \zeta(\vec{u}, p)}{\partial t} - \gamma(\vec{x}, t) + \nabla \cdot \vec{q}(p) \right) d\Omega = 0, \quad (4.268)$$

$$\int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot (\nabla \cdot \vec{u} - \epsilon_v) d\Omega. \quad (4.269)$$

Applying the divergence theorem to the first two equations and incorporating the Neumann boundary conditions yields

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^u : -\boldsymbol{\sigma}(\vec{u}, p_f) d\Omega + \int_{\Gamma_{\tau}} \vec{\psi}_{trial}^u \cdot \vec{\tau}(\vec{x}, t) d\Gamma = 0, \quad (4.270)$$

$$\int_{\Omega} \psi_{trial}^p \left(\frac{\partial \zeta(\vec{u}, p_f)}{\partial t} - \gamma(\vec{x}, t) \right) + \nabla \psi_{trial}^p \cdot (-\vec{q}(p_f)) d\Omega + \int_{\Gamma_q} \psi_{trial}^p q_0(\vec{x}, t) d\Gamma = 0, \text{ and} \quad (4.271)$$

$$\int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot (\nabla \cdot \vec{u} - \epsilon_v) d\Omega = 0 \quad (4.272)$$

Residual Pointwise Functions

Identifying $F(t, s, \dot{s})$ and $G(t, s)$ we have

$$F^u(t, s, \dot{s}) = \int_{\Omega} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{f}(\vec{x}, t)}_{\vec{f}_0^u} + \nabla \vec{\psi}_{trial}^u : \underbrace{-\boldsymbol{\sigma}(\vec{u}, p_f)}_{\vec{f}_1^u} d\Omega + \int_{\Gamma_{\tau}} \vec{\psi}_{trial}^u \cdot \underbrace{\vec{\tau}(\vec{x}, t)}_{\vec{f}_0^u} d\Gamma, \quad (4.273)$$

$$G^u(t, s) = 0, \quad (4.274)$$

$$F^p(t, s, \dot{s}) = \int_{\Omega} \psi_{trial}^p \underbrace{\left(\frac{\partial \zeta(\vec{u}, p_f)}{\partial t} - \gamma(\vec{x}, t) \right)}_{f_0^p} + \nabla \psi_{trial}^p \cdot \underbrace{-\vec{q}(p_f)}_{\vec{f}_1^p} d\Omega + \int_{\Gamma_q} \psi_{trial}^p \underbrace{(q_0(\vec{x}, t))}_{f_0^p} d\Gamma, \quad (4.275)$$

$$G^p(t, s) = 0, \quad (4.276)$$

$$F^{\epsilon_v}(t, s, \dot{s}) = \int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot \underbrace{(\nabla \cdot \vec{u} - \epsilon_v)}_{f_0^{\epsilon_v}} d\Omega. \quad (4.277)$$

$$G^{\epsilon_v} = 0. \quad (4.278)$$

Jacobian Pointwise Functions

Three fields yields potentially 9 Jacobian pointwise functions for the LHS:

$$J_F^{uu} = \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial u} (-\sigma(\vec{u}, p, \epsilon_v)) d\Omega = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial u} (-(\mathbf{C} : \epsilon - \alpha p \mathbf{I})) d\Omega \quad (4.279)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : -\mathbf{C} : \frac{1}{2} (\nabla + \nabla^T) \vec{\psi}_{basis}^u d\Omega = \int_{\Omega} \psi_{trial,i,k}^u \underbrace{(-C_{ikjl})}_{J_{f3}^{uu}} \psi_{basis,j,l}^u d\Omega \quad (4.280)$$

$$J_F^{up} = \frac{\partial F^u}{\partial p} + s_{tshift} \frac{\partial F^u}{\partial \dot{p}} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial p} (-(\mathbf{C} : \epsilon - \alpha p \mathbf{I})) d\Omega = \int_{\Omega} \psi_{trial,i,j}^u \underbrace{(\alpha \delta_{ij})}_{J_{f2}^{up}} \psi_{basis}^p d\Omega \quad (4.281)$$

$$J_F^{u\epsilon_v} = \frac{\partial F^u}{\partial \epsilon_v} + s_{tshift} \frac{\partial F^u}{\partial \dot{\epsilon}_v} = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial \epsilon_v} (-\sigma(\vec{u}, p, \epsilon_v)) d\Omega = \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial \epsilon_v} (-(\mathbf{C} : \epsilon - \alpha p \mathbf{I})) d\Omega \quad (4.282)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial \epsilon_v} [-(2\mu\epsilon + \lambda \mathbf{I}\epsilon_v - \alpha p \mathbf{I})] d\Omega = \int_{\Omega} \psi_{trial,i,j}^u \underbrace{(-\lambda \delta_{ij})}_{J_{f2}^{u\epsilon_v}} \psi_{basis}^{\epsilon_v} d\Omega \quad (4.283)$$

$$J_F^{pu} = \frac{\partial F^p}{\partial u} + s_{tshift} \frac{\partial F^p}{\partial \dot{u}} = 0 \quad (4.284)$$

$$J_F^{pp} = \frac{\partial F^p}{\partial p} + s_{tshift} \frac{\partial F^p}{\partial \dot{p}} = \int_{\Omega} \nabla \psi_{trial}^p \cdot \frac{\partial}{\partial p} - \left[-\frac{\mathbf{k}}{\mu_f} (\nabla p - \vec{f}) \right] d\Omega + s_{tshift} \int_{\Omega} \psi_{trial}^p \frac{\partial}{\partial \dot{p}} \left[\alpha \dot{\epsilon}_v + \frac{\dot{p}}{M} - \gamma(\vec{x}, t) \right] d\Omega \quad (4.285)$$

$$= \int_{\Omega} \nabla \psi_{trial}^p \left(\frac{\mathbf{k}}{\mu_f} \nabla \cdot \psi_{basis}^p \right) d\Omega + \int_{\Omega} \psi_{trial}^p \left(s_{tshift} \frac{1}{M} \right) \psi_{basis}^p d\Omega \quad (4.286)$$

$$= \int_{\Omega} \psi_{trial,k}^p \underbrace{\left(\frac{\mathbf{k}}{\mu_f} \delta_{kl} \right)}_{J_{f3}^{pp}} \psi_{basis,l}^p d\Omega + \int_{\Omega} \psi_{trial}^p \underbrace{\left(s_{tshift} \frac{1}{M} \right)}_{J_{f0}^{pp}} \psi_{basis}^p d\Omega \quad (4.287)$$

$$J_F^{p\epsilon_v} = \frac{\partial F^p}{\partial \epsilon_v} + s_{tshift} \frac{\partial F^p}{\partial \dot{\epsilon}_v} = \int_{\Omega} \psi_{trial}^p \underbrace{(s_{tshift} \alpha)}_{J_{f0}^{p\epsilon_v}} \psi_{basis}^{\epsilon_v} d\Omega \quad (4.288)$$

$$J_F^{\epsilon_v u} = \frac{\partial F^{\epsilon_v}}{\partial u} + s_{tshift} \frac{\partial F^{\epsilon_v}}{\partial \dot{u}} = \int_{\Omega} \psi_{trial}^{\epsilon_v} \nabla \cdot \vec{\psi}_{basis}^u d\Omega = \int_{\Omega} \psi_{basis}^{\epsilon_v} \underbrace{(\delta_{ij})}_{J_{f1}^{\epsilon_v u}} \psi_{basis,i,j}^u d\Omega \quad (4.289)$$

$$J_F^{\epsilon_v p} = \frac{\partial F^{\epsilon_v}}{\partial p} + s_{tshift} \frac{\partial F^{\epsilon_v}}{\partial \dot{p}} = 0 \quad (4.290)$$

$$J_F^{\epsilon_v \epsilon_v} = \frac{\partial F^{\epsilon_v}}{\partial \epsilon_v} + s_{tshift} \frac{\partial F^{\epsilon_v}}{\partial \dot{\epsilon}_v} = \int_{\Omega} \psi_{basis}^{\epsilon_v} \underbrace{(-1)}_{J_{f0}^{\epsilon_v \epsilon_v}} \psi_{basis}^{\epsilon_v} d\Omega \quad (4.291)$$

Porosity State Variable

The default poroelasticity implementation uses a porosity that can vary in space but does not vary in time. PyLith also includes an implementation with porosity as a state variable that allows it to evolve in time as well as vary in space.

For isothermal conditions Detournay and Cheng [1993] derive a differential equation for the change in porosity:

$$\frac{\partial \phi(\vec{x}, t)}{\partial t} = (\alpha(\vec{x}) - \phi(\vec{x}, t)) \left(\dot{\epsilon}_v(\vec{x}, t) + \frac{1 - \alpha(\vec{x})}{K_d(\vec{x})} \dot{p}(\vec{x}, t) \right) \quad (4.292)$$

If we use the approximation

$$\frac{\partial \phi(\vec{x}, t)}{\partial t} = \frac{\phi(\vec{x}, t + \Delta t) - \phi(\vec{x}, t)}{\Delta t}, \quad (4.293)$$

then we can update the porosity after advancing the solution Δt using

$$\phi(\vec{x}, t + \Delta t) = \phi(\vec{x}, t) + \Delta t \left((\alpha(\vec{x}) - \phi(\vec{x}, t)) \left(\dot{\epsilon}_v(\vec{x}, t) + \frac{1 - \alpha(\vec{x})}{K_d(\vec{x})} \dot{p}(\vec{x}, t) \right) \right). \quad (4.294)$$

Note

In updating the porosity state variable, we limit values to between 0 and 1, inclusively.

When updating the state variables, PETSc provides the values of the current solution as well as the auxiliary field. Because the expression for the porosity change depends on the time derivative of the trace strain and fluid pressure, we include those subfields in the solution field.

Danger

In v4.0 we also include the velocity (time derivative of the displacement) in the solution field. This is not necessary, so we plan to remove it in v5.0.

Our full set of governing equations is

$$\vec{s}^T = (\vec{u} \quad p \quad \epsilon_v \quad \vec{v} \quad \dot{p} \quad \dot{\epsilon}_v), \quad (4.295)$$

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^u : -\boldsymbol{\sigma}(\vec{u}, p_f) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^u \cdot \vec{\tau}(\vec{x}, t) d\Gamma = 0, \quad (4.296)$$

$$\int_{\Omega} \psi_{trial}^p \left(\frac{\partial \zeta(\vec{u}, p_f)}{\partial t} - \gamma(\vec{x}, t) \right) + \nabla \psi_{trial}^p \cdot (-\vec{q}(p_f)) d\Omega + \int_{\Gamma_q} \psi_{trial}^p q_0(\vec{x}, t) d\Gamma = 0, \text{ and} \quad (4.297)$$

$$\int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot (\nabla \cdot \vec{u} - \epsilon_v) d\Omega = 0 \quad (4.298)$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{v} \quad (4.299)$$

$$\frac{\partial p}{\partial t} = p_t \quad (4.300)$$

$$\frac{\partial \epsilon_v}{\partial t} = \epsilon_{vt}. \quad (4.301)$$

Residual Pointwise Functions

In addition to the residual functions for the case in which porosity does not evolve in time, we have

$$F^v(t, s, \dot{s}) = \int_{\Omega} \underbrace{\vec{\psi}_{trial}^v \cdot \left(\frac{\partial \vec{u}}{\partial t} - \vec{v} \right)}_{f_0^v} d\Omega, \quad (4.302)$$

$$G^v(t, s) = 0, \quad (4.303)$$

$$F^{p_t}(t, s, \dot{s}) = \int_{\Omega} \underbrace{\psi_{trial}^{p_t} \cdot \left(\frac{\partial p}{\partial t} - p_t \right)}_{f_0^{p_t}} d\Omega, \quad (4.304)$$

$$G^{p_t}(t, s) = 0, \quad (4.305)$$

$$F^{\epsilon_{vt}}(t, s, \dot{s}) = \int_{\Omega} \underbrace{\psi_{trial}^{\epsilon_{vt}} \cdot \left(\frac{\partial \epsilon_v}{\partial t} - \epsilon_{vt} \right)}_{f_0^{\epsilon_{vt}}} d\Omega, \quad (4.306)$$

$$G^{\epsilon_{vt}}(t, s) = 0. \quad (4.307)$$

Jacobian Pointwise Functions

The corresponding additional Jacobian functions are

$$J_F^{vu} = \frac{\partial F^v}{\partial u} + s_{tshift} \frac{\partial F^v}{\partial \dot{u}} = s_{tshift}, \quad (4.308)$$

$$J_F^{vv} = \frac{\partial F^v}{\partial v} + s_{tshift} \frac{\partial F^v}{\partial \dot{v}} = -1, \quad (4.309)$$

$$J_F^{p_t p} = \frac{\partial F^{p_t}}{\partial p} + s_{tshift} \frac{\partial F^{p_t}}{\partial \dot{p}} = s_{tshift}, \quad (4.310)$$

$$J_F^{p_t p_t} = \frac{\partial F^{p_t}}{\partial p_t} + s_{tshift} \frac{\partial F^{p_t}}{\partial \dot{p}_t} = -1, \quad (4.311)$$

$$J_F^{\epsilon_{vt} \epsilon_v} = \frac{\partial F^{\epsilon_{vt}}}{\partial \epsilon_v} + s_{tshift} \frac{\partial F^{\epsilon_{vt}}}{\partial \dot{\epsilon}_t} = s_{tshift}, \quad (4.312)$$

$$J_F^{\epsilon_{vt} \epsilon_{vt}} = \frac{\partial F^{\epsilon_{vt}}}{\partial \epsilon_{vt}} + s_{tshift} \frac{\partial F^{\epsilon_{vt}}}{\partial \dot{\epsilon}_{vt}} = -1, \quad (4.313)$$

$$(4.314)$$

Dynamic

TODO @rwalkerlewis

This section will need to be updated for consistency with the dynamic prescribed slip formulation.

For compatibility with PETSc TS algorithms, we want to turn the second order elasticity equation into two first order equations. We introduce velocity as a unknown, $\vec{v} = \frac{\partial \vec{u}}{\partial t}$, which leads to a slightly different three field problem,

$$\vec{s}^T = (\vec{u} \quad p \quad \vec{v}) \quad (4.315)$$

$$\frac{\partial \vec{u}}{\partial t} = \vec{v} \text{ in } \Omega \quad (4.316)$$

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} - \gamma(\vec{x}, t) + \nabla \cdot \vec{q}(p) = 0 \text{ in } \Omega \quad (4.317)$$

$$\rho_b \frac{\partial \vec{v}}{\partial t} = \vec{f}(\vec{x}, t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) \text{ in } \Omega \quad (4.318)$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau \quad (4.319)$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u \quad (4.320)$$

$$\vec{q} \cdot \vec{n} = q_0(\vec{x}, t) \text{ on } \Gamma_q \quad (4.321)$$

$$p = p_0(\vec{x}, t) \text{ on } \Gamma_p \quad (4.322)$$

For compatibility with PETSc TS explicit time stepping algorithms, we need the left hand side to be $F = (t, s, \dot{s}) = \dot{s}$. We replace the variation of fluid content variable, ζ , with its definition in the conservation of fluid mass equation and solve for the rate of change of pressure,

$$\frac{\partial}{\partial t} \left(\alpha \epsilon_v + \frac{p}{M} \right) - \gamma(\vec{x}, t) + \nabla \cdot \vec{q} = 0 \quad (4.323)$$

$$\alpha \dot{\epsilon}_v + \frac{\dot{p}}{M} - \gamma(\vec{x}, t) + \nabla \cdot \vec{q} = 0 \quad (4.324)$$

$$\frac{\dot{p}}{M} = \gamma(\vec{x}, t) - \alpha \dot{\epsilon}_v - \nabla \cdot \vec{q} \quad (4.325)$$

$$\frac{\dot{p}}{M} = \gamma(\vec{x}, t) - \alpha \left(\nabla \cdot \dot{\vec{u}} \right) - \nabla \cdot \vec{q}. \quad (4.326)$$

We write the volumetric strain in terms of displacement, because this dynamic formulation does not include the volumetric strain as an unknown. Note that for poroelastodynamics we use the generalized Darcy's law [Ding *et al.*, 2013] as

$$\vec{q}(p) = -\frac{\mathbf{k}}{\mu_f} (\nabla p - \vec{f}_f + \rho_f \frac{\partial \vec{v}}{\partial t}), \quad (4.327)$$

where the generalized Darcy's law adds the term $\rho_f \frac{\partial \vec{v}}{\partial t}$. Using trial functions $\vec{\psi}_{trial}^u$, ψ_{trial}^p , and $\vec{\psi}_{trial}^v$, and incorporating the Neumann boundary conditions, the weak form may be written as:

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \left(\frac{\partial \vec{u}}{\partial t} \right) d\Omega = \int_{\Omega} \vec{\psi}_{trial}^u \cdot (\vec{v}) d\Omega \quad (4.328)$$

$$\int_{\Omega} \psi_{trial}^p \left(\frac{1}{M} \frac{\partial p}{\partial t} \right) d\Omega = \int_{\Omega} \psi_{trial}^p \left[\gamma(\vec{x}, t) - \alpha \left(\nabla \cdot \dot{\vec{u}} \right) \right] + \nabla \psi_{trial}^p \cdot \vec{q}(p) d\Omega + \int_{\Gamma_q} \psi_{trial}^p (-q_0(\vec{x}, t)) d\Gamma, \quad (4.329)$$

$$\int_{\Omega} \vec{\psi}_{trial}^v \cdot \left(\rho_b \frac{\partial \vec{v}}{\partial t} \right) d\Omega = \int_{\Omega} \vec{\psi}_{trial}^v \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^v : -\boldsymbol{\sigma}(\vec{u}, p_f) d\Omega + \int_{\Gamma_\tau} \vec{\psi}_{trial}^v \cdot \vec{\tau}(\vec{x}, t) d\Gamma. \quad (4.330)$$

Residual Pointwise Functions

With explicit time stepping the PETSc TS assumes the LHS is $\dot{s}$, so we only need the RHS residual functions:

$$G^u(t, s) = \int_{\Omega} \underbrace{\vec{\psi}_{trial}^u \cdot \vec{v}}_{\vec{g}_0^u} d\Omega, \quad (4.331)$$

$$G^p(t, s) = \int_{\Omega} \underbrace{\psi_{trial}^p \left(\gamma(\vec{x}, t) - \alpha(\nabla \cdot \vec{u}) \right)}_{g_0^p} + \nabla \psi_{trial}^p \cdot \underbrace{\vec{q}(p_f)}_{\vec{g}_1^p} d\Omega + \int_{\Gamma_q} \psi_{trial}^p \underbrace{(-q_0(\vec{x}, t))}_{g_0^p} d\Gamma, \quad (4.332)$$

$$G^v(t, s) = \int_{\Omega} \underbrace{\vec{\psi}_{trial}^v \cdot \vec{f}(\vec{x}, t)}_{\vec{g}_0^v} + \nabla \psi_{trial}^v : \underbrace{-\boldsymbol{\sigma}(\vec{u}, p_f)}_{\mathbf{g}_1^v} d\Omega + \int_{\Gamma_{\tau}} \underbrace{\vec{\psi}_{trial}^u \cdot \vec{\tau}(\vec{x}, t)}_{\vec{g}_0^v} d\Gamma. \quad (4.333)$$

Jacobians Pointwise Functions

These are the pointwise functions associated with M_u , M_p , and M_v for computing the lumped LHS Jacobian. We premultiply the RHS residual function by the inverse of the lumped LHS Jacobian while s_{tshift} remains on the LHS with $\dot{s}$. As a result, we use LHS Jacobian pointwise functions, but set $s_{tshift} = 1$. The LHS Jacobians are:

$$M_u = J_F^{uu} = \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} = \int_{\Omega} \underbrace{\psi_{trial}^u s_{tshift} \delta_{ij} \psi_{basis}^u}_{J_{f0}^{uu}} d\Omega \quad (4.334)$$

$$M_p = J_F^{pp} = \frac{\partial F^p}{\partial p} + s_{tshift} \frac{\partial F^p}{\partial \dot{p}} = \int_{\Omega} \underbrace{\psi_{trial}^p \left(s_{tshift} \frac{1}{M} \right) \psi_{basis}^p}_{J_{f0}^{pp}} d\Omega \quad (4.335)$$

$$M_v = J_F^{vv} = \frac{\partial F^v}{\partial v} + s_{tshift} \frac{\partial F^v}{\partial \dot{v}} = \int_{\Omega} \underbrace{\psi_{trial}^v \rho_b(\vec{x}) s_{tshift} \delta_{ij} \psi_{basis}^v}_{J_{f0}^{vv}} d\Omega \quad (4.336)$$

Nondimensionalization

Starting with the equations governing poroelasticity,

$$\rho_s \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f}(t) - \nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) = \vec{0} \text{ in } \Omega, \quad (4.337)$$

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} + \nabla \cdot \vec{q}(p) - \gamma(\vec{x}, t) = 0 \text{ in } \Omega, \quad (4.338)$$

$$\vec{q}(p) = -\frac{\mathbf{k}}{\mu_f} (\nabla p - \vec{f}_f), \quad (4.339)$$

we define nondimensional values:

$$\vec{x}^* = \frac{\vec{x}}{x_o}, \quad (4.340)$$

$$\vec{u}^* = \frac{\vec{u}}{u_o}, \quad (4.341)$$

$$p^* = \frac{p}{p_o}, \quad (4.342)$$

$$\rho_s^* = \frac{\rho_s}{\rho_o}, \quad (4.343)$$

$$\rho_f^* = \frac{\rho_f}{\rho_o}, \quad (4.344)$$

$$\vec{f}^* = \frac{\vec{f}}{f_o}, \quad (4.345)$$

$$\zeta^* = \frac{\zeta}{\zeta_o}, \quad (4.346)$$

$$\vec{q}^* = \frac{\vec{q}}{q_o}, \quad (4.347)$$

$$\gamma^* = \frac{\gamma}{\gamma_o}, \quad (4.348)$$

$$\boldsymbol{\sigma}^* = \frac{\boldsymbol{\sigma}}{\sigma_o}, \quad (4.349)$$

$$\vec{\tau}^* = \frac{\vec{\tau}}{\sigma_o}. \quad (4.350)$$

We also recognize that

$$\nabla^* = x_o \nabla. \quad (4.351)$$

Substituting into the equations, we have

$$\rho_o \rho^*(\vec{x}^*) \frac{u_o}{t_o^2} \frac{\partial^2 \vec{u}^*}{\partial t^{*2}} - f_o \vec{f}^*(\vec{x}^*, t^*) - \frac{\sigma_o}{x_o} \nabla^* \cdot \boldsymbol{\sigma}^* = \vec{0} \text{ in } \Omega, \quad (4.352)$$

$$\frac{\zeta_o}{t_o} \frac{\partial \zeta^*(\vec{u}, p)}{\partial t^*} + \frac{q_o}{x_o} (\nabla^* \cdot \vec{q}^*(p)) - \gamma_o \gamma^*(\vec{x}, t) = 0 \text{ in } \Omega, \quad (4.353)$$

$$q_o \vec{q}^*(p) = -\frac{\mathbf{k}}{\mu_f} \left(\frac{p_o}{x_o} \nabla^* p^* - f_o \vec{f}_f^* \right). \quad (4.354)$$

Grouping terms we have

$$\left(\rho_o \frac{u_o}{t_o^2} \frac{x_o}{\sigma_o} \right) \rho^*(\vec{x}^*) \frac{\partial^2 \vec{u}^*}{\partial t^{*2}} - \left(f_o \frac{x_o}{\sigma_o} \right) \vec{f}^*(\vec{x}^*, t^*) - \nabla^* \cdot \boldsymbol{\sigma}^* = \vec{0} \text{ in } \Omega, \quad (4.355)$$

$$\left(\frac{\zeta_o}{t_o} \right) \frac{\partial \zeta^*(\vec{u}, p)}{\partial t^*} + \left(\frac{q_o}{x_o} \right) (\nabla^* \cdot \vec{q}^*(p)) - \gamma_o \gamma^*(\vec{x}, t) = 0 \text{ in } \Omega, \quad (4.356)$$

$$q_o \vec{q}^*(p) = - \left(\frac{p_o}{x_o} \frac{\mathbf{k}}{\mu_f} \right) (\nabla^* p^* - f_o \frac{x_o}{p_o} \vec{f}_f^*). \quad (4.357)$$

We also have

$$\zeta(\vec{u}, p) = \alpha (\nabla \cdot \vec{u}) + \frac{p}{M}, \quad (4.358)$$

and substituting in the nondimensionalized values leads to

$$\zeta_o \zeta^*(\vec{u}, p) = \alpha \frac{u_o}{x_o} (\nabla^* \cdot \vec{u}^*) + \frac{p_o}{\mu_o} \frac{p^*}{M^*}, \quad (4.359)$$

where $M^* = \frac{M}{\mu_o}$.

Grouping terms and recognizing that $p_o = \mu_o \frac{u_o}{x_o}$, we have

$$\zeta_o \zeta^*(\vec{u}, p) = \left(\alpha \frac{u_o}{x_o} \right) (\nabla^* \cdot \vec{u}^*) + \frac{u_o}{x_o} \frac{p^*}{M^*}. \quad (4.360)$$

All terms should be nondimensional, which implies

$$\zeta_o = \frac{u_o}{x_o}. \quad (4.361)$$

Returning to the fluid mass balance, we have

$$\left(\frac{u_o}{t_o x_o} \right) \frac{\partial \zeta^*(\vec{u}, p)}{\partial t^*} + \left(\frac{q_o}{x_o} \right) (\nabla^* \cdot \vec{q}^*(p)) - \gamma_o \gamma^*(\vec{x}, t) = 0 \text{ in } \Omega, \quad (4.362)$$

All terms should be nondimensional, which implies

$$q_o = \frac{u_o}{t_o}, \quad (4.363)$$

$$\gamma_o = \frac{1}{t_o} \frac{u_o}{x_o}. \quad (4.364)$$

Regrouping terms leads to

$$\left(\frac{u_o}{t_o x_o} \right) \frac{\partial \zeta^*(\vec{u}, p)}{\partial t^*} + (\nabla^* \cdot \vec{q}^*(p)) - \gamma^*(\vec{x}, t) = 0 \text{ in } \Omega. \quad (4.365)$$

Returning to the equation for Darcy flow, we have

$$\frac{u_o}{t_o} \vec{q}^*(p) = - \left(\frac{\sigma_o}{x_o} \frac{\mathbf{k}}{\mu_f} \right) (\nabla^* p^* - \vec{f}_f^*), \quad (4.366)$$

where we have made use of $q_o = \frac{u_o}{t_o}$, $\sigma_o = p_o$ and $f_o = \frac{\sigma_o}{x_p}$. Rearranging we have,

$$\vec{q}^*(p) = - \left(\frac{\sigma_o t_o}{u_o x_o} \frac{\mathbf{k}}{\mu_f} \right) (\nabla^* p^* - \vec{f}_f^*), \quad (4.367)$$

$$\vec{q}^*(p) = - \left(\frac{\mu_o t_o}{x_o^2} \frac{\mathbf{k}}{\mu_f} \right) (\nabla^* p^* - \vec{f}_f^*). \quad (4.368)$$

This implies

$$\mathbf{k}^* = \frac{\mathbf{k}}{x_o^2}, \quad (4.369)$$

$$\mu_f^* = \frac{\mu_f}{\mu_o t_o}. \quad (4.370)$$

We also find that the time scale is given by

$$t_o = \frac{\mu_{f_o} x_o^2}{\mathbf{k}_o \mu_o}. \quad (4.371)$$

Note

In PyLith v4 and earlier, we used a displacement scale equal to the length scale. Using separate length and displacement scales facilitates accurately solving for displacements many orders of magnitude smaller than the length scale. It also separates the stress scale used to nondimensionalize stress, tractions, and fluid pressure from the rigidity scale.

Table 4.9: Nondimensionalization of poroelasticity/

Quantity	Scale
x_o	Length scale
u_o	Displacement scale
μ_o	Rigidity scale
$t_o = \frac{\mu_{fo}}{k_o} \frac{x_o^2}{\mu_o}$	Time scale
$\sigma_o = p_o = \mu_o \frac{u_o}{x_o}$	Stress scale
$f_o = \frac{\sigma_o}{x_o}$	Body force scale
$\rho_o = \mu_o \frac{t_o^2}{x_o^2}$	Density scale
$q_o = \frac{u_o}{t_o}$	Flow scale
$\gamma_o = \frac{1}{t_o} \frac{u_o}{x_o}$	Source scale
$\mu_f^* = \frac{\mu_f}{\mu_o t_o}$	Viscosity scale
$k_o = \frac{k}{x_o^2}$	Permeability scale

Infinitesimal Strain and Prescribed Fault Slip

We apply the prescribed fault slip formulation that we used for elasticity to poroelasticity. We add a boundary condition to the poroelasticity equation prescribing the jump in the displacement field across the fault,

$$-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t) = \vec{0} \text{ on } \Gamma_f,$$

where $\vec{u}^+$ is the displacement vector on the “positive” side of the fault, $\vec{u}^-$ is the displacement vector on the “negative” side of the fault, $\vec{d}$ is the slip vector on the fault, and $\vec{n}$ is the fault normal which points from the negative side of the fault to the positive side of the fault. We enforce the jump in displacements across the fault using a Lagrange multiplier corresponding to equal and opposite tractions on the two sides of the fault.

Warning

In this formulation, the fault acts as a barrier to fluid flow. That is, there is no coupling in fluid pressure across the fault.

Table 4.10: Mathematical notation for poroelasticity with infinitesimal strain and prescribed fault slip.

Category	Sym- bol	Description
Unknowns	$\vec{u}$	Displacement field
	$\vec{v}$	Velocity field
	p	Pressure field (corresponds to pore fluid pressure)
	ϵ_v	Volumetric (trace) strain
	$\vec{\lambda}$	Lagrange multiplier for fault
Derived quantities	σ	Cauchy stress tensor
	ϵ	Cauchy strain tensor
	ζ	Variation of fluid content (variation of fluid vol. per unit vol. of PM), $\alpha\epsilon_v + \frac{p}{M}$
	ρ_b	Bulk density, $(1 - \phi)\rho_s + \phi\rho_f$
	$\vec{q}$	Darcy flux, $-\frac{k}{\mu_f} \cdot (\nabla p - \vec{f}_f)$
Common constitutive parameters	ρ_s	Solid (matrix) density
	ρ_f	Fluid density
	μ_f	Fluid viscosity
	ϕ	Porosity
	μ	Shear modulus
	K_d	Drained bulk modulus
	α	Biot coefficient, $1 - \frac{K_d}{K_s}$
	M	Biot modulus
	k	Permeability
	$\vec{f}$	Body force per unit volume, for example: $\rho_b \vec{g}$
Source terms	$\vec{f}_f$	Fluid body force, for example: $\rho_f \vec{g}$
	γ	Source density; rate of injected fluid per unit volume of the porous solid

Quasistatic

The strong form of the quasistatic boundary value problem may be expressed as

$$\vec{s}^T = \left(\vec{u} \quad p \quad \epsilon_v \quad \vec{\lambda} \right), \quad (4.372)$$

$$\vec{f}(t) + \nabla \cdot \sigma(\vec{u}, p) = \vec{0} \text{ in } \Omega, \quad (4.373)$$

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} - \gamma(\vec{x}, t) + \nabla \cdot \vec{q}(p) = 0 \text{ in } \Omega, \quad (4.374)$$

$$\nabla \cdot \vec{u} - \epsilon_v = 0 \text{ in } \Omega, \quad (4.375)$$

$$\sigma \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau, \quad (4.376)$$

$$\sigma \cdot \vec{n} = -\vec{\lambda}(\vec{x}, t) \text{ on } \Gamma_{f+}, \quad (4.377)$$

$$\sigma \cdot \vec{n} = +\vec{\lambda}(\vec{x}, t) \text{ on } \Gamma_{f-}, \quad (4.378)$$

$$\vec{q} \cdot \vec{n} = q_0(\vec{x}, t) \text{ on } \Gamma_q, \quad (4.379)$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u, \quad (4.380)$$

$$p = p_0(\vec{x}, t) \text{ on } \Gamma_p, \text{ and} \quad (4.381)$$

$$-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t) = \vec{0} \text{ on } \Gamma_f. \quad (4.382)$$

We place all terms for the elasticity, pressure, an volumetric strain equations on the left-hand-side, consistent with PETSc TS implicit time stepping. We create the weak form by taking the dot product with the trial functions $\vec{\psi}_{trial}^u$,

ψ_{trial}^p , $\psi_{trial}^{\epsilon_v}$, or $\vec{\psi}_{trial}^\lambda$ and integrating over the domain:

$$\int_{\Omega} \vec{\psi}_{trial}^u \cdot \left(\vec{f}(\vec{x}, t) + \nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) \right) d\Omega = 0, \quad (4.383)$$

$$\int_{\Omega} \psi_{trial}^p \left(\frac{\partial \zeta(\vec{u}, p)}{\partial t} - \gamma(\vec{x}, t) + \nabla \cdot \vec{q}(p) \right) d\Omega = 0, \quad (4.384)$$

$$\int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot (\nabla \cdot \vec{u} - \epsilon_v) d\Omega \quad (4.385)$$

$$\int_{\Gamma_f} \vec{\psi}_{trial}^\lambda \cdot \left(-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t) \right) d\Gamma = 0. \quad (4.386)$$

Applying the divergence theorem to the first two equations and incorporating the Neumann boundary conditions yields

$$\int_{\Omega} \psi_{trial}^u \cdot \vec{f}(\vec{x}, t) + \nabla \vec{\psi}_{trial}^u : -\boldsymbol{\sigma}(\vec{u}, p_f) d\Omega + \int_{\Gamma_\tau} \psi_{trial}^u \cdot \vec{\tau}(\vec{x}, t) d\Gamma + \int_{\Gamma_f} \vec{\psi}_{trial}^{u^+} \cdot \left(-\vec{\lambda}(\vec{x}, t) \right) + \vec{\psi}_{trial}^{u^-} \cdot \left(+\vec{\lambda}(\vec{x}, t) \right) d\Gamma = 0, \quad (4.387)$$

$$\int_{\Omega} \psi_{trial}^p \left(\frac{\partial \zeta(\vec{u}, p_f)}{\partial t} - \gamma(\vec{x}, t) \right) + \nabla \psi_{trial}^p \cdot (-\vec{q}(p_f)) d\Omega + \int_{\Gamma_q} \psi_{trial}^p q_0(\vec{x}, t) d\Gamma = 0, \quad (4.388)$$

$$\int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot (\nabla \cdot \vec{u} - \epsilon_v) d\Omega = 0, \text{ and} \quad (4.389)$$

$$\int_{\Gamma_f} \vec{\psi}_{trial}^\lambda \cdot \left(-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t) \right) d\Gamma = 0. \quad (4.390)$$

Residual Pointwise Functions

Identifying $F(t, s, \dot{s})$ and $G(t, s)$ we have

$$F^u(t, s, \dot{s}) = \int_{\Omega} \underbrace{\vec{\psi}_{trial}^u \cdot \vec{f}(\vec{x}, t)}_{\vec{f}_0^u} + \nabla \vec{\psi}_{trial}^u : \underbrace{-\boldsymbol{\sigma}(\vec{u}, p_f)}_{\vec{f}_1^u} d\Omega + \int_{\Gamma_\tau} \underbrace{\psi_{trial}^u \cdot \vec{\tau}(\vec{x}, t)}_{\vec{f}_0^u} d\Gamma + \int_{\Gamma_f} \underbrace{\vec{\psi}_{trial}^{u^+} \cdot (-\vec{\lambda}(\vec{x}, t))}_{\vec{f}_0^u} + \underbrace{\vec{\psi}_{trial}^{u^-} \cdot (+\vec{\lambda}(\vec{x}, t))}_{\vec{f}_0^u} d\Gamma \quad (4.391)$$

$$F^p(t, s, \dot{s}) = \int_{\Omega} \underbrace{\psi_{trial}^p \left(\frac{\partial \zeta(\vec{u}, p_f)}{\partial t} - \gamma(\vec{x}, t) \right)}_{\vec{f}_0^p} + \nabla \psi_{trial}^p \cdot \underbrace{-\vec{q}(p_f)}_{\vec{f}_1^p} d\Omega + \int_{\Gamma_q} \underbrace{\psi_{trial}^p q_0(\vec{x}, t)}_{\vec{f}_0^p} d\Gamma, \quad (4.392)$$

$$F^{\epsilon_v}(t, s, \dot{s}) = \int_{\Omega} \psi_{trial}^{\epsilon_v} \cdot \underbrace{(\nabla \cdot \vec{u} - \epsilon_v)}_{\vec{f}_0^{\epsilon_v}} d\Omega. \quad (4.393)$$

$$F^\lambda(t, s, \dot{s}) = \int_{\Gamma_f} \underbrace{\vec{\psi}_{trial}^\lambda \cdot (-\vec{u}^+ + \vec{u}^- + \vec{d}(\vec{x}, t))}_{\vec{f}_0^\lambda} d\Gamma, \quad (4.394)$$

$$G^u(t, s) = 0, \quad (4.395)$$

$$G^p(t, s) = 0, \quad (4.396)$$

$$G^{\epsilon_v} = 0, \quad (4.397)$$

$$G^\lambda = 0. \quad (4.398)$$

Jacobian Pointwise Functions

Four fields yields potentially 16 Jacobian pointwise functions for the LHS:

$$J_F^{uu} = \frac{\partial F^u}{\partial u} + s_{tshift} \frac{\partial F^u}{\partial \dot{u}} \quad (4.399)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial u} (-\sigma(\vec{u}, p, \epsilon_v)) d\Omega \quad (4.400)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial u} (-(\mathbf{C} : \boldsymbol{\varepsilon} - \alpha p \mathbf{I})) d\Omega \quad (4.401)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : -\mathbf{C} : \frac{1}{2} (\nabla + \nabla^T) \vec{\psi}_{basis}^u d\Omega \quad (4.402)$$

$$= \int_{\Omega} \psi_{trial,i,k}^u \underbrace{(-C_{ikjl})}_{J_{f3}^{uu}} \psi_{basis,j,l}^u d\Omega, \quad (4.403)$$

$$J_F^{up} = \frac{\partial F^u}{\partial p} + s_{tshift} \frac{\partial F^u}{\partial \dot{p}} \quad (4.404)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial p} (-(\mathbf{C} : \boldsymbol{\varepsilon} - \alpha p \mathbf{I})) d\Omega \quad (4.405)$$

$$= \int_{\Omega} \psi_{trial,i,j}^u \underbrace{(\alpha \delta_{ij})}_{J_{f2}^{up}} \psi_{basis}^p d\Omega, \quad (4.406)$$

$$J_F^{u\epsilon_v} = \frac{\partial F^u}{\partial \epsilon_v} + s_{tshift} \frac{\partial F^u}{\partial \dot{\epsilon}_v} \quad (4.407)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial \epsilon_v} (-\sigma(\vec{u}, p, \epsilon_v)) d\Omega \quad (4.408)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial \epsilon_v} (-(\mathbf{C} : \boldsymbol{\varepsilon} - \alpha p \mathbf{I})) d\Omega, \quad (4.409)$$

$$= \int_{\Omega} \nabla \vec{\psi}_{trial}^u : \frac{\partial}{\partial \epsilon_v} [-(2\mu\boldsymbol{\epsilon} + \lambda\mathbf{I}\epsilon_v - \alpha p \mathbf{I})] d\Omega \quad (4.410)$$

$$= \int_{\Omega} \psi_{trial,i,j}^u \underbrace{(-\lambda \delta_{ij})}_{J_{f2}^{u\epsilon_v}} \psi_{basis}^{\epsilon_v} d\Omega, \quad (4.411)$$

$$J_F^{u\lambda} = \frac{\partial F^u}{\partial \lambda} + s_{tshift} \frac{\partial F^u}{\partial \dot{\lambda}} \quad (4.412)$$

$$= \int_{\Gamma_f} \psi_{trial,i}^{u+} \underbrace{(-\delta_{ij})}_{J_{f0}^{u\lambda}} \psi_{basis,j}^{\lambda} + \psi_{trial,i}^{u-} \underbrace{(+\delta_{ij})}_{J_{f0}^{u\lambda}} \psi_{basis,j}^{\lambda} d\Gamma, \quad (4.413)$$

$$J_F^{pu} = \frac{\partial F^p}{\partial u} + s_{tshift} \frac{\partial F^p}{\partial \dot{u}} = 0, \quad (4.414)$$

$$J_F^{pp} = \frac{\partial F^p}{\partial p} + s_{tshift} \frac{\partial F^p}{\partial \dot{p}} \quad (4.415)$$

$$= \int_{\Omega} \nabla \psi_{trial}^p \cdot \frac{\partial}{\partial p} - \left[-\frac{\mathbf{k}}{\mu_f} (\nabla p - \vec{f}) \right] d\Omega + s_{tshift} \int_{\Omega} \psi_{trial}^p \frac{\partial}{\partial \dot{p}} \left[\alpha \dot{\epsilon}_v + \frac{\dot{p}}{M} - \gamma(\vec{x}, t) \right] d\Omega \quad (4.416)$$

$$= \int_{\Omega} \nabla \psi_{trial}^p \left(\frac{\mathbf{k}}{\mu_f} \nabla \cdot \psi_{basis}^p \right) d\Omega + \int_{\Omega} \psi_{trial}^p \left(s_{tshift} \frac{1}{M} \right) \psi_{basis}^p d\Omega \quad (4.417)$$

$$= \int_{\Omega} \psi_{trial,k}^p \underbrace{\left(\frac{\mathbf{k}}{\mu_f} \delta_{kl} \right)}_{J_{f3}^{pp}} \psi_{basis,l}^p d\Omega + \int_{\Omega} \psi_{trial}^p \underbrace{\left(s_{tshift} \frac{1}{M} \right)}_{J_{f0}^{pp}} \psi_{basis}^p d\Omega, \quad (4.418)$$

$$J_F^{p\epsilon_v} = \frac{\partial F^p}{\partial \epsilon_v} + s_{tshift} \frac{\partial F^p}{\partial \dot{\epsilon}_v} \quad (4.419)$$

$$= \int_{\Omega} \psi_{trial}^p \underbrace{(s_{tshift} \alpha)}_{J_{f0}^{p\epsilon_v}} \psi_{basis}^{\epsilon_v} d\Omega,$$

$$J_F^{pl} = \frac{\partial F^p}{\partial \lambda} + s_{tshift} \frac{\partial F^p}{\partial \dot{\lambda}} = 0, \quad (4.421)$$

Bulk Rheologies

In this section we describe the mathematic formulations of the bulk rheologies for poroelasticity. The bulk rheologies include

- Isotropic linear elasticity,

Linear Isotropic

We assume linear elasticity for the solid phase, so we have the following formulation for the stress tensor:

$$\boldsymbol{\sigma}(\vec{u}, p) = \mathbf{C} : \boldsymbol{\epsilon} - \alpha p \mathbf{I} = \lambda \mathbf{I} \epsilon_v + 2\mu \boldsymbol{\epsilon} - \alpha \mathbf{I} p \quad (4.435)$$

where λ and μ are Lamé's parameters, $\lambda = K_d - \frac{2\mu}{3}$, μ is the shear modulus, and the volumetric strain is defined as $\epsilon_v = \nabla \cdot \vec{u}$. For isotropic linear elasticity, all components of C_{ijkl} are zero except for:

$$\begin{aligned} C_{1111} &= C_{2222} = C_{3333} = \lambda + 2\mu, \\ C_{1122} &= C_{1133} = C_{2233} = \lambda, \\ C_{1212} &= C_{2323} = C_{1313} = \mu. \end{aligned}$$

The deviatoric elastic constants are:

$$\begin{aligned} C_{1111}^{dev} &= C_{2222}^{dev} = C_{3333}^{dev} = \frac{4}{3}\mu, \\ C_{1122}^{dev} &= C_{1133}^{dev} = C_{2233}^{dev} = -\frac{2}{3}\mu, \\ C_{1212}^{dev} &= C_{2323}^{dev} = C_{1313}^{dev} = \mu. \end{aligned}$$

4.2.4 Running PyLith

Overview of Running PyLith

Command Line Interface

PyLith is a Python script run from the command line. The most common way to run PyLith is to pass one or more parameter files as arguments on the command line. For example, if the file `strikeslip.cfg` contains the parameters for your simulation, then run

```
$ pylith strikeslip.cfg
```

See [Setting PyLith Parameters](#) for more information about setting parameters via parameter files and the command line.

Tip

When running PyLith in parallel with a large mesh, use `pylith_convertmesh` to convert your finite-element mesh to an HDF5 file in the PETSc mesh format to allow it to be read in parallel.

Running in Parallel on a Desktop or Laptop

You can run PyLith in parallel on a desktop or laptop using the `--nodes=NPROCS` command line argument, where NPROCS is the number of processes to use. The amount of speedup you will see by running in parallel depends on the processor and memory architecture of your computer. On most desktops and laptops, you can obtain reduced runtimes using about one-quarter to one-half of the number of physical cores. Once you saturate the memory bus, using additional processes (cores) will result in little, if any, speedup.

Running on a Offline Desktop or Laptop

The MPI implementation (MPICH) used in the PyLith binary by default uses the network interface. If your desktop or laptop is offline, then you need to tell MPI to use sockets rather than the network interface. You can do this by setting the `IF_PROVIDER` environment variable. For example, in a terminal running `zsh` or `tcsh`, you can use `export IF_PROVIDER=sockets`.

Running in Parallel on a Cluster

Most clusters use a job scheduler to manage user requests for running batch jobs. The command to submit a batch job depends upon the particular job scheduler being used. Further, the command used in a batch script to launch an MPI program varies from one cluster to the next. This command can vary between two clusters, even if the clusters use the same job scheduling system! On some systems, `mpirun` is invoked directly from the batch script. On others, a special wrapper is used instead.

Launchers and Schedulers

Most job schedulers require you to write a shell script to launch a job. Pyre's `launcher` and `scheduler` facilities can generate the shell script and submit the job for you. Many properties associated with `launcher` and `scheduler` depend on the cluster you are using, and are best set in a configuration file. You can put these settings in your personal PyLith configuration file (`$HOME/.pyre/pylithapp/pylithapp.cfg`).

Pyre's `scheduler` facility is used to specify the type of job scheduler system you are using (if any):

```
[pylithapp]
# The valid values for scheduler are `lsf`, `pbs`, `globus`, and `none`.
scheduler = lsf
# Pyre's launcher facility is used to specify the MPI implementation.
# The valid values for launcher include 'mpich' and 'lam-mpi'.
launcher = mpich
```

You may find the `'dry'` option useful while debugging the `launcher` and `scheduler` configuration. This option causes PyLith to perform a “dry run,” dumping the batch script or `mpirun` command to the console, instead of actually submitting it for execution (the output is only meaningful if you're using a batch system).

Listing 4.2: Display the bash script that would be submitted.

```
$ pylith --scheduler.dry
# Display the mpirun command.
$ pylith --launcher.dry
```

Using a Job Scheduler

Many clusters use some implementation of a PBS (e.g., TORQUE/Maui) or LSF batch system. The examples below illustrate use of some of the more important settings. You may need to make use of more options or adjust these to submit jobs on various cluster. These settings are usually placed in `$HOME/.pyre/pylithapp/pylithapp.cfg` or in a system-wide configuration file. They can be overridden on the command line, where one typically specifies the number of compute nodes and number of processes per compute node, the job name, and the allotted time for the job:

```
$ pylith example1.cfg \
--job.queue=debug \
--job.name=example1 \
--job.stdout=example1.log \
--job.stderr=example1.err \
```

(continues on next page)

(continued from previous page)

```
--job.walltime=5*minute \
--nodes=4
```

Important

The value for nodes is equal to the number of compute nodes times the number of processes (usually the number of cores) requested per compute node. Specifying the number of processes per compute node depends on the batch system. For more information on configuring Pyre for your batch system, see CIG's Pythia page <https://geodynamics.org/resources/pythiaXX/>.

LSF Batch System

```
[pylithapp]
scheduler = lsf
# the type of batch system

[pylithapp.lsf]
bsub-options = [-a mpich_gm]
# special options for 'bsub'

[pylithapp.launcher]
command = mpirun.lsf
'mpirun' command to use on our cluster

[pylithapp.job]
queue = normal
# default queue for jobs
```

PBS Batch System

```
[pylithapp]
scheduler = pbs
# the type of batch system

[pylithapp.pbs]
shell = /bin/bash
# submit the job using a bash shell script

# Export all environment variables to the batch job
# Send email to johndoe@mydomain.org when the job begins, ends, or aborts
qsub-options = -V -m bea -M johndoe@mydomain.org

[pylithapp.launcher]
command = mpirun -np ${nodes} -machinefile ${PBS_NODEFILE}
```

For most PBS batch systems you can specify N processes per compute node via the command line argument `--scheduler.ppn=N`.

Running without a Job Scheduler

On a cluster without a job scheduler, you need to explicitly specify the compute nodes on which the job will run. Supposing the compute nodes on your cluster are named n001, n002, ..., etc., and you want to run the job on machines n001, n003, n004, and n005 (maybe n002 is down for the moment). To run an example, create a file named `mynodes.cfg` which specifies the compute nodes to use:

```
[pylithapp.launcher]
nodegen = n%03d
nodelist = [1,3-5]
```

The `nodegen` property is a printf-style format string, used in conjunction with `nodelist` to generate the list of compute node names. The `nodelist` property is a comma-separated list of values in square brackets.

Now, invoke the following:

```
pylith example.cfg mynodes.cfg
```

This strategy gives you the flexibility to create an assortment of `cfg` files (with one `cfg` file for each list of compute nodes) which can be easily paired with different parameter files.

If your list of compute nodes does not change often, you may find it more convenient to specify default values for `nodegen` and `nodelist` in `$HOME/.pyre/pylithapp/pylithapp.cfg` (which is read automatically). Then, you can run any simulation with no additional arguments:

```
pylith example.cfg
```

Warning

This assumes your list of compute nodes has enough computational resources for the simulation in question.

You will notice that a machine file `mpirun.nodes` is generated. It will contain a list of the nodes where PyLith has run.

Defining the Simulation

The parameters for PyLith are specified as a hierarchy or tree of components. The application assembles the hierarchy of components from user input and then calls the `main` function in the top-level module in the same manner as a C or C++ program. The behavior of the application is determined by the components included in the hierarchy as specified by the user. The Pyre framework provides the interface for defining this hierarchy. [PyLith Components](#) includes detailed descriptions of the components provided by PyLith. Pyre properties correspond to simple settings in the form of strings, integers, and real numbers. Pyre facilities correspond to software modules. Facilities may have their own facilities (branches in the tree) and any number of properties. See [Fig. 4.4](#) for the general concept of Pyre facilities and properties.

Organization of Simulation Components

The components in a PyLith simulation generally fall into four main categories:

Topology

Components associated with the spatial discretization of the domain, such as the finite-element mesh;

Physics

Components specifying the physics to be solved, such as materials associated with a governing equation, bulk rheologies, boundary conditions, and fault interface conditions;

Physics Implementation

Components that perform the finite-element operations, such as integration of the residual and system Jacobian; and

Observers

Components that get notified of updates to the solution and state variables, such as writers for saving the solution to a file.

The physics components provide the point-wise functions (kernels) used by the physics implementation components, the auxiliary field, and the layout of the derived field (subfields computed from the auxiliary field and solution, such as stress and strain).

Simulation Input and Output

Fig. 4.7 shows the inputs and outputs for a PyLith simulation. The user supplies:

1. Mesh information. This includes the topology of the finite-element mesh (coordinates of vertices and how the vertices are connected into cells), a material identifier for each cell, and sets of vertices associated with boundary conditions, faults, and output (for subsets of the mesh). This information can be provided using the PyLith mesh ASCII format (see [PyLith Mesh ASCII File](#) for the format specification), by importing the information from Cubit or Gmsh mesh generation software, or an HDF5 file in the PETSc mesh format. Refer to [Finite-Element Mesh](#) for more information.
2. A set of parameters describing the problem. These parameters describe the type of problem to be run, solver information, time-stepping information, boundary conditions, materials, etc. This information can be provided from the command-line or by using a `cfg` file.
3. Spatial databases specifying the values for the material properties and boundary conditions. Arbitrarily complex spatial variations in boundary and fault conditions and material properties may be given in the spatial database (see [Examples](#) and [Spatialdata Documentation](#)).

PyLith writes solution information, such as solution fields and state variables, to either VTK/VTU files or HDF5/Xdmf files using the observer components. ParaView and Visit as well as several other visualization tools can read both types of files. PyLith includes `pylith_viz` for visualizing PyLith HDF files using PyVista. Post-processing of output is generally performed using HDF5 files accessed via a Python script and the `h5py` package or a Matlab script.



Tip

When running PyLith in parallel, convert your finite-element mesh to an HDF5 file in the PETSc mesh format to allow it to be read in parallel. Refer to section [pylith_convertmesh](#) for more information.

Nondimensionalization

PyLith scales all parameters provided by the user so that the simulation solves the equations using nondimensional quantities. This permits application of PyLith to problems across a vast range of spatial and temporal scales. The scales used to nondimensionalize the problem are length, pressure, density, and time. SpatialData provides two scales objects to make it easy to provide reasonable scales for quasi-static and dynamic elasticity boundary value problems. Refer to [Nondimensionalization](#) for details.

Finite-Element Implementation User Interface

In specifying simulation parameters, some details of the finite-element implementation using the PETSc DMPlex is exposed to the user. In this section we describe the data structures to give the user greater context for understanding what the parameters mean.

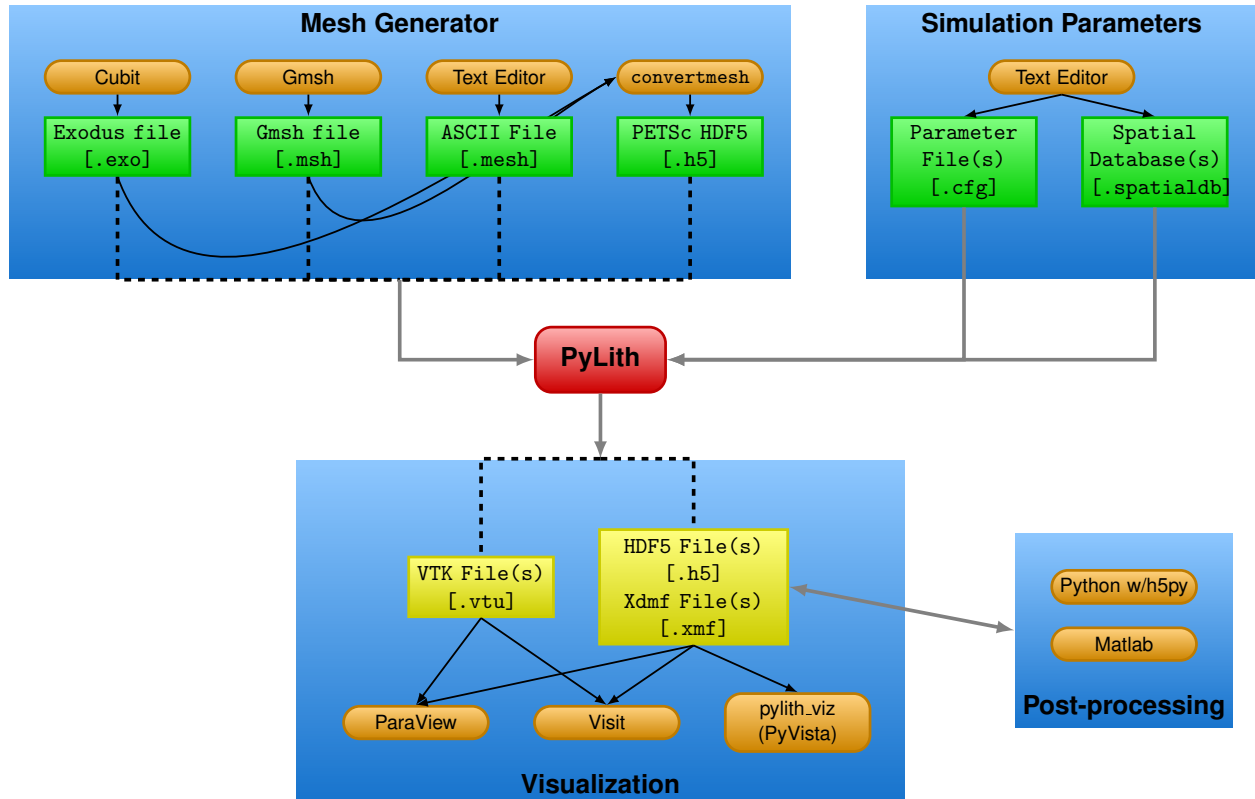


Fig. 4.7: PyLith requires a finite-element mesh (three different mechanisms for generating a mesh are currently supported), simulation parameters, and spatial databases (defining the spatial variation of various parameters). PyLith writes the solution output to either VTK/VTU or HDF5/Xdmf files, which can be visualized with ParaView or Visit. PyLith includes *pylith_viz* for visualizing PyLith HDF files using PyVista. Post-processing is generally done using the HDF5 files with Python or Matlab scripts.

 Tip

See *Code Layout* for a detailed discussion of organization of the PyLith code.

Fields and Subfields

Finite-element coefficients for the finite-element basis functions (sometimes thought of as the values at vertices, on edges and faces, or in cells) are stored in a `Field`. A `Field` is composed of a `PetscSection`, which associates the points (vertices, edges, faces, and cells) with the finite-element coefficients, and a `PetscVec`, which is a vector storing the finite-element coefficients. A `Field` may hold a single subfield, such as displacement, or it may hold several subfields, such as the density, shear modulus, and bulk modulus for an isotropic, linear elastic material.

Spatial discretization is specified for each subfield. That is, each subfield within a `Field` can have a different discretization. For example, a displacement field may use a second order discretization while a pressure field may use a first order discretization. If we have uniform material properties, we use a zero order discretization (uniform values within a cell) to reduce the storage requirements.

The two main types of fields are the solution field and auxiliary fields.

Solution Field

The solution field contains all of the finite-element coefficients corresponding to the problem solution. As discussed in the multiphysics finite-element formulation in *Finite-Element Formulation with PETSc*, if the governing equations have multiple unknowns, such as displacement and fluid pressure for poroelasticity, then the solution field will have multiple subfields. See *Solution component* for details of the user interface and predefined containers for common subfield collections.

Auxiliary Field

We specify parameters for materials, boundary conditions, and fault interfaces using fields we refer to as the `auxiliary` field. Each scalar, vector, tensor, or other parameter is held in a separate subfield. We also store state variables in the auxiliary field, with each state variable as a different subfield. This provides a single container for the collection of spatially varying parameters while maintaining the flexibility to specify the discretization of each parameter separately.

Discretization

The discretization of a field is given in terms of the topology (vertices, edges, faces, and cells) associated with the field and the basis order and quadrature order. The basis order refers to the highest order in the basis functions. For example, a basis order of 0 has just a constant term whereas a basis order of 2 for a polynomial basis has constant, linear, and quadratic terms.

 Warning

Currently, the quadrature order **MUST** be the same for all subfields in a simulation. This restriction may be relaxed in the future. PyLith verifies that the quadrature order is the same for all subfields, and it will indicate if a subfield has a quadrature order that does not match the quadrature order of the first solution subfield.

Setting PyLith Parameters

There are several methods for setting input parameters for the `pylith` executable: via the command line or by using a text file in `cfg` or `pml` format. Both facilities and properties may have default values, so you only need to set values when you want to deviate from the default behavior.

Units

All dimensional parameters require units. The units are specified using Python syntax, so square meters is `m**2`. Whitespace is not allowed in the string, for units and dimensioned quantities are multiplied by the units string; for example, two meters per second is `2.0*m/s`. Available units are shown in [Table 4.11](#).

Table 4.11: Pyre supported units. Aliases are in parentheses.

Scale	Available Units
length	meter (m), micrometer (um, micron), millimeter (mm), centimeter (cm), kilometer (km), inch, foot, yard, mile
time	second (s), nanosecond (ns), microsecond (us), millisecond (ms), minute, hour, day, year
mass	kilogram (kg), gram (g), centigram (cg), milligram (mg), ounce, pound, ton
pressure	pascal (Pa), kPa, MPa, GPa, bar, millibar, atmosphere (atm)

Using the Command Line

The `--help` command line argument displays links to useful resources for learning PyLith.

Pyre uses the following syntax to change properties from the command line. To change the value of a property of a component, use `--COMPONENT.PROPERTY=VALUE`. Each component is attached to a facility, so the option above can also be written as `--FACILITY.PROPERTY=VALUE`. Each facility has a default component attached to it. A different component can be attached to a facility by `--FACILITY=NEW_COMPONENT`.

PyLith's command-line arguments can control Pyre and PyLith properties and facilities, MPI settings, and PETSc settings. You can get a list of all of these top-level properties along with a description using the `--help-properties` command-line argument. To get information on user-configurable facilities and components, use the `--help-components` command-line argument. To find out about the properties associated with a given component, use the `--COMPONENT.help-properties` command-line argument.

```
$ pylith --problem.help-properties
# Show problem components.
$ pylith --problem.help-components
# Show bc components (bc is a component of problem).
$ pylith --problem.bc.help-components
# Show bc properties.
$ pylith --problem.bc.help-properties
```

Using a .cfg File

Entering more than a few parameters via the command line is cumbersome. You will generally find it easier to collect parameters into a `cfg` file. The file is composed of one or more sections which are formatted as follows:

```
[pylithapp.COMPONENT1.COMPONENT2]
# This is a comment.

FACILITY3 = COMPONENT3
PROPERTY1 = VALUE1
PROPERTY2 = VALUE2
```

 Tip

We strongly recommend that you use `cfg` files for your work. The files are syntax-highlighted in most editors, such as vim, Emacs, Atom, and VS Code.

Using a .pml File

A `pml` file is an XML file that specifies parameter values in a highly structured format. It is composed of nested sections which are formatted as follows:

```
<component name="COMPONENT1">
  <component name="COMPONENT2">
    <property name="PROPERTY1">VALUE1</property>
    <property name="PROPERTY2">VALUE2</property>
  </component>
</component>
```

XML files are intended to be read and written by software, not edited manually by humans. The `pml` file format is intended for applications in which PyLith input files are generated by another program, e.g., a GUI, web application, or a high-level structured editor. This file format will not be discussed further here, but if you are interested in using `pml` files, note that `pml` files and `cfg` files can be used interchangeably; in the following discussion, a file with a `pml` extension can be substituted anywhere a `cfg` file can be used.

Specification and Placement of Configuration Files

Configuration files may be specified on the command line:

```
$ pylith example.cfg
```

In addition, the Pyre framework searches for configuration files named `pylithapp.cfg` in several predefined locations. You may put settings in any or all of these locations, depending on the scope you want the settings to have:

1. `$PREFIX/etc/pylithapp.cfg` for system-wide settings;
2. `$HOME/.pyre/pylithapp/pylithapp.cfg` for user settings and preferences; and
3. `./pylithapp.cfg`, for local settings.

All of the example problems are set up using configuration files and specific problems are defined by including the appropriate configuration file on the command-line.

 Important

The Pyre framework will search these directories for `cfg` files matching the names of components (for example, `timedependent.cfg`, `faultcohesivekin.cfg`, `greensfns.cfg`, etc) and will attempt to assign all parameters in those files to the respective component.

 Important

Parameters given directly on the command line will override any input contained in a configuration file. Configuration files given on the command line override all others. The `pylithapp.cfg` files placed in (3) will override those in (2), (2) overrides (1), and (1) overrides only the built-in defaults.

 Tip

See *Utilities* and *PyLith Parameter Viewer* for several helpful utilities for viewing PyLith parameters and finding examples using specific features.

PyLith Application

The top-level object is the PyLith application with four facilities:

metadata

Simulation metadata;

petsc

PETSc settings.

problem

Problem to run, including materials, boundary conditions, etc.; and

dump_parameters

Dump parameters used and version information to file.

The problem facility defines of the boundary value problem to solve and contains the most information. It defines the finite-element discretization, materials, faults, initial conditions, boundary conditions, and output.

 See also

PyLithApp Component

Simulation Metadata

New in v3.0.0.

We use metadata to provide a concise summary of a simulation. The metadata gives structure to information previously placed in comments within the parameter files while also making this information machine readable. The metadata makes it possible for a Python script to launch an entire suite of simulations and search for simulation parameter files based on the metadata content (see *Utilities* for more information). For example, users can search examples that use a given feature.

At a minimum the metadata must include: (1) a description, (2) the command line arguments necessary to run the simulation, and (3) the PyLith version(s) that are compatible with the input files. We strongly encourage users to include all of the metadata in their own PyLith parameter files.

 See also

SimulationMetadata Component

PETSc Options

PyLith relies on PETSc for finite-element data structures, linear and nonlinear solvers, and time-stepping algorithms. PETSc has its own object-oriented interface for specifying runtime options. PyLith provides two mechanisms for passing options to PETSc:

1. Default values that are controlled by a few flags in the PyLith Problem `petsc_defaults` facility, and
2. PyLith `petsc` facility which passes options directly to PETSc.

When using the default values, PyLith selects solver and preconditioner options based on the governing equations. In most cases these default values will give good performance and you do not need to specify any PETSc options. User-specified values always take precedence over the default values.

Default PETSc Options

New in v3.0.0.

We separate the defaults into a few categories to make it easy to select desired options.

monitors

Options for basic monitoring of the solver;

initial_guess

Options for improving initial guesses at each time step;

adaptive_time_stepping

Options for adaptive time stepping

collective_io

Options for collective input/output; and

testing

Options used in testing.

solver

Options for the preconditioner and solver;

parallel

Options used when running in parallel (can be used in serial as well);

See also

See [PetscDefaults Component](#) for more information about the Pyre interface for specifying default PETSc options.

Monitoring Options

The monitoring options are enabled by default and provide a few lines of output per time step summarizing the operation of the linear and nonlinear solvers and time stepping. Additional monitoring can be turned on using the user-specified options.

Table 4.12: Description of PETSc monitoring options

Option	Description
ts_monitor	Show time-stepping progress.
ksp_monitor	Show preconditioned residual norm.
ksp_error_if_not_converged	Generate an error if linear solver does not converge.
ksp_converged_reason	Indicate why iterating stopped in linear solve.
snes_monitor	Show residual norm for each nonlinear solve iteration.
snes_error_if_not_converged	Generate an error if nonlinear solver does not converge.
snes_converged_reason	Indicate why iterating stopped in nonlinear solve.
snes_linesearch_monitor	Show line search information in nonlinear solve.

Listing 4.3: Default PETSc options for monitoring the solver. These settings are also in `share/settings/petsc_monitoring.cfg`.

```
[pylithapp.petsc]
# Turn on TS, KSP, and SNES monitors
ts_monitor =
ksp_monitor =
snes_monitor =

ksp_converged_reason = True
snes_converged_reason = True

# Trigger error if linear or nonlinear solvers fail to converge
ts_error_if_step_fails = True
ksp_error_if_not_converged = True
snes_error_if_not_converged = True
```

Initial Guess Options

Improve initial guesses of the solution when time stepping.

Listing 4.4: Default PETSc options for initial guesses of the solution. These settings are also in `share/settings/petsc_initial_guess.cfg`.

```
[pylithapp.petsc]
ksp_guess_type = pod
ksp_guess_pod_size = 8
```

Adaptive Time-Stepping Options

New in v5.0.0.

Adaptive time stepping (not enabled by default) adjusts the time step based upon the rate of deformation. We use the [basic PETSc adaptive time-stepping algorithm](#). Tolerances (`ts_atol` and `ts_rtol`) of around 0.05 work reasonably well in many simulations with viscoelasticity or poroelasticity. Increasing the tolerances leads to relatively larger time steps and smaller tolerances lead to small time steps. The PETSc documentation provides more details about how the time steps are selected.

Listing 4.5: Default PETSc options for adaptive time stepping. These settings are also in `share/settings/petsc_tsadaptive.cfg`.

```
[pylithapp.petsc]
ts_adapt_type = basic
ts_adapt_safety = 0.2
ts_adapt_reject_safety = 0.1
ts_atol = 0.05
ts_rtol = 0.05

ts_adapt_monitor = true
```

Collective I/O Options

The collective input and output options are enabled by default and turn on HDF5 collective output. We use parallel HDF5 implementation, which in turn relies on MPI IO. Many MPI IO implementations require collective input and output to be enabled for parallel HDF5 output even if only one process is being used.

Listing 4.6: Default PETSc options for collective output. These settings are also in `share/settings/petsc_collectiveio.cfg`.

```
[pylithapp.petsc]
viewer_hdf5_collective = True
```

Testing Options

The options in the testing category are intended for use in internal testing. These options help identify memory leaks in PETSc data structures and inconsistent back traces.

Listing 4.7: Default PETSc options for testing. These settings are also in `share/settings/petsc_testing.cfg`.

```
[pylithapp.petsc]
malloc_dump = True
```

Solver Options

The solver options are enabled by default. PyLith selects options based on the governing equation, formulation, presence of a fault, and whether the simulation is running in parallel. In some cases the solver settings for running in parallel are different than those for running in serial; in such cases, the settings for running in parallel often give comparable or better performance. If you have a moderate or large simulation, you should enable the parallel settings. Additionally, PyLith specifies general options related to the solver tolerances and triggering errors if the linear or nonlinear solver fails to converge. The different sets of defaults are detailed in the following code blocks.

Warning

When running in parallel in cases in which a fault face is split across processes, the current solver settings will result in a diverged solution. We attempt to prevent this from happening by specifying a penalty for splitting across the fault; however, sometimes the partitioner will still split a fault face across processes. In cases in which the default settings fail and the solver diverges, you can fall back to the previous settings by using the field split preconditioner in `share/settings/solver_elasticity_fault_fieldsplit.cfg`. Simply add this `.cfg` file to your command line options.

This issue will go away once we implement parallel mesh loading.

Note

If you do use the field split fall back, you need to be aware that it also has deficiencies. The split fields and algebraic multigrid preconditioning currently fails in problems with a nonzero null space. This most often occurs when a problem contains multiple faults that extend through the entire domain and create subdomains without any Dirichlet boundary conditions. The workaround is to use the `ilu` preconditioner. However, it only works in serial. An alternative is to use the `asm` preconditioner (Additive Schwarz) which works in parallel and serial.

Solver tolerances

PyLith will set default solver tolerances whenever the solver defaults are enabled.

Table 4.13: Summary of PETSc solver tolerances.

Option	Description
<code>ksp_rtol</code>	Stop iterating when the preconditioned KSP residual norm has this amount relative to its starting value.
<code>ksp_atol</code>	Stop iterating when the preconditioned KSP residual normal is smaller than this value.
<code>snes_rtol</code>	Stop iterating when the SNES residual norm has this amount relative to its starting value.
<code>snes_atol</code>	Stop iterating when the SNES residual normal is smaller than this value.

Listing 4.8: PETSc solver tolerances used whenever the solver defaults are enabled. These settings are also in `share/settings/petsc_solver_tolerances.cfg`.

```
[pylithapp.petsc]
ksp_rtol = 1.0e-14
ksp_atol = 1.0e-7

snes_rtol = 1.0e-14
snes_atol = 5.0e-7
```

Quasistatic Elasticity

Listing 4.9: PETSc options used for quasistatic elasticity without a fault. These settings are also in `share/settings/solver_elasticity.cfg`.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = gamg
mg_fine_ksp_max_it = 5
```

The Lagrange multiplier corresponding to the tractions on the fault introduces a saddle point in the system of equations. We could use a Schur complement approach, but we have found that grouping the degrees of freedom on each side of the fault into blocks and using a variable point-block Jacobi preconditioner provides better results; the number of iterations remains nearly constant with increased problem size and the overall solution time is low.

Listing 4.10: PETSc options used for quasistatic elasticity with a fault. These settings are also in `share/settings/solver_elasticity_fault.cfg`.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = gamg
ksp_gmres_restart = 100

dm_reorder_section = True
dm_reorder_section_type = cohesive
mg_fine_pc_type = vpbjacobi
mg_fine_ksp_max_it = 5
```

Quasistatic Incompressible Elasticity

The pressure field introduces a saddle point in the system of equations, so we use a Schur complement approach.

Listing 4.11: PETSc options used for quasistatic incompressible elasticity in serial.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = fieldsplit
pc_fieldsplit_type = schur

pc_fieldsplit_schur_factorization_type = lower
pc_fieldsplit_schur_precondition = full

fieldsplit_displacement_pc_type = lu
fieldsplit_pressure_pc_type = lu
```

Listing 4.12: PETSc options used for quasistatic incompressible elasticity in parallel. These settings are also in `share/settings/solver_incompressible_elasticity.cfg`.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = fieldsplit
pc_fieldsplit_type = schur

pc_fieldsplit_schur_factorization_type = lower
pc_fieldsplit_schur_precondition = full

fieldsplit_displacement_pc_type = ml
fieldsplit_pressure_pc_type = bjacobi
```

Quasistatic Poroelasticity

Listing 4.13: PETSc options used for quasistatic poroelasticity. These settings are also in `share/settings/solver_poroelasticity.cfg`.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = fieldsplit
pc_fieldsplit_type = multiplicative
pc_fieldsplit_0_fields = 2
pc_fieldsplit_1_fields = 1
pc_fieldsplit_2_fields = 0

fieldsplit_trace_strain_pc_type = bjacobi
fieldsplit_pressure_pc_type = bjacobi

fieldsplit_displacement_pc_type = ml
fieldsplit_displacement_ksp_type = gmres
```

Listing 4.14: PETSc options used for quasistatic poroelasticity with a porosity state variable. These settings are also in `share/settings/solver_poroelasticity_statevars.cfg`.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = fieldsplit
pc_fieldsplit_type = multiplicative
pc_fieldsplit_0_fields = 2
pc_fieldsplit_1_fields = 1
pc_fieldsplit_2_fields = 0
pc_fieldsplit_3_fields = 3
pc_fieldsplit_4_fields = 4
pc_fieldsplit_5_fields = 5

fieldsplit_trace_strain_pc_type = bjacobi
fieldsplit_pressure_pc_type = bjacobi

fieldsplit_displacement_pc_type = ml
fieldsplit_displacement_ksp_type = gmres

fieldsplit_velocity_pc_type = bjacobi
fieldsplit_pressure_t_pc_type = bjacobi
fieldsplit_trace_strain_t_pc_type = bjacobi
```

Listing 4.15: PETSc options used for quasistatic poroelasticity with a fault. These settings are also in `share/settings/solver_poroelasticity_fault.cfg`.

```
[pylithapp.petsc]
ts_type = beuler
pc_type = fieldsplit
pc_fieldsplit_type = multiplicative
pc_fieldsplit_0_fields = 2
pc_fieldsplit_1_fields = 1
```

(continues on next page)

(continued from previous page)

```

pc_fieldsplit_2_fields = 0
pc_fieldsplit_3_fields = 3

fieldsplit_trace_strain_pc_type = bjacobi
fieldsplit_pressure_pc_type = bjacobi

fieldsplit_displacement_pc_type = ml
fieldsplit_displacement_ksp_type = gmres
fieldsplit_displacement_ksp_gmres_restart = 100
fieldsplit_displacement_mg_fine_pc_type = vpbjacobi
fieldsplit_displacement_mg_fine_ksp_max_it = 5

fieldsplit_lagrange_multiplier_fault_pc_type = jacobi

dm_reorder_section = True
dm_reorder_section_type = cohesive

```

User-Specified PETSc Options

PETSc provides a few options for understanding solver settings and performance. The options `ksp_view` and `snes_view` control the display of the details of the linear and nonlinear solvers, respectively. The `log_view` option controls output of logging information at the end of a simulation.

Table 4.14: Description of PETSc logging and solving viewing options

Option	Description
<code>log_view</code>	Show logging objects and events.
<code>ksp_view</code>	Show linear solver parameters.
<code>snes_view</code>	Show nonlinear solver parameters.

Listing 4.16: Examples of using the logging and viewing options.

```

[pylithapp.petsc]

# Show detailed information about the linear solver.
ksp_view = True

# Show detailed information about the nonlinear solver.
snes_view = True

# Write logging information to stdout.
log_view = True

# Write logging information to an ASCII file `pylith_log.txt`.
log_view = ascii:pylith_log.txt

```

Customizing Solver Options

For most problems we use the GMRES method from Saad and Schultz [1986] for the linear solver; this is the linear solver PETSc uses as the default. See [PETSc linear solver table](#) for a list of PETSc options for linear solvers and preconditioners.

 Tip

It is important to keep in mind the resolution of the model and observations when setting solver tolerances. For example, matching observations with an accuracy of 1.0 mm does not require solving the equations to an accuracy of 0.0001 mm.

Utilities

The PyLith distribution contains several utilities for working with PyLith simulations and processing output. These Python scripts are all installed into the same `bin` directory as the `pylith` application with the exception of the `pyre_doc.py` script which is installed as part of Pythia/Pyre.

pylith_viz

Visualize output from PyLith

pyre_doc

Display the Pyre properties and facilities available for a given component.

pylith_convertmesh

Convert a mesh from one format to another along with other operations, such as reordering the mesh.

pylith_cfgsearch

Search and display metadata in `.cfg` files.

pylith_runner

Run all PyLith simulations in a given path.

pylith_dumpparameters

Dump simulation parameters, including default values, to a file for viewing.

pylith_eqinfo

Compute earthquake rupture metrics from PyLith output.

pylith_genxdmf

Generate Xdmf files from HDF5 files written by PyLith.

pylith_powerlaw_gendb

Generate a spatial database with power-law bulk rheology parameters for PyLith.

pylith_viz

New in v4.1.0.

This utility provides an interactive graphical user interface for plotting output from PyLith simulations using [PyVista](#). We demonstrate use of this utility in the examples.

 Important

`pylith_viz` works with **HDF5 files** that use the same layout as those written by PyLith.

```
pylith_viz --filenames FILENAMES [--help] subcommand subcommand_options
```

-help

Display help information for script.

-filenames FILENAMES

Comma separated list of relative paths to HDF5 files from PyLith.

subcommand

Subcommand to use for plotting data. Available subcommands are `plot_mesh`, `plot_field`, and `warp_grid`.

subcommand_options

Options specific to subcommand.

plot_mesh Subcommand

Plot the mesh and, optionally, the mesh quality. When plotting the mesh quality, you can use the slider to adjust the threshold for the range of metric values shown. This makes it easier to identify cells with the poorest and best mesh quality.

```
pylith_viz --filenames FILENAMES plot_mesh [--mesh-quality METRIC] [--help]
```

-help

Display help information for subcommand.

-mesh-quality METRIC

Plot the mesh quality using METRIC. Refer to the [VTK mesh quality documentation](#) for definitions of the metrics.

plot_field Subcommand

Plot a diagnostic, solution, or derived field. If the data has more than one time step, then a slider will be enabled so that you can plot the data at different time steps. Additionally, you can press the `n` key to increment time and the `p` key to decrement time.

The colormap and its range are selected based upon the component. When you plot the magnitude component, the plotting uses the `plasma` sequential colormap with the minimum value set to 0 and the maximum value set to the maximum absolute magnitude over the time span of the data. When you plot a vector or tensor component, the plotting uses the a blue-red diverging colormap with zero at the center of the colormap and the range set to the maximum absolute value of the component over the time span of the data.

```
pylith_viz --filenames FILENAMES plot_field [--field FIELD_NAME] [--component COMPONENT]
↔ [--hide-edges] [--help]
```

-help

Display help information for subcommand.

-field FIELD_NAME

Plot field FIELD_NAME (default: displacement).

-component COMPONENT

Component to display (default: magnitude).

-hide-edges

Do not show cell edges.

warp_grid Subcommand

Plot a solution field and warp the grid according to the displacement vector. You can specify a scale factor for how much the deformation is exaggerated using a slider or a command line argument. The maximum exaggeration is 5.0 times the initial exaggeration.

If the data has more than one time step, then a slider will be enabled so that you can plot the data at different time steps. Additionally, you can press the `n` key to increment time and the `p` key to decrement time.

The colormap and its range are selected based upon the component. When you plot the magnitude component, the plotting uses the plasma sequential colormap with the minimum value set to 0 and the maximum value set to the maximum absolute magnitude over the time span of the data. When you plot a vector or tensor component, the plotting uses the a blue-red diverging colormap with zero at the center of the colormap and the range set to the maximum absolute value of the component over the time span of the data.

```
pylith_viz --filenames FILENAMES warp_grid [--field FIELD_NAME] [--component COMPONENT]
↪ [--exaggeration EXAGGERATION] [--hide-outline] [--hide-edges] [--help]
```

-help

Display help information for subcommand.

-field FIELD_NAME

Plot field FIELD_NAME (default: displacement).

-component COMPONENT

Component to display (default: magnitude).

-exaggeration EXAGGERATION

Scale factor for exaggerating the displacement deformation (default: 1000).

-hide-outline

Turn off showing the grid outline with the warped grid.

-hide-edges

Do not show cell edges on warped grid.

pyre_doc.py

New in v3.0.0.

This utility is part of the Pythia/Pyre framework. It will be installed to the bin directory where Pythia/Pyre is installed. This utility extracts the Python docstrings and help information for Pyre components.

Warning

This utility does not work on the PyLithApp application object because it is a mpi.Application object.

```
pyre_doc.py [--help] [--short] OBJECT
```

-help

Display help information for script.

-short

Display only the docstrings for the specified module or object. The default is to display the information for the specified object and all child objects, such as classes within a module.

OBJECT

Full path to Python module or object, such as `pylith.problems.Problem` (module) or `pylith.problems.Problem.Problem` (object).

Listing 4.17: Example of running `pyre_doc.py` on `pylith.problems.Problem`

```
$ pyre_doc.py pylith.problems.Problem
No help available for module pylith.problems.Problem.
```

(continues on next page)

(continued from previous page)

```

class Problem
Python abstract base class for crustal dynamics problems.

    FACTORY: problem

facilities of 'problem':
    bc=<component name>: Boundary conditions.
        current value: 'emptybin', from {default}
        configurable as: emptybin, bc
    defaults=<component name>: Default options for problem.
        current value: 'problem_defaults', from {default}
        configurable as: problem_defaults, defaults
    gravity_field=<component name>: Database used for gravity field.
        current value: 'nullcomponent', from {default}
        configurable as: nullcomponent, gravity_field
    interfaces=<component name>: Interior surfaces with constraints or constitutive_
↪models.
        current value: 'emptybin', from {default}
        configurable as: emptybin, interfaces
    materials=<component name>: Materials in problem.
        current value: 'homogeneous', from {default}
        configurable as: homogeneous, materials
    scales=<component name>: Nondimensionalizer for problem.
        current value: 'nondimelasticquasistatic', from {default}
        configurable as: nondimelasticquasistatic, scales
    solution=<component name>: Solution field for problem.
        current value: 'solution', from {default}
        configurable as: solution
    solution_observers=<component name>: Observers (e.g., output) for solution.
        current value: 'singlesolnobserver', from {default}
        configurable as: singlesolnobserver, solution_observers
properties of 'problem':
    formulation=<str>: Formulation for equations.
        default value: 'quasistatic'
        current value: 'quasistatic', from {default}
        validator: (in ['quasistatic', 'dynamic', 'dynamic_imex'])
    help=<bool>: prints a screen that describes my traits
        default value: False
        current value: False, from {default}
    help-components=<bool>: prints a screen that describes my subcomponents
        default value: False
        current value: False, from {default}
    help-persistence=<bool>: prints a screen that describes my persistent store
        default value: False
        current value: False, from {default}
    help-properties=<bool>: prints a screen that describes my properties
        default value: False
        current value: False, from {default}
    solver=<str>: Type of solver to use ['linear', 'nonlinear'].
        default value: 'linear'
        current value: 'linear', from {default}

```

(continues on next page)

(continued from previous page)

```
validator: (in ['linear', 'nonlinear'])
```

pylith_convertmesh

New in v5.0.0.

This utility converts a finite-element mesh from one file format to another. The primary use case is converting an Exodus II file from Cubit or a Gmsh file to an HDF5 file in the PETSc mesh format, which PyLith can read in parallel. All other mesh formats are read in serial (loaded onto a single process) and then the mesh is distributed among the processes.

The utility uses the `pylith.initializers.Initializer` to perform the conversion. By default, the conversion includes

1. Read the mesh using an implementation of `MeshIO` (default is `MeshIOPetsc`).
2. Reorder the mesh using the reverse Cuthill-McKee algorithm.
3. Write the mesh using an implementation of `MeshIO` (default is `MeshIOPetsc`).

This is a Pyre application, so you can get help using the same command line help options as those available for PyLith.

Listing 4.18: Example of running `pylith_convertmesh` with command line arguments for getting help.

```
pylith_convertmesh [--help] [--help-components] [--help-properties]
```

Listing 4.19: Example of running `pylith_convertmesh` to convert a Gmsh file `mesh_tri.msh` to an HDF5 file in PETSc mesh format `mesh_tri.h5`.

```
pylith_convertmesh --mesh_initializer.phases.read_mesh.reader.filename=mesh_tri.mesh --
↪mesh_initializer.phases.write_mesh.writer.filename=mesh_tri.h5
```

Listing 4.20: Equivalent parameter file for running `pylith_convertmesh` to convert a Gmsh file `mesh_tri.msh` to an HDF5 file in PETSc mesh format `mesh_tri.h5`.

```
[mesh_initializer.phases.read_mesh.reader]
filename = mesh_tri.mesh

[mesh_initializer.phases.write_mesh.writer]
filename = mesh_tri.h5
```

Listing 4.21: Example of running `pylith_convertmesh` to convert an Exodus II file `mesh_tri.exo` to an HDF5 file in PETSc mesh format `mesh_tri.h5`.

```
pylith_convertmesh --mesh_initializer.phases.read_mesh.reader=pylith.meshio.MeshIOCubit -
↪mesh_initializer.phases.read_mesh.reader.filename=mesh_tri.exo --mesh_initializer.
↪phases.write_mesh.writer.filename=mesh_tri.h5
```

Listing 4.22: Equivalent parameter file for running `pylith_convertmesh` to convert an Exodus II file `mesh_tri.exo` to an HDF5 file in PETSc mesh format `mesh_tri.h5`.

```
[mesh_initializer.phases.read_mesh]
reader = pylith.meshio.MeshIOCubit
reader.filename = mesh_tri.exo

[mesh_initializer.phases.write_mesh.writer]
filename = mesh_tri.h5
```

pylith_cfgsearch

New in v3.0.0.

This utility searches and displays the metadata in `.cfg` files based on criteria provided via the command line.

```
pylith_cfgsearch [--help] [--path SEARCHPATH] [--display DISPLAY] [--verbose] [--
→keywords KEYWORDS]
  [--features FEATURES] [--authors AUTHORS] [--version VERSION] [--incompatible]
```

-help

Display help information for script.

-path SEARCHPATH

Search path for `.cfg` files (default: `./`).

-display DISPLAY

List of metadata to display in search results (default: `all`).

-keywords KEYWORDS

Comma delimited list of keywords for filtering search results (default: `None`).

-features FEATURES

Comma delimited list of features for filtering search results (default: `None`).

-authors AUTHORS

Comma delimited list of authors for filtering search results (default: `None`).

-version VERSION

PyLith version for filtering search results (default: `None`).

-verbose

Report missing metadata (default: `False`).

-incompatible

Filter search results to show incompatible parameter files (default: `False`).

Listing 4.23: Example of running `pylith_cfgsearch` in `examples/strikeslip-2d`.

```
$ pylith_cfgsearch
step01_slip.cfg v1.0.0; requires PyLith >=3.0 and <5.0
Coseismic prescribed slip with zero displacement Dirichlet boundary conditions.
Authors: Brad Aagaard
Keywords: example, 2D, strike slip, prescribed slip
Features:
```

(continues on next page)

(continued from previous page)

```

    Triangular cells, pylith.meshio.MeshIOCubit, pylith.problems.TimeDependent,
    ↪pylith.materials.Elasticity,
      pylith.materials.IsotropicLinearElasticity, pylith.faults.FaultCohesiveKin,
    ↪pylith.faults.KinSrcStep, field split
      preconditioner, Schur complement preconditioner, pylith.bc.
    ↪DirichletTimeDependent, spatialdata.spatialdb.UniformDB,
      pylith.meshio.OutputSolnBoundary, pylith.meshio.DataWriterHDF5, Static simulation
    pylith step01_slip.cfg
step02_slip_velbc.cfg v1.0.0; requires PyLith >=3.0 and <5.0
    Coseismic prescribed slip with velocity Dirichlet boundary conditions.
    Authors: Brad Aagaard
    Keywords: example, 2D, strike slip, prescribed slip, velocity boundary conditions
    Features:
      Triangular cells, pylith.meshio.MeshIOCubit, pylith.problems.TimeDependent,
    ↪pylith.materials.Elasticity,
      pylith.materials.IsotropicLinearElasticity, pylith.faults.FaultCohesiveKin,
    ↪pylith.faults.KinSrcStep, field split
      preconditioner, Schur complement preconditioner, pylith.bc.
    ↪DirichletTimeDependent, spatialdata.spatialdb.UniformDB,
      pylith.meshio.OutputSolnBoundary, pylith.meshio.DataWriterHDF5, Quasi-static
    ↪simulation, spatialdata.spatialdb.SimpleDB
    pylith step02_slip_velbc.cfg
step03_multislip_velbc.cfg v1.0.0; requires PyLith >=3.0 and <5.0
    Coseismic prescribed slip with multiple ruptures and velocity Dirichlet boundary
    ↪conditions.
    Authors: Brad Aagaard
    Keywords: example, 2D, strike slip, prescribed slip, multiple fault ruptures,
    ↪velocity boundary conditions
    Features:
      Triangular cells, pylith.meshio.MeshIOCubit, pylith.problems.TimeDependent,
    ↪pylith.materials.Elasticity,
      pylith.materials.IsotropicLinearElasticity, pylith.faults.FaultCohesiveKin,
    ↪pylith.faults.KinSrcStep, field split
      preconditioner, Schur complement preconditioner, pylith.bc.
    ↪DirichletTimeDependent, spatialdata.spatialdb.UniformDB,
      pylith.meshio.OutputSolnBoundary, pylith.meshio.DataWriterHDF5, Quasi-static
    ↪simulation, spatialdata.spatialdb.SimpleDB
    pylith step03_multislip_velbc.cfg

```

Listing 4.24: Example of running `pylith_cfgsearch` in `examples/strikeslip-2d`, limiting output to the description and keywords.

```

$ pylith_cfgsearch --display=description,keywords
step01_slip.cfg
    Coseismic prescribed slip with zero displacement Dirichlet boundary conditions.
    Keywords: example, 2D, strike slip, prescribed slip
step02_slip_velbc.cfg
    Coseismic prescribed slip with velocity Dirichlet boundary conditions.
    Keywords: example, 2D, strike slip, prescribed slip, velocity boundary conditions
step03_multislip_velbc.cfg
    Coseismic prescribed slip with multiple ruptures and velocity Dirichlet boundary
    ↪conditions.

```

(continues on next page)

(continued from previous page)

Keywords: example, 2D, strike slip, prescribed slip, multiple fault ruptures,
 ↪ velocity boundary conditions

Listing 4.25: Example of running `pylith_cfgsearch` in `examples/strikeslip-2d`, filtering search results to quasistatic simulations.

```
$ pylith_cfgsearch --features="Quasi-static simulation"
step02_slip_velbc.cfg v1.0.0; requires PyLith >=3.0 and <5.0
  Coseismic prescribed slip with velocity Dirichlet boundary conditions.
  Authors: Brad Aagaard
  Keywords: example, 2D, strike slip, prescribed slip, velocity boundary conditions
  Features:
    Triangular cells, pylith.meshio.MeshIOCubit, pylith.problems.TimeDependent,
    ↪ pylith.materials.Elasticity,
      pylith.materials.IsotropicLinearElasticity, pylith.faults.FaultCohesiveKin,
    ↪ pylith.faults.KinSrcStep, field split
      preconditioner, Schur complement preconditioner, pylith.bc.
    ↪ DirichletTimeDependent, spatialdata.spatialdb.UniformDB,
      pylith.meshio.OutputSolnBoundary, pylith.meshio.DataWriterHDF5, Quasi-static,
    ↪ simulation, spatialdata.spatialdb.SimpleDB
      pylith step02_slip_velbc.cfg
step03_multislip_velbc.cfg v1.0.0; requires PyLith >=3.0 and <5.0
  Coseismic prescribed slip with multiple ruptures and velocity Dirichlet boundary
  ↪ conditions.
  Authors: Brad Aagaard
  Keywords: example, 2D, strike slip, prescribed slip, multiple fault ruptures,
  ↪ velocity boundary conditions
  Features:
    Triangular cells, pylith.meshio.MeshIOCubit, pylith.problems.TimeDependent,
    ↪ pylith.materials.Elasticity,
      pylith.materials.IsotropicLinearElasticity, pylith.faults.FaultCohesiveKin,
    ↪ pylith.faults.KinSrcStep, field split
      preconditioner, Schur complement preconditioner, pylith.bc.
    ↪ DirichletTimeDependent, spatialdata.spatialdb.UniformDB,
      pylith.meshio.OutputSolnBoundary, pylith.meshio.DataWriterHDF5, Quasi-static,
    ↪ simulation, spatialdata.spatialdb.SimpleDB
      pylith step03_multislip_velbc.cfg
```

pylith_runner

New in v3.0.0.

The runner utility searches a directory path for `.cfg` files with arguments in the simulation metadata (see [PyLith Application](#) for details). It uses those arguments to run PyLith simulations.

Important

The runner utility only knows how to run PyLith simulations, it does not know how to do pre- or post-processing.

```
pylith_runner [--help] [--path SEARCHPATH] [--verbose]
```

-help

Display help information for script.

-path SEARCHPATH

Search path for .cfg files (default: ./).

-verbose

Report missing metadata (default: False).

Listing 4.26: Example of using `pylith_runner` to run all simulations in `examples/box-2d` (output omitted).

```
$ pylith_runner --path=examples/box-2d
```

pylith_dumpparameters

This utility dumps all PyLith parameters collected from `cfg` files, the command line, and any other sources, into a file. The output file can be a simple text file or a JSON file. The JSON file can be viewed using the *PyLith Parameter Viewer*.

```
pylith_dumpparameters [--quiet] [--format=ascii|json] [--filename=FILENAME]
```

-quiet

Don't include description, location, or aliases in output.

-format=ascii|json

Format of output file (default=json).

-filename=FILENAME

Name of output file.

pylith_eqinfo

This utility computes the moment magnitude, seismic moment, seismic potency, and average slip at user-specified time snapshots from PyLith fault HDF5 output. The utility works with output from simulations with either prescribed slip or spontaneous rupture. The moment magnitudes, seismic moment, and seismic potency for 2D simulation have limited value, because they use on a 1D fault. Currently, we compute the shear modulus from a user-specified spatial database at the centroid of the fault cells. In the future we plan to automatically account for lateral variations in shear modulus across the fault when calculating the seismic moment. The average slip, rupture area, seismic moment, and seismic potency are all given in SI units.

Tip

From Wu and Chen [2003] the seismic moment, M_0 , when considering a lateral variation in the shear modulus across the fault is

$$M_0 = \mu_{eff} AD \quad (4.436)$$

where A is the rupture area, D is the average slip, and μ_{eff} is the effective shear modulus given by

$$\mu_{eff} = \frac{1}{2} (\mu^+ + \mu^-) \left(1 - \left(\frac{\mu^+ - \mu^-}{\mu^+ + \mu^-} \right)^2 \right) \quad (4.437)$$

where μ^+ and μ^- are the shear modulus on each side of the fault.

The Python script is a Pyre application, so its parameters can be specified using `cfg` files (`eqinfoapp.cfg` will be read if it exists) or command line arguments just like PyLith.

The Pyre properties and facilities include:

output_filename

Filename for output of slip information.

coordsys

Coordinate system associated with mesh in simulation.

faults

Array of fault names.

filename_pattern

Filename pattern in C/Python format for creating filename for each fault. Default is `output/fault_\\%s.h5`.

snapshots

Array of timestamps for slip snapshots (`[-1]` means use last time step in file, which is the default).

snapshot_units

Units for timestamps in array of snapshots.

db_properties

Spatial database for elastic properties.

Listing 4.27: Example `cfg` file for running `pylith_eqinfo` in `examples/strikeslip-2d` to compute the earthquake magnitude, seismic moment, and average slip for `step04_varslip`.

```
[eqinfoapp]
output_filename = output/step04_varslip-eqinfo.py

coordsys.space_dim = 2

faults = [fault]
filename_pattern = output/step04_varslip-%s.h5

db_properties = spatialdata.spatialdb.UniformDB
db_properties.description = Fault properties
db_properties.values = [density, Vs]
db_properties.data = [2500.0*kg/m**3, 3.46*km/s]
```

pylith_genxdmf

This utility generates Xdmf files from HDF5 files that conform to the layout used by PyLith. It is a simple Python script with a single command line argument with the file pattern of HDF5 files for which Xdmf files should be generated.

Typically, it is used to regenerate Xdmf files that get corrupted or lost due to renaming and moving. It is also useful in updating Xdmf files when users add fields to HDF5 files during post-processing.

```
pylith_genxdmf --files=FILE_OR_FILE_PATTERN
```

The default value for `FILE_OR_FILE_PATTERN` is `*.h5`.

Warning

If the HDF5 files contain external datasets, then this utility should be run from the same relative path to the HDF5 files as when they were created. For example, if a PyLith simulation was run from directory `work` and HDF5 files were generated in `output/work`, then the utility should be run from the directory `work`. Furthermore, a visualization tool, such as ParaView, should also be started from the working directory `work`.

pylith_powerlaw_gendb

This Pyre application generates a SimpleDB spatial database file with power-law viscoelastic material properties for use with PyLith. The inputs are spatial databases with values often available from laboratory experiments, such as activation energy, temperature, power-law coefficient, and the power-law exponent. An additional parameter defines the units of the activation energy. You must also specify either a reference stress or a reference strain rate.

You place all of the application parameters in `powerlaw_gendb.cfg`, which the application will read by default. See *Step 8: Slip on Two Faults and Power-law Viscoelastic Materials* for an example of how to use `pylith_powerlaw_gendb`.

PyLith Parameter Viewer

New in v2.2.0.

The PyLith Parameter Viewer provides a graphical user interface for viewing the parameters associated with a PyLith simulation and the version information for PyLith and its dependencies. This viewer is an updated and interactive interface to the information generated by the `pylith_dumpparameters` script. It displays the hierarchy of components and the parameters for each one, including default values.

Installation

The PyLith Parameter Viewer is included in the PyLith binary distributions and PyLith Docker container for versions 2.1.5 and later. Additionally, the PyLith Installer will install the Parameter Viewer by default. For manual installation you can download the PyLith Parameter Viewer tarball from the PyLith software page <https://geodynamics.org/resources/pylith/supportingdocs/>. After downloading the tarball, unpack it. We recommend unpacking the tarball in the top-level PyLith directory.

```
$ tar -xvf pylith_parameters-1.1.0.tgz
```

Running the Parameter Viewer

The steps to run the parameter viewer are:

1. Generate the parameter JSON file.
2. Start the web server (if not already running).
3. Load the parameter JSON file.

Generate the parameter JSON file

The parameter viewer uses a JSON file with all of the parameters collected from `cfg` files, command line arguments, etc as input. This file can be generated using `pylith_dumpparameters` and, by default, it will be generated whenever a PyLith simulation is run. When using `pylith_dumpparameters` the name of the parameter file can be set via a command line argument. When using PyLith the `DumpParametersJSON` component contains a property for the name of the file. You can set the filename on the command line

```
$ pylith --dump_parameters.filename=FILENAME.json
```

or within a `cfg` file

```
[pylithapp.dump_parameters]
filename = FILENAME.json
```

Currently, the JSON parameter file cannot be used to run a PyLith simulation. This feature will be added in an upcoming release.

Start the web server

Change to the directory containing the `pylith_paramviewer` script (usually the `parametersgui` directory under the top-level `pylith` directory), and run the `pylith_paramviewer` script. This will start a simple Python-based web server on your local computer. Alternatively, you can use the online version at https://geodynamics.github.io/pylith_parameters/.

```
$ cd parametersgui
$ ./pylith_paramviewer
```

The script will instruct you to point your web browser to a local port on your computer. The default is `http://127.0.0.1:9000`. You can change the default port using the `--port` command line argument to the `pylith_paramviewer` script.

Using the Parameter Viewer

When you point your web browser to the correct port, you should see the PyLith Parameter Viewer as shown in Fig. 4.8. Click the **Choose File** button and navigate to the desired JSON parameter file. The viewer tarball includes a sample parameter file `sample_parameters.json`. Click the **Reload** button to reload the same JSON parameter file if you regenerate it. To select a new JSON parameter file, click the **Choose File** button and navigate to the desired file.

Version Information

Click on the **Version** tab to examine the version information. This tab displays the same version information shown with the `--version` command line argument to `pylith` in an easy to read layout. This includes information about the platform on which `pylith` or `pylith_dumpparameters` was run, the PyLith version, and versions of the dependencies, as shown in Fig. 4.9.

Parameter Information

Click on the **Parameters** tab to examine the hierarchy of components and the parameters for each. You can expand/collapse the Component Hierarchy tree in the left panel by clicking on the triangles or facility name in blue to the left of the equals sign (Fig. 4.10). Clicking on the component in red to the right of the equals sign will show its parameters in the right panel (Fig. 4.10). The selected facility in the left panel whose parameters are shown in the right panel will be highlighted via a gray background (Fig. 4.11).

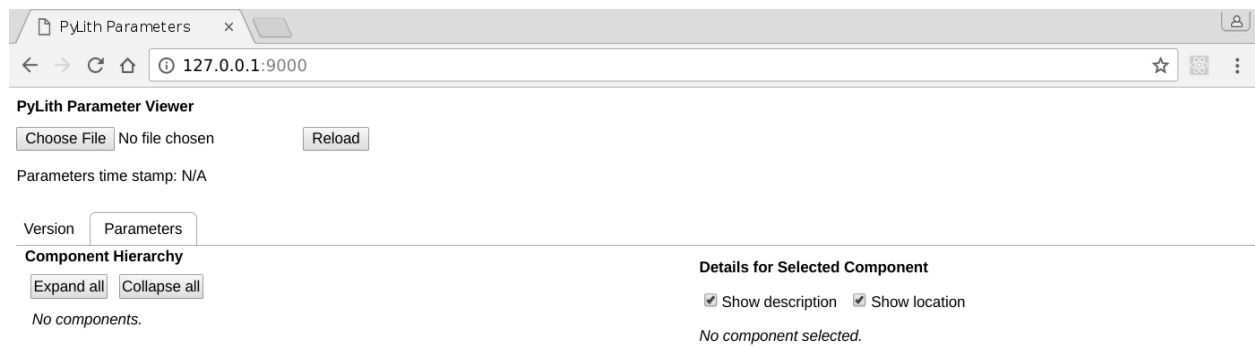


Fig. 4.8: Screenshot of PyLith Parameter Viewer in web browser upon startup.

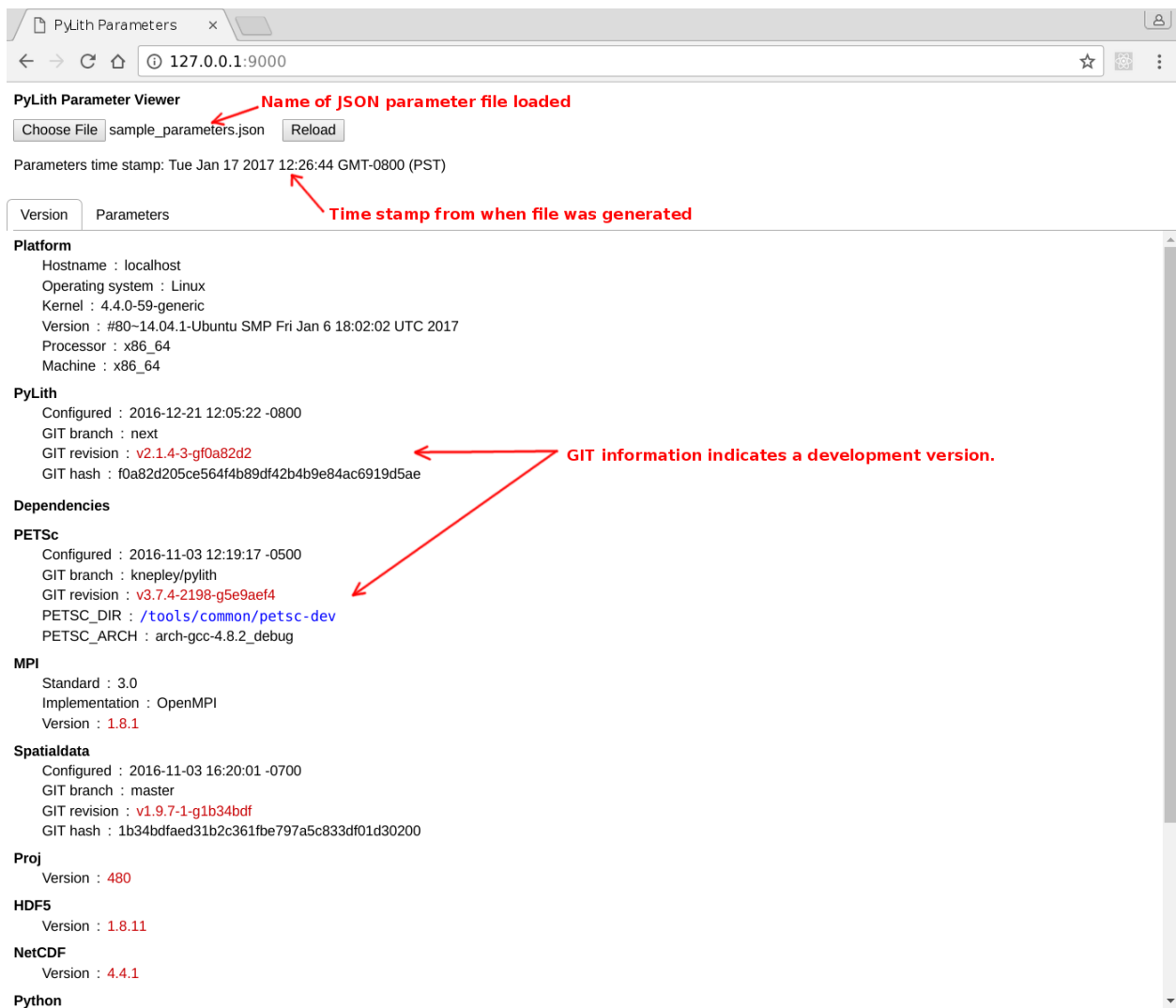


Fig. 4.9: Screenshot of Version tab of the PyLith Parameter Viewer with sample JSON parameter file.

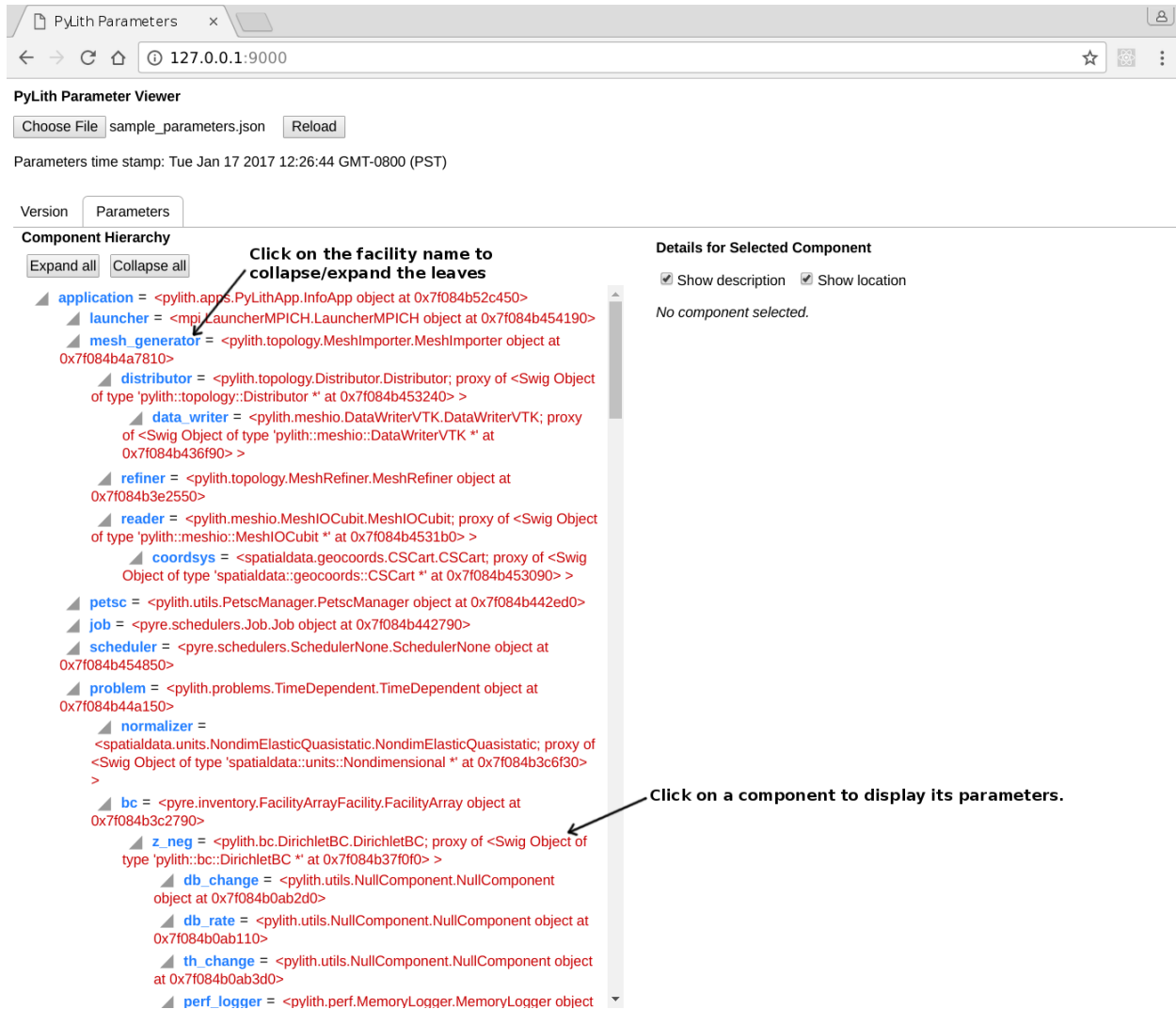


Fig. 4.10: Screenshot of Parameters tab of the PyLith Parameter Viewer with sample JSON parameter file before selecting a component in the left panel.

The screenshot shows the PyLith Parameter Viewer application. The browser address bar displays '127.0.0.1:9000'. The application title is 'PyLith Parameters'. Below the title bar, there is a 'Choose File' button and a text input field containing 'sample_parameters.json', followed by a 'Reload' button. The parameters time stamp is 'Tue Jan 17 2017 12:26:44 GMT-0800 (PST)'. The interface has two tabs: 'Version' and 'Parameters', with 'Parameters' being the active tab. The main content area is divided into two panels. The left panel, titled 'Component Hierarchy', shows a tree structure of components. The right panel, titled 'Details for Selected Component', shows the details for the selected component, 'z_neg'.

Component Hierarchy

- Expand all Collapse all
- application = <pylith.apps.PyLithApp.InfoApp object at 0x7f084b52c450>
 - launcher = <mpi.LauncherMPICH.LauncherMPICH object at 0x7f084b454190>
 - mesh_generator = <pylith.topology.MeshImporter.MeshImporter object at 0x7f084b4a7810>
 - distributor = <pylith.topology.Distributor.Distributor; proxy of <Swig Object of type 'pylith::topology::Distributor *' at 0x7f084b453240> >
 - data_writer = <pylith.meshio.DataWriterVTK.DataWriterVTK; proxy of <Swig Object of type 'pylith::meshio::DataWriterVTK *' at 0x7f084b436f90> >
 - refiner = <pylith.topology.MeshRefiner.MeshRefiner object at 0x7f084b3e2550>
 - reader = <pylith.meshio.MeshIOcubit.MeshIOcubit; proxy of <Swig Object of type 'pylith::meshio::MeshIOcubit *' at 0x7f084b4531b0> >
 - coordsys = <spatialdata.geocoords.CSCart.CSCart; proxy of <Swig Object of type 'spatialdata::geocoords::CSCart *' at 0x7f084b453090> >
 - petsc = <pylith.utils.PetscManager.PetscManager object at 0x7f084b442ed0>
 - job = <pyre.schedulers.Job.Job object at 0x7f084b442790>
 - scheduler = <pyre.schedulers.SchedulerNone.SchedulerNone object at 0x7f084b454850>
 - problem = <pylith.problems.TimeDependent.TimeDependent object at 0x7f084b44a150>
 - normalizer = <spatialdata.units.NondimElasticQuasistatic.NondimElasticQuasistatic; proxy of <Swig Object of type 'spatialdata::units::Nondimensional *' at 0x7f084b3c6f30> >
 - bc = <pyre.inventory.FacilityArrayFacility.FacilityArray object at 0x7f084b3c2790>
 - z_neg = <pylith.bc.DirichletBC.DirichletBC; proxy of <Swig Object of type 'pylith::bc::DirichletBC *' at 0x7f084b37f0f0> >
 - db_change = <pylith.utils.NullComponent.NullComponent object at 0x7f084b0ab2d0>
 - db_rate = <pylith.utils.NullComponent.NullComponent object at 0x7f084b0ab110>
 - th_change = <pylith.utils.NullComponent.NullComponent object at 0x7f084b0ab3d0>
 - perf_logger = <pylith.perf.MemoryLogger.MemoryLogger object at 0x7f084b0ab3d0>

Details for Selected Component

☒ Show description ☒ Show location

z_neg = <pylith.bc.DirichletBC.DirichletBC; proxy of <Swig Object of type 'pylith::bc::DirichletBC *' at 0x7f084b37f0f0> >

Component information

Full path : [application.problem.bc.z_neg]
 Configurable as : dirichletbc, z_neg
 Description : No description available.
 Set from : {default}

Properties

bc_dof (list) = [2]
 Description : Indices of boundary condition DOF (0=1st DOF, 1=2nd DOF, etc).
 Set from : {file='step01.cfg', line=91, column=-1}

up_dir (list) = [0, 0, 1]
 Description : Direction perpendicular to horizontal tangent direction that is not collinear with normal direction.
 Set from : {default}

label (str) = face_zneg
 Description : Label identifier for boundary.
 Set from : {file='step01.cfg', line=92, column=-1}

Facilities (subcomponents)

db_change = <pylith.utils.NullComponent.NullComponent object at 0x7f084b0ab2d0>
 Configurable as : nullcomponent, db_change
 Description : Database with temporal change in values.
 Set from : {default}

db_rate = <pylith.utils.NullComponent.NullComponent object at 0x7f084b0ab110>
 Configurable as : nullcomponent, db_rate
 Description : Database with rate of change values.
 Set from : {default}

th_change = <pylith.utils.NullComponent.NullComponent object at 0x7f084b0ab3d0>
 Configurable as : nullcomponent, th_change
 Description : Database with time history.

Fig. 4.11: Screenshot of Parameters tab of the PyLith Parameter Viewer with sample JSON parameter file with `z_neg` facility selected.

Troubleshooting

Tips and Hints For Running PyLith

- Examine the examples for a problem similar to the one you want to run and dissect it in detail.
- Start with a uniform-resolution coarse mesh to debug the problem setup. Increase the resolution as necessary to resolve the solution fields of interest (resolving stresses/strains may require a higher resolution than that for resolving displacements).
- Merge materials using the same material model. This will result in only one VTK or HDF5 file for each material model rather than several files.
- The rate of convergence in quasistatic (implicit) problems can sometimes be improved by renumbering the vertices in the finite-element mesh to reduce the bandwidth of the sparse matrix. PyLith can use the reverse Cuthill-McKee algorithm to reorder the vertices and cells.
- If you encounter errors or warnings, run `pylith_dumpparameters` or use the `--help`, `--help-components`, or `--help-properties` command-line arguments when running PyLith to check the parameters to make sure PyLith is using the parameters you intended.
- Use the `--petsc.log_view`, `--petsc.ksp_monitor`, `--petsc.ksp_view`, `--petsc.ksp_converged_reason`, and `--petsc.snes_converged_reason` command-line arguments (or set them in a parameter file) to view PyLith performance and monitor the convergence.
- Turn on additional journals (Refer to Section [Journal](#)) to monitor what the code is doing.

See [Troubleshooting \(2D\)](#) for examples of how to troubleshoot running PyLith simulations. Also consult the PyLith category in the [CIG community forum](#) to see if someone else encountered a similar issue.

Error Message Format

Configuration

Errors caught in simple verification of simulation parameters will display information in the form:

```
>> FILE:LINE:
-- error (pyre.inventory)
-- COMPONENT <- VALUE
-- MESSAGE
```

where the assignment of value `VALUE` to component `COMPONENT` at line `LINE` in file `FILE` generates the error message `MESSAGE`.

Python

Errors caught in Python code will display the Python backtrace followed by the error message.

C++

New in v5.0.0.

Errors caught in C++ code will display the error message using an error or firewall journal channel, the Python backtrace, the exception message, and then the full C++ backtrace with raw symbols names for the functions. You can use the `c++filt` utility to demangle the function names.

Common Error Messages

Import Error and Missing Library

```
ImportError: liblapack.so.2: cannot open shared object file: No such file or directory
```

PyLith cannot find one of the libraries. You need to set up your environment variables (e.g., PATH, PYTHONPATH, and LD_LIBRARY_PATH) to match your installation. If you are using the PyLith binary on Linux or macOS, run the command `source setup.sh` in the directory where you unpacked the distribution. This will set up your environment variables for you. If you are building PyLith from source, please consult the instructions for building from source.

Unrecognized Property 'p4wd'

```
-- error (pyre.inventory) } \\  
-- p4wd <- 'true' } \\  
-- unrecognized property 'p4wd' } \\  
>> command line:: } \\  
-- error (pyre.inventory) } \\  
-- p4pg <- 'true' } \\  
-- unrecognized property ' p4pg'}
```

Verify that the `mpirun` command included in the PyLith package is the first one on your PATH by running the command `which mpirun`. If it is not, adjust your PATH environment variable accordingly.

Detected zero pivot in LU factorization

```
-- Solving equations.  
[0] PETSC ERROR: -----  
Error Message -----  
[0] PETSC ERROR: Detected zero pivot in LU factorization  
see http://www.mcs.anl.gov/petsc/petsc-as/documentation/faq.html#ZeroPivot!
```

This usually occurs when the null space of the system Jacobian is nonzero, such as the case of a problem without Dirichlet boundary conditions on any boundary. If this arises and no additional Dirichlet boundary conditions are desired, then the workaround is to change the preconditioner from LU to ILU. See *PETSc Options* for additional information.

Bus Error

This often indicates that PyLith is using incompatible versions of libraries. This can result from changing your environment variables after configuring or installing PyLith (when building from source) or from errors in setting the environment variables PATH, LD_LIBRARY_PATH, and PYTHONPATH. If the former case, simply reconfigure and rebuild PyLith. In the latter case, check your environment variables (order matters!) to make sure PyLith finds the desired directories before system directories.

Segmentation Fault

A segmentation fault usually results from an invalid read/write to memory. It might be caused by an error that wasn't trapped or a bug in the code. Please report these cases so that we can fix these problems (either trap the error and provide the user with an informative error message, or fix the bug). If this occurs with any of the problems distributed with PyLith, simply submit a bug report (see *Getting Help and Reporting Bugs*) indicating which problem you ran and your platform.

Important

PETSc will often report errors as segmentation faults even if the underlying problem is not an invalid read/write. If you see PETSc reporting a segmentation fault, examine the output carefully for other error messages that indicate the real issue is something else.

If the crash occurs for a problem you created, it is a great help if you can try to reproduce the crash with a very simple problem (e.g., adjust the boundary conditions or other parameters of one of the examples to reproduce the segmentation fault). Submit a bug report along with log files showing the backtrace from a debugger (e.g., gdb) and the valgrind log file (only available on Linux platforms). You can generate a backtrace using the debugger by using the `--petsc.start_in_debugger` command-line argument:

```
$ pylith [--args...] --petsc.start_in_debugger
(gdb) continue
(gdb) backtrace
```

To use valgrind to detect the memory error, first go to your working directory and run the problem with `--launcher.dry`:

```
$ pylith [--args...] --launcher.dry
```

Instead of actually running the problem, this causes PyLith to dump the mpirun/mpiexec command it will execute. Copy and paste this command into your shell so you can run it directly. Insert the full path to valgrind before the full path to mpinemesiis and tell valgrind to use a log file:

```
$ mpirun /path/to/valgrind --log-file=valgrind-log /path/to/mpinemesiis --pyre-start
[...lots of junk...]
```

Diagnosing solver issues

SNES does not converge in one iteration for a linear problem

The PETSc nonlinear solver (SNES) should converge in one iteration for a linear problem. If it does not, it usually means that either the Jacobian and residual are not consistent or there is a problem with the preconditioner. The following steps can help diagnose the source of the problem.

1. Check the consistency of the Jacobian and residual using `--petsc.snes_check_jacobian=1.0e-6` and `--petsc.snes_check_jacobian_view`.
2. Use LU for all preconditioners on a small problem and verify that the true residual matches the preconditioner residual. Helpful PETSc settings include `--petsc.ksp_monitor_true_residual` and `--petsc.pc_type=lu`.
3. Use the desired preconditioner and compare the true and preconditioned residuals. If they differ, then the preconditioner may not be appropriate for the boundary value problem or you may need to adjust the scales for nondimensionalization. Manually verify that the terms in the Jacobian will be $O(1)$, adjusting the scales for nondimensionalization accordingly.
4. If using `preonly` for the KSP type (often the default), then try switching to `gmres` with a KSP tolerance of $1.0e-12$. If that works, then try backing off on the tolerance. Helpful PETSc settings include `--petsc.ksp_view=true`, `--petsc.ksp_type=gmres` and `--petsc.ksp_rtol=1.0e-12`.

4.2.5 Mesh generation with Cubit and Gmsh

Overview

Cubit and Gmsh can generate quadrilateral or triangular meshes in 2D and hexahedral or tetrahedral meshes in 3D. In each case, you first create the geometry, specify the meshing algorithm and discretization size, and then generate the mesh. You can build up the geometry from points, curves, surfaces, and volumes or use the geometry engines to construct the domain using simple shapes.

2D meshing

Constructing surfaces from points and curves for 2D meshing with Cubit and Gmsh is very similar. Cubit provides more geometric operations than Gmsh, but many simple geometric operations in Gmsh can be implemented by the user when using the Python interface. Generating unstructured quadrilateral meshes for complex geometry is often easier in Cubit, whereas generating meshes with complex specification of discretization size is often easier with Gmsh.

3D meshing

Cubit provides an extensive suite of tools for constructing complex 3D geometry. This includes building surfaces and performing geometric operations on surfaces and volumes. The suite of tools in the Gmsh geometry engine is more limited; the OpenCASCADE engine interface provides additional tools. With either Cubit or Gmsh, you can use external CAD tools to generate the geometry. As in the case with 2D meshing, generating unstructured hexahedral meshes is often easier in Cubit, whereas generating meshes with complex specification of discretization size is often easier in Gmsh.

Marking surfaces

New in v5.0.0.

Beginning with v5.0.0, PyLith supports marking surfaces associated with boundary conditions, fault interfaces, and other surfaces using the cell faces rather than vertices. In Cubit, marking faces rather than vertices corresponds to using sidesets rather than nodesets. In Gmsh, marking faces rather than vertices means a physical group needs to contain only entities at the highest dimension (surfaces in 3D and curves in 2D). Marking surfaces using vertices will be removed in v6.0.0.

Reading meshes in parallel

New in v5.0.0.

When running PyLith with multiple processes (parallel processing), Exodus II files from Cubit and Gmsh files are read in serial and then distributed among the processes. This means that one process must be able to hold the entire mesh in memory, and this approach does not scale to very large simulations. The preferred approach for large simulations is to convert the finite-element mesh to an HDF5 file in the PETSc mesh format. This format is designed for parallel I/O, and PyLith will read the mesh in parallel. Refer to [pylith_convertmesh](#) for more information about converting mesh files.

Cubit

Danger

Starting in late 2024, Cubit is no longer available for macOS.

For many years we constructed Cubit meshes using journal files while leveraging the Cubit APREPRO scripting language. The Cubit Python interface has improved over this time, and with the use of Python scripting with Gmsh, we have replaced some of the Cubit journal files with Python scripts. In a Python script, you can run any Cubit command

using the `cubit.cmd()` function; however, for a more genuine Python experience, we recommend using the available Python functions. Using Python scripts provides access to the complete programming language, making it easier to read in information from external files (for example, fault traces).

Sidesets

New in v5.0.0.

PyLith supports reading sidesets for boundary conditions, fault interfaces, and output on surfaces. We strongly recommend using sidesets rather than nodesets.

Complex fault surfaces

Cubit has limited support for embedding fault surfaces into a volume domain. We have found that Cubit will sometimes generate a suitable mesh with a single embedded fault, but usually it fails to generate a mesh when we have two embedded faults. The workaround is to extend each fault surface to the edges of a volume (refer to `examples/crustal-strikeslip-3d` for an example).

Python scripting

In the PyLith examples, when we generate a Cubit mesh we create a Python script `generate_cubit.py`. The script is setup to run from within Cubit or externally. We usually write the script in an external editor with Python language support and then copy and paste commands into the Cubit Python command panel.

Useful Cubit functions

The Cubit Python API is available in an appendix of the manual. For example, the Cubit Python API for version 16.16 is available in [CubitInterface Namespace](#). s `cubit.cmd(CMD: str)` : Run Cubit command CMD.

`cubit.get_last_id(entity_type: str)`

Get id of last entity create of type `entity_type`.

`cubit.get_id_from_name(name: str)`

Get id of enty with name `name`.

`cubit.get_id_string(entity_ids: list)`

Get string with list of ids for list of entity ids `entity_ids`.

`cubit.get_idless_signature(entity_type: str, id: int)`

Get the idless signatude of a Cubit entity type `entity_type` with id `id`.

`cubit.vertex(id: int)`

Get vertex given entity id `id`.

`cubit.curve(id: int)`

Get curve given entity id `id`.

`cubit.surface(id: int)`

Get surface given entity id `id`.

`cubit.volume(id: int)`

Get volume given entity id `id`.

`cubit.body(id: int)`

Get body given entity id `id`.

Gmsh

We only recently started using Gmsh and have only used the [Python interface](#). Gmsh also offers a simple scripting language, similar to Cubit journal files. The Gmsh Python interface integrates well with the rest of Python; it can be installed so that it is compatible with the Python interpreter used by PyLith. This means one can leverage additional Python packages, such as geographic projection libraries. Gmsh includes its own geometry engine as well as an interface to the OpenCASCADE engine.

Warning

Gmsh does not construct quadrilateral or hexahedral meshes directly; instead, it first constructs a triangular or tetrahedral mesh and then combines triangles or tetrahedra to form quadrilaterals or hexahedra.

Sometimes it will not be able to remove all triangular or hexahedral cells, resulting in meshes with multiple shapes, which PyLith does not support.

Gmsh tips

The function `gmsh.model.geo.synchronize()` copies entities from the geometry engine (built in or OpenCASCADE) into the model. You need to call this function after you create geometry and *before* you attempt to use it in the model.

Built-in CAD engine

This is a list of some of the more useful functions in the built-in CAD engine. Some functions include optional parameters that are not shown; refer to the Python API documentation for complete specification of the API.

`gmsh.model.geo.add_point(x: float, y: float, z: float) -> int`

Create a point at (x, y, z) and return the tag of the point.

`gmsh.model.geo.add_line(startTag: int, endTag: int) -> int`

Create a line from point `startTag` to point `endTag` and return the tag of the line.

`gmsh.model.geo.add_spline(pointTags: list) -> int`

Create a spline going through the points in `pointTags` and return the tag of the spline.

`gmsh.model.geo.add_curve_loop(curveTags: list) -> int`

Create a curve loop formed by the curves `curveTags` and return the tag of the curve loop.

`gmsh.model.geo.add_plane_surface(wireTags: list) -> int`

Create a planar surface defined by one or more curve loops `wireTags` and return the tag of the surface. The first curve loop defines the exterior boundary, and any additional curve loops define holes. If the curve loops are not on a plane, no surface will be created.

`gmsh.model.geo.add_surface_filling(wireTags: list) -> int`

Create a surface, filling the curve loops in `wireTags` using transfinite interpolation, and return the tag of the surface. Only a single curve loop is supported and it must contain 3 or 4 curves.

`gmsh.model.geo.add_surface_loop(surfaceTags: list) -> int`

Create a surface loop formed by `surfaceTags` and return the tag of the surface.

`gmsh.model.geo.add_volume(shellTags: list) -> int`

Create a volume formed by `shellTags` and return the tag of the volume. The first surface loop defines the exterior boundary, and any additional surface loops define holes.

`gmsh.model.geo.extrude(dimTags: list, dx: float, dy: float, dz: float) -> list`

Extrude the entities `dimTags` (list of (dim, tag) tuples) by translating the entities along (dx, dy, dz) and returning a list of (dim, tag) tuples of the newly created entities.

gmsh.model.geo.translate(dimTags: list, dx: float, dy: float, dz: float)

Translate the entities dimTags (list of (dim, tag) tuples) along (dx, dy, dz).

gmsh.model.geo.remove_all_duplicates()

Remove all duplicate entities.

gmsh.model.geo.split_curve(tag, pointTags) -> list

Split the curve tag (line, spline, or b-spline) at the specified points pointTags and return a list of the newly created curves.

OpenCASCADE geometry engine

gmsh.model.occ.add_point(x: float, y: float, z: float) -> int

Create a point at (x, y, z) and return the tag of the point.

gmsh.model.occ.add_line(startTag: int, endTag: int) -> int

Create a line from point startTag to point endTag and return the tag of the line.

gmsh.model.occ.add_spline(pointTags: list) -> int

Create a spline going through the points in pointTags and return the tag of the spline.

gmsh.model.occ.add_curve_loop(curveTags: list) -> int

Create a curve loop formed by the curves curveTags and return the tag of the curve loop.

gmsh.model.occ.add_rectangle(x, y, z, dx, dy) -> int

Create a rectangle with lower left corner at (x, y, z) and upper right corner at (x+dx, y+dy, z) and return the tag of the rectangle.

gmsh.model.occ.add_plane_surface(wireTags: list) -> int

Create a planar surface defined by one or more curve loops wireTags and return the tag of the surface. The first curve loop defines the exterior boundary, and any additional curve loops define holes. If the curve loops are not on a plane, no surface will be created.

gmsh.model.occ.add_surface_filling(wireTag: int, pointTags: list=[]) -> int

Create a surface, filling the curve loop wireTag while going through points pointTags and return the tag of the surface.

gmsh.model.occ.add_bspline_surface(pointTags: list, numPointsU: int) -> int

Create a b-spline surface with control points pointTags (list with numPointsU*numPointsV points) and return the tag of the surface.

gmsh.model.occ.add_trimmed_surface(surfaceTag: int, wireTags: list=[]) -> int

Trim the surface surfaceTag with the wires wireTags and return the tag of the trimmed surface. The first wire defines the external contour, and any additional wires define holes.

gmsh.model.occ.add_surface_loop(surfaceTags: list) -> int

Create a surface loop formed by surfaceTags and return the tag of the surface.

gmsh.model.occ.add_volume(shellTags: list) -> int

Create a volume formed by shellTags and return the tag of the volume. The first surface loop defines the exterior boundary, and any additional surface loops define holes.

gmsh.model.occ.add_sphere(x: float, y: float, z: float, radius: float) -> int

Add a sphere centered at (x, y, z) with radius radius and return the tag of the sphere.

gmsh.model.occ.add_box(x: float, y: float, z: float, dx: float, dy: float, dz: float) -> int

Add a prism with a corner at (x, y, z) and extent (dx, dy, dz) and return the tag of the sphere.

gmsh.model.occ.extrude(dimTags: list, dx: float, dy: float, dz: float) -> list

Extrude the entities dimTags (list of (dim, tag) tuples) by translating the entities along (dx, dy, dz) and returning a list of (dim, tag) tuples of the newly created entities.

gmsh.model.occ.translate(dimTags: list, dx: float, dy: float, dz: float)

Translate the entities dimTags (list of (dim, tag) tuples) along (dx, dy, dz).

gmsh.model.occ.fuse(objectDimTags: list, toolDimTags: list) -> tuple

Compute the union of entities objectDimTags (list of (dim, tag) tuples) and toolDimTags (list of (dim, tag) tuples). The return value is a tuple with the a list of (dim, tag) tuples of the objects created and a mapping from the origin entities to the new entities.

gmsh.model.occ.intersect(objectDimTags: list, toolDimTags: list) -> tuple

Compute the intersection of entities objectDimTags (list of (dim, tag) tuples) and toolDimTags (list of (dim, tag) tuples). The return value is a tuple with the a list of (dim, tag) tuples of the objects created and a mapping from the origin entities to the new entities.

gmsh.model.occ.cut(objectDimTags: list, toolDimTags: list) -> tuple

Compute the difference of entities objectDimTags (list of (dim, tag) tuples) and toolDimTags (list of (dim, tag) tuples). The return value is a tuple with the a list of (dim, tag) tuples of the objects created and a mapping from the origin entities to the new entities.

gmsh.model.occ.fragment(objectDimTags: list, toolDimTags: list) -> tuple

Compute the boolean fragments resulting from the intersection of entities objectDimTags (list of (dim, tag) tuples) and toolDimTags (list of (dim, tag) tuples). The return value is a tuple with the a list of (dim, tag) tuples of the objects created and a mapping from the origin entities to the new entities.

gmsh.model.occ.remove_all_duplicates()

Remove all duplicate entities.

Troubleshooting

Using the Python debugger

You can use the Python debugger to have a Python script break at a specific point, so you can inspect the current state. For example, insert `import pdb; pdb.set_trace()` at the desired breakpoint.

Missing geometry

Did you forget to call `gmsh.model.geo.synchronize()` after creating the geometry?

PETSc error when reading mesh

When a curve is not embedded in a surface or a surface is not embedded in a volume, Gmsh will generate independent meshes on each entity. This will generate the following error when the mesh is read via the MeshIOPetsc mesh importer.

```
[0]PETSC ERROR: ----- Error Message -----
↪-----
[0]PETSC ERROR: No support for this operation for this object type
[0]PETSC ERROR: Could not determine Plex facet for Gmsh element 23 (Plex cell 48)

-- lines omitted --

[0]PETSC ERROR: #1 DMPlexCreateGmsh() at petsc-dev/src/dm/impls/plex/plexgmsh.c:1766
[0]PETSC ERROR: #2 DMPlexCreateGmshFromFile() at petsc-dev/src/dm/impls/plex/plexgmsh.
↪c:1470
[0]PETSC ERROR: #3 DMPlexCreateFromFile() at petsc-dev/src/dm/impls/plex/plexcreate.
↪c:5889
[0]PETSC ERROR: #4 DMPlexCreateFromOptions_Internal() at petsc-dev/src/dm/impls/plex/
```

(continues on next page)

(continued from previous page)

```
↪plexcreate.c:4001
[0]PETSC ERROR: #5 DMSetFromOptions_Plex() at petsc-dev/src/dm/impls/plex/plexcreate.
↪c:4418
[0]PETSC ERROR: #6 DMSetFromOptions() at petsc-dev/src/dm/interface/dm.c:914
[0]PETSC ERROR: #7 virtual void pylith::meshio::MeshIOPetsc::_read()() at src/cig/pylith/
↪libsrc/pylith/meshio/MeshIOPetsc.cc:147
```

gmsh_utils

In Gmsh we use physical groups to associate cells with materials and mark entities for boundary conditions and faults. The names of physical groups for materials must follow the syntax `material-id:TAG`, where TAG is the tag of the physical group. PyLith includes a Python module `pylith.meshio.gmsh_utils` to make it easy to generate a PyLith compatible Gmsh file.

The function `create_material()` generates physical groups following the required naming convention of `material-id:TAG` given the tag and names of entities. Similarly, the function `create_group()` will construct physical groups for boundary conditions and faults compatible with PyLith.

The physical groups for boundary conditions and faults must include entities at the topological dimension of the boundary condition as well as all lower dimensions. For example, for a boundary condition on curves a physical group must include the entities on the curves as well as the vertices defining the curves. For a boundary condition on surfaces a physical group must include the entities on the surfaces as well as the curves and vertices defining the surfaces.

GenerateMesh Application Template

The `gmsh_utils` module also includes a application template object (Python abstract base class) called `GenerateMesh` for writing Python scripts that generate meshes using Gmsh. The application template defines the steps for generating the mesh with a separate function (to be implemented by the user) for each step:

1. `initialize()`: Initialize Gmsh;
2. `create_geometry()`: Create the geometry (implemented in user application);
3. `mark()`: Create physical groups for boundary conditions, faults, and materials (implemented in user application);
4. `generate_mesh()`: Generate the finite-element mesh (implemented in user application);
5. `write()`: Save the mesh to a file; and
6. `finalize()`: Start the Gmsh graphical user interface, if requested, and then finalize Gmsh.

The command line arguments specify which step(s) to run, the output filename, and whether to invoke the Gmsh graphical user interface upon completing the steps:

```
--geometry
    Generate the geometry by calling create_geometry().

--mark
    Create physical groups by calling mark().

--generate
    Generate the mesh by calling generate_mesh().

--write
    Save the mesh by calling write().

--name
    Name of the mesh in Gmsh (default="mesh").
```



```
--filename=FILENAME
    Name of output mesh file (default="mesh.msh").

--ascii
    Write mesh to ASCII file (default is binary).

--cell=[tri,quad,tet,hex]
    Generate mesh with specified cell type.

--gui
    Start the Gmsh graphical user interface after running steps.
```

The application template always calls the `initialize()` and `finalize()` methods. Additionally, the application will run any prerequisite steps. For example, specifying `--generate` will trigger creating the geometry and physical groups before generating the mesh.

The application is discussed in more detail in the examples.

MaterialGroup

`MaterialGroup` is a Python data class that holds information about a physical group associated with a material. The data members include:

```
tag (int)
    Integer tag for the physical group.

entities (list)
    List (array) of entities for the material.
```

The `MaterialGroup` data class include a method `create_physical_group()` that will create a physical group from the information in the `MaterialGroup`.

BoundaryGroup

New in v5.0.0.

`BoundaryGroup` (formerly `VertexGroup`) is a Python data class that holds information about a physical group associated with a boundary or fault. The data members include:

```
name (str)
    Name for the physical group.

tag (int)
    Integer tag for the physical group.

dim (int)
    Dimension of the entities (0=points, 1=curves, 2=surfaces)

entities
    List (array) of entities for the boundary condition or fault.
```

The `BoundaryGroup` data class include a method `create_physical_group()` that will create a physical group from the information in the `BoundaryGroup`. The `create_physical_group()` method has an argument `recursive` with a default value of `False`; this will create a physical group with entities passed as an argument. Using `recursive=True` will create a physical group with the entities passed as an argument as well as lower dimension entities; this was required in PyLith before version 5.0.0.

4.2.6 Defining Simulations

Types of Simulations

PyLith currently supports two types of problems:

- Time dependent, and
- Static Green's functions.

Time-Dependent Problem (`TimeDependent`)

This type of problem solves a time-dependent boundary value problem and applies to static, quasistatic, and dynamic simulations. We use the PETSc object to manage the time-stepping, including the selection of the time-stepping algorithm. By default PyLith uses the backward Euler time-stepping algorithm.

Pyre User Interface

See *`TimeDependent Component`* for Pyre properties and facilities and configuration examples.

Initial Conditions

New in v3.0.0.

The initial conditions for a simulation are specified via a combination of initial values for the solution and initial values for state variables. The initial values for state variables are specified via the spatial databases for the auxiliary field of each material. In this section we discuss how to set the initial values of the solution field.

`InitialConditionDomain`

We use this object when we want to specify the initial values of solution subfields across the entire domain using a single spatial database.

Pyre User Interface

See *`InitialConditionDomain Component`* for Pyre properties and facilities and configuration examples.

`InitialConditionPatch`

We use this object when we want to specify the initial values of solution subfields across patches of the domain defined by materials.

Important

Initial conditions over a patch currently only work for Gmsh input files. Exodus II files from Cubit do not contain the information needed by the current PyLith implementation.

In creating the physical group in Gmsh, you must include the cells and all lower dimension entities (faces, edges, and vertices). The easiest way to do this is to use the `VertexGroup` provided in the `pylith.meshio.gmsh_utils` Python module. By default, the lower dimension entities will be included in the physical group.

i Pyre User Interface

See *InitialConditionPatch Component* for Pyre properties and facilities and configuration examples.

Numerical Damping in Explicit Time Stepping**⚠ Danger**

Not yet reimplemented in v3.x.

Green's Functions Problem (GreensFns)

This type of problem applies to computing static Green's functions for elastic deformation. The `GreensFns` problem loops over a suite of fault slip impulses and computes the static solution for each impulse using the linear solver. In the output files, the deformation at each "time step" is the deformation for a different slip impulse. The fault slip impulses are specified using `FaultCohesiveImpulses` for the fault. See *Fault Slip Impulses (FaultCohesiveImpulses)* for more information.

⚠ Warning

The `GreensFns` problem generates slip impulses on a fault. PyLith currently requires that impulses be applied to a single fault of type `FaultCohesiveImpulses`.

i Pyre User Interface

See *GreensFns Component* for Pyre properties and facilities and configuration examples.

Finite-Element Mesh

The finite-element mesh specifies the geometry and topology of the discretization of the domain. It must be generated using external software before running PyLith. PyLith supports triangular and quadrilateral cells in 2D and tetrahedral and hexahedral cells in 3D. PyLith does not support meshes with mixtures of cell shapes (meshes with triangular and quadrilateral cells in 2D or meshes with tetrahedral, hexahedral and wedges in 3D). The vertex ordering must follow the convention shown in *Fig. 4.12* and *Fig. 4.13*. The cells define the geometry of the domain; the basis order and quadrature order used to discretize the solution subfields are specified separately.

The mesh information specifies the vertex coordinates and the vertices composing each cell in the mesh. The mesh information must also define at least one boundary for which displacement (Dirichlet) boundary conditions will be provided. In most simulations, there will be several groups of faces (or vertices) for boundary conditions, each with a unique identifying label. For example, one group might define a surface of the mesh where displacement (Dirichlet) boundary conditions will be applied, another might define a surface where traction (Neumann) boundary conditions will be applied, while a third might specify a surface that defines a fault. Similarly, the mesh information contains cell labels that define the material type for each cell in the mesh. For a mesh with a single material type, there will only be a single label for every cell in the mesh. See *Materials* and *Boundary Conditions* for more detailed discussions of setting the materials and boundary conditions.

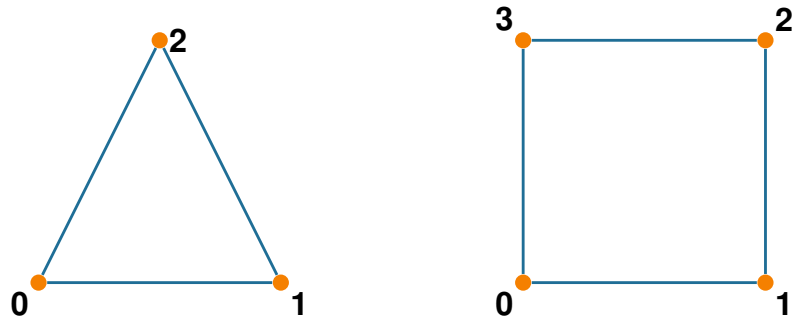


Fig. 4.12: Cells available for 2D problems are the triangle and the quadrilateral.

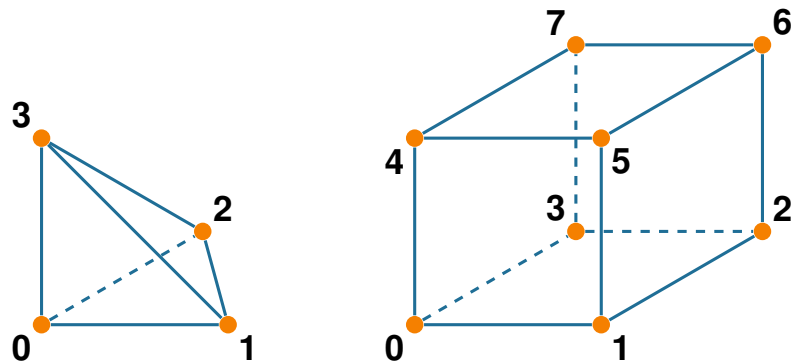


Fig. 4.13: Cells available for 3D problems are the tetrahedron and the hexahedron.

Mesh Initialization

The `Initializer` object manages initializing the mesh, which involves reading it in from a file and performing some pre-processing operations. The phases (steps) in the initialization process depend on whether the mesh can be read in parallel or it must be read in serial.

Pyre User Interface

See *Initializer component*

Serial Phases

Phases for reading in a mesh in serial.

1. Read mesh
2. Reorder mesh
3. Distribute mesh
4. Insert fault interfaces
5. Refine mesh

i Pyre User InterfaceSee *Serial component***Parallel Phases**

Phases for reading in a mesh in parallel.

1. Read mesh
2. Distribute mesh (fine tune distribution)
3. Insert fault interfaces
4. Refine mesh

i Pyre User InterfaceSee *Parallel component***Initialization Phases****MeshReader**

Read a finite-element mesh from a file.

MeshWriter

Write a finite-element mesh to a file.

MeshReordering

Reorder the cells and vertices of a finite-element mesh.

MeshDistributor

Distribute a finite-element mesh among processes.

MeshInsertInterfaces

Insert cohesive cells for faults.

MeshRefine

Refine a finite-element mesh.

MeshReader and MeshWriter

PyLith supports reading meshes generated by Cubit (Exodus II files) and Gmsh, as well as mesh files converted to HDF5 files in the PETSc mesh format. We have implemented our own readers for Exodus II files. For Gmsh and HDF5 files we use the reader included in PETSc; PETSc also supports several other formats, but they have not been tested for use with PyLith. Mesh vertices or faces can be associated with boundary conditions, fault interfaces, or output by creating groups of vertices or faces. Materials are marked by the same label `material-id` with different label values for each material. The PyLith readers create the labels from the mesh input files. Refer to *Mesh generation with Cubit and Gmsh* for information about generating finite-element meshes using Cubit and Gmsh and converting Exodus II files from Cubit and Gmsh files to HDF5 files in the PETSc format.

ii Important

Marking faces (edges in 2D), which is a new capability in version 5.0, is preferred. Marking vertices is deprecated and will be removed in version 6.0.

i Pyre User Interface*MeshReader Component**MeshWriter Component***ASCII Mesh Files - MeshIOAscii**

The `MeshIOAscii` object is intended for reading small, simple ASCII files containing a mesh constructed by hand. We use this file format in small tests. *PyLith Mesh ASCII File* describes the format of the files.

Table 4.15: Translation of ASCII mesh ‘tags’ to PyLith mesh ‘label’ and ‘label_value’.

MeshIOAscii entity	label	label_value
material-ids	material-id (hardwired)	value
Group name	name	1 (default)

i Pyre User Interface*MeshIOAscii Component***Cubit (Exodus II) Mesh Files - MeshIOCubit**

The `MeshIOCubit` object reads the NetCDF Exodus II files output from Cubit. Beginning with Cubit 11.0, the names of the nodesets are included in the Exodus II files and PyLith can use these nodeset names or revert to using the nodeset ids.

Table 4.16: Translation of Cubit mesh ‘tags’ to PyLith mesh ‘label’ and ‘label_value’.

Cubit entity	label	label_value
Material block	material-id (hardwired)	Block value
Nodeset	Nodeset name	1 (default)
Sideset	Sideset name	1 (default)

i Pyre User Interface*MeshIOCubit Component***⚠ Warning**

There are two versions of Cubit: Sandia National Laboratory provides a version to U.S. government agencies, and Coreform provides another version to all other users. The two versions used to be essentially the same, but the differences have started to grow. We strive to provide Cubit Journal scripts that work with both versions without modification, but this is becoming more difficult.

Please be aware that we cannot guarantee that all Cubit Journal files will work with all versions of Cubit. You may need to make small adjustments (usually updating geometry ids) to get them to work with the version of Cubit you are using.

Cubit is no longer supported on macos, so we are phasing out use of Cubit in PyLith examples.

Gmsh and HDF5 Files - MeshIOPetsc

The MeshIOPetsc object supports reading a variety of mesh formats. We have only thoroughly tested this interface using Gmsh files and HDF5 files in the PETSc mesh format. Refer to [pylith_convertmesh](#) for how to convert an Exodus II file from Cubit or a Gmsh file to an HDF5 file in the PETSc mesh format.

Table 4.17: Translation of Gmsh mesh ‘tags’ to PyLith mesh ‘label’ and ‘label_value’.

Gmsh entity	label	label_value
Material physical groups	material-id (hardwired)	tag
Boundary condition physical groups	Physical group name	tag

Important

A Gmsh file must end in .msh for the reader to recognize that it is a Gmsh file.

The HDF5 file must end in .h5 for the reader to recognize that it is an HDF5 file.

Tip

You can view the mesh quality in Gmsh using Tools→Statistics. We prefer the condition number quality metric, which Gmsh provides as SICN (signed inverse of the condition number). Click on 3D next to SICN to color the cells by mesh quality. Click on Plot to view the cumulative distribution of the metric over the cells.

Pyre User Interface

MeshIOPetsc Component

MeshReordering

Reordering the mesh so that vertices and cells connected topologically reside close together in memory improves overall performance. We use the revere Cuthill-McKee algorithm, which is available in PETSc, for reordering.

MeshDistributor

The distributor uses a partitioner to compute which cells should be placed on each process, computes the overlap among the processes, and then distributes the mesh among the processes. The default partitioner is METIS/ParMETIS.

Pyre User Interface

Distributor Component

MeshInsertInterfaces

Insert fault interfaces into the mesh by creating cohesive cells. Refer to *Fault Interface Conditions* for more information.

MeshRefine

Refining the mesh adjusts the discretization size for a specific simulation. The typical use case is uniformly refining a mesh to decrease the discretization size by the same amount over the entire domain.

RefineUniform

This refiner is used to decrease node spacing by a power of two by recursively subdividing each cell by a factor of two. In a 2D triangular mesh a node is inserted at the midpoint of each edge, splitting each cell into four cells (see Fig. 4.14). In a 2D quadrilateral mesh a node is inserted at the midpoint of each edge and at the centroid of the cell, splitting each cell into four cells. In a 3D tetrahedral mesh a node is inserted at the midpoint of each edge, splitting each cell into eight cells. In a 3D hexahedral mesh a node is inserted at the midpoint of each edge, the centroid of each face, and at the centroid of the cell, splitting each cell into eight cells.

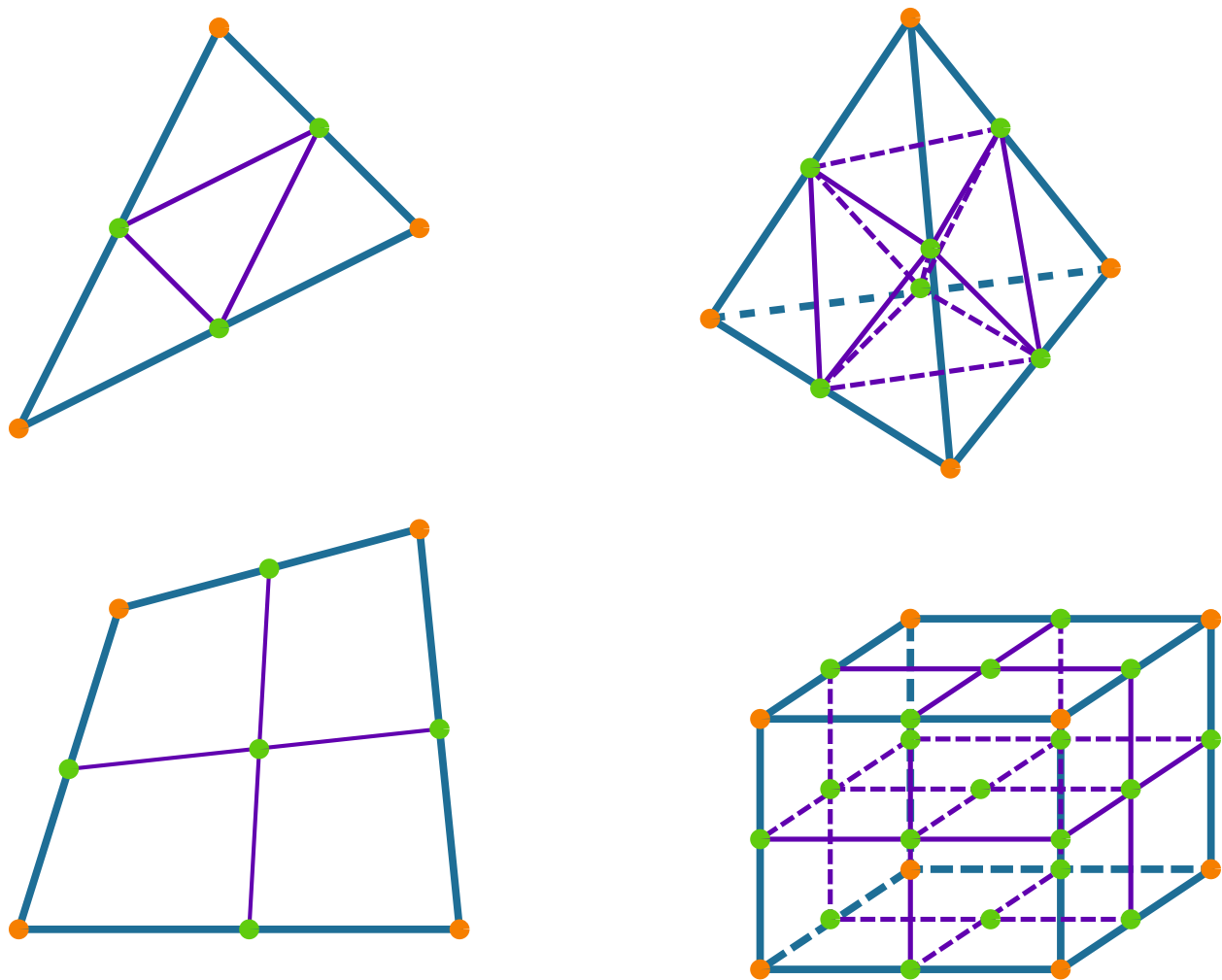


Fig. 4.14: Global uniform mesh refinement of 2D and 3D linear cells. The blue lines and orange circles identify the edges and vertices in the original cells. The purple lines and green circles identify the new edges and vertices added to the original cells to refine the mesh by a factor of two.

Refinement occurs after distribution of the mesh among processes. This allows one to run much larger simulations by (1) permitting the mesh generator to construct a mesh with a node spacing larger than that needed in the simulation and (2) operations performed in serial during the simulation setup phase, such as, adjusting the topology to insert cohesive cells and distribution of the mesh among processors uses this much smaller coarse mesh. For 2D problems the global mesh refinement increases the maximum problem size by a factor of 4^n , and for 3D problems it increases the maximum problem size by a factor of 8^n , where n is the number of recursive refinement levels. For a tetrahedral mesh, the element quality decreases with refinement so n should be limited to 1-2.

Nondimensionalization

Nondimensionalization permits solving a system equations across a vast range of scales. We are primarily interested in elasticity-related boundary value problems, so we define displacement, length, rigidity, time, and temperature scales.

See also

In the governing equations section, we discuss the nondimensionalization of each governing equation.

- *Elasticity*
- *Incompressible elasticity*
- *Poroelectricity*

Scales

The Python `General` object is the generic interfaces to the underlying C++ `Scales` object that manages the scales used to nondimensionalize values.

Tip

The default nondimensionalization is reasonable for many problems; however, it may be necessary to change the default values in some cases.

QuasistaticElasticity

`QuasistaticElasticity` provides convenient scales for nondimensionalizing quasistatic elasticity and incompressible elasticity boundary value problems. The scales are specified in terms of the displacement scale, length scale, shear modulus, and time scale.

`displacement_scale`

Nominal size of the displacement

`length_scale`

Nominal size of the geometry controlling the deformation (for example, fault dimension or size of the domain)

`shear_modulus`

Nominal shear modulus

`time_scale`

Time scale of deformation (for example, viscoelastic relaxation time)

Table 4.18: Nondimensional scales for quasistatic elasticity boundary value problems.

Scale	Name	Default Value
length	displacement_scale	1 m
length	length_scale	100 km
pressure	shear_modulus	10 GPa
time	time_scale	100 years

Pyre User Interface

See *QuasistaticElasticity component*.

DynamicElasticity

DynamicElasticity provides convenient scales for nondimensionalizing dynamic elasticity boundary values problems. The scales are specified in terms of the displacement scale, length scale, shear modulus, and shear wave speed. The time scale is computed based on the time it takes a shear wave (v_s) to propagate the length scale (l_o).

$$t_o = l_o / v_s$$

displacement_scale

Nominal size of the displacement

length_scale

Nominal size of the geometry controlling the deformation (for example, fault dimension or size of the domain)

shear_modulus

Nominal shear modulus

shear_wave_speed

Nominal shear wave speed

Table 4.19: Nondimensional scales for dynamic elasticity boundary value problems.

Scale	Name	Default Value
length	displacement_scale	1 m
length	length_scale	100 km
pressure	shear_modulus	10 GPa
—	shear_wave_speed	3 km/s

Pyre User Interface

See *DynamicElasticity component*.

QuasistaticPoroelasticity

QuasistaticPoroelasticity provides convenient scales for nondimensionalizing quasistatic poroelasticity boundary value problems. The scales are specified in terms of the displacement scale, length scale, shear modulus, viscosity,

and permeability. The time scale is computed based on the length scale (l_o), rigidity (μ_o), viscosity (μ_{f_o}), and permeability (k_o).

$$t_o = \frac{\mu_{f_o}}{k_o} \frac{l_o^2}{\mu_o}$$

displacement_scale

Nominal size of the displacement

length_scale

Nominal size of the geometry controlling the deformation (for example, fault dimension or size of the domain)

shear_modulus

Nominal shear modulus

viscosity

Nominal viscosity

permeability

Nominal permeability

Table 4.20: Nondimensional scales for quasistatic poroelasticity boundary value problems.

Scale	Name	Default Value
length	displacement_scale	1 m
length	length_scale	100 km
pressure	shear_modulus	10 GPa
–	viscosity	0.001 Pa-s
–	permeability	1.0e-13 m ²

Pyre User Interface

See *QuasistaticPoroelasticity component*.

Output

PyLith produces four kinds of output:

- Information written to stdout (default) or other devices that is controlled by the Pyre journal settings;
- Information written to stdout that is controlled by PETSc options;
- Information about the progress of the simulation written to a text file that is controlled by progress monitors (see *Progress Monitors*);
- Output of the solution over the domain, external boundary, or at discrete points (see *Solution Observers*); and
- Output of the solution or state variables over a material, interface, or external boundary (see *Physics Observer*).

Journal

New in v5.0.0.

The Pyre journal module controls information written to stdout (terminal window). The default PyLith behavior is to show the application-flow info journal channel and all warnings, errors, and firewalls. Errors and firewalls will throw Python or C++ exceptions, which will cause PyLith to abort.

Listing 4.28: A Python or C++ call to a journal channel writes a message to stdout (terminal window). The first line contains `FILE:LINE:FUNCTION` in which the call to journal originates in function `FUNCTION` at line `LINE` in file `FILE`. The second line contains `SEVERITY (CHANNEL)` in which `SEVERITY` is `info`, `debug`, `warning`, `error`, or `firewall` and `CHANNEL` is the name of the journal channel. The third and any subsequent lines contain the journal message.

```
>> FILE:LINE:FUNCTION
-- SEVERITY (CHANNEL)
-- MESSAGE
```

Devices

The examples set the journal device to `color-console`, which will use colors to highlight journal information sent to the terminal window (console).

Listing 4.29: Setting the journal device to `color-console` to use colors to highlight journal information sent to the terminal window (console).

```
[pylithapp.journal]
device = color-console

# Color scheme for use on terminals with a dark background (default)
device.renderer.color-scheme = dark-bg

# Light color scheme for use on terminals with a light background
device.renderer.color-scheme = light-bg
```

Tip

If you do not see colors in your journal output but see escape characters and other garbled text in the journal output, you likely need to set your `TERM` environment variable. We recommend setting `TERM` to `xterm-256color` in your environment (Bash: `export TERM=xterm-256color`).

Channels

You can enable and disable journal channels by setting their state.

Listing 4.30: Enabling and disabling journal channels in a `PyLithApp` parameter file.

```
[pylithapp]
# Disable application-flow info channel; this is a special case.
show_application_flow = False

[pylithapp.journal]
# Enable 'debug-parameters' info channel.
info.debug-parameters = True

# Turn off deprecation warnings (not recommended).
warning.deprecation = False
```

Within PyLith, we use journals with predefined names to allow the user to select how much information is written to stdout (terminal window) at runtime.

Info

- `about`: General information about application.
- `application-flow`: High-level information simulation status.
- `application-flow-detail-3`: Additional details about simulation status.
- `debug-parameters`: Detailed reporting of simulation parameters being set in C++ code.

Warning

- `user-input`: Warnings related to user input.
- `deprecation`: Warnings about using deprecated features.

Error

- `user-input`: Errors related to user input.
- `external`: Errors related to external libraries.
- `output`: Errors related to output.

Firewall

- `internal`: Internal error.
- `logic`: Internal logic error.

Debug

- `auxiliary-fields`: Show auxiliary fields.
- `solution`: Show solution field.
- `residual`: Show residual.
- `mms-test`: Show debugging information for MMS test.
- `full-scale-test`: Show debugging information for full-scale test.
- `integration-kernels`: Show integration kernels.
- `solver`: Show solver information.
- `mesh`: Show mesh information.
- `mesh-detail-3`: Show additional mesh information.
- `mesh-detail-5`: Show detailed mesh information.

Progress Monitors

The progress monitors make it easy to monitor the general progress of long simulations, especially on clusters where stdout is not always easily accessible. The progress monitors update a simulation's current progress by writing information to a text file. The information includes time stamps, percent completed, and an estimate of when the simulation will finish.

ProgressMonitorTime

This is the default progress monitor for time-stepping problems. The monitor calculates the percent completed based on the time at the current time step and the total simulated time of the simulation, not the total number of time steps (which may be unknown in simulations with adaptive time stepping).

➡ See also

ProgressMonitorTime Component

ProgressMonitorStep

This is the default progress monitor for problems with a specified number of steps, such as Green's function problems. The monitor calculates the percent completed based on the number of steps (e.g., Green's function impulses completed).

➡ See also

ProgressMonitorStep Component

Observers

Observer objects manage transferring the solution and state variables to other objects, including output and external software. Currently, the only type of observers implemented in PyLith are ones that produce output.

Output Observers

PyLith currently supports output to HDF5/Xdmf and VTK files, which can be imported directly into a number of visualization tools, such as ParaView, Visit, and MayaVi. The HDF5 files can also be directly accessed via Matlab and Python modules, such as h5py. PyLith v3.x supports output of the solution subfields, all auxiliary fields (material properties, boundary condition parameters, and fault interface parameters, etc.) and fields derived from the auxiliary field and/or solution, such as strain and stress.

The HDF5 writer provides parallel binary output, whereas the VTK writer provides serial ASCII output. Additionally, with the VTK writer each time step is written to a separate file; the HDF5 writer puts all information for a given domain, boundary condition, or fault interface into a single file.

Output observers have a data writer (see [Data Writers](#)) for writing the data in a specified format and a trigger (see [Output Triggers](#)) for specifying how often to write the output in a time-dependent simulation.

📌 Important

Fields with a basis order of 1 are written as vertex fields, whereas fields with a basis order of 0 are written as cell fields. Fields with a basis order of 0 are kept at a basis order of 0 when output. Fields with a basis order of 1 or more can be output with a basis order of 0 or 1.

New in v4.0.0.

The output observers produce files with either diagnostic information (info files) or solution and state variable information (data files). The default behavior is the files include all available information. The info files include the auxiliary subfields at the beginning of the simulation along with surface orientation information for faults and boundary conditions. The data files include all solution subfields, state variables, and derived fields (fields computed from the solution, such as Cauchy stress and strain).

For boundary conditions the orientation information is provided in terms of x, y, and z components of the unit vectors for the surface normal and tangential directions. In 3D the “vertical” tangential direction is the cross product of the surface normal and horizontal tangential direction; it is in the +z direction for a vertical boundary. In the case of the horizontal boundary, the horizontal tangential direction is in the +x direction and the “vertical” tangential direction is in the +y direction. For a fault surface the horizontal tangential direction generally corresponds to the along-strike direction and the “vertical” tangential direction generally corresponds to the up-dip direction; the exception is a 2D simulation for a vertical cross-section in which the “horizontal” tangential direction corresponds to the dip direction.

The orientation information is useful for transforming from components written in terms of a surface (for example, left lateral, reverse, and opening fault tractions), into the model coordinate system (for example, the global Cartesian coordinate system). Given unit vector components n_x, n_y, n_z (normal direction), h_x, h_y, h_z (horizontal tangential direction or along-strike direction), and v_x, v_y, v_z (vertical tangential direction or up-dip direction), we can transform a vector in the boundary (or fault) coordinate system (a_n, a_h, a_v) into the global coordinate system using

$$a_x = a_n n_x + a_h h_x + a_v v_x \quad (4.438)$$

$$a_y = a_n n_y + a_h h_y + a_v v_y \quad (4.439)$$

$$a_z = a_n n_z + a_h h_z + a_v v_z \quad (4.440)$$

This transformation is useful for plotting fault tractions and slip in 3D visualization tools such as ParaView that require vectors in the model coordinate system.

New in v4.2.0.

Output observers can output information on a finer resolution mesh than the one used in the simulation. This is useful when using a basis order of 2 or greater for the discretization of the solution and writing output at a basis order of 1. In this case, output using a mesh that is 4-8 times finer and fields discretized using a basis order of 1 (usually required by visualization tools) will do a good job representing the fields with a basis order of 2 on the mesh used in the simulation. This feature is used in [examples/strikeslip-2d/step04-varslip](#).

Solution Observers

The solution observers get notified of updates to the solution. [Table 4.21](#) lists the current implementations of solution observers, which are used for output.

Table 4.21: Solution observers.

Object	Use Cases
OutputSoln	Output of the solution over the domain
OutputSolnBoundary	Output of the solution over an external boundary
OutputSolnPoints	Output of the solution at discrete points

See also

[OutputSoln Component](#), [OutputSolnBoundary Component](#), and [OutputSolnPoints Component](#).

Output at Discrete Points

In many situations with recorded observations, one would like to extract the solution at the same locations as the recorded observation. The locations are specified in a text file.

PointsList Reader

This object corresponds to a simple text file containing a list of points (one per line) where output is desired. See *Points List File* for file format specifications. The points are specified in the coordinate system specified by `OutputSolnPoints`. The coordinates will be transformed into the coordinate system of the mesh prior to interpolation.

See also

PointsList Component

Physics Observer

Analogous to the `OutputSoln` objects, which provide a means to output the solution, the physics objects (material, boundary conditions, and fault interfaces) have `OutputPhysics` objects to provide output of the solution, properties, state variables, etc.

Data Writers

HDF5 Output

HDF5 files provide a flexible framework for storing simulation data with datasets in groups logically organized in a tree structure analogous to files in directories. HDF5 output offers parallel, multi-dimensional array output in binary files, so it is much faster and more convenient than the VTK output which uses ASCII files and separate files for each time step. Standards for organizing datasets and groups in HDF5 files do not exist for general finite-element software in geodynamics. Consequently, PyLith uses its own simple layout show in Fig. 4.15. In order for visualization tools, such as ParaView, to determine which datasets to read and where to find them in the hierarchy of groups within the HDF5 file, we create an Xdmf (eXtensible Data Model and Format, <https://www.xdmf.org>) metadata file that provides this information. This file is written when PyLith closes the HDF5 file at the end of the simulation. In order to visualize the datasets in an HDF5 file, one simply opens the corresponding Xdmf file (the extension is `xmf`) in ParaView or Visit. The Xdmf file contains the relative path to the HDF5 file so the files can be moved but must be located together in the same directory.

Important

The Xdmf format supports representation of two- and three-dimensional coordinates of points, scalar fields, and three-dimensional vector and tensor fields but not two-dimensional vector or tensor fields. Consequently, for two-dimensional vector fields we build a three-component vector from the two-component vector (x and y components) and a separate zero scalar field (z component). For tensor fields, we create a scalar field for each of the tensor components, adding the component as a suffix to the name of the field.

Table 4.22: General ordering and names of vector and tensor components in HDF5 output.

Vector Field Type	Components
vector	x, y, z
tensor	xx, yy, zz, xy, yz, xz

PyLith provides two different data writers for HDF5 files. The `DataWriterHDF5` object writes all information into the HDF5 file, whereas the `DataWriterHDF5Ext` object writes the data to external binary files and only the metadata to the HDF5 file. HDF5 files do not contain self-correcting features that allow a file to be read if part of a dataset is corrupted.

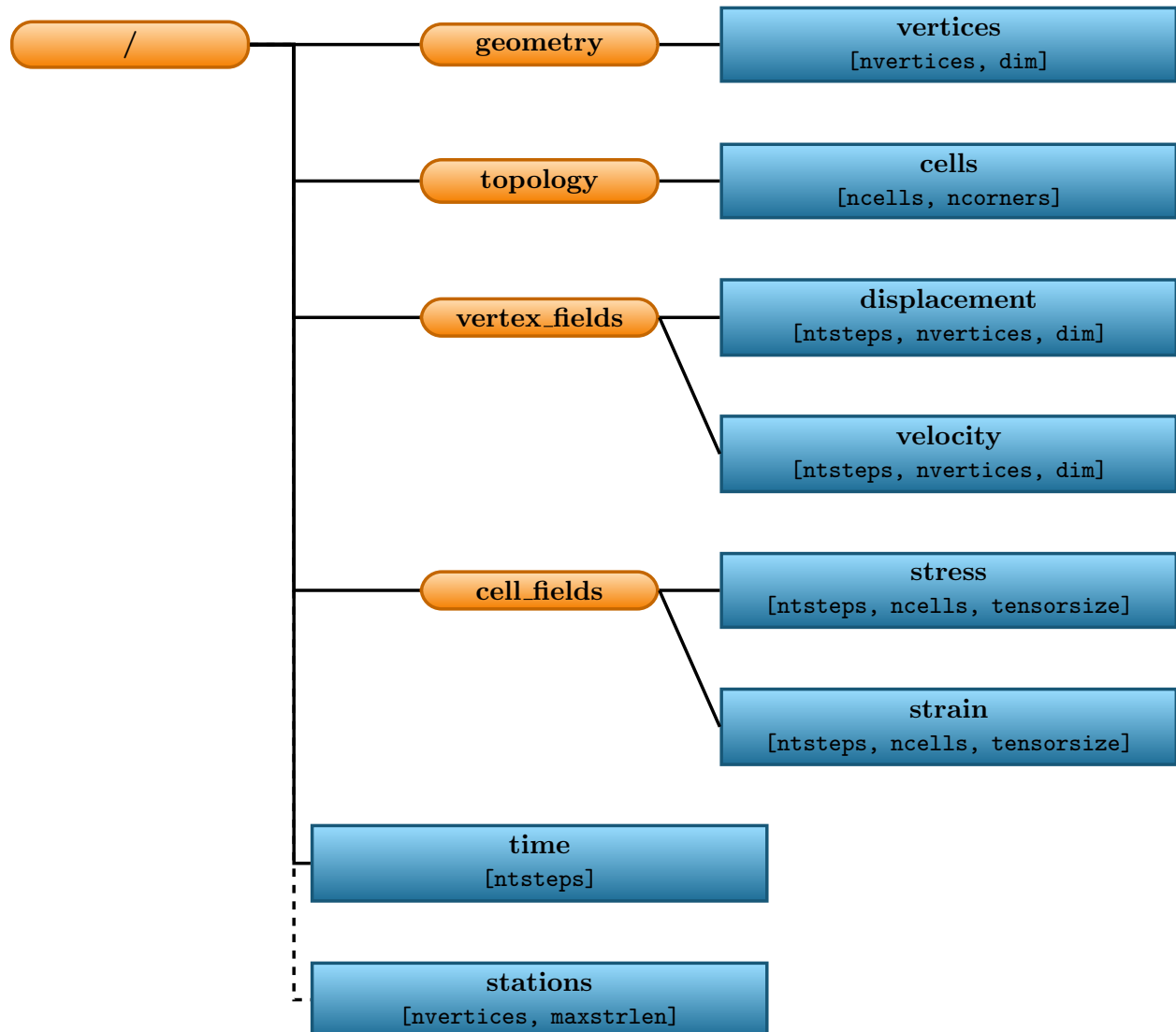


Fig. 4.15: General layout of a PyLith HDF5 file. The orange rectangles with rounded corners identify the groups and the blue rectangles with sharp corners identify the datasets. The dimensions of the data sets are shown in parentheses. Most HDF5 files will contain either `vertex_fields` or `cell_fields` but not both.

This type of error can occur if a job terminates abnormally in the middle or at the end of a simulation on a large cluster or other parallel machine. Fortunately, HDF5 also offers the ability to store datasets in external binary files with the locations specified by links in the HDF5 file. Note that the use of external data files results in one data file per dataset in addition to the HDF5 and Xdmf files. The external data files use the name of the HDF5 file with the dataset name added to the prefix and the h5 suffix replaced by dat. The HDF5 files include relative paths to the external data files, so these files can also be moved, but they, too, must be kept together in the same directory. This provides a more robust method of output because one can generate an HDF5 file associated with the uncorrupted portions of the external data files should an error occur. Currently, PyLith does not include a utility to do this, but we plan to add one in a future release. Thus, there are two options when writing PyLith output to HDF5 files: (1) including the datasets directly in the HDF5 files themselves using the `DataWriterHDF5` object or (2) storing the datasets in external binary files with just metadata in the HDF5 files using the `DataWriterHDF5Ext` object. Both methods provide similar performance because they will use MPI I/O if it is available.

Warning

Storing the datasets within the HDF5 file in a parallel simulation requires that the HDF5 library be configured with the `--enable-parallel` option. The binary PyLith packages include this feature and it is a default setting in building HDF5 via the PyLith Installer.

Accessing the datasets for additional analysis or visualization is nearly identical in the two methods because the use of external data files is completely transparent to the user except for the presence of the additional files. Note that in order for ParaView to find the HDF5 and external data files, it must be run from the same relative location where the simulation was run. For example, if the simulation was run from a directory `work` and the HDF5/Xdmf files were written to `work/output`, then ParaView should be run from the `work` directory.

See also

[DataWriterHDF5 Component](#) and [DataWriterHDF5Ext Component](#)

HDF5 Utilities

HDF5 includes several utilities for examining the contents of HDF5 files. `h5dump` is very handy for dumping the hierarchy, dimensions of datasets, attributes, and even the dataset values to stdout.

```
# Dump the entire HDF5 file (not useful for large files).
$ h5dump mydata.h5

# Dump the hierarchy of an HDF5 file.
$ h5dump -n mydata.h5

# Dump the hierarchy with dataset dimensions and attributes.
$ h5dump -H mydata.h5

# Dump dataset 'vertices' in group '/geometry' to stdout.
$ h5dump -d /geometry/vertices mydata.h5
```

We also provide a utility `pylith_genxdmf` (see {ref}`sec-user-pylith-genxdmf`) that generates an appropriate Xdmf file from a PyLith HDF5 file. This is very useful if you add fields to HDF5 files in post-processing and wish to view the results in ParaView or Visit.

VTK Output

PyLith writes VTU files for VTK output, which are in the XML format. The XML files contain metadata in XML tags and raw data for the vertex coordinates, mesh topology, and fields over vertices and/or cells. Each time step is written to a different file. The time stamp is included in the filename with the decimal point removed. This allows automatic generation of animations with many visualization packages that use VTK files. The default time stamp is the time in seconds, but this can be changed using the normalization constant to give a time stamp in years, tens of years, or any other value.

Warning

We discourage use of the VTK output as the files are not easily read from post-processing scripts. We strongly recommend using HDF5 output instead, which is the default starting in PyLith v3.0.0.

See also

DataWriterVTK Component

Output Triggers

New in v3.0.0.

By default PyLith will write the requested output after every time step. In many cases we prefer to save the solution, state variables, etc at a coarser temporal resolution. `OutputTriggerStep` controls the decimation of the output by time step, and `OutputTriggerTime` controls the decimation of the output via elapsed time. For a constant time step these can be equivalent.

Decimate by time step

`OutputTriggerStep` decimates the output by skipping a user-specified number of time steps.

See also

OutputTriggerStep Component

Decimate by time

`OutputTriggerTime` decimates the output by skipping a user-specified elapsed time between output time slices.

Tip

Due to roundoff error in determining the simulation time over many time steps, a simulation may occasionally skip writing output unexpectedly when using `OutputTriggerTime`. The best workaround is to use an `elapsed_time` that is a fraction of the time step size smaller than the desired elapsed time, such as `0.9999year` instead of `1.0year`.

See also

OutputTriggerTime Component

4.2.7 Physics Implementations

Materials

The material objects encapsulate the bulk behavior of the domain. This includes both the governing equations and the associated bulk rheology.

Specifying Material Properties

Associating material properties with a given cell involves several steps.

1. In the mesh generation process, assign a material identifier to each cell.
2. Define material property groups corresponding to each material identifier. In Cubit/Trelis this is done by creating the blocks as part of the boundary conditions.
3. Provide the settings for each material group in the parameters, i.e., `cfg` file.
4. Specify the parameters for the material properties, e.g., linear variation in density with depth, using a spatial database file. This allows variation of the material properties across cells with the same material identifier.

Material Implementations

Elasticity

You can use the `Elasticity` component to solve the elasticity equation with or without inertia. Whether inertia or body forces are included is determined by the `Elasticity` property settings. Gravitational body forces are included if the `gravity_field` is set in the `Problem`. Table 4.23 lists the various elastic, viscoelastic, and elastoplastic bulk constitutive models implemented for the elasticity equation.

Table 4.23: Elasticity bulk rheologies

Bulk Rheology	Description
<code>IsotropicLinearElasticity</code>	Isotropic, linear elasticity
<code>IsotropicLinearMaxwell</code>	Isotropic, linear Maxwell viscoelasticity
<code>IsotropicLinearGenMaxwell</code>	Isotropic, generalized Maxwell viscoelasticity
<code>IsotropicPowerLaw</code>	Isotropic, power-law viscoelasticity
<code>IsotropicDruckerPrager</code>	Isotropic, Drucker-Prager elastoplasticity

Warning

The `IsotropicDruckerPrager` rheology has not yet been implemented in PyLith v3.

Table 4.24: Properties defining elasticity bulk rheologies

Subfield	L	LM	GM	PL	DP	Compo- nents
<code>density</code>	X	X	X	X	X	
<code>vp</code> (P-wave speed)	X	X	X	X	X	
<code>vs</code> (S-wave speed)	X	X	X	X	X	
<code>body_force</code>	O	O	O	O	O	x, y, z

continues on next page

Table 4.24 – continued from previous page

Subfield	L	LM	GM	PL	DP	Compo- nents
gravitational	O	O	O	O	O	x, y, z
shear_modulus	I	I	I	I	I	
bulk_modulus	I	I	I	I	I	
reference_strain	O	O	O	O	O	xx, yy, zz, xy, yz, xz
reference_stress	O	O	O	O	O	xx, yy, zz, xy, yz, xz
maxwell_time		I				
viscosity		X				
viscosity_1			X			
viscosity_2			X			
viscosity_3			X			
shear_ratio_1			X			
shear_ratio_2			X			
shear_ratio_3			X			
total_strain		X	X	X		xx, yy, zz, xy, yz, xz
viscous_strain		X		X		xx, yy, zz, xy, yz, xz
viscous_stress			X			xx, yy, zz, xy, yz, xz
viscous_strain			X			xx, yy, zz, xy, yz, xz
viscous_stress			X			xx, yy, zz, xy, yz, xz
power_law_exponent				X		
power_law_reference				X		
power_law_ratio				X		
deviatoric_stress				X		
cohesion					X	
friction_angle					X	
dilatation_angle					X	
alpha_yield					I	
alpha_flow					I	
beta					I	
plastic_strain					X	xx, yy, zz, xy, yz, xz

X: required value in auxiliary field spatial database

O: optional value in auxiliary field spatial database

I: internal auxiliary subfield; computed from spatial database values L: isotropic, linear elasticity

ML: isotropic linear Maxwell viscoelasticity

GM: isotropic generalized linear Maxwell viscoelasticity

PL: isotropic power-law viscoelasticity

DP: isotropic Drucker-Prager elastoplasticity

Table 4.25: Derived subfields that are available for output for elasticity bulk rheologies.

Subfield	L	LM	GM	PL	DP	Components
cauchy_stress	✓	✓	✓	✓	✓	xx, yy, zz, xy, yz, xz
cauchy_strain	✓	✓	✓	✓	✓	xx, yy, zz, xy, yz, xz

➡ See also

See [Elasticity Component](#) for the Pyre properties and facilities and configuration examples.

Incompressible Elasticity

You can use the `IncompressibleElasticity` component to solve the quasistatic incompressible elasticity equation. Estimating realistic distributions of initial stress fields consistent with gravitational body forces can be quite difficult due to our lack of knowledge of the deformation history. A simple way to approximate the lithostatic load is to solve for the stress field imposed by gravitational body forces assuming an incompressible elastic material. This limits the volumetric deformation. In this context we do not include inertia, so the `IncompressibleElasticity` component does not include an inertial term. Gravitational body forces are included if the `gravity_field` is set in the `Problem`. Table 4.26 lists the elastic bulk rheology implemented for the incompressible elasticity equation.

Table 4.26: Incompressible elasticity bulk rheology.

Bulk Rheology	Description
<code>IsotropicLinearIncompElasticity</code>	Isotropic, linear incompressible elasticity

Table 4.27: Properties defining incompressible elasticity bulk rheologies.

Subfield	L	LM	GM	PL	Components
density	X	X	X	X	
vp (P-wave speed)	X	X	X	X	
vs (S-wave speed)	X	X	X	X	
body_force	O	O	O	O	x, y, z
gravitational_acceleration	O	O	O	O	x, y, z
shear_modulus	I	I	I	I	
bulk_modulus	I	I	I	I	
reference_stress	O	O	O	O	xx, yy, zz, xy, yz, xz
reference_strain	O	O	O	O	xx, yy, zz, xy, yz, xz

X: required value in auxiliary field spatial database

O: optional value in auxiliary field spatial database

I: internal auxiliary subfield; computed from spatial database values L: isotropic, linear elasticity

ML: isotropic linear Maxwell viscoelasticity

GM: isotropic generalized linear Maxwell viscoelasticity

PL: isotropic power-law viscoelasticity

Table 4.28: Derived subfields that are available for output for incompressible elasticity bulk rheologies.

Subfield	L	LM	GM	PL	Components
cauchy_stress	✓	✓	✓	✓	xx, yy, zz, xy, yz, xz
cauchy_strain	✓	✓	✓	✓	xx, yy, zz, xy, yz, xz

➡ See also

See [IncompressibleElasticity Component](#) for the Pyre properties and facilities and configuration examples.

Poroelectricity

You can use the Poroelectricity component to solve the poroelectricity equation with or without inertia. Whether inertia or body forces are included is determined by the Poroelectricity property settings. Gravitational body forces are included if the `gravity_field` is set in the Problem. [Table 4.29](#) lists the poroelectric bulk rheology implemented for the poroelectricity equation.

Table 4.29: Elasticity bulk rheologies.

Bulk Rheology	Description
IsotropicLinearPoroelectricity	Isotropic, linear poroelectricity

Table 4.30: Properties defining elasticity bulk rheologies.

Subfield	L	Components
solid_density	X	
fluid_density	X	
fluid_viscosity	X	
porosity	X	
body_force	O	x, y, z
gravitational_acceleration	O	x, y, z
shear_modulus	X	
drained_bulk_modulus	X	
fluid_bulk_modulus	X	
biot_coefficient	X	
isotropic_permeability	X	
tensor_permeability	O	xx, yy, zz, xy, yz, xz
reference_stress	O	xx, yy, zz, xy, yz, xz
reference_strain	O	xx, yy, zz, xy, yz, xz

X: required value in auxiliary field spatial database

O: optional value in auxiliary field spatial database

L: isotropic, linear poroelectricity

Table 4.31: Derived subfields that are available for output for poroelasticity bulk rheologies.

Subfield	L	Components
cauchy_stress	X	xx, yy, zz, xy, yz, xz
cauchy_strain	X	xx, yy, zz, xy, yz, xz
bulk_density	X	
water_content	X	

When porosity is enabled as a state variable, it will be included in the output along with the derived subfields.

➡ See also

See *Poroelasticity Component* for the Pyre properties and facilities and configuration examples.

➡ See also

See *Governing Equations* for the derivation of the finite-element formulation for each of the materials.

Boundary Conditions

Assigning boundary conditions

There are three basic steps in assigning a specific boundary condition to a portion of the domain.

1. Create sets of vertices in the mesh generation process for each boundary condition.
2. Set the parameters for each boundary condition group using `cfg` files or command line arguments.
3. Specify the spatial variation in parameters for the boundary condition using a spatial database.

Marking boundaries when meshing

The procedure for marking boundaries depends on the mesh generator. For meshes specified using the PyLith mesh ASCII format, cell faces are specified using `face-groups` (see *PyLith Mesh ASCII File*). In Cubit cell faces are marked using sidesets. Note also that we currently associate boundary conditions with string identifiers. Finally, note that the boundary conditions must be associated with a simply-connected surface; that is, surfaces must be connected and not contain holes.

Arrays of boundary condition components

A dynamic array of boundary condition components associates a name (string) with each boundary condition. The default boundary condition for each component in the array is `DirichletTimeDependent`. Other boundary conditions can be bound to the named items in the array by assigning the component type to the named boundary condition.

Listing 4.31: Array of boundary conditions in a `cfg` file

```
[pylithapp.problem]
# Array of four boundary conditions
bc = [x_neg, x_pos, y_pos, z_neg]
# Default boundary condition is DirichletBC
# Keep default value for x_neg and x_pos but assign new types to y_pos and z_neg
```

(continues on next page)

(continued from previous page)

```
bc.y_pos = pylith.bc.AbsorbingDampers
bc.z_neg = pylith.bc.NeumannTimeDependent
```

Diagnostic information

The diagnostic information includes the outward normal direction (`normal_dir`) and the two tangential directions (`horizontal_tangential_dir` and `vertical_tangential_dir`). The default basis order for discretizing these directions is 1, so these produce `vertex_fields` as opposed to `cell_fields` (basis order of 0).

Boundary condition implementations

Time-Dependent Boundary Conditions

Several boundary conditions use a common formulation for the spatial and temporal variation of the boundary condition parameters,

$$f(\vec{x}) = f_0(\vec{x}) + \dot{f}_1(\vec{x})(t - t_1(\vec{x})) + f_2(\vec{x})a(t - t_2(\vec{x})), \quad (4.441)$$

where

- $f(\vec{x})$
may be a scalar or vector parameter,
- $f_0(\vec{x})$
is a constant value (independent of time),
- $\dot{f}_1(\vec{x})$
is a constant rate of change in the value with time,
- $t_1(\vec{x})$
is the onset time for the constant rate of change,
- $f_2(\vec{x})$
is the amplitude for the temporal modulation,
- $a(t)$
is the variation in amplitude with time,
- $t_2(\vec{x})$
is the onset time for the temporal modulation, and
- $\vec{x}$
is the position of a location in space.

This common formulation permits easy specification of a scalar or vector with a constant value, constant rate of change of a value, or modulation of a value in time. One can specify just the initial value, just the rate of change of the value (along with the corresponding onset time), or just the modulation in amplitude (along with the corresponding temporal variation and onset time), or any combination of the three.

Time-Dependent Dirichlet Boundary Conditions

You can use the `DirichletTimeDependent` boundary conditions to prescribe a solution subfield on a boundary of the finite-element mesh. The spatial database files for the Dirichlet boundary condition specify the parameters for the time-dependent expression.

Important

The spatial database files for Dirichlet boundary conditions must contain values for all degrees of freedom (x and y for 2-D, and x, y, and z for 3-D even if they are not constrained. This limitation is imposed by the PETSc DMPlex interface.

Table 4.32: Values in the auxiliary field spatial databases used for Dirichlet time-dependent boundary conditions.

Flag	Required Values
use_initial	initial_amplitude_x, initial_amplitude_y, initial_amplitude_z
use_rate	rate_start_time, rate_amplitude_x, rate_amplitude_y, rate_amplitude_z
use_time_histc	time_history_start, time_history_amplitude_x, time_history_amplitude_y, time_history_amplitude_z

See also

DirichletTimeDependent Component for the Pyre properties and facilities and configuration examples.

Neumann Time-Dependent Boundary Conditions

Neumann boundary conditions are surface tractions applied over a boundary. As with the `DirichletTimeDependent` condition, each Neumann boundary condition can only be applied to a simply-connected surface on an external boundary. The spatial database file the auxiliary subfields for the Neumann boundary condition specify the parameters for the time-dependent expressions.

Table 4.33: Values in the auxiliary field spatial database used for Neuman time-dependent boundary conditions.

Di-men-sion	Flag	Required Values
2	use_initial	initial_amplitude_normal, initial_amplitude_tangential
	use_rate	rate_start_time, rate_amplitude_normal, rate_amplitude_tangential
	use_time_h	time_history_start, time_history_amplitude_normal, time_history_amplitude_tangential
3	use_initial	initial_amplitude_normal, initial_amplitude_tangential_1, initial_amplitude_tangential_2
	use_rate	rate_start_time, rate_amplitude_normal, rate_amplitude_tangential_1, rate_amplitude_tangential_2
	use_time_h	time_history_start, time_history_amplitude_normal, time_history_amplitude_tangential_1, time_history_amplitude_tangential_2

 See also

NeumannTimeDependent Component for the Pyre properties and facilities and configuration examples.

Absorbing Boundary Conditions

You can use the `AbsorbingDampers` boundary condition to prevent seismic waves reflecting off of a boundary. Normally incident dilatational and shear waves are perfectly absorbed. Waves incident at other angles are only partially absorbed. This boundary condition is simpler than a perfectly matched layer (PML) boundary condition but does not perform quite as well, especially for surface waves. If the waves arriving at the absorbing boundary are relatively small in amplitude compared to the amplitudes of primary interest, this boundary condition gives reasonable results.

The auxiliary field spatial database contains the bulk rheology properties for an isotropic, linear elastic material (density, V_s (S-wave speed), and V_p (P-wave speed)). You can simply use the same spatial database that was used to specify the elastic properties of the material.

 See also

See *AbsorbingDampers Component* for the Pyre properties and facilities and configuration examples.

See *Absorbing Boundary* for the derivation of the finite-element implementation of the absorbing boundary.

 See also

See *Governing Equations* for the derivation of the finite-element formulation for each of the boundary conditions.

Fault Interface Conditions

Fault interfaces are used to create dislocations (jumps in the displacement field) in the model. The dislocations arise from slip across a fault surface. Both shear and tensile dislocations are supported. Dislocations in 2D are specified in terms of left-lateral-slip and fault opening, and in 3D they are specified in terms of left-lateral-slip, reverse-slip, and fault opening. PyLith supports kinematic (prescribed) slip and dynamic (spontaneous) rupture simulations.

 Warning

Spontaneous rupture is not available in PyLith v3.0+; we plan to have it reimplemented in v6.0 or v7.0

Conventions

Slip corresponds to relative motion across a fault surface. [Fig. 4.16](#) shows the orientation of the slip vector in 3D with respect to the fault surface and coordinate axes. PyLith automatically determines the local orientation of the fault surface. This alleviates the user from having to compute the strike, dip, and rake angles over potentially complex, nonplanar fault surfaces. Instead, the user specifies fault parameters in terms of lateral motion, reverse motion, and fault opening as shown in [Fig. 4.17](#).

Fault Implementation

In order to create relative motion across the fault surface in the finite-element mesh, PyLith adjusts the topology of the mesh by inserting zero area (2D) or volume (3D) cohesive cells along the fault surface. The cohesive cells allow control of the relative motion between vertices on the two sides of the fault. [Fig. 4.18](#) illustrates the results of inserting

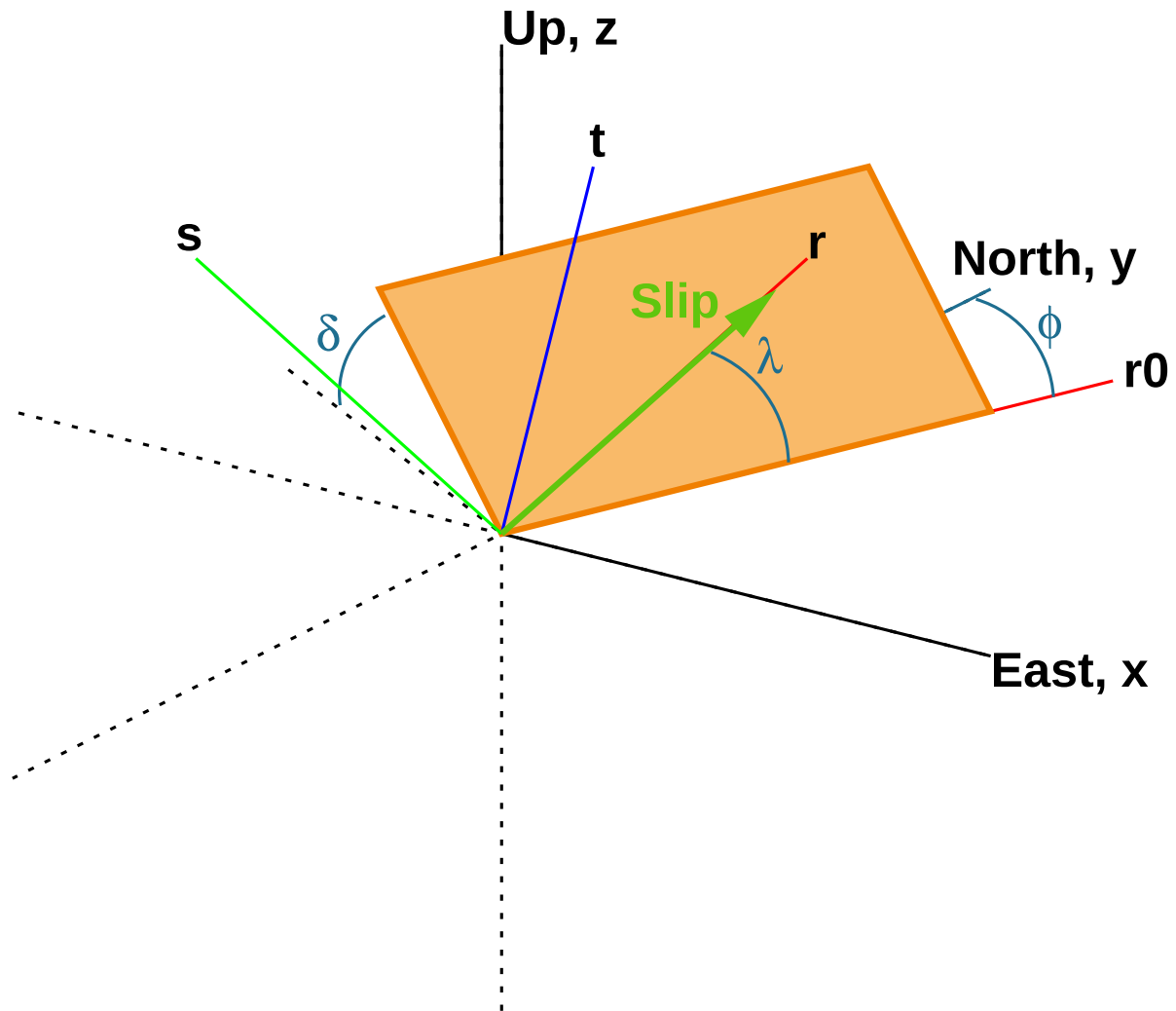


Fig. 4.16: Orientation of a fault surface in 3D, where ϕ denotes the angle of the fault strike, δ denotes the angle of the fault dip, and λ the rake angle.

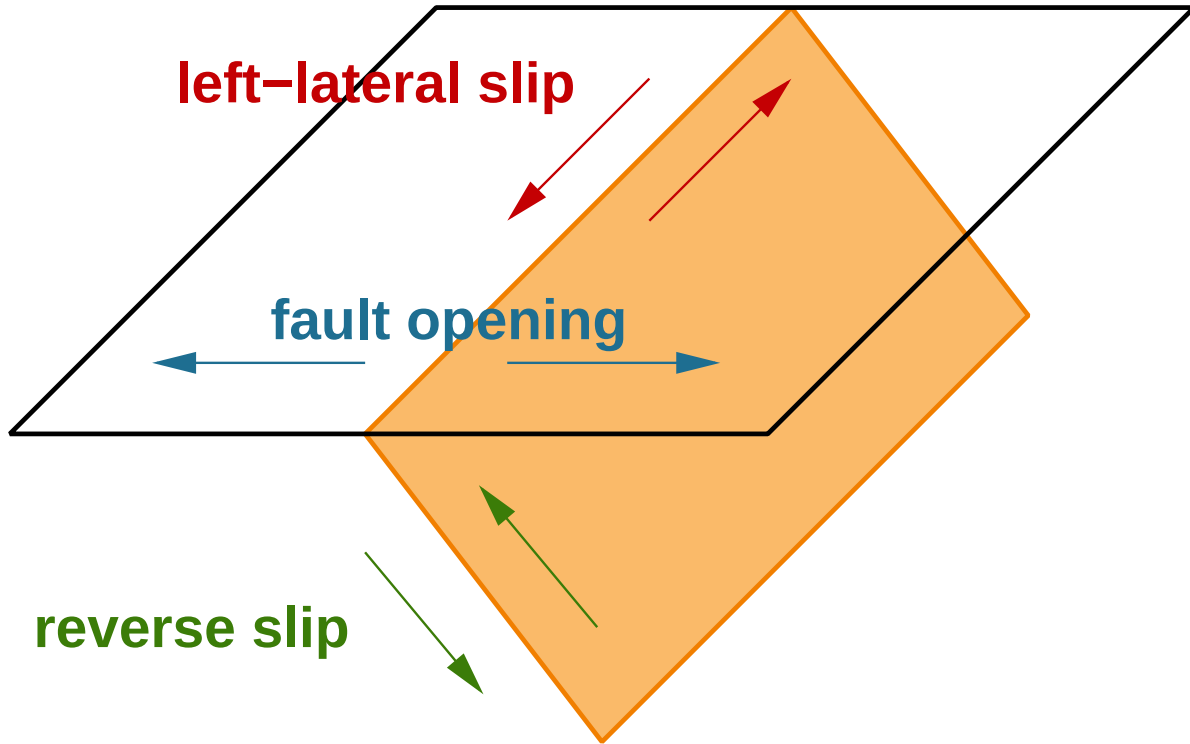


Fig. 4.17: Sign conventions associated with fault slip. Positive values are associated with left-lateral, reverse, and fault opening motions.

cohesive cells in a mesh consisting of triangular cells. This example also shows the distinction between how buried fault edges are handled differently than fault edges that reach the edge of the domain, such as the ground surface.

For faults that have buried edges, splitting the mesh apart and inserting the cohesive cells becomes complex at the buried edges due to the ambiguity of defining where the fault ends and how to insert the cohesive cell. Starting in PyLith v2.0.0 we changed how the buried edges of the fault are managed. An additional group of fault nodes is specified (for example, via a nodeset from Cubit) that marks the buried edges of the fault (see Fig. 4.19). This allows the cohesive cell insertion algorithm to adjust the topology so that cohesive cells are inserted up to the buried edge of the fault but no additional degrees of freedom are added on the fault edge. This naturally forces slip to zero along the buried edges.

Important

In 2D, positive in-plane slip is left-lateral; this applies to all faults regardless of whether the domain is a horizontal slice (strike-slip motion) or a vertical slice (reverse or normal motion). In 3D the reference directions are used to resolve the ambiguity in the along-strike and dip-dir directions. Left-lateral slip is in the along-strike direction, which is defined to be $\vec{t}_1 = \vec{r}_1 \times \vec{n}$, where $\vec{r}_1$ is the first reference direction; to match seismological conventions, $\vec{r}_1$ should approximately be in the vertical or up-dip direction. The default value is (0, 0, 1), which works as long as the fault surface is not horizontal at any location. Reverse slip is in the dip direction, which is defined to be $\vec{t}_2 = \vec{n} \times \vec{t}_1$. Because the fault normal direction is arbitrary, the dip direction can be either up dip or down dip. We resolve the ambiguity by using the second reference direction to define the preferred approximate fault normal direction.

By default the output observers write both diagnostic information (for example, fault orientation directions) and the slip at each time step. The fault coordinate system is shown in Fig. 4.17. The vectors in the fault coordinate system can be transformed to the global coordinate system using the direction vectors in the diagnostic output (*Output Observers*).

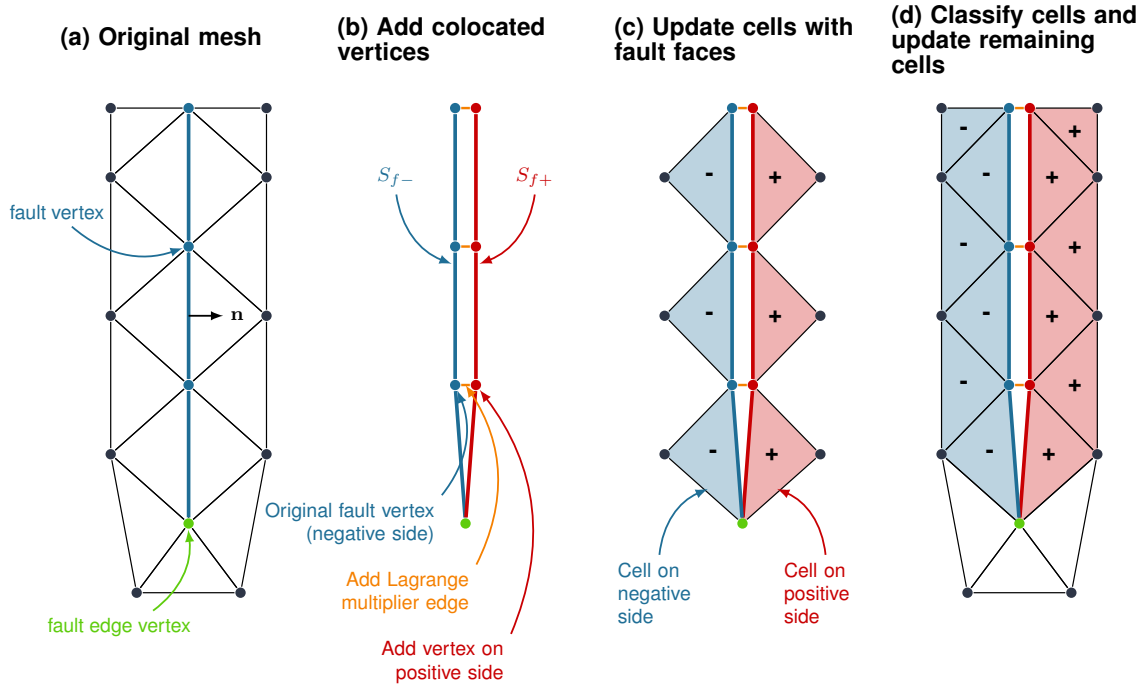


Fig. 4.18: Example of cohesive cells inserted into a mesh of triangular cells. The zero thickness cohesive cells control slip on the fault via the relative motion between the vertices on the positive and negative sides of the fault.

Prescribed Slip (FaultCohesiveKin)

Prescribed slip, often called kinematic earthquake ruptures, use the `FaultCohesiveKin` component to prescribe the slip as a function of time on the fault surface. Slip may evolve simultaneously over the fault surface instantaneously in a single time step (as is usually done in quasistatic simulations) or propagate over the fault surface over hundreds and up to thousands of time steps (as is usually done in a dynamic simulation).

Multiple earthquake ruptures can be specified on a single fault surface. This permits repeatedly rupturing the same portion of a fault or combining earthquake rupture on one subset of the fault surface with steady aseismic slip on another subset (the two subsets may overlap in both time and space). An array of kinematic earthquake rupture components associates a name (string) with each kinematic rupture. The default dynamic array contains a single earthquake rupture, `rupture`.

The default discretization of the `slip` auxiliary subfield is a basis order of 1 (linear variation on the fault). This discretization will also be used for all auxiliary subfields associated with parameters of the slip time function.

➡ See also

See [FaultCohesiveKin Component](#) for the Pyre properties and facilities and configuration examples.

Prescribed Slip Parameters (KinSrc)

The kinematic rupture parameters include the origin time and slip time function. The slip initiation time in the slip time function is relative to the origin time (default is 0). This means that slip initiates at a point at a time corresponding to the sum of the kinematic rupture's origin time and the slip initiation time for that point.

PyLith supports specification of the evolution of fault slip using analytical expressions for the slip time history at each point, where the parameters for the slip time function may vary over the fault surface. The following slip time functions

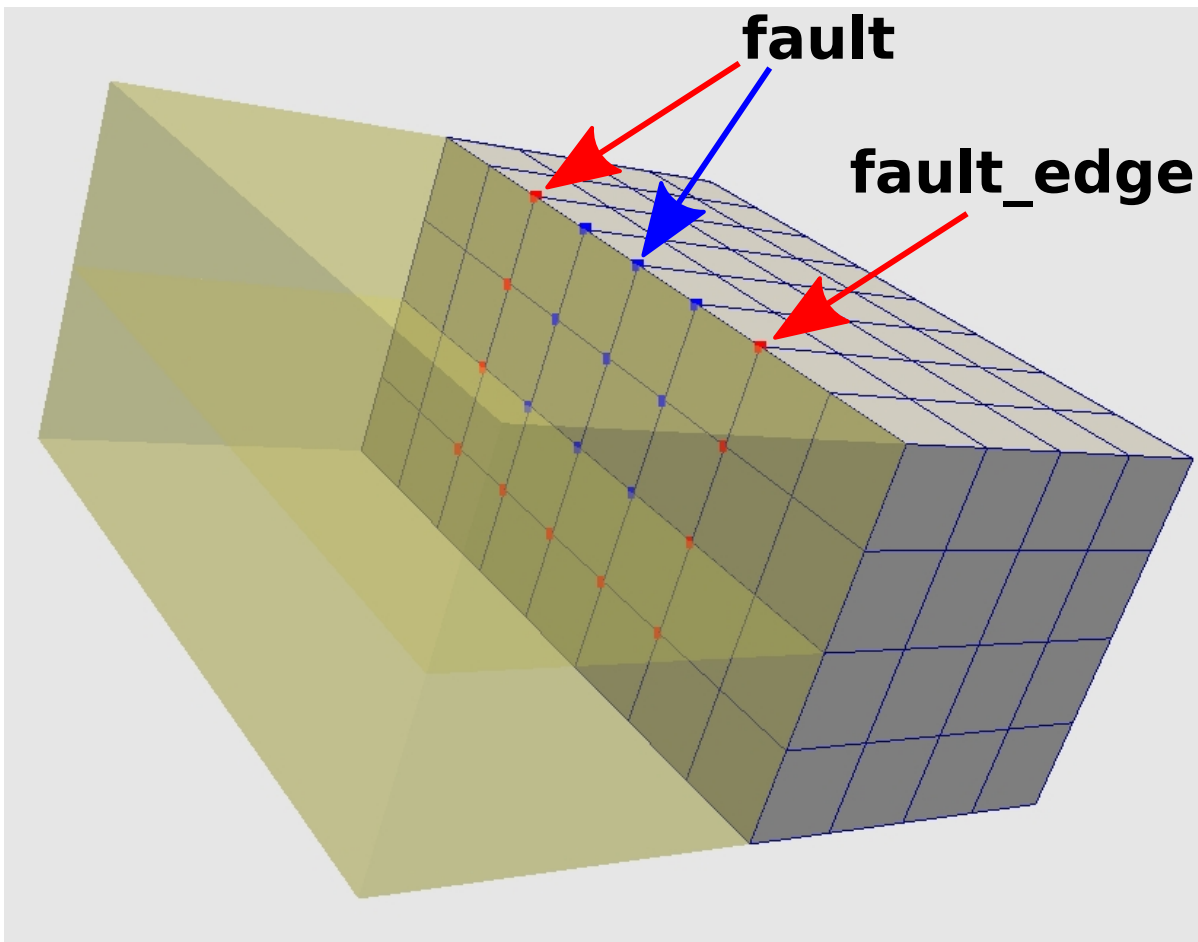


Fig. 4.19: Example of how faults with buried edges must be described with two sets of vertices. All of the vertices on the fault are included in the `fault` group; the subset of vertices along the buried edges are included in the `fault_edge` group. In 2D the fault edges are just a single vertex as shown in [Fig. 4.18](#)

are available:

KinSrcStep

a step function for quasistatic modeling of earthquake rupture,

KinSrcConstRate

a constant rate function for steady slip (for example, creep),

KinSrcRamp

a constant rate function for a specified time interval,

KinSrcBrune

a slip time function corresponding to the integral of Brune's far-field time function for dynamic modeling of earthquake rupture,

KinSrcLiuCos

a slip time function built on cosine functions for dynamic modeling of earthquake rupture, and

KinSrcTimeHistory

a user-specified slip time function.

Step-Function Slip Time Function (KinSrcStep)

This slip function prescribes a step in slip at a given time at a point:

$$D(t) = \begin{cases} 0 & 0 \leq t < t_r \\ D_{final} & t \geq t_r \end{cases} \quad (4.442)$$

where $D(t)$ is slip at time t , D_{final} is the final slip, and t_r is the slip initiation time (time when rupture reaches the location). The slip is specified independently for each of the components of slip, and the slip and slip starting time may vary over the fault surface.

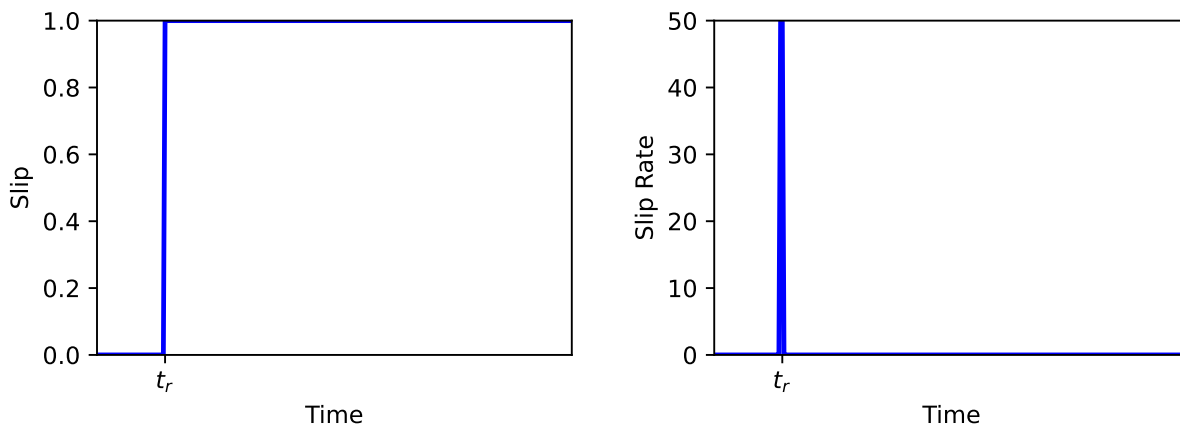


Fig. 4.20: Step slip time function.

Table 4.34: Values in the auxiliary field spatial database for KinSrcStep.

Subfield	Components
initiation_time (t_r)	–
final_slip	opening, left_lateral, reverse

➡ See also

See [KinSrcStep Component](#) for the Pyre properties and facilities and configuration examples.

Constant Slip Rate Slip Time Function (KinSrcConstRate)

This slip function prescribes a constant slip rate for the evolution of slip at a point:

$$D(t) = \begin{cases} 0 & 0 \leq t < t_r \\ V(t - t_r) & t \geq t_r \end{cases} \quad (4.443)$$

where $D(t)$ is slip at time t , V is the slip rate, and t_r is the slip initiation time (time when rupture reaches the location). The slip rate is specified independently for each of the components of slip, and the slip rate and slip starting time may vary over the fault surface.

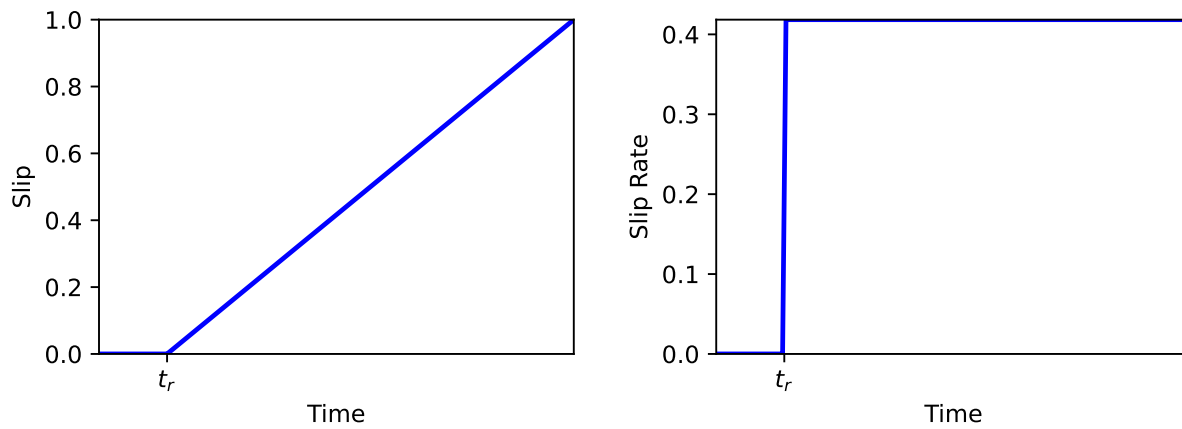


Fig. 4.21: Constant rate slip time function.

Table 4.35: Values in the auxiliary field spatial database for KinSrc-ConstRate.

Subfield	Components
initiation_time (t_r)	–
slip_rate	opening, left_lateral, reverse

➔ See also

See [KinSrcConstRate Component](#) for the Pyre properties and facilities and configuration examples.

Ramp Slip Time Function (KinSrcRamp)

This slip function prescribes a constant slip rate over a time window with a smooth initiation and termination:

$$D(t) = \begin{cases} 0 & t - t_r \leq 0 \\ \frac{D_{fina}}{C_{acc}} f_1(t) & t - t_r \leq \frac{1}{2} t_{acc} \\ \frac{D_{fina}}{C_{acc}} f_2(t) & t - t_r \leq t_{acc} \\ \frac{D_{fina}}{C_{acc}} f_3(t) & t - t_r \leq t_{rise} - t_{acc} \\ \frac{D_{fina}}{C_{acc}} f_4(t) & t - t_r \leq t_{rise} - \frac{1}{2} t_{acc} \\ \frac{D_{fina}}{C_{acc}} f_5(t) & t - t_r \leq t_{rise} \\ \frac{D_{fina}}{C_{acc}} f_6(t) & t - t_r > t_{rise} \end{cases} \quad (4.444)$$

$$f_1(t) = \frac{1}{6} t^3 \quad (4.445)$$

$$f_2(t) = \frac{1}{2} t_{acc} t^2 - \frac{1}{6} t^3 - \frac{1}{4} t_{acc}^2 t + \frac{1}{24} t_{acc}^3 \quad (4.446)$$

$$f_3(t) = \frac{1}{4} t_{acc}^2 t - \frac{1}{8} t_{acc}^3 \quad (4.447)$$

$$f_4(t) = -\frac{1}{6} t^3 + \frac{1}{2} (t_{rise} - t_{acc}) t^2 - \frac{1}{2} (t_{rise} - t_{acc})^2 t + \frac{1}{4} t_{acc}^2 t + \frac{1}{6} (t_{rise} - t_{acc})^3 - \frac{1}{8} t_{acc}^3 \quad (4.448)$$

$$f_5(t) = \frac{1}{6} t^3 - \frac{1}{2} t_{rise} t^2 + \frac{1}{2} t_{rise}^2 t - \frac{1}{3} t_1^3 + \frac{1}{2} t_{rise} t_1^2 - \frac{1}{2} t_{rise}^2 t_1 \quad (4.449)$$

$$+ \frac{1}{2} (t_{rise} - t_{acc}) t_1^2 - \frac{1}{2} (t_{rise} - t_{acc})^2 t_1 + \frac{1}{4} t_{acc}^2 t_1 + \frac{1}{6} (t_{rise} - t_{acc})^3 - \frac{1}{8} t_{acc}^3 \quad (4.450)$$

$$f_6(t) = \frac{1}{6} (t_{rise}^3 - \frac{1}{3} (t_1^3 + \frac{1}{2} t_{rise} (t_1^2 - \frac{1}{2} (t_{rise}^2 t_1 + \frac{1}{2} (t_{rise} - t_{acc}) t_1^2 - \frac{1}{2} (t_{rise} - t_{acc})^2 t_1$$

$$+ \frac{1}{4} t_{acc}^2 t_1 + \frac{1}{6} (t_{rise} - t_{acc})^3 - \frac{1}{8} t_{acc}^3 \quad (4.452)$$

$$C_{acc} = \frac{1}{6} t_{rise}^3 - 1.0/3.0 (t_{rise} - \frac{1}{2} t_{acc})^3 + \frac{1}{2} t_{rise} (t_{rise} - \frac{1}{2} t_{acc})^2 - \frac{1}{2} t_{rise}^2 (t_{rise} - \frac{1}{2} t_{acc}) \quad (4.453)$$

$$+ \frac{1}{2} (t_{rise} - t_{acc}) (t_{rise} - \frac{1}{2} t_{acc})^2 - \frac{1}{2} (t_{rise} - t_{acc})^2 (t_{rise} - \frac{1}{2} t_{acc}) + \frac{1}{4} (t_{acc}^2 (t_{rise} - \frac{1}{2} t_{acc}) \quad (4.454)$$

$$+ \frac{1}{6} (t_{rise} - t_{acc})^3 - \frac{1}{8} (t_{acc})^3 \quad (4.455)$$

$$t_1 = t_{rise} - \frac{1}{2} t_{acc} \quad (4.456)$$

where $D(t)$ is slip at time t , t_r is the slip initiation time (time when rupture reaches the location), t_{rise} is the rise time, and t_{acc} is the duration of the acceleration impulse. The slip is specified independently for each of the components of slip, and the slip rate and slip starting time may vary over the fault surface.

Table 4.36: Values in the auxiliary field spatial database for KinSrcRamp.

Subfield	Components
initiation_time (t_r)	—
rise_time (t_{rise})	—
impulse_duration (t_{acc})	—
final_slip	opening, left_lateral, reverse

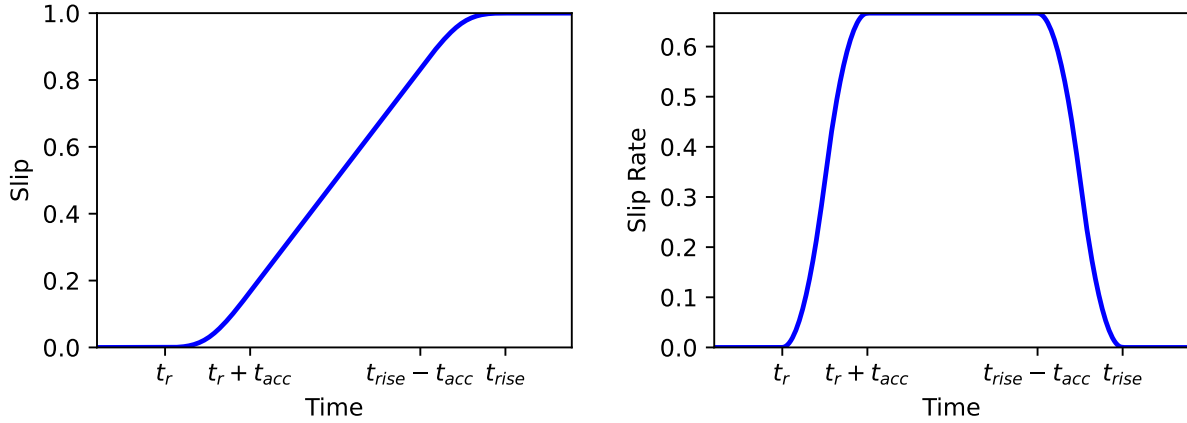


Fig. 4.22: Ramp slip time function.

➔ See also

See [KinSrcRamp Component](#) for the Pyre properties and facilities and configuration examples.

Brune Slip Time Function (KinSrcBrune)

This slip function corresponds to the integral of Brune's far-field time function [Brune, 1970] and prescribes a rapid rise in slip rate followed by a very gradual slowdown:

$$D(t) = \begin{cases} 0 & 0 \leq t < t_r \\ D_{final} \left(1 - \exp\left(-\frac{t-t_r}{t_1}\right) \left(1 + \frac{t-t_r}{t_1}\right) \right) & t \geq t_r \end{cases} \quad (4.457)$$

$$t_1 = 0.6195 t_{rise} \quad (4.458)$$

where $D(t)$ is slip at time t , D_{final} is the final slip at the location, t_r is the slip initiation time (time when rupture reaches the location), and t_{rise} is the rise time. Because the slip time function approaches the final slip asymptotically, we use the time it takes for the slip to reach 95% of the final slip value as the rise time.

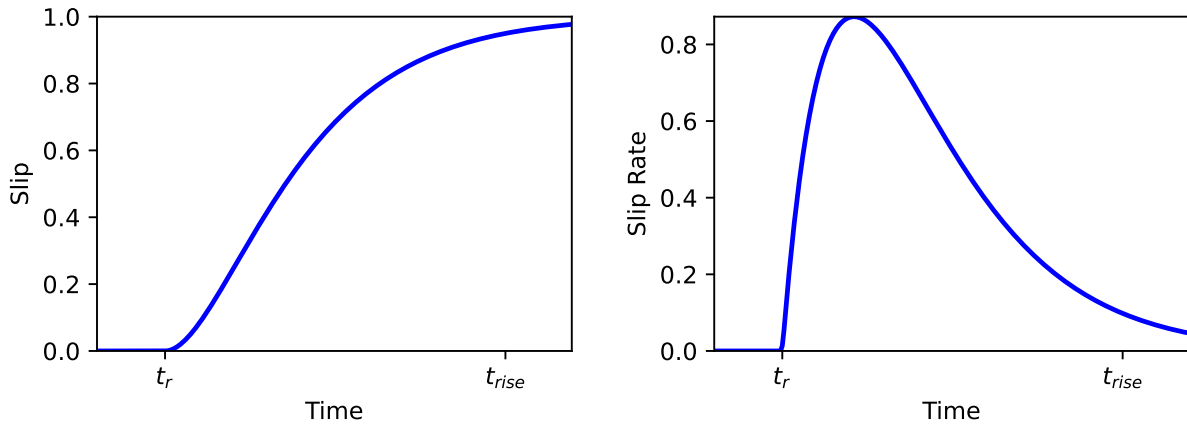


Fig. 4.23: Brune slip time function.

Table 4.37: Values in the auxiliary field spatial database for KinSrcBrune.

Subfield	Components
initiation_time (t_r)	–
rise_time (t_{rise})	–
final_slip	opening, left_lateral, reverse

➔ See also

See [KinSrcBrune Component](#) for the Pyre properties and facilities and configuration examples.

Liu-Cosine Slip Time Function (KinSrcLiuCosine)

This slip time function, proposed by Liu, Archuleta, and Hartzell for use in ground-motion modeling [Liu *et al.*, 2006], combines several cosine and sine functions together to create a slip time history with a sharp rise and gradual termination with a finite duration of slip. The evolution of slip at a point follows:

$$D(t) = \begin{cases} D_{final}C_n \left(0.7t - 0.7\frac{t_1}{\pi} \sin \frac{\pi t}{t_1} - 1.2\frac{t_1}{\pi} \left(\cos \frac{\pi t}{2t_1} - 1 \right) \right) & 0 \leq t < t_1 \\ D_{final}C_n \left(1.0t - 0.7\frac{t_1}{\pi} \sin \frac{\pi t}{t_1} + 0.3\frac{t_2}{\pi} \sin \frac{\pi(t-t_1)}{t_2} + \frac{1.2}{\pi}t_1 - 0.3t_1 \right) & t_1 \leq t < 2t_1 \\ D_{final}C_n \left(0.7 - 0.7 \cos \frac{\pi t}{t_1} + 0.6 \sin \frac{\pi t}{2t_1} \right) & 2t_1 \leq t \leq t_0 \end{cases} \quad (4.459)$$

$$C_n = \frac{\pi}{1.4\pi t_1 + 1.2t_1 + 0.3\pi t_2} \quad (4.460)$$

$$t_0 = 1.525t_{rise} \quad (4.461)$$

$$t_1 = 0.13t_0 \quad (4.462)$$

$$t_2 = t_0 - t_1 \quad (4.463)$$

where $D(t)$ is slip at time t , D_{final} is the final slip at the location, t_r is the slip initiation time (time when rupture reaches the location), and t_{rise} is the rise time.

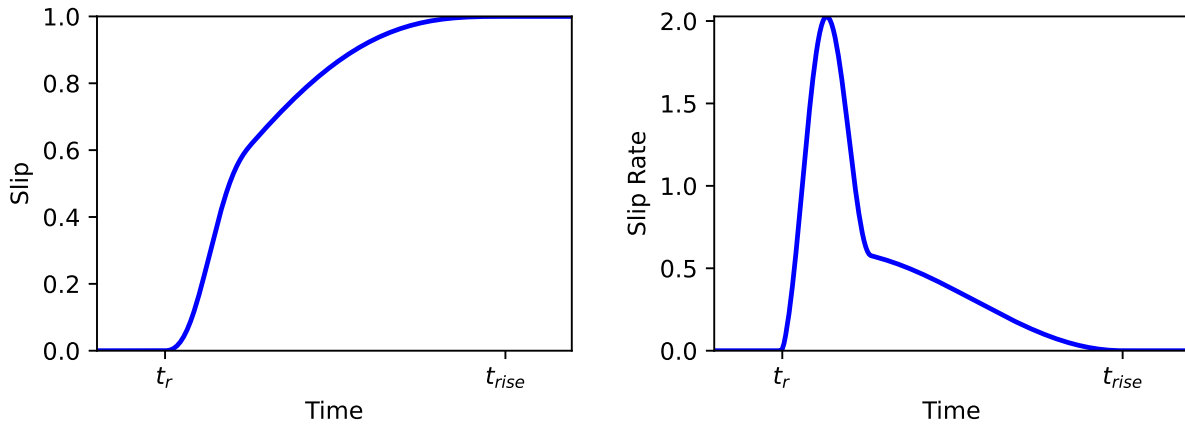


Fig. 4.24: Liu-cosine slip time function.

Table 4.38: Values in the auxiliary field spatial database for KinSrcLiuCos.

Subfield	Components
initiation_time (t_r)	–
rise_time (t_{rise})	–
final_slip	opening, left_lateral, reverse

See also

See *KinSrcLiuCos Component* for the Pyre properties and facilities and configuration examples.

User-Time History Slip Time Function (KinSrcTimeHistory)

This slip time function reads the slip time function from a data file, so it can have an arbitrary shape. The slip and slip initiation times are specified using spatial databases, so the slip time function should use a normalized amplitude (0 → 1).

Table 4.39: Values in the auxiliary field spatial database for KinSrcTimeHistory.

Subfield	Components
initiation_time	–
final_slip	opening, left_lateral, reverse

See also

See *KinSrcTimeHistory Component* for the Pyre properties and facilities and configuration examples.

Output

The derived subfield available for prescribed slip is the change in tractions on the fault surface.

 Tip

To compute the change in tractions on a locked fault, prescribe zero slip on the fault.

Table 4.40: Derived subfields that are available for output for prescribed slip.

Subfield	Components
traction_change	normal, along strike, up dip

Fault Slip Impulses (`FaultCohesiveImpulses`)

Computing static Green's functions using the `GreensFns` problem (*Green's Functions Problem (`GreensFns`)*) requires a specialized fault implementation, `FaultCohesiveImpulses`, to set up the slip impulses. The parameters controlling the slip impulses include the slip components to be used (left-lateral, reverse, or fault opening) and the amplitude of the impulses (for example, to use only a subset of a fault).

The slip auxiliary subfield specifies the patch of the fault on which to apply slip impulses. Additionally, the basis order for the representation of the slip auxiliary subfield and the solution fields controls the spatial representation of the slip impulse. The default basis order for all of these fields is 1, corresponding to a linear variation in slip, fault traction, and displacement. A basis order of 0 will approximate uniform slip within a fault cell using the finite-element representation of the displacement field and fault Lagrange multiplier. For example, a basis order of 0 for the slip auxiliary subfield and a basis order of 1 for the displacement and fault Lagrange multiplier fields will result in nearly uniform slip within the fault cell with a linear taper into the adjacent surrounding fault cells.

Impulses will be applied at any point on the fault with a slip component greater than the specified threshold. Slip impulses can be applied on any subset of the slip components, such as just the left lateral and reverse components in 3D and not fault opening.

Table 4.41: Impulse degrees of freedom

Impulse degree of freedom	Slip component
0	fault opening
1	left lateral
2	reverse (3D only)

 Pyre User Interface

See *`FaultCohesiveImpulses` Component* for the Pyre properties and facilities and configuration examples.

Output

The derived subfield available for Green's functions is the change in tractions on the fault surface.

Table 4.42: Derived subfields that are available for output for Green's functions.

Subfield	Components
traction_change	normal, along strike, up dip

4.2.8 PyLith Components

Component Implementations

Problems

TimeDependent

Time-dependent boundary value problems, including problems with a single time step (static).

GreensFns

Computation of Green's functions for static fault slip.

Materials

Elasticity

Elasticity with and without faults.

IncompressibleElasticity

Elasticity formulation for nearly incompressible materials.

Poroelasticity

Poroelasticity

Bulk rheologies

IsotropicLinearElasticity

Isotropic, linear elastic bulk rheology for Elasticity.

IsotropicLinearMaxwell

Isotropic, linear Maxwell visoelastic bulk rheology for Elasticity.

IsotropicLinearGenMaxwell

Isotropic, linear generalized Maxwell visoelastic bulk rheology for Elasticity.

IsotropicPowerLaw

Isotropic power-law visoelastic bulk rheology for Elasticity.

IsotropicLinearIncompElasticity

Isotropic, linear elastic bulk rheology for IncompressibleElasticity.

IsotropicLinearPoroelasticity

Isotropic, linear elastic bulk rheology for Poroelasticity.

Boundary conditions

DirichletTimeDependent

Time-dependent Dirichlet boundary condition.

NeumannTimeDependent

Time-dependent Neumann boundary condition.

AbsorbingDampers

Absorbing boundary condition for dynamic simulations.

Interface conditions

FaultCohesiveKin

Earthquake ruptures using prescribed slip (kinematic rupture model) in TimeDependent problems.

FaultCohesiveImpulses

Slip impulses for computing Green's functions in GreensFns problems.

Kinematic earthquake ruptures

KinSrcStep

Slip time function using a step function for use in FaultCohesiveKin.

KinSrcRamp

Slip time function using a ramp function for use in FaultCohesiveKin.

KinSrcConstRate

Slip time function using a constant rate function for use in FaultCohesiveKin.

KinSrcBrune

Slip time function using the Brune far-field time function for use in FaultCohesiveKin.

KinSrcLiuCos

Slip time function using three cosine functions for use in FaultCohesiveKin.

KinSrcTimeHistory

User-specified slip time function for use in FaultCohesiveKin.

Initial conditions

InitialConditionDomain

Initial conditions (solution values) over the domain for use in TimeDependent.

InitialConditionPatch

Initial conditions (solution values) over a patch of the domain for use in TimeDependent.

Solution fields

SolnDisp

Solution field array with displacement subfield.

SolnDispLagrange

Solution field with displacement and lagrange_multiplier_fault subfields.

SolnDispPres

Solution field with displacement and pressure subfields.

SolnDispPresLagrange

Solution field with displacement, pressure, and lagrange_multiplier_fault subfields.

SolnDispPresTracStrain

Solution field with displacement, pressure, and trace_strain subfields.

SolnDispPresTracStrainLagrange

Solution field with displacement, pressure, trace_strain and lagrange_multiplier_fault subfields.

SolnDispPresTracStrainVelPdotTdot

Solution field with displacement, pressure, trace_strain, velocity, pressure_dot, and trace_strain_dot subfields.

SolnDispPresTracStrainVelPdotTdotLagrange

Solution field with displacement, pressure, trace_strain, velocity, pressure_dot, trace_strain_dot, and lagrange_multiplier_fault subfields.

SolnDispPresVel

Solution field with displacement, pressure, and velocity subfields.

SolnDispVel

Solution field with displacement and velocity subfields.

SolnDispVellLagrange

Solution field with displacement, velocity, lagrange_multiplier_fault subfields.

Mesh initialization***Initializer***

Manager for initializing the mesh.

Serial

Initialization phases for reading a mesh in serial (default).

Parallel

Initialization phases for reading a mesh in parallel.

Convert

Initialization phases for converting a mesh form one format to another (used by ConvertMeshApp).

MeshReader

Read a finite-element mesh.

MeshWriter

Writer a finite-element mesh.

MeshReordering

Reorder vertices and cells in the finite-element mesh.

MeshDistributor

Distribute the finite-element mesh among processes.

MeshInsertInterfaces

Insert fault interfaces (cohesive cells) into the finite-element mesh.

MeshRefiner

Adjust the discretization of the finite-element mesh.

Mesh readers***OAscii***

Reader for PyLith ASCII finite-element mesh files.

MeshIOCubit

Reader for Cubit Exodus II finite-element mesh files.

MeshIOPetsc

Reader for Gmsh finite-element mesh files and meshes generated with PETSc command line arguments.

Mesh distributors***Distributor***

Distributor for distributing mesh information among processes.

Mesh refiners***RefineUniform***

Uniform refinement for decreasing discretization size by a power of 2.

Solution output

OutputSolnDomain

Output of solution subfields over the entire domain.

OutputSolnBoundary

Output of solution subfields over an external boundary of the domain.

OutputSolnPoints

Output of solution subfields at specific points in the domain.

Output writers

DataWriterHDF5

Write output to HDF5 files.

DataWriterHDF5Ext

Write output to HDF5 files with datasets in external binary files written using MPI-IO.

DataWriterVTK

Write output to VTK files.

Output triggers

OutputTriggerStep

Trigger output by skipping set number of time steps.

OutputTriggerTime

Trigger output by skipping a set time interval.

Progress monitors

ProgressMonitorStep

Trigger output of simulation progress based on number of time steps.

ProgressMonitorTime

Trigger output of simulation progress based on a time interval.

SpatialData implementations

<https://spatialdata.readthedocs.io/en/latest/user/components/index.html>

apps

ConvertMeshApp

Full name: `pylith.apps.ConvertMeshApp`

Python ConvertMeshApp application.

Pyre Facilities

- `reader=<component name>`: Reader for input mesh file.
 - **current value**: ‘meshiopetsc’, from {default}
 - **configurable as**: meshiopetsc, reader
- `writer=<component name>`: Writer for output mesh file.

- **current value:** ‘meshiopetsc’, from {default}
- **configurable as:** meshiopetsc, writer

Pyre Properties

- **check_topology=<bool>:** Check topology of imported mesh.
 - **default value:** True
 - **current value:** True, from {default}

EqInfoApp

Full name
pylith.apps.EqInfoApp

Journal name
eqinfoapp

Pyre Facilities

- **coordsys:** Coordinate system associated with mesh.
 - **current value:** ‘cscart’, from {default}
 - **configurable as:** cscart, coordsys
- **db_properties:** Spatial database for elastic properties.
 - **current value:** ‘simplifiedb’, from {default}
 - **configurable as:** simplifiedb, db_properties
- **weaver:** the pretty printer of my configuration as an XML document
 - **current value:** ‘weaver’, from {default}
 - **configurable as:** weaver

Pyre Properties

- **faults=<list>:** Array of fault names.
 - **default value:** []
 - **current value:** [], from {default}
- **filename_pattern=<str>:** Pattern for fault files.
 - **default value:** ‘output/fault_%s.h5’
 - **current value:** ‘output/fault_%s.h5’, from {default}
- **output_filename=<str>:** Filename for output.
 - **default value:** ‘eqstats.py’
 - **current value:** ‘eqstats.py’, from {default}
- **snapshot_units=<dimensional>:** Units for timestamps in array of snapshots.
 - **default value:** 1*s
 - **current value:** 1*s, from {default}

- `snapshots=<list>`: Array of timestamps for slip snapshots (-1 == last time step).
 - **default value**: [-1]
 - **current value**: [-1], from {default}
- `typos=<str>`: Specifies the handling of unknown properties and facilities
 - **default value**: 'pedantic'
 - **current value**: 'pedantic', from {default}
 - **validator**: (in ['relaxed', 'strict', 'pedantic'])

PyLithApp

Full name: `pylith.apps.PyLithApp`

Python PyLithApp application.

Pyre Facilities

- `dump_parameters=<component name>`: Dump parameters used and version information to file.
 - **current value**: 'dumpparamters', from {default}
 - **configurable as**: dumpparamters, dump_parameters
- `job=<component name>`: (no documentation available)
 - **current value**: 'job', from {default}
 - **configurable as**: job
- `launcher=<component name>`: (no documentation available)
 - **current value**: 'mpich', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/mpi/launchers/mpich.odt'} via {default}
 - **configurable as**: mpich, launcher
- `mesh_generator=<component name>`: Generates or imports the computational mesh.
 - **current value**: 'meshimporter', from {default}
 - **configurable as**: meshimporter, mesh_generator
- `metadata=<component name>`: Simulation metadata.
 - **current value**: 'metadata', from {default}
 - **configurable as**: metadata
- `petsc=<component name>`: Manager for PETSc options.
 - **current value**: 'petsc', from {default}
 - **configurable as**: petsc
- `problem=<component name>`: Computational problem to solve.
 - **current value**: 'timedependent', from {default}
 - **configurable as**: timedependent, problem
- `scheduler=<component name>`: (no documentation available)

- **current value:** ‘scheduler-none’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/schedulers/none.odt’} via {default}
- **configurable as:** scheduler-none, scheduler
- **weaver=<component name>:** the pretty printer of my configuration as an XML document
 - **current value:** ‘weaver’, from {default}
 - **configurable as:** weaver

Pyre Properties

- **include-citations=<bool>:** At end of simulation, display information on how to cite PyLith and components used.
 - **default value:** False
 - **current value:** False, from {default}
- **initialize_only=<bool>:** Stop simulation after initializing problem.
 - **default value:** False
 - **current value:** False, from {default}
- **nodes=<int>:** number of machine nodes
 - **default value:** 1
 - **current value:** 1, from {default}
- **start_python_debugger=<bool>:** Start python debugger at beginning of main().
 - **default value:** False
 - **current value:** False, from {default}
- **typos=<str>:** Specifies the handling of unknown properties and facilities
 - **default value:** ‘pedantic’
 - **current value:** ‘pedantic’, from {default}
 - **validator:** (in [‘relaxed’, ‘strict’, ‘pedantic’])

bc

AbsorbingDampers

Full name

pylith.bc.AbsorbingDampers

Journal name

absorbingdampers

Absorbing dampers boundary condition.

Implements BoundaryCondition.

Pyre Facilities

- **auxiliary_subfields:** Discretization information for auxiliary subfields.
 - **current value:** ‘auxiliary_subfields’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}

- **configurable as:** auxiliary_subfields
- db_auxiliary_field: Database for physical property parameters.
 - **current value:** ‘simplified’, from {default}
 - **configurable as:** simplified, db_auxiliary_field
- derived_subfields: Discretization of derived subfields.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, derived_subfields
- observers: Observers (e.g., output).
 - **current value:** ‘singlephysicsobserver’, from {default}
 - **configurable as:** singlephysicsobserver, observers

Pyre Properties

- field=<str>: Solution subfield associated with boundary condition.
 - **default value:** ‘displacement’
 - **current value:** ‘displacement’, from {default}
- label=<str>: Name of label identifying boundary.
 - **default value:** ‘’
 - **current value:** ‘’, from {default}
 - **validator:** <function validateLabel at 0x124bbc4c0>
- label_value=<int>: Value of label identifying boundary (tag of physical group in Gmsh files).
 - **default value:** 1
 - **current value:** 1, from {default}

Example

Example of setting AbsorbingDampers Pyre properties and facilities in a parameter file.

```
[bc]
label = boundary_xpos
field = velocity

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Material properties for absorbing boundary
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500*kg/m**3, 1.0*km/s, 1.732*km/s]

auxiliary_subfields.density.basis_order = 0
auxiliary_subfields.vp.basis_order = 0
auxiliary_subfields.vs.basis_order = 0
```

AuxSubfieldsAbsorbingDampers

Full name

pylith.bc.AuxSubfieldsAbsorbingDampers

Journal name

auxsubfieldsabsorbingdampers

Auxiliary subfields for the absorbing dampers boundary condition.

Pyre Facilities

- density: Mass density subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, density
- vp: Dilatational (P) wave speed subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, vp
- vs: Shear (S) wave speed subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, vs

Example

Example of setting AuxSubfieldsAbsorbingDampers Pyre properties and facilities in a parameter file.

```
[absorbing_dampers_auxiliary_subfields]
density.basis_order = 0
vp.basis_order = 0
vs.basis_order = 0
```

AuxSubfieldsTimeDependent

Full name

pylith.bc.AuxSubfieldsTimeDependent

Journal name

auxfieldstimedependent

Auxiliary subfields for time-dependent boundary conditions.

The boundary conditions values have the functional form:

$$f(x, t) = f_0(x) + \dot{f}_1(x)(t - t_1(x)) + f_2(x)a(t - t_2(x)) \quad (4.464)$$

The association of these functions with the auxiliary subfields is:

$f_0(x)$
initial_amplitude

$\dot{f}_1(x)$
rate_amplitude

$t_1(x)$
rate_start

$f_2(x)$
time_history_amplitude
 $t_2(x)$
time_history_start

Pyre Facilities

- **initial_amplitude**: Initial amplitude, $f_0(x)$, subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, initial_amplitude
- **rate_amplitude**: Rate amplitude, $\dot{f}_1(x)$, subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, rate_amplitude
- **rate_start_time**: Rate starting time, $t_1(x)$, subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, rate_start_time
- **time_history_amplitude**: Time history amplitude, $f_2(x)$, subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, time_history_amplitude
- **time_history_start_time**: Time history starting time, $t_2(x)$, subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, time_history_start_time

Example

Example of setting AuxSubfieldsTimeDependent Pyre properties and facilities in a parameter file.

```
[time_dependent_subfields]
initial_amplitude.basis_order = 1
rate_amplitude.basis_order = 0
rate_start_time.basis_order = 1
time_history_amplitude.basis_order = 1
time_history_start_time.basis_order = 1
```

BoundaryCondition

Full name

pylith.bc.BoundaryCondition

Journal name

boundarycondition

Abstract base class for boundary conditions.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for auxiliary subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, auxiliary_subfields
- **db_auxiliary_field**: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field
- **derived_subfields**: Discretization of derived subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, derived_subfields
- **observers**: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers

Pyre Properties

- **field=<str>**: Solution subfield associated with boundary condition.
 - **default value**: ‘displacement’
 - **current value**: ‘displacement’, from {default}
- **label=<str>**: Name of label identifying boundary.
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
 - **validator**: <function validateLabel at 0x124bbc4c0>
- **label_value=<int>**: Value of label identifying boundary (tag of physical group in Gmsh files).
 - **default value**: 1
 - **current value**: 1, from {default}

DirichletTimeDependent

Full name

pylith.bc.DirichletTimeDependent

Journal name

dirichlettimedependent

Dirichlet (prescribed values) time-dependent boundary condition.

This boundary condition sets values of a single solution subfield on a boundary. To set multiple solution subfields on a boundary, use multiple Dirichlet boundary conditions.

 See also

See [*AuxSubfieldsTimeDependent Component*](#) for the functional form of the time dependence.

Implements `BoundaryCondition`.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for auxiliary subfields.
 - **current value**: `'auxiliary_subfields'`, from `{file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}`
 - **configurable as**: `auxiliary_subfields`
- `db_auxiliary_field`: Database for physical property parameters.
 - **current value**: `'simpledb'`, from `{default}`
 - **configurable as**: `simpledb`, `db_auxiliary_field`
- `derived_subfields`: Discretization of derived subfields.
 - **current value**: `'emptybin'`, from `{default}`
 - **configurable as**: `emptybin`, `derived_subfields`
- `observers`: Observers (e.g., output).
 - **current value**: `'singlephysicsobserver'`, from `{default}`
 - **configurable as**: `singlephysicsobserver`, `observers`
- `time_history`: Time history with normalized amplitude.
 - **current value**: `'nullcomponent'`, from `{default}`
 - **configurable as**: `nullcomponent`, `time_history`

Pyre Properties

- `constrained_dof=<array>`: Array of constrained degrees of freedom (0=1st DOF, 1=2nd DOF, etc).
 - **default value**: `[]`
 - **current value**: `[]`, from `{default}`
- `field=<str>`: Solution subfield associated with boundary condition.
 - **default value**: `'displacement'`
 - **current value**: `'displacement'`, from `{default}`
- `label=<str>`: Name of label identifying boundary.
 - **default value**: `''`
 - **current value**: `''`, from `{default}`
 - **validator**: `<function validateLabel at 0x124bbc4c0>`
- `label_value=<int>`: Value of label identifying boundary (tag of physical group in Gmsh files).
 - **default value**: `1`
 - **current value**: `1`, from `{default}`

- `use_initial=<bool>`: Use initial term in time-dependent expression.
 - **default value**: True
 - **current value**: True, from {default}
- `use_rate=<bool>`: Use rate term in time-dependent expression.
 - **default value**: False
 - **current value**: False, from {default}
- `use_time_history=<bool>`: Use time history term in time-dependent expression.
 - **default value**: False
 - **current value**: False, from {default}

Example

Example of setting `DirichletTimeDependent` Pyre properties and facilities in a parameter file.

```
# Dirichlet (prescribed displacements) boundary condition constraining the x and y
↪degrees of freedom on the +y boundary.
[pylithapp.problem.bc.bc_ypos]
constrained_dof = [0, 1]
label = boundary_ypos
field = displacement

use_initial = False
use_time_history = True
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Displacement Dirichlet BC +y boundary
db_auxiliary_field.values = [time_history_amplitude_x, time_history_amplitude_y, time_
↪history_start_time]
db_auxiliary_field.data = [1.0*m, 0.0*m, 0.0]

time_history = spatialdata.spatialdb.TimeHistory
time_history.description = Impulse time history
time_history.filename = impulse.timedb
```

NeumannTimeDependent

Full name

`pylith.bc.NeumannTimeDependent`

Journal name

`neumanntimedependent`

Neumann time-dependent boundary condition. Implements `BoundaryCondition`.

This boundary condition applies a Neumann boundary condition for a single solution subfield on a boundary. To apply Neumann boundary conditions for multiple solution subfields on a boundary, use multiple Neumann boundary conditions.

Important

The components are specified in the local normal-tangential coordinate system for the boundary. Ambiguities in specifying the shear (tangential) tractions in 3D problems are resolved using the `ref_dir_1` and `ref_dir_2` properties. The first tangential direction is $\vec{z} \times \vec{r}_1$ unless these are colinear, then $\vec{r}_2$ (`ref_dir_2`) is used. The second tangential direction is $\vec{n} \times \vec{t}_1$.

➡ See also

See *AuxSubfieldsTimeDependent Component* for the functional form of the time dependence.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for auxiliary subfields.
 - **current value**: ‘auxiliary_subfields’, from {file=’/Users/baagaard/software/unix/micromamba/envs/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **configurable as**: auxiliary_subfields
- `db_auxiliary_field`: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field
- `derived_subfields`: Discretization of derived subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, derived_subfields
- `observers`: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers
- `time_history`: Time history with normalized amplitude as a function of time.
 - **current value**: ‘nullcomponent’, from {default}
 - **configurable as**: nullcomponent, time_history

Pyre Properties

- `field=<str>`: Solution subfield associated with boundary condition.
 - **default value**: ‘displacement’
 - **current value**: ‘displacement’, from {default}
- `label=<str>`: Name of label identifying boundary.
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
 - **validator**: <function validateLabel at 0x108fedc60>
- `label_value=<int>`: Value of label identifying boundary (tag of physical group in Gmsh files).
 - **default value**: 1

- **current value:** 1, from {default}
- **ref_dir_1=<list>:** First choice for reference direction to discriminate among tangential directions in 3D.
 - **default value:** [0.0, 0.0, 1.0]
 - **current value:** [0.0, 0.0, 1.0], from {default}
 - **validator:** <function validateDir at 0x108fedf80>
- **ref_dir_2=<list>:** Second choice for reference direction to discriminate among tangential directions in 3D.
 - **default value:** [0.0, 1.0, 0.0]
 - **current value:** [0.0, 1.0, 0.0], from {default}
 - **validator:** <function validateDir at 0x108d365c0>
- **scale_name=<str>:** Type of scale for nondimensionalizing Neumann boundary condition ('stress' for elasticity).
 - **default value:** 'stress'
 - **current value:** 'stress', from {default}
 - **validator:** (in ['displacement', 'stress'])
- **use_initial=<bool>:** Use initial term in time-dependent expression.
 - **default value:** True
 - **current value:** True, from {default}
- **use_rate=<bool>:** Use rate term in time-dependent expression.
 - **default value:** False
 - **current value:** False, from {default}
- **use_time_history=<bool>:** Use time history term in time-dependent expression.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting NeumannTimeDependent Pyre properties and facilities in a parameter file.

```
# Neumann (traction) boundary condition in 2D on -y boundary.
[pylithapp.problem.bc.bc_yneg]
label = boundary_yneg
field = displacement
scale_name = stress

use_initial = False
use_time_history = True
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Displacement Neumann BC +y boundary
db_auxiliary_field.values = [time_history_amplitude_tangential, time_history_amplitude_
↪normal, time_history_start_time]
db_auxiliary_field.data = [2.0*MPa, -1.0*MPa, 0.0]

time_history = spatialdata.spatialdb.TimeHistory
```

(continues on next page)

(continued from previous page)

```
time_history.description = Impulse time history
time_history.filename = impulse.timedb
```

ZeroDB

Full name

pylith.bc.ZeroDB

Journal name

zerodb

Special case of a UniformDB spatial database with uniform zero initial amplitude values for degrees of freedom.

Implements SpatialDB.

Pyre Properties

- data=<list>: Values in spatial database.
 - **default value:** []
 - **current value:** [], from {default}
- description=<str>: Description for database.
 - **default value:** ‘
 - **current value:** ‘’, from {default}
 - **validator:** <function validateDescription at 0x124922820>
- label=<str>: Label for ZeroDB spatial database.
 - **default value:** ‘Zero initial amplitude spatial database.’
 - **current value:** ‘Zero initial amplitude spatial database.’, from {default}
- values=<list>: Names of values in spatial database.
 - **default value:** []
 - **current value:** [], from {default}

Example

Example of setting ZeroDB Pyre properties and facilities in a parameter file.

```
# Zero displacement boundary condition constraining the y degree of freedom on the -y_
↪boundary.
[pylithapp.problem.bc.bc_yneg]
constrained_dof = [1]
label = boundary_yneg
field = displacement

db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet displacement boundary condition on the -y_
↪boundary
```

faults

AuxSubfieldsFault

Full name`pylith.faults.AuxSubfieldsFault`**Journal name**`auxsubfieldfault`

Auxiliary subfields associated with a fault.

Pyre Facilities

- `slip`: Slip subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, slip

Example

Example of setting AuxSubfieldsFault Pyre properties and facilities in a parameter file.

```
# We set the basis order to represent linear variations in the slip subfield.
[pylithapp.problem.interfaces.fault.auxiliary_fields]
slip.basis_order = 1
```

FaultCohesive

Full name`pylith.faults.FaultCohesive`**Journal name**`fault`

Abstract base class for a fault surface implemented with cohesive cells.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for auxiliary subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, auxiliary_subfields
- `db_auxiliary_field`: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field
- `derived_subfields`: Discretization of derived subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, derived_subfields
- `observers`: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers

Pyre Properties

- `edge=<str>`: Name of label identifier for buried fault edges.
 - **default value:** ‘
 - **current value:** ‘’, from {default}
- `edge_value=<int>`: Value of label identifier for buried fault edges.
 - **default value:** 1
 - **current value:** 1, from {default}
- `label=<str>`: Name of label identifier for fault.
 - **default value:** ‘
 - **current value:** ‘’, from {default}
 - **validator:** <function validateLabel at 0x124b02ca0>
- `label_value=<int>`: Value of label identifier for fault.
 - **default value:** 1
 - **current value:** 1, from {default}
- `ref_dir_1=<list>`: First choice for reference direction to discriminate among tangential directions in 3D; should roughly match the up-dir direction.
 - **default value:** [0.0, 0.0, 1.0]
 - **current value:** [0.0, 0.0, 1.0], from {default}
 - **validator:** <function validateDir at 0x108bf6840>
- `ref_dir_2=<list>`: Second choice for reference direction to discriminate among tangential directions in 3D; should roughly match the fault normal direction.
 - **default value:** [0.0, 1.0, 0.0]
 - **current value:** [0.0, 1.0, 0.0], from {default}
 - **validator:** <function validateDir at 0x124b02dc0>

FaultCohesiveImpulses

Full name

`pylith.faults.FaultCohesiveImpulses`

Journal name

`faultcohesiveimpulses`

Fault surface with slip impulses for Green’s functions implemented with cohesive cells.

The components

Implements `FaultCohesiveKin`.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for auxiliary subfields.
 - **current value:** ‘auxiliary_subfields’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’ }

- **configurable as:** auxiliary_subfields
- **db_auxiliary_field:** (no documentation available)
 - **current value:** ‘nullcomponent’, from {default}
 - **configurable as:** nullcomponent, db_auxiliary_field
- **derived_subfields:** Discretization of derived subfields.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, derived_subfields
- **observers:** Observers (e.g., output).
 - **current value:** ‘singlephysicsobserver’, from {default}
 - **configurable as:** singlephysicsobserver, observers

Pyre Properties

- **edge=<str>:** Name of label identifier for buried fault edges.
 - **default value:** ‘
 - **current value:** ‘’, from {default}
- **edge_value=<int>:** Value of label identifier for buried fault edges.
 - **default value:** 1
 - **current value:** 1, from {default}
- **impulse_dof=<list>:** Indices of impulse components; 0=fault opening, 1=left lateral, 2=reverse (3D only).
 - **default value:** []
 - **current value:** [], from {default}
 - **validator:** <function validateDOF at 0x124b029d0>
- **label=<str>:** Name of label identifier for fault.
 - **default value:** ‘
 - **current value:** ‘’, from {default}
 - **validator:** <function validateLabel at 0x124b02ca0>
- **label_value=<int>:** Value of label identifier for fault.
 - **default value:** 1
 - **current value:** 1, from {default}
- **ref_dir_1=<list>:** First choice for reference direction to discriminate among tangential directions in 3D; should roughly match the up-dir direction.
 - **default value:** [0.0, 0.0, 1.0]
 - **current value:** [0.0, 0.0, 1.0], from {default}
 - **validator:** <function validateDir at 0x108bf6840>
- **ref_dir_2=<list>:** Second choice for reference direction to discriminate among tangential directions in 3D; should roughly match the fault normal direction.
 - **default value:** [0.0, 1.0, 0.0]

- **current value:** [0.0, 1.0, 0.0], from {default}
- **validator:** <function validateDir at 0x124b02dc0>
- **threshold=<dimensional>:** Threshold for non-zero amplitude.
 - **default value:** 1e-06*m
 - **current value:** 1e-06*m, from {default}
 - **validator:** (greater than or equal to 0*m)

Example

Example of setting FaultCohesiveImpulses Pyre properties and facilities in a parameter file.

```
[pylithapp.greensfns]
interfaces = [fault]
interfaces.fault = pylith.faults.FaultCohesiveImpulses

[pylithapp.greensfns.interfaces.fault]
label = fault
label_value = 20

# Impulses for left-lateral slip (dof=1)
impulse_dof = [1]
threshold = 0.5

# Create impulses at all points on the fault by specifying a uniform amplitude of 1.0.
# Impulses will be applied at any location with a slip component greater than the
↪ threshold.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Slip impulse amplitude
db_auxiliary_field.values = [slip_left_lateral, slip_opening]
db_auxiliary_field.data = [1.0*m, 0.0*m]

# Represent the impulse as a linear variation in slip centered on each point.
auxiliary_subfields.slip.basis_order = 1
```

FaultCohesiveKin

Full name

pylith.faults.FaultCohesiveKin

Journal name

faultcohesivekin

Fault surface with kinematic (prescribed) slip implemented with cohesive cells.

The fault may have an arbitrary number of kinematic sources for coseismic slip and creep. They are superimposed at each time step to create the prescribed slip on the fault.

Implements FaultCohesive.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for auxiliary subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, auxiliary_subfields
- **db_auxiliary_field**: (no documentation available)
 - **current value**: ‘nullcomponent’, from {default}
 - **configurable as**: nullcomponent, db_auxiliary_field
- **derived_subfields**: Discretization of derived subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, derived_subfields
- **eq_ruptures**: Kinematic earthquake sources information.
 - **current value**: ‘singlerupture’, from {default}
 - **configurable as**: singlerupture, eq_ruptures
- **observers**: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers

Pyre Properties

- **edge=<str>**: Name of label identifier for buried fault edges.
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
- **edge_value=<int>**: Value of label identifier for buried fault edges.
 - **default value**: 1
 - **current value**: 1, from {default}
- **label=<str>**: Name of label identifier for fault.
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
 - **validator**: <function validateLabel at 0x124b02ca0>
- **label_value=<int>**: Value of label identifier for fault.
 - **default value**: 1
 - **current value**: 1, from {default}
- **ref_dir_1=<list>**: First choice for reference direction to discriminate among tangential directions in 3D; should roughly match the up-dir direction.
 - **default value**: [0.0, 0.0, 1.0]
 - **current value**: [0.0, 0.0, 1.0], from {default}
 - **validator**: <function validateDir at 0x108bf6840>

- `ref_dir_2=<list>`: Second choice for reference direction to discriminate among tangential directions in 3D; should roughly match the fault normal direction.
 - **default value**: [0.0, 1.0, 0.0]
 - **current value**: [0.0, 1.0, 0.0], from {default}
 - **validator**: <function validateDir at 0x124b02dc0>

Example

Example of setting FaultCohesiveKin Pyre properties and facilities in a parameter file.

```
# Specify prescribed slip on a fault via two earthquakes in a 2D domain.
[pylithapp.problem.interfaces.fault]
label = fault

observers.observer.data_fields = [slip, traction_change]

# Two earthquakes with different slip time functions.
eq_ruptures = [quake10, quake50]
quake10 = pylith.faults.KinSrcBrune
quake50 = pylith.faults.KinSrcLiuCosine

# Rupture parameters for the first earthquake.
[pylithapp.problem.interfaces.fault.eq_ruptures.quake10]
origin_time = 10*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -2.0*m, 0.0*m]

# Rupture parameters for the second earthquake.
[pylithapp.problem.interfaces.fault.eq_ruptures.quake50]
origin_time = 50*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -1.0*m, 0.0*m]
```

KinSrc

Full name

pylith.faults.KinSrc

Journal name

kinsrc

Abstract base class for a prescribed slip source.

Pyre Facilities

- `db_auxiliary_field`: Database for slip time function parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field

Pyre Properties

- `origin_time=<dimensional>`: Origin time for slip source.
 - **default value**: 0*s
 - **current value**: 0*s, from {default}

KinSrcBrune

Full name

`pylith.faults.KinSrcBrune`

Journal name

`kinsrcbrune`

Brune [1970] far-field slip time function.

Implements `KinSrc`.

Pyre Facilities

- `db_auxiliary_field`: Database for slip time function parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field

Pyre Properties

- `origin_time=<dimensional>`: Origin time for slip source.
 - **default value**: 0*s
 - **current value**: 0*s, from {default}

Example

Example of setting `KinSrcBrune` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
origin_time = 10*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Brune far-field slip time function auxiliary field_
↪ spatial database
db_auxiliary_field.values = [initiation_time, rise_time, final_slip_left_lateral, final_
↪ slip opening]
db_auxiliary_field.data = [0.0*s, 3.0*s, -2.0*m, 0.0*m]
```

KinSrcConstRate

Full name

pylith.faults.KinSrcConstRate

Journal name

kinsrcconstrate

Constant slip rate slip time function.

Implements KinSrc.

Pyre Facilities

- **db_auxiliary_field**: Database for slip time function parameters.
 - **current value**: 'simplifiedb', from {default}
 - **configurable as**: simplifiedb, db_auxiliary_field

Pyre Properties

- **origin_time=<dimensional>**: Origin time for slip source.
 - **default value**: 0*s
 - **current value**: 0*s, from {default}

Example

Example of setting KinSrcConstRate Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
origin_time = 100*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Constant slip rate slip time function auxiliary field.
↪spatial database
db_auxiliary_field.values = [initiation_time, slip_rate_left_lateral, slip_rate_opening]
db_auxiliary_field.data = [0.0*s, -2.0*mm/year, 0.0*mm/year]
```

KinSrcLiuCos

Full name

pylith.faults.KinSrcLiuCos

Journal name

kinsrcliucos

Liu *et al.* [2006] cosine-sine slip time function.

Implements KinSrc.

Pyre Facilities

- **db_auxiliary_field**: Database for slip time function parameters.
 - **current value**: 'simplifiedb', from {default}
 - **configurable as**: simplifiedb, db_auxiliary_field

Pyre Properties

- `origin_time=<dimensional>`: Origin time for slip source.
 - **default value:** 0*s
 - **current value:** 0*s, from {default}

Example

Example of setting KinSrcLiuCos Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
origin_time = 10*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Liu cosine-sine slip time function auxiliary field_
↳spatial database
db_auxiliary_field.values = [initiation_time, rise_time, final_slip_left_lateral, final_
↳slip_opening]
db_auxiliary_field.data = [0.0*s, 3.0*s, -2.0*m, 0.0*m]
```

KinSrcRamp

Full name

pylith.faults.KinSrcRamp

Journal name

kinsrcramp

Linear ramp slip time function.

Implements KinSrc.

Pyre Facilities

- `db_auxiliary_field`: Database for slip time function parameters.
 - **current value:** 'simplifiedb', from {default}
 - **configurable as:** simplifiedb, db_auxiliary_field

Pyre Properties

- `origin_time=<dimensional>`: Origin time for slip source.
 - **default value:** 0*s
 - **current value:** 0*s, from {default}

Example

Example of setting KinSrcRamp Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
origin_time = 10*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
```

(continues on next page)

(continued from previous page)

```
db_auxiliary_field.description = Ramp slip time function auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, rise_time, final_slip_left_lateral, final_
↪slip_opening]
db_auxiliary_field.data = [0.0*s, 3.0*s, -2.0*m, 0.0*m]
```

KinSrcStep

Full name

pylith.faults.KinSrcStep

Journal name

kinsrcstep

Step slip time function.

Implements KinSrc.

Pyre Facilities

- `db_auxiliary_field`: Database for slip time function parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, `db_auxiliary_field`

Pyre Properties

- `origin_time=<dimensional>`: Origin time for slip source.
 - **default value**: 0*s
 - **current value**: 0*s, from {default}

Example

Example of setting KinSrcStep Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
origin_time = 10*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Step slip time function auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪opening]
db_auxiliary_field.data = [0.0*s, -2.0*m, 0.0*m]
```

KinSrcTimeHistory

Full name

pylith.faults.KinSrcTimeHistory

Journal name

kinsrcetimehistory

Time history slip time function.

The slip time function is given in a TimeHistory database.

Implements KinSrc.

Pyre Facilities

- `db_auxiliary_field`: Database for slip time function parameters.
 - **current value**: ‘simplifiedb’, from {default}
 - **configurable as**: simplifiedb, db_auxiliary_field
- `time_history`: Time history with normalized amplitude as a function of time.
 - **current value**: ‘timehistory’, from {default}
 - **configurable as**: timehistory, time_history

Pyre Properties

- `origin_time=<dimensional>`: Origin time for slip source.
 - **default value**: 0*s
 - **current value**: 0*s, from {default}

Example

Example of setting KinSrcTimeHistory Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
origin_time = 10*year

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Time history slip time function auxiliary field spatial_
↪database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪opening]
db_auxiliary_field.data = [0.0*s, -2.0*m, 0.0*m]

time_history.description = Slip time function time history
time_history.filename = slipfn.timedb
```

SingleRupture

Full name

pylith.faults.SingleRupture

Journal name

singlerupture

Kinematic slip source container with one source.

See also

See *FaultCohesiveKin Component*.

Pyre Facilities

- rupture: Kinematic slip source (for example, an earthquake rupture) for fault.
 - **current value:** ‘kinsrcstep’, from {default}
 - **configurable as:** kinsrcstep, rupture

initializers

Convert

Full name

pylith.initializers.Convert

Journal name

convert_phases

Mesh initialization phases for reordering and converting to a different format.

 See also

See *Initializer Component*.

Pyre Facilities

- read_mesh: Read mesh.
 - **current value:** ‘mesh_reader’, from {default}
 - **configurable as:** mesh_reader, read_mesh
- reorder_mesh: Reorder mesh using reverse Cuthill-McKee algorithm.
 - **current value:** ‘reorder_mesh’, from {default}
 - **configurable as:** reorder_mesh
- write_mesh: Write mesh.
 - **current value:** ‘mesh_writer’, from {default}
 - **configurable as:** mesh_writer, write_mesh

InitializePhase

Full name

pylith.initializers.InitializePhase

Journal name

initialize_phase

Abstract base class for mesh initialization phases.

Initializer

Full name

pylith.initializers.Initializer

Journal name

mesh_initializer

Manager for reading and setting up a finite-element mesh.

Pyre Facilities

- phases: Phases for mesh initialization.
 - **current value:** ‘serial_phases’, from {default}
 - **configurable as:** serial_phases, phases

Example

Example of setting Initializer Pyre properties and facilities in a parameter file.

```
[pylithapp.mesh_initializer]
phases = pylith.initializers.Serial

# Equivalent manual construction of Serial phases
phases = [read_mesh, reorder_mesh, distribute_mesh, insert_interfaces, refine_mesh]
read_mesh = pylith.initializers.MeshReader
reorder_mesh = pylith.initializers.MeshReordering
distribute_mesh = pylith.initializers.MeshDistributor
insert_interfaces = pylith.initializers.InsertInterfaces
refine_mesh = pylith.initializers.MeshRefiner
```

MeshDistributor

Full name

pylith.initializers.MeshDistributor

Journal name

mesh_refiner

Distribute mesh.

Implements InitializePhase.

Pyre Facilities

- distributor: Mesh distributor.
 - **current value:** ‘mesh_distributor’, from {default}
 - **configurable as:** mesh_distributor, distributor

Example

Example of setting MeshDistributor Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.distribute_mesh]
distributor = pylith.topology.Distributor
```

MeshInsertInterfaces

Full name

pylith.initializers.MeshInsertInterfaces

Journal name`mesh_insert_interfaces`

Insert fault interfaces using PETSc transform operation.

Implements `InitializePhase`.

Example

Example of setting `MeshInsertInterfaces` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.insert_interfaces]
# No parameters
```

MeshReader**Full name**`pylith.initializers.MeshReader`**Journal name**`mesh_reader`

Read mesh.

Implements `InitializePhase`.

Pyre Facilities

- reader: Mesh reader.
 - **current value:** ‘meshiopetsc’, from {default}
 - **configurable as:** meshiopetsc, reader

Example

Example of setting `MeshReader` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.read_mesh]
reader = pylith.meshio.MeshIOPetsc
```

MeshRefiner**Full name**`pylith.initializers.MeshRefiner`**Journal name**`mesh_refiner`

Refine mesh.

Implements `InitializePhase`.

Pyre Facilities

- refiner: Mesh refiner.
 - **current value:** ‘refineuniform’, from {default}
 - **configurable as:** refineuniform, refiner

Example

Example of setting MeshRefiner Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.refine_mesh]
refiner = pylith.topology.RefineUniform
```

MeshReordering

Full name

pylith.initializers.MeshReordering

Journal name

reorder_mesh

Reorder mesh using the reverse Cuthill-McKee algorithm.

Implements InitializePhase.

Example

Example of setting MeshReordering Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.reorder_mesh]
# No parameters
```

MeshWriter

Full name

pylith.initializers.MeshWriter

Journal name

mesh_writer

Write mesh.

Implements InitializePhase.

Pyre Facilities

- writer: Mesh writer.
 - **current value:** ‘meshiopetsc’, from {default}
 - **configurable as:** meshiopetsc, writer

Example

Example of setting MeshWriter Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.write_mesh]
writer = pylith.meshio.MeshIOPetsc
```

Parallel

Full name`pylith.initializers.Parallel`**Journal name**`parallel_phases`

Mesh initialization phases for reading in parallel.

 See also

See *Initializer Component*.

Pyre Facilities

- `distribute_mesh`: Distribute mesh among processes.
 - **current value**: ‘mesh_refiner’, from {default}
 - **configurable as**: mesh_refiner, distribute_mesh
- `insert_interfaces`: Insert interfaces using PETSc mesh transform operation.
 - **current value**: ‘mesh_insert_interfaces’, from {default}
 - **configurable as**: mesh_insert_interfaces, insert_interfaces
- `read_mesh`: Read mesh in parallel.
 - **current value**: ‘mesh_reader’, from {default}
 - **configurable as**: mesh_reader, read_mesh
- `refine_mesh`: Refine mesh.
 - **current value**: ‘mesh_refiner’, from {default}
 - **configurable as**: mesh_refiner, refine_mesh

Example

Example of setting Parallel Pyre properties and facilities in a parameter file.

```
# Equivalent manual construction
phases = [read_mesh, distribute_mesh, insert_interfaces, refine_mesh]
read_mesh = pylith.initializers.MeshReader
distribute_mesh = pylith.initializers.MeshDistributor
insert_interfaces = pylith.initializers.InsertInterfaces
refine_mesh = pylith.initializers.MeshRefiner
```

Serial

Full name`pylith.initializers.Serial`**Journal name**`serial_phases`

Mesh initialization phases for reading in serial.

 See also

See *Initializer Component*.

Pyre Facilities

- `distribute_mesh`: Distribute mesh among processes.
 - **current value**: ‘mesh_refiner’, from {default}
 - **configurable as**: mesh_refiner, distribute_mesh
- `insert_interfaces`: Insert interfaces using PETSc mesh transform operation.
 - **current value**: ‘mesh_insert_interfaces’, from {default}
 - **configurable as**: mesh_insert_interfaces, insert_interfaces
- `read_mesh`: Read mesh in serial.
 - **current value**: ‘mesh_reader’, from {default}
 - **configurable as**: mesh_reader, read_mesh
- `refine_mesh`: Refine mesh.
 - **current value**: ‘mesh_refiner’, from {default}
 - **configurable as**: mesh_refiner, refine_mesh
- `reorder_mesh`: Reorder mesh using reverse Cuthill-McKee algorithm.
 - **current value**: ‘reorder_mesh’, from {default}
 - **configurable as**: reorder_mesh

Example

Example of setting Serial Pyre properties and facilities in a parameter file.

```
# Equivalent manual construction
phases = [read_mesh, distribute_mesh, insert_interfaces, refine_mesh]
read_mesh = pylith.initializers.MeshReader
distribute_mesh = pylith.initializers.MeshDistributor
insert_interfaces = pylith.initializers.InsertInterfaces
refine_mesh = pylith.initializers.MeshRefiner
```

materials

AuxSubfieldsElasticity

Full name

pylith.materials.AuxSubfieldsElasticity

Journal name

auxsubfieldselasticity

Auxiliary subfields associated with the elasticity equation.

Setting the parameters for a subfield does not turn on its use. The *Elasticity Component* has flags for including or excluding terms in the elasticity equation.

Pyre Facilities

- `body_force`: Body force subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `body_force`
- `density`: Density subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `density`
- `gravitational_acceleration`: Gravitational acceleration subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `gravitational_acceleration`

Example

Example of setting `AuxSubfieldsElasticity` Pyre properties and facilities in a parameter file.

```
# We set the basis order to represent linear variations in the density and body
# force subfields and a uniform gravitational acceleration subfield.
[pylithapp.problem.materials.mat_elastic.auxiliary_fields]
density.basis_order = 1
body_force.basis_order = 1
gravitational_acceleration.basis_order = 0
```

AuxSubfieldsIsotropicLinearElasticity

Full name

`pylith.materials.AuxSubfieldsIsotropicLinearElasticity`

Journal name

`auxsubfieldsisotropiclinearelasticity`

Auxiliary subfields associated with the isotropic linear elastic bulk rheology.

Important

The auxiliary subfields (internal representation of material properties) do not necessarily match the values in the spatial database. For example, the spatial database uses density, V_p , and V_s instead of density, shear modulus, and bulk modulus because that is how they are usually characterized in seismic velocity models. PyLith converts the values provided by the user in a spatial database to the internal representation stored in the auxiliary field.

Pyre Facilities

- `bulk_modulus`: Bulk modulus subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `bulk_modulus`
- `reference_strain`: Reference strain subfield.
 - **current value**: ‘subfield’, from {default}

- **configurable as:** subfield, reference_strain
- reference_stress: Reference stress subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, reference_stress
- shear_modulus: Shear modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, shear_modulus

Example

Example of setting AuxSubfieldsIsotropicLinearElasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_elastic.rheology.auxiliary_fields]
shear_modulus.basis_order = 1
bulk_modulus.basis_order = 1
reference_stress.basis_order = 0
reference_strain.basis_order = 0
```

AuxSubfieldsIsotropicLinearGenMaxwell

Full name

pylith.materials.AuxSubfieldsIsotropicLinearGenMaxwell

Journal name

auxfieldsisotropiclineargenmaxwell

Auxiliary subfields associated with the isotropic generalized Maxwell viscoelastic bulk rheology.

Pyre Facilities

- bulk_modulus: Bulk modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, bulk_modulus
- maxwell_time: Maxwell time subfield for 3 Maxwell elements.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, maxwell_time
- reference_strain: Reference strain subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, reference_strain
- reference_stress: Reference stress subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, reference_stress
- shear_modulus: Shear modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, shear_modulus

- `shear_modulus_ratio`: Shear modulus ratio subfield for 3 Maxwell elements.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `shear_modulus_ratio`
- `total_strain`: Total strain subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `total_strain`
- `viscous_strain`: Viscous strain subfield for 3 Maxwell elements.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `viscous_strain`

Example

Example of setting `AuxSubfieldsIsotropicLinearGenMaxwell` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_genmaxwell.rheology.auxiliary_fields]
shear_modulus.basis_order = 1
bulk_modulus.basis_order = 1
maxwell_time.basis_order = 1
shear_modulus_ratio.basis_order = 1
total_strain.basis_order = 1
viscous_strain.basis_order = 1
reference_stress.basis_order = 0
reference_strain.basis_order = 0
```

AuxSubfieldsIsotropicLinearMaxwell

Full name

`pylith.materials.AuxSubfieldsIsotropicLinearMaxwell`

Journal name

`auxfieldsisotropiclinearmaxwell`

Auxiliary subfields associated with the isotropic linear Maxwell viscoelastic bulk rheology.

Pyre Facilities

- `bulk_modulus`: Bulk modulus subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `bulk_modulus`
- `maxwell_time`: Maxwell time subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `maxwell_time`
- `reference_strain`: Reference strain subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `reference_strain`
- `reference_stress`: Reference stress subfield.

- **current value:** ‘subfield’, from {default}
- **configurable as:** subfield, reference_stress
- shear_modulus: Shear modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, shear_modulus
- total_strain: Total strain subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, total_strain
- viscous_strain: Viscous strain subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, viscous_strain

Example

Example of setting AuxSubfieldsIsotropicLinearMaxwell Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_maxwell.rheology.auxiliary_fields]
shear_modulus.basis_order = 1
bulk_modulus.basis_order = 1
maxwell_time.basis_order = 1
total_strain.basis_order = 1
viscous_strain.basis_order = 1
reference_stress.basis_order = 0
reference_strain.basis_order = 0
```

AuxSubfieldsIsotropicLinearPoroelectricity

Full name

pylith.materials.AuxSubfieldsIsotropicLinearPoroelectricity

Journal name

auxfieldsisotropiclinearporelasticity

Auxiliary subfields associated with the isotropic linear poroelastic bulk rheology.

Pyre Facilities

- biot_coefficient: Biot coefficient subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, biot_coefficient
- biot_modulus: Biot modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, biot_modulus
- drained_bulk_modulus: Drained bulk modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, drained_bulk_modulus

- **isotropic_permeability**: Isotropic permeability subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, isotropic_permeability
- **reference_strain**: Reference strain subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, reference_strain
- **reference_stress**: Reference stress subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, reference_stress
- **shear_modulus**: Shear modulus subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, shear_modulus
- **tensor_permeability**: Tensor permeability subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, tensor_permeability

Example

Example of setting `AuxSubfieldsIsotropicLinearPoroelasticity` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_poroelastic.rheology.auxiliary_fields]
shear_modulus.basis_order = 1
biot_coefficient.basis_order = 0
isotropic_permeability.basis_order = 0
drained_bulk_modulus.basis_order = 1
biot_modulus.basis_order = 1
```

AuxSubfieldsIsotropicPowerLaw

Full name

`pylith.materials.AuxSubfieldsIsotropicPowerLaw`

Journal name

`auxfieldsisotropicpowerlaw`

Auxiliary subfields associated with the isotropic power-law viscoelastic bulk rheology.

Pyre Facilities

- **bulk_modulus**: Bulk modulus subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, bulk_modulus
- **deviatoric_stress**: Deviatoric stress subfield.
 - **current value**: ‘subfield’, from {default}

- **configurable as:** subfield, deviatoric_stress
- power_law_exponent: Power-law exponent subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, power_law_exponent
- power_law_reference_strain_rate: Power-law reference strain rate subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, power_law_reference_strain_rate
- power_law_reference_stress: Power-law reference stress subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, power_law_reference_stress
- reference_strain: Reference strain subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, reference_strain
- reference_stress: Reference stress subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, reference_stress
- shear_modulus: Shear modulus subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, shear_modulus
- viscous_strain: Viscous strain subfield.
 - **current value:** ‘subfield’, from {default}
 - **configurable as:** subfield, viscous_strain

Example

Example of setting AuxSubfieldsIsotropicPowerLaw Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_powerlaw.rheology.auxiliary_fields]
shear_modulus.basis_order = 1
bulk_modulus.basis_order = 1
power_law_reference_strain_rate = 1
power_law_reference_stress = 1
power_law_exponent.basis_order = 1
viscous_strain.basis_order = 1
deviatoric_stress.basis_order = 1
reference_stress.basis_order = 0
reference_strain.basis_order = 0
```

AuxSubfieldsPoroelasticity

Full name

pylith.materials.AuxSubfieldsPoroelasticity

Journal name

auxsubfieldsporoelasticity

Auxiliary subfields associated with the poroelasticity equation.

Setting the parameters for a subfield does not turn on its use. The *Poroelasticity Component* has flags for including or excluding terms in the poroelasticity equation.

Pyre Facilities

- **body_force**: Body force subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, body_force
- **fluid_density**: Fluid density subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, fluid_density
- **fluid_viscosity**: Fluid viscosity subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, fluid_viscosity
- **gravitational_acceleration**: Gravitational acceleration subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, gravitational_acceleration
- **porosity**: Porosity subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, porosity
- **solid_density**: Solid density subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, solid_density
- **source_density**: Source density subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, source_density

Example

Example of setting AuxSubfieldsPoroelasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_poroelastic.auxiliary_fields]
porosity.basis_order = 1
solid_density.basis_order = 1
fluid_density.basis_order = 0
fluid_viscosity.basis_order = 1
body_force.basis_order = 1
source_density.basis_order = 1
gravitational_acceleration.basis_order = 0
```

DerivedSubfieldsElasticity

Full name

pylith.materials.DerivedSubfieldsElasticity

Journal name

derivedsubfieldselasticity

Derived subfields associated with the elasticity equation.

For elastic materials these derived subfields are available for output in addition to the solution and auxiliary subfields.

Pyre Facilities

- `cauchy_strain`: Cauchy strain subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `cauchy_strain`
- `cauchy_stress`: Cauchy stress subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `cauchy_stress`

Example

Example of setting DerivedSubfieldsElasticity Pyre properties and facilities in a parameter file.

```
# The basis order for stress and strain should be at least 1 less than the basis order
↪ for displacement.
[pylithapp.problem.materials.mat_elastic.derived_subfields]
cauchy_stress.basis_order = 0
cauchy_strain.basis_order = 0
```

DerivedSubfieldsPoroelasticity

Full name

pylith.materials.DerivedSubfieldsPoroelasticity

Journal name

derivedsubfieldsporoelasticity

Derived subfields associated with the poroelasticity equation.

For poroelastic materials these derived subfields are available for output in addition to the solution and auxiliary subfields.

Pyre Facilities

- `bulk_density`: Bulk density subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `bulk_density`
- `cauchy_strain`: Cauchy strain subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `cauchy_strain`

- `cauchy_stress`: Cauchy stress subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `cauchy_stress`
- `water_content`: Water content subfield.
 - **current value**: ‘subfield’, from {default}
 - **configurable as**: subfield, `water_content`

Example

Example of setting `DerivedSubfieldsPoroelasticity` Pyre properties and facilities in a parameter file.

```
# The basis order for stress and strain should be at least 1 less than the basis order.
↪ for displacement.
[pylithapp.problem.materials.mat_elastic.derived_subfields]
cauchy_stress.basis_order = 0
cauchy_strain.basis_order = 0
bulk_density.basis_order = 1
water_content.basis_order = 1
```

Elasticity

Full name

`pylith.materials.Elasticity`

Journal name

`elasticity`

Material behavior governed by the elasticity equation.

Implements `Material`.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for auxiliary subfields.
 - **current value**: ‘auxiliary_subfields’, from {file=’/Users/baagaard/software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **configurable as**: `auxiliary_subfields`
- `bulk_rheology`: Bulk rheology for elastic material.
 - **current value**: ‘isotropiclinearelasticity’, from {default}
 - **configurable as**: `isotropiclinearelasticity`, `bulk_rheology`
- `db_auxiliary_field`: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: `simpledb`, `db_auxiliary_field`
- `derived_subfields`: Discretization of derived subfields.
 - **current value**: ‘derived_subfields’, from {file=’/Users/baagaard/software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}

- **configurable as:** derived_subfields
- **observers:** Observers (e.g., output).
 - **current value:** ‘singlephysicsobserver’, from {default}
 - **configurable as:** singlephysicsobserver, observers

Pyre Properties

- **description=<str>:** Descriptive label for material (deprecated in v5.0; will be removed in v6.0).
 - **default value:** ‘’
 - **current value:** ‘’, from {default}
 - **validator:** <function validateDescription at 0x10480b2e0>
- **label=<str>:** Name of label for material. Currently only ‘material-id’ is allowed.
 - **default value:** ‘material-id’
 - **current value:** ‘material-id’, from {default}
 - **validator:** (in [‘material-id’])
- **label_value=<int>:** Value of label for material.
 - **default value:** 1
 - **current value:** 1, from {default}
- **use_body_force=<bool>:** Include body force term in elasticity equation.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting Elasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_elastic]
label_value = 4
use_body_force = False
bulk_rheology = pylith.materials.IsotropicLinearElasticity

auxiliary_subfields.density.basis_order = 0
derived_subfields.cauchy_stress.basis_order = 1
derived_subfields.cauchy_strain.basis_order = 1
```

Homogeneous

Full name

pylith.materials.Homogeneous

Journal name

homogeneous

Materials container with one material.

 See also

See *Problems Component*.

Pyre Facilities

- **material**: Material in problem.
 - **current value**: ‘isotropiclinearelasticity’, from {default}
 - **configurable as**: isotropiclinearelasticity, material

IncompressibleElasticity

Full name

pylith.materials.IncompressibleElasticity

Journal name

incompressibleelasticity

Material behavior governed by the elasticity equation.

Implements Material.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for auxiliary subfields.
 - **current value**: ‘auxiliary_subfields’, from {file=’/Users/baagaard/software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **configurable as**: auxiliary_subfields
- **bulk_rheology**: Bulk rheology for elastic material.
 - **current value**: ‘isotropiclinearincompelasticity’, from {default}
 - **configurable as**: isotropiclinearincompelasticity, bulk_rheology
- **db_auxiliary_field**: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field
- **derived_subfields**: Discretization of derived subfields.
 - **current value**: ‘derived_subfields’, from {file=’/Users/baagaard/software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **configurable as**: derived_subfields
- **observers**: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers

Pyre Properties

- `description=<str>`: Descriptive label for material (deprecated in v5.0; will be removed in v6.0).
 - **default value:** ‘’
 - **current value:** ‘’, from {default}
 - **validator:** <function validateDescription at 0x10480b2e0>
- `label=<str>`: Name of label for material. Currently only ‘material-id’ is allowed.
 - **default value:** ‘material-id’
 - **current value:** ‘material-id’, from {default}
 - **validator:** (in [‘material-id’])
- `label_value=<int>`: Value of label for material.
 - **default value:** 1
 - **current value:** 1, from {default}
- `use_body_force=<bool>`: Include body force term in elasticity equation.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting IncompressibleElasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_incompelastic]
label_value = 3
use_body_force = True
bulk_rheology = pylith.materials.IsotropicLinearIncompElasticity

auxiliary_subfields.density.basis_order = 0
auxiliary_subfields.body_force.basis_order = 0
derived_subfields.cauchy_stress.basis_order = 1
derived_subfields.cauchy_strain.basis_order = 1
```

IsotropicLinearElasticity

Full name

pylith.materials.IsotropicLinearElasticity

Journal name

isotropiclinearelasticity

Isotropic linear elastic bulk rheology.

Implements RheologyElasticity.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for physical properties and state variables.
 - **current value:** ‘auxiliary_subfields’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}

- **configurable as:** auxiliary_subfields

Pyre Properties

- use_reference_state=<bool>: Use reference stress/strain state.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting IsotropicLinearElasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_elastic.rheology]
use_reference_state = False

auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.bulk_modulus.basis_order = 0
```

IsotropicLinearGenMaxwell

Full name

pylith.materials.IsotropicLinearGenMaxwell

Journal name

isotropiclineargenmaxwell

Isotropic generalized Maxwell viscoelastic bulk rheology.

Implements RheologyElasticity.

Pyre Facilities

- auxiliary_subfields: Discretization information for physical properties and state variables.
 - **current value:** 'auxiliary_subfields', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set' }
 - **configurable as:** auxiliary_subfields

Pyre Properties

- use_reference_state=<bool>: Use reference stress/strain state.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting IsotropicLinearGenMaxwell Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_genmaxwell.rheology]
use_reference_state = False

auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.bulk_modulus.basis_order = 0
```

IsotropicLinearIncompElasticity

Full name

pylith.materials.IsotropicLinearIncompElasticity

Journal name

isotropiclinearincompelasticity

Isotropic linear incompressible elastic bulk rheology.

Implements RheologyIncompressibleElasticity.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for physical properties and state variables.
 - **current value**: 'auxiliary_subfields', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}
 - **configurable as**: auxiliary_subfields

Pyre Properties

- **use_reference_state=<bool>**: Use reference stress/strain state.
 - **default value**: False
 - **current value**: False, from {default}

Example

Example of setting IsotropicLinearIncompElasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_incompelastic.rheology]
use_reference_state = False

auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.bulk_modulus.basis_order = 0
```

IsotropicLinearMaxwell

Full name

pylith.materials.IsotropicLinearMaxwell

Journal name

isotropiclinearmaxwell

Isotropic linear Maxwell viscoelastic bulk rheology.

Implements RheologyElasticity.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for physical properties and state variables.
 - **current value**: 'auxiliary_subfields', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}
 - **configurable as**: auxiliary_subfields

Pyre Properties

- `use_reference_state=<bool>`: Use reference stress/strain state.
 - **default value**: False
 - **current value**: False, from {default}

Example

Example of setting `IsotropicLinearMaxwell` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_maxwell.rheology]
use_reference_state = False

auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.bulk_modulus.basis_order = 0
```

IsotropicLinearPoroelasticity

Full name

`pylith.materials.IsotropicLinearPoroelasticity`

Journal name

`isotropiclinearporoelasticity`

Isotropic linear incompressible elastic bulk rheology.

Implements `RheologyIncompressibleElasticity`.

Pyre Facilities

- `auxiliary_subfields`: Discretization information for physical properties and state variables.
 - **current value**: `'auxiliary_subfields'`, from `{file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}`
 - **configurable as**: `auxiliary_subfields`

Pyre Properties

- `use_reference_state=<bool>`: Use reference stress/strain state.
 - **default value**: False
 - **current value**: False, from {default}
- `use_tensor_permeability=<bool>`: Use tensor permeability.
 - **default value**: False
 - **current value**: False, from {default}

Example

Example of setting `IsotropicLinearPoroelasticity` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_incompelastic.rheology]
use_reference_state = False

auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.bulk_modulus.basis_order = 0
```

IsotropicPowerLaw

Full name

pylith.materials.IsotropicPowerLaw

Journal name

isotropicpowerlaw

Isotropic power law viscoelastic bulk rheology.

Implements RheologyElasticity.

Pyre Facilities

- **auxiliary_subfields:** Discretization information for physical properties and state variables.
 - **current value:** 'auxiliary_subfields', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}
 - **configurable as:** auxiliary_subfields

Pyre Properties

- **use_reference_state=<bool>:** Use reference stress/strain state.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting IsotropicPowerLaw Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_powerlaw.rheology]
use_reference_state = False

auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.bulk_modulus.basis_order = 0
```

Material

Full name

pylith.materials.Material

Journal name

material

Abstract base class for a bulk material.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for auxiliary subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, auxiliary_subfields
- **db_auxiliary_field**: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field
- **derived_subfields**: Discretization of derived subfields.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, derived_subfields
- **observers**: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers

Pyre Properties

- **description=<str>**: Descriptive label for material (deprecated in v5.0; will be removed in v6.0).
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
 - **validator**: <function validateDescription at 0x10608f2e0>
- **label=<str>**: Name of label for material. Currently only ‘material-id’ is allowed.
 - **default value**: ‘material-id’
 - **current value**: ‘material-id’, from {default}
 - **validator**: (in [‘material-id’])
- **label_value=<int>**: Value of label for material.
 - **default value**: 1
 - **current value**: 1, from {default}

Poroelasticity

Full name

pylith.materials.Poroelasticity

Journal name

poroelasticity

Material behavior governed by the poroelasticity equation.

Implements `Material`.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for auxiliary subfields.
 - **current value**: ‘auxiliary_subfields’, from {file=’/Users/baagaard/software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **configurable as**: auxiliary_subfields
- **bulk_rheology**: Bulk rheology for poroelastic material.
 - **current value**: ‘isotropiclinearporoelasticity’, from {default}
 - **configurable as**: isotropiclinearporoelasticity, bulk_rheology
- **db_auxiliary_field**: Database for physical property parameters.
 - **current value**: ‘simpledb’, from {default}
 - **configurable as**: simpledb, db_auxiliary_field
- **derived_subfields**: Discretization of derived subfields.
 - **current value**: ‘derived_subfields’, from {file=’/Users/baagaard/software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **configurable as**: derived_subfields
- **observers**: Observers (e.g., output).
 - **current value**: ‘singlephysicsobserver’, from {default}
 - **configurable as**: singlephysicsobserver, observers

Pyre Properties

- **description=<str>**: Descriptive label for material (deprecated in v5.0; will be removed in v6.0).
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
 - **validator**: <function validateDescription at 0x10480b2e0>
- **label=<str>**: Name of label for material. Currently only ‘material-id’ is allowed.
 - **default value**: ‘material-id’
 - **current value**: ‘material-id’, from {default}
 - **validator**: (in [‘material-id’])
- **label_value=<int>**: Value of label for material.
 - **default value**: 1
 - **current value**: 1, from {default}
- **use_body_force=<bool>**: Include body force term in Poroelasticity equation.
 - **default value**: False
 - **current value**: False, from {default}
- **use_source_density=<bool>**: Include source_density term in Poroelasticity equation.
 - **default value**: False

- **current value:** False, from {default}
- **use_state_variables=<bool>:** Update porosity state variable using compaction formulation.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting Poroelasticity Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.materials.mat_poroelastic]
label_value = 3
use_body_force = True
use_source_density = False
use_state_variables = True
bulk_rheology = pylith.materials.IsotropicLinearPoroelasticity

auxiliary_subfields.density.basis_order = 0
auxiliary_subfields.body_force.basis_order = 0
```

RheologyElasticity

Full name

pylith.materials.RheologyElasticity

Journal name

rheologyelasticity

Abstract base class for bulk rheology of elastic material.

Pyre Facilities

- **auxiliary_subfields:** Discretization information for physical properties and state variables.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, auxiliary_subfields

RheologyIncompressibleElasticity

Full name

pylith.materials.RheologyIncompressibleElasticity

Journal name

rheologyincompressibleelasticity

Abstract base class for bulk rheology of incompressible elastic material.

Pyre Facilities

- **auxiliary_subfields:** Discretization information for physical properties and state variables.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, auxiliary_subfields

RheologyPoroelasticity

Full name

pylith.materials.RheologyPoroelasticity

Journal name

rheologyporoelasticity

Abstract base class for bulk rheology of poroelastic material.

Pyre Facilities

- **auxiliary_subfields**: Discretization information for physical properties and state variables.
 - **current value**: ‘emptybin’, from {default}
 - **configurable as**: emptybin, auxiliary_subfields

meshio

DataWriter

Full name

pylith.meshio.DataWriter

Journal name

datawriter

Abstract base class writing solution, auxiliary, and derived subfields.

DataWriterHDF5

Full name

pylith.meshio.DataWriterHDF5

Journal name

datawriterhdf5

Writer of solution, auxiliary, and derived subfields to an HDF5 file.

Implements DataWriter.

Pyre Properties

- **filename=<str>**: Name of HDF5 file.
 - **default value**: ‘’
 - **current value**: ‘’, from {default}

Example

Example of setting DataWriterHDF5 Pyre properties and facilities in a parameter file.

```
[data_writer]
filename = domain_solution.h5
```

DataWriterHDF5Ext

Full name

pylith.meshio.DataWriterHDF5Ext

Journal name

datawriterhdf5

Writer of solution, auxiliary, and derived subfields to an HDF5 file with datasets stored in external binary files.

Implements DataWriter.

Pyre Properties

- `filename=<str>`: Name of HDF5 file.
 - **default value:** ‘
 - **current value:** ‘, from {default}

Example

Example of setting DataWriterHDF5Ext Pyre properties and facilities in a parameter file.

```
[data_writer]
filename = domain_solution.h5
```

DataWriterVTK

Full name

pylith.meshio.DataWriterVTK

Journal name

datawritervtk

Writer of solution, auxiliary, and derived subfields to a VTK file.

Implements DataWriter.

Pyre Properties

- `filename=<str>`: Name of VTK file.
 - **default value:** ‘
 - **current value:** ‘, from {default}
- `float_precision=<int>`: Precision of floating point values in output.
 - **default value:** 6
 - **current value:** 6, from {default}
 - **validator:** (greater than 0)
- `time_constant=<dimensional>`: Values used to normalize time stamp in filename.
 - **default value:** 1*s
 - **current value:** 1*s, from {default}
 - **validator:** (greater than 0*s)

- `time_format=<str>`: C style format string for time stamp in filename.
 - **default value**: ‘%f’
 - **current value**: ‘%f’, from {default}

Example

Example of setting `DataWriterVTK` Pyre properties and facilities in a parameter file.

```
[data_writer]
filename = domain_solution.vtk
time_format = %0.2f
time_constant = 1.0*year
float_precision = 6
```

MeshIOAscii

Full name
pylith.meshio.MeshIOAscii

Journal name
meshioascii

Reader for finite-element meshes using a simple ASCII format.

Warning

The coordinate system associated with the mesh must be a Cartesian coordinate system, such as a generic Cartesian coordinate system or a geographic projection.

Implements `MeshIOObj`.

Pyre Facilities

- `coordsys`: Coordinate system associated with mesh.
 - **current value**: ‘cscart’, from {default}
 - **configurable as**: cscart, coordsys

Pyre Properties

- `filename=<str>`: Name of mesh file
 - **default value**: ‘’
 - **current value**: ‘’, from {default}

Example

Example of setting `MeshIOAscii` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.read_mesh.reader]
filename = mesh_quad.txt
coordsys.space_dim = 2
```

MeshIOCubit

Full name

`pylith.meshio.MeshIOCubit`

Journal name

`meshiocubit`

Reader for finite-element meshes from Exodus II files (usually from Cubit).

 Warning

The coordinate system associated with the mesh must be a Cartesian coordinate system, such as a generic Cartesian coordinate system or a geographic projection.

Implements `MeshIOObj`.

Pyre Facilities

- `coordsys`: Coordinate system associated with mesh.
 - **current value**: `'cscart'`, from `{default}`
 - **configurable as**: `cscart`, `coordsys`

Pyre Properties

- `filename=<str>`: Name of Cubit Exodus file.
 - **default value**: `'mesh.exo'`
 - **current value**: `'mesh.exo'`, from `{default}`

Example

Example of setting `MeshIOCubit` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.read_mesh.reader]
filename = mesh_quad.exo
coordsys.space_dim = 2
```

MeshIOObj

Full name

`pylith.meshio.MeshIOObj`

Journal name

`meshio`

Abstract base class for finite-element mesh readers.

MeshIOPetsc

Full name

`pylith.meshio.MeshIOPetsc`

Journal name

meshiopetsc

Python object for a variety of reading/writing finite-element meshes using PETSc. Currently, the primary use of this object is to import meshes from Gmsh.

Warning

The coordinate system associated with the mesh must be a Cartesian coordinate system, such as a generic Cartesian coordinate system or a geographic projection.

Implements MeshIOObj.

Pyre Facilities

- coordsys: Coordinate system associated with mesh.
 - **current value:** 'cscart', from {default}
 - **configurable as:** cscart, coordsys

Pyre Properties

- filename=<str>: Name of mesh file for reading with PETSc.
 - **default value:** ''
 - **current value:** '', from {default}
- gmsh_mark_recursive=<bool>: Gmsh file marks faces, edges, and vertices rather than just faces (3D) or edges (2D).
 - **default value:** False
 - **current value:** False, from {default}
- options_prefix=<str>: Name of PETSc options prefix for this mesh.
 - **default value:** ''
 - **current value:** '', from {default}

Example

Example of setting MeshIOPetsc Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.mesh_initializer.phases.read_mesh.reader]
filename = mesh_quad.msh
gmsh_mark_vertices = False
coordsys.space_dim = 2
```

OutputObserver**Full name**

pylith.meshio.OutputObserver

Journal name

outputobserver

Abstract base class for managing output of solution information.

Pyre Facilities

- **trigger**: Trigger defining how often output is written.
 - **current value**: ‘outputtriggerstep’, from {default}
 - **configurable as**: outputtriggerstep, trigger
- **writer**: Writer for data.
 - **current value**: ‘datawriterhdf5’, from {default}
 - **configurable as**: datawriterhdf5, writer

Pyre Properties

- **output_basis_order=<int>**: Basis order for output.
 - **default value**: 1
 - **current value**: 1, from {default}
 - **validator**: (in [0, 1])
- **refine_levels=<int>**: Number of mesh refinement levels for output.
 - **default value**: 0
 - **current value**: 0, from {default}
 - **validator**: (greater than or equal to 0)

OutputPhysics

Full name

pylith.meshio.OutputPhysics

Journal name

outputphysics

Output for objects implementing physics (materials and boundary conditions).

Tip

Most output information can be configured at the problem level using the *ProblemDefaults Component*.

Implements OutputObserver.

Pyre Facilities

- **trigger**: Trigger defining how often output is written.
 - **current value**: ‘outputtriggerstep’, from {default}
 - **configurable as**: outputtriggerstep, trigger
- **writer**: Writer for data.
 - **current value**: ‘datawriterhdf5’, from {default}

- **configurable as:** datawriterhdf5, writer

Pyre Properties

- **data_fields=<list>:** Names of solution, auxiliary, and derived subfields to include in data output.
 - **default value:** ['all']
 - **current value:** ['all'], from {default}
- **info_fields=<list>:** Names of auxiliary subfields to include in info output.
 - **default value:** ['all']
 - **current value:** ['all'], from {default}
- **output_basis_order=<int>:** Basis order for output.
 - **default value:** 1
 - **current value:** 1, from {default}
 - **validator:** (in [0, 1])
- **refine_levels=<int>:** Number of mesh refinement levels for output.
 - **default value:** 0
 - **current value:** 0, from {default}
 - **validator:** (greater than or equal to 0)

Example

Example of setting OutputPhysics Pyre properties and facilities in a parameter file.

```
[observer]
# Skip two time steps between output.
trigger = pylith.meshio.OutputTriggerStep
trigger.num_skip = 2

# Write output to HDF5 file with name `boundary_xpos.h5`.
writer = pylith.meshio.DataWriterHDF5
writer.filename = boundary_xpos.h5

output_basis_order = 1
```

OutputSoln

Full name

pylith.meshio.OutputSoln

Journal name

outputsoln

Abstract base class for output of solution subfields.

Implements OutputObserver.

Pyre Facilities

- **trigger**: Trigger defining how often output is written.
 - **current value**: ‘outputtriggerstep’, from {default}
 - **configurable as**: outputtriggerstep, trigger
- **writer**: Writer for data.
 - **current value**: ‘datawriterhdf5’, from {default}
 - **configurable as**: datawriterhdf5, writer

Pyre Properties

- **data_fields=<list>**: Names of solution subfields to include in output.
 - **default value**: [‘all’]
 - **current value**: [‘all’], from {default}
- **output_basis_order=<int>**: Basis order for output.
 - **default value**: 1
 - **current value**: 1, from {default}
 - **validator**: (in [0, 1])
- **refine_levels=<int>**: Number of mesh refinement levels for output.
 - **default value**: 0
 - **current value**: 0, from {default}
 - **validator**: (greater than or equal to 0)

OutputSolnBoundary

Full name

pylith.meshio.OutputSolnBoundary

Journal name

outputsolnsubset

Output of solution subfields over an external boundary.

Tip

Most output information can be configured at the problem level using the *ProblemDefaults Component*.

Implements OutputSoln.

Pyre Facilities

- **trigger**: Trigger defining how often output is written.
 - **current value**: ‘outputtriggerstep’, from {default}
 - **configurable as**: outputtriggerstep, trigger
- **writer**: Writer for data.

- **current value:** ‘datawriterhdf5’, from {default}
- **configurable as:** datawriterhdf5, writer

Pyre Properties

- **data_fields=<list>:** Names of solution subfields to include in output.
 - **default value:** [‘all’]
 - **current value:** [‘all’], from {default}
- **label=<str>:** Name of label identifier for external boundary.
 - **default value:** ‘’
 - **current value:** ‘’, from {default}
 - **validator:** <function validateLabel at 0x104835e40>
- **label_value=<int>:** Value of label identifier for external boundary (tag of physical group in Gmsh files).
 - **default value:** 1
 - **current value:** 1, from {default}
- **output_basis_order=<int>:** Basis order for output.
 - **default value:** 1
 - **current value:** 1, from {default}
 - **validator:** (in [0, 1])
- **refine_levels=<int>:** Number of mesh refinement levels for output.
 - **default value:** 0
 - **current value:** 0, from {default}
 - **validator:** (greater than or equal to 0)

Example

Example of setting OutputSolnBoundary Pyre properties and facilities in a parameter file.

```
[observer]
data_fields = [displacement]

label = boundary_xpos

# Skip two time steps between output.
trigger = pylith.meshio.OutputTriggerStep
trigger.num_skip = 2

# Write output to HDF5 file with name `boundary_xpos.h5`.
writer = pylith.meshio.DataWriterHDF5
writer.filename = boundary_xpos.h5

output_basis_order = 1
```

OutputSolnDomain

Full name

pylith.meshio.OutputSolnDomain

Journal name

outputsolndomain

Output of solution subfields over the simulation domain.

 Tip

Most output information can be configured at the problem level using the *ProblemDefaults Component*.

Implements OutputSoln.

Pyre Facilities

- **trigger**: Trigger defining how often output is written.
 - **current value**: ‘outputtriggerstep’, from {default}
 - **configurable as**: outputtriggerstep, trigger
- **writer**: Writer for data.
 - **current value**: ‘datawriterhdf5’, from {default}
 - **configurable as**: datawriterhdf5, writer

Pyre Properties

- **data_fields=<list>**: Names of solution subfields to include in output.
 - **default value**: [‘all’]
 - **current value**: [‘all’], from {default}
- **output_basis_order=<int>**: Basis order for output.
 - **default value**: 1
 - **current value**: 1, from {default}
 - **validator**: (in [0, 1])
- **refine_levels=<int>**: Number of mesh refinement levels for output.
 - **default value**: 0
 - **current value**: 0, from {default}
 - **validator**: (greater than or equal to 0)

Example

Example of setting OutputSolnDomain Pyre properties and facilities in a parameter file.

```
[observer]
data_fields = [displacement]
```

(continues on next page)

(continued from previous page)

```
# Skip two time steps between output.
trigger = pylith.meshio.OutputTriggerStep
trigger.num_skip = 2

# Write output to HDF5 file with name `domain.h5`.
writer = pylith.meshio.DataWriterHDF5
writer.filename = domain.h5

# Output with a basis order of 1 and refine mesh 3x (cells are 1/8 size).
# Refining the output mesh is useful with a basis order of 2 or greater in the solution.
output_basis_order = 1
refine_levels = 3
```

OutputSolnPoints

Full name

pylith.meshio.OutputSolnPoints

Journal name

outputsolnpoints

Output of solution subfields at discrete points in the domain.

Tip

Most output information can be configured at the problem level using the *ProblemDefaults Component*.

Implements OutputSoln.

Pyre Facilities

- reader: Reader for points list.
 - **current value:** ‘pointslst’, from {default}
 - **configurable as:** pointslst, reader
- trigger: Trigger defining how often output is written.
 - **current value:** ‘outputtriggerstep’, from {default}
 - **configurable as:** outputtriggerstep, trigger
- writer: Writer for data.
 - **current value:** ‘datawriterhdf5’, from {default}
 - **configurable as:** datawriterhdf5, writer

Pyre Properties

- data_fields=<list>: Names of solution subfields to include in output.
 - **default value:** [‘all’]
 - **current value:** [‘all’], from {default}
- label=<str>: Label identifier for points (used in constructing default filenames).

- **default value:** ‘points’
 - **current value:** ‘points’, from {default}
- **output_basis_order=<int>:** Basis order for output.
 - **default value:** 1
 - **current value:** 1, from {default}
 - **validator:** (in [0, 1])
- **refine_levels=<int>:** Number of mesh refinement levels for output.
 - **default value:** 0
 - **current value:** 0, from {default}
 - **validator:** (greater than or equal to 0)

Example

Example of setting OutputSolnPoints Pyre properties and facilities in a parameter file.

```
[observer]
label = stations
data_fields = [displacement]

# List of points where we want output.
reader = pylith.meshio.PointsList
reader.filename = stations.txt

# Skip two time steps between output.
trigger = pylith.meshio.OutputTriggerStep
trigger.num_skip = 2

# Write output to HDF5 file with name `domain.h5`.
writer = pylith.meshio.DataWriterHDF5
writer.filename = domain.h5

output_basis_order = 1
```

OutputTrigger

Full name

pylith.meshio.OutputTrigger

Journal name

outputtrigger

Abstract base class for managing how often output is written.

OutputTriggerStep

Full name

pylith.meshio.OutputTriggerStep

Journal name

outputtriggerstep

Define how often output is written in terms of solution steps.

Implements OutputTrigger.

Pyre Properties

- `num_skip=<int>`: Number of solution steps to skip between writes (0 means write every time step).
 - **default value:** 0
 - **current value:** 0, from {default}
 - **validator:** (greater than or equal to 0)

Example

Example of setting OutputTriggerStep Pyre properties and facilities in a parameter file.

```
[output_trigger]
num_skip = 2
```

OutputTriggerTime

Full name

`pylith.meshio.OutputTriggerTime`

Journal name

`outputtriggertime`

Define how often output is written in terms of elapsed simulation time.

Tip

Due to floating point roundoff, it is usually a good idea to use a value that is a fraction of a time step smaller than the desired value.

Implements OutputTrigger.

Pyre Properties

- `elapsed_time=<dimensional>`: Elapsed time between writes.
 - **default value:** 0*s
 - **current value:** 0*s, from {default}

Example

Example of setting OutputTriggerTime Pyre properties and facilities in a parameter file.

```
[output_trigger]
elapsed_time = 0.9999*year
```

PointsList

Full name

pylith.meshio.PointsList

Journal name

pointslist

Reader for a list of points from an ASCII file.

 See also

See *OutputSolnPoints Component*.

Pyre Facilities

- coordsys: Coordinate system associated with points.
 - **current value:** ‘cscart’, from {default}
 - **configurable as:** cscart, coordsys

Pyre Properties

- comment_delimiter=<str>: Delimiter for comments.
 - **default value:** ‘#’
 - **current value:** ‘#’, from {default}
- filename=<str>: Filename for list of points.
 - **default value:** ‘’
 - **current value:** ‘’, from {default}
 - **validator:** <function validateFilename at 0x11f3209d0>
- value_delimiter=<str>: Delimiter used to separate values.
 - **default value:** None
 - **current value:** None, from {default}

Example

Example of setting PointsList Pyre properties and facilities in a parameter file.

```
[points]
filename = stations.txt
comment_delimiter = #
value_delimiter = ,

coordsys = spatialdata.geocoords.CSCart
coordsys.space_dim = 2
```


problems

GreensFns

Full name

`pylith.problems.GreensFns`

Journal name

`greensfns`

Static Green's function problem type with each Green's function corresponding to a fault-slip impulse.

Implements Problem.

Pyre Facilities

- `bc`: Boundary conditions.
 - **current value**: 'emptybin', from {default}
 - **configurable as**: emptybin, bc
- `defaults`: Default options for problem.
 - **current value**: 'problem_defaults', from {default}
 - **configurable as**: problem_defaults, defaults
- `gravity_field`: Database used for gravity field.
 - **current value**: 'nullcomponent', from {default}
 - **configurable as**: nullcomponent, gravity_field
- `interfaces`: Interior surfaces with constraints or constitutive models.
 - **current value**: 'emptybin', from {default}
 - **configurable as**: emptybin, interfaces
- `materials`: Materials in problem.
 - **current value**: 'homogeneous', from {default}
 - **configurable as**: homogeneous, materials
- `mesh_initializer`: Mesh initializer.
 - **current value**: 'mesh_initializer', from {default}
 - **configurable as**: mesh_initializer
- `petsc_defaults`: Flags controlling which default PETSc options to use.
 - **current value**: 'petscdefaults', from {default}
 - **configurable as**: petscdefaults, petsc_defaults
- `progress_monitor`: Simple progress monitor via text file.
 - **current value**: 'progressmonitorstep', from {default}
 - **configurable as**: progressmonitorstep, progress_monitor
- `scales`: Scales for nondimensionalizing boundary value problem.
 - **current value**: 'quasistaticelasticity', from {default}
 - **configurable as**: quasistaticelasticity, scales

- **solution**: Solution field for problem.
 - **current value**: ‘solution’, from {default}
 - **configurable as**: solution
- **solution_observers**: Observers (e.g., output) for solution.
 - **current value**: ‘singlesolnobserver’, from {default}
 - **configurable as**: singlesolnobserver, solution_observers

Pyre Properties

- **formulation=<str>**: Formulation for equations.
 - **default value**: ‘quasistatic’
 - **current value**: ‘quasistatic’, from {default}
 - **validator**: (in [‘quasistatic’, ‘dynamic’, ‘dynamic_imex’])
- **label=<str>**: Name of label identifier for fault surface on which to impose impulses.
 - **default value**: ‘fault’
 - **current value**: ‘fault’, from {default}
- **label_value=<int>**: Value of label identifier for fault surface on which to impose impulses.
 - **default value**: 1
 - **current value**: 1, from {default}
- **solver=<str>**: Type of solver to use [‘linear’, ‘nonlinear’].
 - **default value**: ‘nonlinear’
 - **current value**: ‘nonlinear’, from {default}
 - **validator**: (in [‘linear’, ‘nonlinear’])

Example

Example of setting GreensFns Pyre properties and facilities in a parameter file.

```
[pylithapp]
problem = pylith.problems.GreensFns

[pylithapp.greensfns]
label = fault
label_value = 1

# Set appropriate default solver settings.
set_solver_defaults = True

interfaces = [fault]
interfaces.fault = pylith.faults.FaultCohesiveImpulses

[pylithapp.greensfns.interfaces.fault]
label = fault
label_value = 20
```

(continues on next page)

(continued from previous page)

```

# Impulses for left-lateral slip (dof=1)
impulse_dof = [1]
threshold = 0.5

# Create impulses at all points on the fault by specifying a uniform amplitude of 1.0.
# Impulses will be applied at any location with a slip component greater than the
↳ threshold.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Slip impulse amplitude
db_auxiliary_field.values = [slip_left_lateral, slip_opening]
db_auxiliary_field.data = [1.0*m, 0.0*m]

# Represent the impulse as a linear variation in slip centered on each point.
auxiliary_subfields.slip.basis_order = 1

[pylithapp.greensfns.petsc_defaults]
solver = True
monitors = True

```

InitialCondition

Full name

pylith.problems.InitialCondition

Journal name

initialconditions

Abstract base class for specifying initial conditions for the solution.

Pyre Properties

- subfields=<list>: Names of solution subfields for initial condition.
 - **default value:** ['displacement']
 - **current value:** ['displacement'], from {default}

InitialConditionDomain

Full name

pylith.problems.InitialConditionDomain

Journal name

initialconditionsdomain

Initial conditions for the solution over the entire domain.

Implements InitialCondition.

Pyre Facilities

- db: Spatial database with values for initial condition.
 - **current value:** 'simplifiedb', from {default}
 - **configurable as:** simplifiedb, db

Pyre Properties

- `subfields=<list>`: Names of solution subfields for initial condition.
 - **default value**: ['displacement']
 - **current value**: ['displacement'], from {default}

Example

Example of setting `InitialConditionDomain` Pyre properties and facilities in a parameter file.

```
# Create a single initial condition over the domain.
[pylithapp.problem]
ic = [domain]
ic.domain = pylith.problems.InitialConditionDomain

[pylithapp.problem.ic.domain]
db = spatialdata.spatialdb.SimpleGridDB
db.description = Initial conditions over domain
db.filename = sheardisp_ic.spatialdb
```

InitialConditionPatch

Full name

`pylith.problems.InitialConditionPatch`

Journal name

`initialconditionspatch`

Initial conditions over a portion of the domain (patch).

Warning

Initial conditions over a patch currently only work for Gmsh input files. Exodus II files from Cubit do not contain the information needed by the current PyLith implementation.

Implements `InitialCondition`.

Pyre Facilities

- `db`: Spatial database with values for initial condition.
 - **current value**: 'simpledb', from {default}
 - **configurable as**: simpledb, db

Pyre Properties

- `label=<str>`: Name of label for patch.
 - **default value**: 'material-id'
 - **current value**: 'material-id', from {default}
- `label_value=<int>`: Value of label associated with initial condition patch, usually the material label value.
 - **default value**: 1

- **current value:** 1, from {default}
- **subfields=<list>:** Names of solution subfields for initial condition.
 - **default value:** ['displacement']
 - **current value:** ['displacement'], from {default}

Example

Example of setting InitialConditionPatch Pyre properties and facilities in a parameter file.

```
# Create separate initial conditions for two materials.
# This is often useful if the materials have different properties.
[pylithapp.problem]
ic = [mat1, mat2]
ic.mat1 = pylith.problems.InitialConditionPatch
ic.mat2 = pylith.problems.InitialConditionPatch

[pylithapp.problem.ic.mat1]
label_value = 1
db = spatialdata.spatialdb.SimpleGridDB
db.description = Initial conditions over material 1
db.filename = shearmat1_ic.spatialdb

[pylithapp.problem.ic.mat2]
label_value = 2
db = spatialdata.spatialdb.SimpleGridDB
db.description = Initial conditions over material 2
db.filename = shearmat2_ic.spatialdb
```

Physics

Full name

pylith.problems.Physics

Journal name

physics

Abstract base class for objects defining physics.

Pyre Facilities

- **auxiliary_subfields:** Discretization information for auxiliary subfields.
 - **current value:** 'emptybin', from {default}
 - **configurable as:** emptybin, auxiliary_subfields
- **db_auxiliary_field:** Database for physical property parameters.
 - **current value:** 'simpledb', from {default}
 - **configurable as:** simpledb, db_auxiliary_field
- **derived_subfields:** Discretization of derived subfields.
 - **current value:** 'emptybin', from {default}
 - **configurable as:** emptybin, derived_subfields

- **observers:** Observers (e.g., output).
 - **current value:** ‘singlephysicsobserver’, from {default}
 - **configurable as:** singlephysicsobserver, observers

Problem

Full name

`pylith.problems.Problem`

Journal name

`problem`

Abstract base class for a problem.

The default formulation, solution field, and scales for nondimensionalization are appropriate for solving the quasi-static elasticity equation.

By default, we use the nonlinear solver. This facilitates verifying that the residual and Jacobian are consistent. If the nonlinear (SNES) solver requires multiple iterations to converge for these linear problems, then we know there is an error in the problem setup.

Pyre Facilities

- **bc:** Boundary conditions.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, bc
- **defaults:** Default options for problem.
 - **current value:** ‘problem_defaults’, from {default}
 - **configurable as:** problem_defaults, defaults
- **gravity_field:** Database used for gravity field.
 - **current value:** ‘nullcomponent’, from {default}
 - **configurable as:** nullcomponent, gravity_field
- **interfaces:** Interior surfaces with constraints or constitutive models.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, interfaces
- **materials:** Materials in problem.
 - **current value:** ‘homogeneous’, from {default}
 - **configurable as:** homogeneous, materials
- **mesh_initializer:** Mesh initializer.
 - **current value:** ‘mesh_initializer’, from {default}
 - **configurable as:** mesh_initializer
- **petsc_defaults:** Flags controlling which default PETSc options to use.
 - **current value:** ‘petscdefaults’, from {default}
 - **configurable as:** petscdefaults, petsc_defaults

- **scales**: Scales for nondimensionalizing boundary value problem.
 - **current value**: ‘quasistaticelasticity’, from {default}
 - **configurable as**: quasistaticelasticity, scales
- **solution**: Solution field for problem.
 - **current value**: ‘solution’, from {default}
 - **configurable as**: solution
- **solution_observers**: Observers (e.g., output) for solution.
 - **current value**: ‘singlesolnobserver’, from {default}
 - **configurable as**: singlesolnobserver, solution_observers

Pyre Properties

- **formulation=<str>**: Formulation for equations.
 - **default value**: ‘quasistatic’
 - **current value**: ‘quasistatic’, from {default}
 - **validator**: (in [‘quasistatic’, ‘dynamic’, ‘dynamic_imex’])
- **solver=<str>**: Type of solver to use [‘linear’, ‘nonlinear’].
 - **default value**: ‘nonlinear’
 - **current value**: ‘nonlinear’, from {default}
 - **validator**: (in [‘linear’, ‘nonlinear’])

ProblemDefaults

Full name

pylith.problems.ProblemDefaults

Journal name

problem_defaults

Default options for a problem. Specifying defaults at the problem level (here) will override defaults for individual components. Non-default values specified for individual components will override the problem defaults (specified here).

Pyre Properties

- **name=<str>**: Name for the problem (used with output_directory for default output filenames).
 - **default value**: ‘’
 - **current value**: ‘’, from {default}
 - **validator**: <function validateName at 0x124c42280>
- **output_basis_order=<int>**: Default basis order for output.
 - **default value**: 1
 - **current value**: 1, from {default}
 - **validator**: (in [0, 1])

- `output_directory=<str>`: Directory for output.
 - **default value:** ‘output’
 - **current value:** ‘output’, from {default}
- `quadrature_order=<int>`: Finite-element quadrature order.
 - **default value:** 1
 - **current value:** 1, from {default}
 - **validator:** (greater than 0)

Example

Example of setting `ProblemDefaults` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.defaults]
output_directory = output
name = step01
quadrature_order = 1
output_basis_order = 0
```

ProgressMonitor

Full name

`pylith.problems.ProgressMonitor`

Journal name

`progressmonitor`

Abstract base class for simulation progress monitor.

Pyre Properties

- `filename=<str>`: Name of output file.
 - **default value:** ‘’
 - **current value:** ‘’, from {default}
- `update_percent=<float>`: Frequency of progress updates (percent).
 - **default value:** 5.0
 - **current value:** 5.0, from {default}
 - **validator:** (greater than 0)

ProgressMonitorStep

Full name

`pylith.problems.ProgressMonitorStep`

Journal name

`progressmonitorstep`

Progress monitor for problems with a given number of steps, such as Green’s functions problem.

If you do not set the filename for the progress monitor, then PyLith will create one using the simulation name from the application defaults settings.

Implements ProgressMonitor.

Pyre Properties

- `filename=<str>`: Name of output file.
 - **default value:** ‘ ’
 - **current value:** ‘ ’, from {default}
- `update_percent=<float>`: Frequency of progress updates (percent).
 - **default value:** 5.0
 - **current value:** 5.0, from {default}
 - **validator:** (greater than 0)

Example

Example of setting ProgressMonitorStep Pyre properties and facilities in a parameter file.

```
[pylithapp.time-dependent.progress_monitor]
filename = output/greensfns01-progress.txt
```

ProgressMonitorTime

Full name

`pylith.problems.ProgressMonitorTime`

Journal name

`progressmonitortime`

Progress monitor for time-dependent problem.

If you do not set the filename for the progress monitor, then PyLith will create one using the simulation name from the application defaults settings.

Pyre Properties

- `filename=<str>`: Name of output file.
 - **default value:** ‘ ’
 - **current value:** ‘ ’, from {default}
- `t_units=<str>`: Units used for simulation time in output.
 - **default value:** ‘year’
 - **current value:** ‘year’, from {default}
- `update_percent=<float>`: Frequency of progress updates (percent).
 - **default value:** 5.0
 - **current value:** 5.0, from {default}
 - **validator:** (greater than 0)

Example

Example of setting ProgressMonitorTime Pyre properties and facilities in a parameter file.

```
[pylithapp.timedependent.progress_monitor]
filename = output/step01-progress.txt
t_units = year
```

SolnDisp

Full name

pylith.problems.SolnDisp

Journal name

solndisp

Container for solution subfields with displacement subfield.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value:** ‘subfelddisplacement’, from {default}
 - **configurable as:** subfelddisplacement, displacement

SolnDispLagrange

Full name

pylith.problems.SolnDispLagrange

Journal name

solndisplagrange

Container for solution subfields with displacement and fault Lagrange multiplier subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value:** ‘subfelddisplacement’, from {default}
 - **configurable as:** subfelddisplacement, displacement
- lagrange_multiplier_fault: Fault Lagrange multiplier subfield.
 - **current value:** ‘subfieldlagrangefault’, from {default}
 - **configurable as:** subfieldlagrangefault, lagrange_multiplier_fault

Example

Example of setting SolnDispLagrange Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange
```

SolnDispPres

Full name

pylith.problems.SolnDispPres

Journal name

solndisppres

Container for solution subfields with displacement and pressure subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value**: ‘subfielddisplacement’, from {default}
 - **configurable as**: subfielddisplacement, displacement
- pressure: Pressure subfield.
 - **current value**: ‘subfieldpressure’, from {default}
 - **configurable as**: subfieldpressure, pressure

Example

Example of setting SolnDispPres Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPres
```

SolnDispPresLagrange

Full name

pylith.problems.SolnDispPresLagrange

Journal name

solndisppres

Container for solution subfields with displacement, pressure, and fault Lagrange multiplier subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value**: ‘subfielddisplacement’, from {default}
 - **configurable as**: subfielddisplacement, displacement
- lagrange_multiplier_fault: Fault Lagrange multiplier subfield.
 - **current value**: ‘subfieldlagrangefault’, from {default}
 - **configurable as**: subfieldlagrangefault, lagrange_multiplier_fault
- pressure: Pressure subfield.
 - **current value**: ‘subfieldpressure’, from {default}
 - **configurable as**: subfieldpressure, pressure

Example

Example of setting SolnDispPresLagrange Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresLagrange
```

SolnDispPresTracStrain

Full name

pylith.problems.SolnDispPresTracStrain

Journal name

solndispprestracstrain

Container for solution subfields with displacement, pore pressure, and trace strain subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value:** ‘subfielddisplacement’, from {default}
 - **configurable as:** subfielddisplacement, displacement
- pressure: Pressure subfield.
 - **current value:** ‘subfieldpressure’, from {default}
 - **configurable as:** subfieldpressure, pressure
- trace_strain: Trace strain subfield.
 - **current value:** ‘subfieldtracestrain’, from {default}
 - **configurable as:** subfieldtracestrain, trace_strain

Example

Example of setting SolnDispPresTracStrain Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresTracStrain
```

SolnDispPresTracStrainLagrange

Full name

pylith.problems.SolnDispPresTracStrainLagrange

Journal name

SolnDispPresTracStrainLagrange

Container for solution subfields with displacement, pore pressure, trace strain subfields, and fault Lagrange multiplier subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value:** ‘subfielddisplacement’, from {default}
 - **configurable as:** subfielddisplacement, displacement
- lagrange_multiplier_fault: Fault Lagrange multiplier subfield.
 - **current value:** ‘subfieldlagrangefault’, from {default}
 - **configurable as:** subfieldlagrangefault, lagrange_multiplier_fault
- pressure: Pressure subfield.
 - **current value:** ‘subfieldpressure’, from {default}
 - **configurable as:** subfieldpressure, pressure
- trace_strain: Trace strain subfield.
 - **current value:** ‘subfieldtracestrain’, from {default}
 - **configurable as:** subfieldtracestrain, trace_strain

Example

Example of setting SolnDispPresTracStrainLagrange Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresTracStrainLagrange
```

SolnDispPresTracStrainVelPdotTdot

Full name

pylith.problems.SolnDispPresTracStrainVelPdotTdot

Journal name

solndispprestracstrainveltpdotpdot

Container for solution subfields with displacement, pore pressure, and trace strain subfields, along with their time derivatives.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value:** ‘subfielddisplacement’, from {default}
 - **configurable as:** subfielddisplacement, displacement
- pressure: Pressure subfield.
 - **current value:** ‘subfieldpressure’, from {default}
 - **configurable as:** subfieldpressure, pressure
- pressure_t: Pressure_t subfield.
 - **current value:** ‘subfieldpressure_t’, from {default}
 - **configurable as:** subfieldpressure_t, pressure_t
- trace_strain: Trace strain subfield.

- **current value:** ‘subfieldtracstrain’, from {default}
 - **configurable as:** subfieldtracstrain, trace_strain
- trace_strain_t: Trace strain_t subfield.
 - **current value:** ‘subfieldtracstrain_t’, from {default}
 - **configurable as:** subfieldtracstrain_t, trace_strain_t
- velocity: Velocity subfield.
 - **current value:** ‘subfieldvelocity’, from {default}
 - **configurable as:** subfieldvelocity, velocity

Example

Example of setting SolnDispPresTracStrainVelPdotTdot Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresTracStrainVelPdotTdot
```

SolnDispPresTracStrainVelPdotTdotLagrange

Full name

pylith.problems.SolnDispPresTracStrainVelPdotTdotLagrange

Journal name

solindispprestracstrainvelpdottdotlagrange

Python solution field with displacement, pressure, and trace strain subfields, along with their time derivatives, and a lagrange fault.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value:** ‘subfelddisplacement’, from {default}
 - **configurable as:** subfelddisplacement, displacement
- lagrange_multiplier_fault: Fault Lagrange multiplier subfield.
 - **current value:** ‘subfieldlagrangefault’, from {default}
 - **configurable as:** subfieldlagrangefault, lagrange_multiplier_fault
- pressure: Pressure subfield.
 - **current value:** ‘subfieldpressure’, from {default}
 - **configurable as:** subfieldpressure, pressure
- pressure_t: PressureT subfield.
 - **current value:** ‘subfieldpressure_t’, from {default}
 - **configurable as:** subfieldpressure_t, pressure_t
- trace_strain: Trace strain subfield.
 - **current value:** ‘subfieldtracstrain’, from {default}
 - **configurable as:** subfieldtracstrain, trace_strain

- `trace_strain_t`: `TraceStrainT` subfield.
 - **current value**: ‘subfieldtracestrain_t’, from {default}
 - **configurable as**: subfieldtracestrain_t, trace_strain_t
- `velocity`: `Velocity` subfield.
 - **current value**: ‘subfieldvelocity’, from {default}
 - **configurable as**: subfieldvelocity, velocity

Example

Example of setting `SolnDispPresTracStrainVelPdotTdotLagrange` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresTracStrainVelPdotTdotLagrange
```

SolnDispPresVel

Full name

`pylith.problems.SolnDispPresVel`

Journal name

`solndisppresvel`

Container for solution subfields with displacement, pore pressure, and velocity subfields.

Pyre Facilities

- `displacement`: `Displacement` subfield.
 - **current value**: ‘subfelddisplacement’, from {default}
 - **configurable as**: subfelddisplacement, displacement
- `pressure`: `Pressure` subfield.
 - **current value**: ‘subfieldpressure’, from {default}
 - **configurable as**: subfieldpressure, pressure
- `velocity`: `Velocity` subfield.
 - **current value**: ‘subfieldvelocity’, from {default}
 - **configurable as**: subfieldvelocity, velocity

Example

Example of setting `SolnDispPresVel` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresVel
```

SolnDispVel

Full name

pylith.problems.SolnDispVel

Journal name

solndispvel

Container for solution subfields with displacement and velocity subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value**: ‘subfielddisplacement’, from {default}
 - **configurable as**: subfielddisplacement, displacement
- velocity: Velocity subfield.
 - **current value**: ‘subfieldvelocity’, from {default}
 - **configurable as**: subfieldvelocity, velocity

Example

Example of setting SolnDispVel Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispVel
```

SolnDispVelLagrange

Full name

pylith.problems.SolnDispVelLagrange

Journal name

solndispvel

Container for solution subfields with displacement, velocity, and fault Lagrange multiplier subfields.

Pyre Facilities

- displacement: Displacement subfield.
 - **current value**: ‘subfielddisplacement’, from {default}
 - **configurable as**: subfielddisplacement, displacement
- lagrange_multiplier_fault: Fault Lagrange multiplier subfield.
 - **current value**: ‘subfieldlagrangefault’, from {default}
 - **configurable as**: subfieldlagrangefault, lagrange_multiplier_fault
- velocity: Velocity subfield.
 - **current value**: ‘subfieldvelocity’, from {default}
 - **configurable as**: subfieldvelocity, velocity

Example

Example of setting SolnDispVelLagrange Pyre properties and facilities in a parameter file.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispVelLagrange
```

Solution

Full name

pylith.problems.Solution

Journal name

solution

Abstract base class for solution field for problem.

Pyre Facilities

- subfields: Subfields in solution.
 - **current value:** 'solndisp', from {default}
 - **configurable as:** solndisp, subfields

SolutionSubfield

Full name

pylith.problems.SolutionSubfield

Journal name

solution_subfield

Base class for defining attributes of a subfield within a field.

Pyre Properties

- alias=<str>: Name for subfield.
 - **default value:** ''
 - **current value:** '', from {default}
 - **validator:** <function validateAlias at 0x124c314c0>
- basis_order=<int>: Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- cell_basis=<str>: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** 'default'
 - **current value:** 'default', from {default}
 - **validator:** (in ['simplex', 'tensor', 'default'])
- dimension=<int>: Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1

- **current value:** -1, from {default}
- **finite_element_space=<str>:** Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** 'polynomial'
 - **current value:** 'polynomial', from {default}
 - **validator:** (in ['polynomial', 'point'])
- **is_basis_continuous=<bool>:** Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>:** Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

SubfieldDisplacement

Full name

`pylith.problems.SubfieldDisplacement`

Journal name

`subfielddisplacement`

Object for defining attributes of the displacement solution subfield.

Implements `SolutionSubfield`.

Pyre Properties

- **alias=<str>:** Name for subfield.
 - **default value:** ''
 - **current value:** 'displacement', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}
 - **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>:** Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- **cell_basis=<str>:** Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** 'default'
 - **current value:** 'default', from {default}
 - **validator:** (in ['simplex', 'tensor', 'default'])
- **dimension=<int>:** Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}

- `finite_element_space=<str>`: Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** 'polynomial'
 - **current value:** 'polynomial', from {default}
 - **validator:** (in ['polynomial', 'point'])
- `is_basis_continuous=<bool>`: Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- `quadrature_order=<int>`: Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldDisplacement Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.displacement]
alias = displacement
basis_order = 1
```

SubfieldLagrangeFault

Full name

pylith.problems.SubfieldLagrangeFault

Journal name

subfieldlagrangefault

Object for defining attributes of the fault Lagrange multiplier solution subfield.

Implements SolutionSubfield.

Pyre Properties

- `alias=<str>`: Name for subfield.
 - **default value:** ''
 - **current value:** 'lagrange_multiplier_fault', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}
 - **validator:** <function validateAlias at 0x124c314c0>
- `basis_order=<int>`: Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- `cell_basis=<str>`: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** 'default'
 - **current value:** 'default', from {default}

- **validator:** (in ['simplex', 'tensor', 'default'])
- **dimension=<int>:** Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>:** Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** 'polynomial'
 - **current value:** 'polynomial', from {default}
 - **validator:** (in ['polynomial', 'point'])
- **is_basis_continuous=<bool>:** Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>:** Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldLagrangeFault Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.lagrange_multiplier_fault]
alias = lagrange_multiplier_fault
basis_order = 1
```

SubfieldPressure

Full name

pylith.problems.SubfieldPressure

Journal name

subfieldpressure

Object for defining attributes of the pressure solution subfield.

Implements SolutionSubfield.

Pyre Properties

- **alias=<str>:** Name for subfield.
 - **default value:** ''
 - **current value:** 'pressure', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}
 - **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>:** Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}

- `cell_basis=<str>`: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** 'default'
 - **current value:** 'default', from {default}
 - **validator:** (in ['simplex', 'tensor', 'default'])
- `dimension=<int>`: Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- `finite_element_space=<str>`: Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** 'polynomial'
 - **current value:** 'polynomial', from {default}
 - **validator:** (in ['polynomial', 'point'])
- `is_basis_continuous=<bool>`: Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- `quadrature_order=<int>`: Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldPressure Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.pressure]
alias = pressure
basis_order = 1
```

SubfieldPressureDot

Full name

pylith.problems.SubfieldPressureDot

Journal name

subfieldpressure_t

Object for defining attributes of the time derivative of pressure solution subfield.

Implements SolutionSubfield.

Pyre Properties

- `alias=<str>`: Name for subfield.
 - **default value:** ''
 - **current value:** 'pressure_t', from {file='/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py', line=26, function='set'}

- **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>:** Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- **cell_basis=<str>:** Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** 'default'
 - **current value:** 'default', from {default}
 - **validator:** (in ['simplex', 'tensor', 'default'])
- **dimension=<int>:** Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>:** Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** 'polynomial'
 - **current value:** 'polynomial', from {default}
 - **validator:** (in ['polynomial', 'point'])
- **is_basis_continuous=<bool>:** Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>:** Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldPressureDot Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.pressure_t]
alias = pressure_t
basis_order = 1
```

SubfieldTemperature

Full name

pylith.problems.SubfieldTemperature

Journal name

subfieldtemperature

Object for defining attributes of the temperature solution subfield.

Implements SolutionSubfield.

Pyre Properties

- **alias=<str>**: Name for subfield.
 - **default value:** ‘’
 - **current value:** ‘temperature’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>**: Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- **cell_basis=<str>**: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** ‘default’
 - **current value:** ‘default’, from {default}
 - **validator:** (in [‘simplex’, ‘tensor’, ‘default’])
- **dimension=<int>**: Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>**: Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** ‘polynomial’
 - **current value:** ‘polynomial’, from {default}
 - **validator:** (in [‘polynomial’, ‘point’])
- **is_basis_continuous=<bool>**: Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>**: Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldTemperature Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.temperature]
alias = absolute_temperature
basis_order = 1
```

SubfieldTraceStrain

Full name

pylith.problems.SubfieldTraceStrain

Journal name

subfieldtracestrain

Object for defining attributes of the trace strain solution subfield.

Implements SolutionSubfield.

Pyre Properties

- **alias=<str>**: Name for subfield.
 - **default value:** ‘’
 - **current value:** ‘trace_strain’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>**: Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- **cell_basis=<str>**: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** ‘default’
 - **current value:** ‘default’, from {default}
 - **validator:** (in [‘simplex’, ‘tensor’, ‘default’])
- **dimension=<int>**: Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>**: Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** ‘polynomial’
 - **current value:** ‘polynomial’, from {default}
 - **validator:** (in [‘polynomial’, ‘point’])
- **is_basis_continuous=<bool>**: Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>**: Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldTraceStrain Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.trace_strain]
alias = trace_strain
basis_order = 1
```

SubfieldTraceStrainDot

Full name

pylith.problems.SubfieldTraceStrainDot

Journal name

subfieldtracetrain_t

Object for defining attributes of the time derivative of trace strain solution subfield.

Implements SolutionSubfield.

Pyre Properties

- **alias=<str>**: Name for subfield.
 - **default value:** ‘’
 - **current value:** ‘trace_strain_t’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>**: Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- **cell_basis=<str>**: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** ‘default’
 - **current value:** ‘default’, from {default}
 - **validator:** (in [‘simplex’, ‘tensor’, ‘default’])
- **dimension=<int>**: Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>**: Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** ‘polynomial’
 - **current value:** ‘polynomial’, from {default}
 - **validator:** (in [‘polynomial’, ‘point’])
- **is_basis_continuous=<bool>**: Is basis continuous?
 - **default value:** True

- **current value:** True, from {default}
- **quadrature_order=<int>:** Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldTraceStrainDot Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.trace_strain_t]
alias = trace_strain_t
basis_order = 1
```

SubfieldVelocity

Full name

pylith.problems.SubfieldVelocity

Journal name

subfieldvelocity

Object for defining attributes of the velocity solution subfield.

Implements SolutionSubfield.

Pyre Properties

- **alias=<str>:** Name for subfield.
 - **default value:** “
 - **current value:** ‘velocity’, from {file=’/software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pythia/pyre/inventory/ConfigurableClass.py’, line=26, function=’set’}
 - **validator:** <function validateAlias at 0x124c314c0>
- **basis_order=<int>:** Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- **cell_basis=<str>:** Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.
 - **default value:** ‘default’
 - **current value:** ‘default’, from {default}
 - **validator:** (in [‘simplex’, ‘tensor’, ‘default’])
- **dimension=<int>:** Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>:** Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** ‘polynomial’

- **current value:** ‘polynomial’, from {default}
- **validator:** (in [‘polynomial’, ‘point’])
- **is_basis_continuous=<bool>:** Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>:** Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

Example

Example of setting SubfieldVelocity Pyre properties and facilities in a parameter file.

```
[pylithapp.problems.solution.subfields.velocity]
alias = velocity
basis_order = 1
```

TimeDependent

Full name

pylith.problems.TimeDependent

Journal name

timedependent

Static, quasistatic, or dynamic time-dependent problem.

Implements Problem.

Pyre Facilities

- **bc:** Boundary conditions.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, bc
- **defaults:** Default options for problem.
 - **current value:** ‘problem_defaults’, from {default}
 - **configurable as:** problem_defaults, defaults
- **gravity_field:** Database used for gravity field.
 - **current value:** ‘nullcomponent’, from {default}
 - **configurable as:** nullcomponent, gravity_field
- **ic:** Initial conditions.
 - **current value:** ‘emptybin’, from {default}
 - **configurable as:** emptybin, ic
- **interfaces:** Interior surfaces with constraints or constitutive models.
 - **current value:** ‘emptybin’, from {default}

- **configurable as:** emptybin, interfaces
- materials: Materials in problem.
 - **current value:** ‘homogeneous’, from {default}
 - **configurable as:** homogeneous, materials
- mesh_initializer: Mesh initializer.
 - **current value:** ‘mesh_initializer’, from {default}
 - **configurable as:** mesh_initializer
- petsc_defaults: Flags controlling which default PETSc options to use.
 - **current value:** ‘petscdefaults’, from {default}
 - **configurable as:** petscdefaults, petsc_defaults
- progress_monitor: Simple progress monitor via text file.
 - **current value:** ‘progressmonitortime’, from {default}
 - **configurable as:** progressmonitortime, progress_monitor
- scales: Scales for nondimensionalizing boundary value problem.
 - **current value:** ‘quasistaticelasticity’, from {default}
 - **configurable as:** quasistaticelasticity, scales
- solution: Solution field for problem.
 - **current value:** ‘solution’, from {default}
 - **configurable as:** solution
- solution_observers: Observers (e.g., output) for solution.
 - **current value:** ‘singlesolnobserver’, from {default}
 - **configurable as:** singlesolnobserver, solution_observers

Pyre Properties

- end_time=<dimensional>: End time for problem.
 - **default value:** 3.15576e+06*s
 - **current value:** 3.15576e+06*s, from {default}
 - **validator:** (greater than or equal to 0*s)
- formulation=<str>: Formulation for equations.
 - **default value:** ‘quasistatic’
 - **current value:** ‘quasistatic’, from {default}
 - **validator:** (in [‘quasistatic’, ‘dynamic’, ‘dynamic_imex’])
- initial_dt=<dimensional>: Initial time step.
 - **default value:** 3.15576e+07*s
 - **current value:** 3.15576e+07*s, from {default}
 - **validator:** (greater than 0*s)

- `max_timesteps=<int>`: Maximum number of time steps.
 - **default value:** 20000
 - **current value:** 20000, from {default}
 - **validator:** (greater than 0)
- `notify_observers_ic=<bool>`: Notify observers of solution with initial conditions.
 - **default value:** False
 - **current value:** False, from {default}
- `solver=<str>`: Type of solver to use ['linear', 'nonlinear'].
 - **default value:** 'nonlinear'
 - **current value:** 'nonlinear', from {default}
 - **validator:** (in ['linear', 'nonlinear'])
- `start_time=<dimensional>`: Start time for problem.
 - **default value:** 0*s
 - **current value:** 0*s, from {default}

Example

Example of setting TimeDependent Pyre properties and facilities in a parameter file.

```
# Set boundary conditions, faults, and materials
bc = [boundary_xpos, boundary_xneg]
interfaces = [san_andreas, hayward]
materials = [crust, mantle]

# Create an initial condition over the domain
ic = [domain]

# Turn on gravitational body forces
gravity_field = spatialdata.spatialdb.GravityField

# Set the scales for nondimensionalizing the problem
scales = pylith.scales.QuasistaticElasticity

# Set the subfields in the solution
solution = = pylith.problems.SolnDispLagrange

# Output the solution for the domain and ground surface
solution_observers = [domain, ground_surface]

# Use the quasistatic formulation, linear solver, and set appropriate default solver
↳ settings.
formulation = quasistatic
solver = linear

# Use a maximum of 20 time steps to simulation from -0.5 years to 2.0 years with an
↳ initial time step of 0.5 years.
# The first time step will compute the solution at time 0.
```

(continues on next page)

(continued from previous page)

```
start_time = -0.5*year
end_time = 2.0*year
initial_dt = 0.5*year
max_timesteps = 20

[pylithapp.greensfns.petsc_defaults]
solver = True
monitors = True
```

scales

General

Full name`pylith.scales.General`**Journal name**`scales`

Abstract base class for nondimensionalizing problems.

QuasistaticElasticity

Full name`pylith.scales.QuasistaticElasticity`**Journal name**`quasistaticelasticity`

Convenience object for nondimensionalizing quasi-static elasticity problems.

Implements General.

Pyre Properties

- `displacement_scale=<dimensional>`: Nominal displacement scale in boundary value problem.
 - **default value:** `1*m`
 - **current value:** `1*m`, from {default}
 - **validator:** (greater than `0*m`)
- `length_scale=<dimensional>`: Length scale in boundary value problem (size of feature controlling displacement, fault).
 - **default value:** `100000*m`
 - **current value:** `100000*m`, from {default}
 - **validator:** (greater than `0*m`)
- `shear_modulus=<dimensional>`: Nominal shear modulus in boundary value problem.
 - **default value:** `1e+10m**-1kg*s**-2`
 - **current value:** `1e+10m**-1kg*s**-2`, from {default}
 - **validator:** (greater than `0m**-1kg*s**-2`)
- `time_scale=<dimensional>`: Time scale of boundary value problem (for example, viscoelastic relaxation time).

- **default value:** 3.15576e+09*s
- **current value:** 3.15576e+09*s, from {default}
- **validator:** (greater than 0*s)

Example

Example of setting QuasistaticElasticity Pyre properties and facilities in a parameter file.

```
[normalizer]
length_scale = 1.0*km
displacement_scale = 50.0*km
shear_modulus = 25.0*GPa
time_scale = 100.0*year
```

QuasistaticPoroelasticity

Full name
pylith.scales.QuasistaticPoroelasticity

Journal name
quasistaticporoelasticity

Convenience object for nondimensionalizing quasi-static poroelasticity problems.

Implements General.

Pyre Properties

- **displacement_scale=<dimensional>:** Nominal displacement scale in boundary value problem.
 - **default value:** 1*m
 - **current value:** 1*m, from {default}
 - **validator:** (greater than 0*m)
- **length_scale=<dimensional>:** Length scale in boundary value problem (size of feature controlling displacement, fault).
 - **default value:** 100000*m
 - **current value:** 100000*m, from {default}
 - **validator:** (greater than 0*m)
- **permeability=<dimensional>:** Nominal permeability in boundary value problem.
 - **default value:** 1e-13*m**2
 - **current value:** 1e-13*m**2, from {default}
 - **validator:** (greater than 0m**-Ikg*s**-2)
- **shear_modulus=<dimensional>:** Nominal shear modulus in boundary value problem.
 - **default value:** 1e+10m**-Ikg*s**-2
 - **current value:** 1e+10m**-Ikg*s**-2, from {default}
 - **validator:** (greater than 0m**-Ikg*s**-2)
- **viscosity=<dimensional>:** Nominal fluid viscosity in boundary value problem.

- **default value:** $0.001\text{m}^{**}\text{-Ikg*s}^{**}\text{-1}$
- **current value:** $0.001\text{m}^{**}\text{-Ikg*s}^{**}\text{-1}$, from {default}
- **validator:** (greater than $0\text{m}^{**}\text{-Ikg*s}^{**}\text{-2}$)

Example

Example of setting QuasistaticPoroelasticity Pyre properties and facilities in a parameter file.

```
[normalizer]
length_scale = 1.0*km
displacement_scale = 50.0*km
shear_modulus = 25.0*GPa
viscosity = 0.001*Pa*s
permeability = 1.0e-12*m**2
```

DynamicElasticity

Full name

pylith.scales.DynamicElasticity

Journal name

dynamicelasticity

Convenience object for nondimensionalizing dynamic elasticity problems.

Implements General.

Pyre Properties

- **displacement_scale=<dimensional>:** Nominal displacement scale in boundary value problem.
 - **default value:** $1*\text{m}$
 - **current value:** $1*\text{m}$, from {default}
 - **validator:** (greater than $0*\text{m}$)
- **length_scale=<dimensional>:** Length scale in boundary value problem (size of feature controlling displacement, fault).
 - **default value:** $100000*\text{m}$
 - **current value:** $100000*\text{m}$, from {default}
 - **validator:** (greater than $0*\text{m}$)
- **shear_modulus=<dimensional>:** Nominal shear modulus in boundary value problem.
 - **default value:** $1\text{e}+10\text{m}^{**}\text{-Ikg*s}^{**}\text{-2}$
 - **current value:** $1\text{e}+10\text{m}^{**}\text{-Ikg*s}^{**}\text{-2}$, from {default}
 - **validator:** (greater than $0\text{m}^{**}\text{-Ikg*s}^{**}\text{-2}$)
- **shear_wave_speed=<dimensional>:** Nominal shear wave speed in boundary value problem.
 - **default value:** $3000\text{ms}^{**}\text{-1}$
 - **current value:** $3000\text{ms}^{**}\text{-1}$, from {default}
 - **validator:** (greater than $0*\text{s}$)

Example

Example of setting DynamicElasticity Pyre properties and facilities in a parameter file.

```
[normalizer]
length_scale = 1.0*km
displacement_scale = 50.0*km
shear_modulus = 25.0*GPa
velocity_scale = 3.0*km/s
```

topology

Distributor

Full name

pylith.topology.Distributor

Journal name

mesh_distributor

Distributor of the the mesh among processes.

Pyre Facilities

- data_writer: Data writer for partition information.
 - **current value:** 'datawriterhdf5', from {default}
 - **configurable as:** datawriterhdf5, data_writer

Pyre Properties

- partitioner=<str>: Name of mesh partitioner (PETSc must be built with partitioner).
 - **default value:** 'parmetis'
 - **current value:** 'parmetis', from {default}
 - **validator:** (in ['parmetis', 'chaco', 'simple'])
- use_edge_weighting=<bool>: Use edge weighting (parmetis only).
 - **default value:** True
 - **current value:** True, from {default}
- write_partition=<bool>: Write partition information to file.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting Distributor Pyre properties and facilities in a parameter file.

```
[pylithapp.mesh_generator.distributor]
partitioner = parmetis
use_edge_weighting = True
write_partition = False
data_writer = pylith.meshio.DataWriterHDF5
```

RefineMesh

Full name

pylith.topology.RefineMesh

Journal name

refiner

Abstract base class for refining a mesh in parallel.

RefineUniform

Full name

pylith.topology.RefineUniform

Journal name

refineuniform

Uniform global mesh refinement in parallel.

Implements MeshRefiner.

Pyre Properties

- `levels=<int>`: Number of refinement levels.
 - **default value:** 0
 - **current value:** 0, from {default}
 - **validator:** (greater than or equal to 0)

Example

Example of setting RefineUniform Pyre properties and facilities in a parameter file.

```
# Refine mesh twice to reduce size of cell edges by a factor of 4.  
[pylithapp.mesh_generator.refiner]  
levels = 0
```

Subfield

Full name

pylith.topology.Subfield

Journal name

subfield

Python object for defining discretization of a subfield.

Pyre Properties

- `basis_order=<int>`: Order of basis functions.
 - **default value:** 1
 - **current value:** 1, from {default}
- `cell_basis=<str>`: Type of cell basis functions (simplex, tensor, or default). Default is to use type matching cell type.

- **default value:** ‘default’
 - **current value:** ‘default’, from {default}
 - **validator:** (in [‘simplex’, ‘tensor’, ‘default’])
- **dimension=<int>:** Topological dimension associated with subfield (=-1 will use dimension of domain).
 - **default value:** -1
 - **current value:** -1, from {default}
- **finite_element_space=<str>:** Finite-element space (polynomial or point). Point space corresponds to delta functions at quadrature points.
 - **default value:** ‘polynomial’
 - **current value:** ‘polynomial’, from {default}
 - **validator:** (in [‘polynomial’, ‘point’])
- **is_basis_continuous=<bool>:** Is basis continuous?
 - **default value:** True
 - **current value:** True, from {default}
- **quadrature_order=<int>:** Order of numerical quadrature.
 - **default value:** -1
 - **current value:** -1, from {default}

utils

CollectVersionInfo

Full name

pylith.utils.CollectVersionInfo

Journal name

collectversioninfo

Collect version information for PyLith and its dependencies.

DumpParameters

Full name

pylith.utils.DumpParameters

Journal name

dumpparameters

Abstract base class for dumping PyLith parameter information to a file.

DumpParametersAscii

Full name

pylith.utils.DumpParametersAscii

Journal name

dumpparameters

Dump PyLith parameter information to an ASCII file.

If you do not set the filename for the progress monitor, then PyLith will create one using the simulation name from the application defaults settings.

Implements DumpParameters.

Pyre Properties

- filename=<str>: Name of file written with parameters.
 - **default value:** ‘
 - **current value:** ‘, from {default}
- indent=<int>: Number of spaces to indent.
 - **default value:** 4
 - **current value:** 4, from {default}
- verbose=<bool>: Include description, location, and aliases.
 - **default value:** True
 - **current value:** True, from {default}

Example

Example of setting DumpParametersAscii Pyre properties and facilities in a parameter file.

```
[pylithapp]
dump_parameters = pylith.utils.DumpParametersAscii

[pylithapp.dump_parameters]
filename = output/parameters.txt
verbose = True
```

DumpParametersJson

Full name
pylith.utils.DumpParametersJson

Journal name
dumpparamters

Dump PyLith parameter information to an ASCII file.

If you do not set the filename for the progress monitor, then PyLith will create one using the simulation name from the application defaults settings.

Implements DumpParameters.

Pyre Properties

- filename=<str>: Name of file written with parameters.
 - **default value:** ‘
 - **current value:** ‘, from {default}
- indent=<int>: Number of spaces to indent, use a negative number for no newlines.

- **default value:** 4
- **current value:** 4, from {default}
- **style=<str>:** Style of JSON file [compact, normal].
 - **default value:** 'normal'
 - **current value:** 'normal', from {default}
 - **validator:** (in ['normal', 'compact'])

Example

Example of setting DumpParametersJson Pyre properties and facilities in a parameter file.

```
[pylithapp]
dump_parameters = pylith.utils.DumpParametersJson

[pylithapp.dump_parameters]
filename = output/parameters.json
style = normal
verbose = True
```

EmptyBin

Full name
pylith.utils.EmptyBin

Journal name
emptybin

Empty container for a collection of objects.

NullComponent

Full name
pylith.utils.NullComponent

Journal name
nullcomponent

Empty Pyre component.

PetscDefaults

Full name
pylith.utils.PetscDefaults

Journal name
petscdefaults

Flags controlling use of default PETSc settings. No user-specified settings will be overwritten.

Pyre Properties

- **adaptive_time_stepping=<bool>:** Use adaptive time stepping.
 - **default value:** False

- **current value:** False, from {default}
- **collective_io=<bool>:** Use default PETSc collective I/O options.
 - **default value:** True
 - **current value:** True, from {default}
- **impulse_ts=<bool>:** Use custom PETSc TS for impulses.
 - **default value:** False
 - **current value:** False, from {default}
- **initial_guess=<bool>:** Use initial guess options.
 - **default value:** True
 - **current value:** True, from {default}
- **monitors=<bool>:** Use default solver monitors.
 - **default value:** True
 - **current value:** True, from {default}
- **parallel=<bool>:** Use solver settings normally used when running in parallel.
 - **default value:** False
 - **current value:** False, from {default}
- **solver=<bool>:** Use default solver settings based on governing equations.
 - **default value:** True
 - **current value:** True, from {default}
- **testing=<bool>:** Use default PETSc testing options.
 - **default value:** False
 - **current value:** False, from {default}

Example

Example of setting `PetscDefaults` Pyre properties and facilities in a parameter file.

```
[pylithapp.problem.petsc_defaults]
solver = True
parallel = False
monitors = True
initial_guess = True
adaptive_time_stepping = False
collective_io = True
testing = False
```

PetscManager

Full name

`pylith.utils.PetscManager`

Journal name

`petsc`

Manage PETSc options.

Example

Example of setting PetscManager Pyre properties and facilities in a parameter file.

```
[pylithapp.petsc]
ts_monitor = true
ksp_monitor = true
ksp_converged_reason = true
snes_monitor = true
snes_converged_reason = true
snes_linesearch_monitor = true
```

PropertyList

Full name
pylith.utils.PropertyList

Journal name
propertylist

Container for holding a list of items (one per line).

SimulationMetadata

Full name
pylith.utils.SimulationMetadata

Journal name
metadata

Metadata for simulation.

When using base to specify other files with metadata, the other files will append to the keywords and features lists, whereas other metadata will be overwritten (the same behavior as other Pyre properties).

Pyre Properties

- arguments=<list>: Command line arguments for running simulation.
 - **default value:** []
 - **current value:** [], from {default}
- authors=<list>: Creator(s) of simulation.
 - **default value:** []
 - **current value:** [], from {default}
- base=<list>: Parameter files with metadata that complement this metadata.
 - **default value:** []
 - **current value:** [], from {default}
- description=<str>: Description of simulation.
 - **default value:** “
 - **current value:** “, from {default}

- **validator**: <function notEmptyString at 0x124cdf820>
- **features**=<list>: PyLith features used in simulation.
 - **default value**: []
 - **current value**: [], from {default}
- **keywords**=<list>: Keywords describing simulation.
 - **default value**: []
 - **current value**: [], from {default}
- **pylith_version**=<list>: PyLith versions compatible with simulation input files.
 - **default value**: []
 - **current value**: [], from {default}
- **version**=<str>: Version number for simulation.
 - **default value**: ‘
 - **current value**: ‘, from {default}

Example

Example of setting SimulationMetadata Pyre properties and facilities in a parameter file.

```
[pylithapp.metadata]
base = [pylithapp.cfg]
description = Axial extension using Dirichlet boundary conditions.
keywords = [example, 2D, box, axial extension]
features = [
    Quadrilateral cells,
    pylith.meshio.MeshIOAscii,
    pylith.problems.TimeDependent,
    pylith.materials.Elasticity,
    pylith.materials.IsotropicLinearElasticity,
    spatialdata.spatialdb.UniformDB,
    pylith.meshio.DataWriterHDF5
]
authors = [Brad Aagaard]
version = 1.0.0
arguments = [step01_axialdisp.cfg]
pylith_version = [>=3.0, <5.0]
```

4.2.9 Examples

Overview

This section includes several suites of examples available in the `examples` directory. Each suite includes several “steps” which are examples that increase in complexity from one “step” to the next. In some cases, a later step may make use of output from an earlier step; these cases are clearly documented. [Table 4.43](#) classifies the level of difficulty of each example suite and provides a general description of the type of problems discussed.

Table 4.43: Overview of example suites.

Example Suite	Difficulty	Description
<i>box-2d</i>	novice	Simple axial and shear deformation in static and quasi-static simulations in 2D box with a mesh in an ASCII text file.
<i>box-3d</i>	novice	Same as 2d/box but with a 3D box and a mesh from Gmsh or Cubit.
<i>strikeslip-2d</i>	beginner	Prescribed coseismic slip and multiple earthquake ruptures in 2D with a mesh from Gmsh or Cubit.
<i>reverse-2d</i>	beginner	Gravity, surface loads, and prescribed coseismic slip on multiple reverse faults in 2D with a mesh from Gmsh or Cubit.
<i>crustal-strikesli</i>	intermediate	Coseismic deformation on 2D intersecting strike-slip faults with a mesh from Gmsh or Cubit.
<i>crustal-strikesli</i>	intermediate	Coseismic deformation on 3D intersecting strike-slip faults with a mesh from Gmsh or Cubit.
<i>subduction-2d</i>	intermediate	Coseismic, postseismic, and creep deformation using a 2D subduction zone cross-section with a mesh from Gmsh or Cubit.
<i>subduction-3d</i>	intermediate	Close to research-complexity for a 3D subduction zone with a mesh from Cubit.
<i>magma-2d</i>	intermediate	Magma reservoir using poroelasticity.
<i>poroelastic-outer</i>	intermediate	Hydration in the outer-rise of subducting oceanic lithosphere using poroelasticity.
<i>troubleshooting-2</i>	novice	Troubleshooting errors in simulation in put files.

The *crustal-strikeslip-2d*, *crustal-strikeslip-3d*, and *subduction-3d* example suites are the most advanced. If you want to use PyLith in your research, we recommend that you work through relevant beginner examples and then these more sophisticated examples.

Tip

You can use the `pylith_cfgsearch` utility (see [Utilities](#)) to search for examples based on keywords and features.

Prerequisites

Before you begin any of the examples, you will need to install PyLith following the instructions in [Installation](#). You should also read [Running PyLith](#). Complete sets of input files are included in the examples. However, if you wish to generate the finite-element meshes yourself, you will also need Gmsh (available from <https://gmsh.info> and included in the PyLith binary package), Coreform Cubit (available from <https://coreform.com/>), or Cubit (available to US federal government agencies from <https://cubit.sandia.gov/>).

Input Files

The files needed to work through the examples are found in the `examples` directory under the top-level PyLith directory. All of the files used in the example problems are extensively documented with comments.

Visualizing PyLith Output

You can visualize the results using the `pylith_viz` utility or other general scientific visualization tools, such as [ParaView](#). ParaView includes built-in documentation that is accessed by clicking on the Help menu item. Some additional documentation is available on the ParaView Wiki site <https://www.paraview.org/Wiki/ParaView>.

Examples

Axial and Shear Deformation (2D Box)

The files are in the directory `examples/box-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

***.mesh**

Finite-element mesh files generated manually using a text editor.

***.spatialdb**

Spatial database files associated with the spatial databases.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This suite of examples demonstrates some basic concepts of using PyLith to solve the static and quasistatic boundary elasticity equation in a 2D box (Fig. 4.25) with uniform material properties. This example incrementally adds complexity through a series of steps:

Step 1

Axial extension with Dirichlet (displacement) boundary conditions.

Step 2

Shear deformation with Dirichlet (displacement) boundary conditions.

Step 3

Shear deformation with Dirichlet (displacement) and Neumann (traction) boundary conditions.

Step 4

Same as Step 2 but with initial conditions equal to the analytical solution.

Step 5

Shear deformation with time-dependent Dirichlet (displacement) and Neumann (traction) boundary conditions.

Example Workflow

Mesh Description

For this small, simple problem we constructed the finite-element mesh manually in a text editor using the *MeshIOAscii file format*. Fig. 4.26 shows our mesh with quadrilateral cells and a discretization size of 4 km from the file `quad.mesh`; we also provide a mesh with triangular cells in `tri.mesh`. The mesh file contains the coordinates of the vertices, the index of the vertices in each cell, the material identifier for each cell, and groups of cell faces associated with boundaries. The cells are defined by its vertices listed in a counterclockwise traversal of the cell boundary; the starting point is arbitrary. For example, cell 0 is defined by the vertices (0, 5, 6, 1); we also could have defined the cell using (5, 6, 1, 0), (6, 1, 0, 5), or (1, 0, 5, 6). We use the cell face groups to apply boundary conditions.

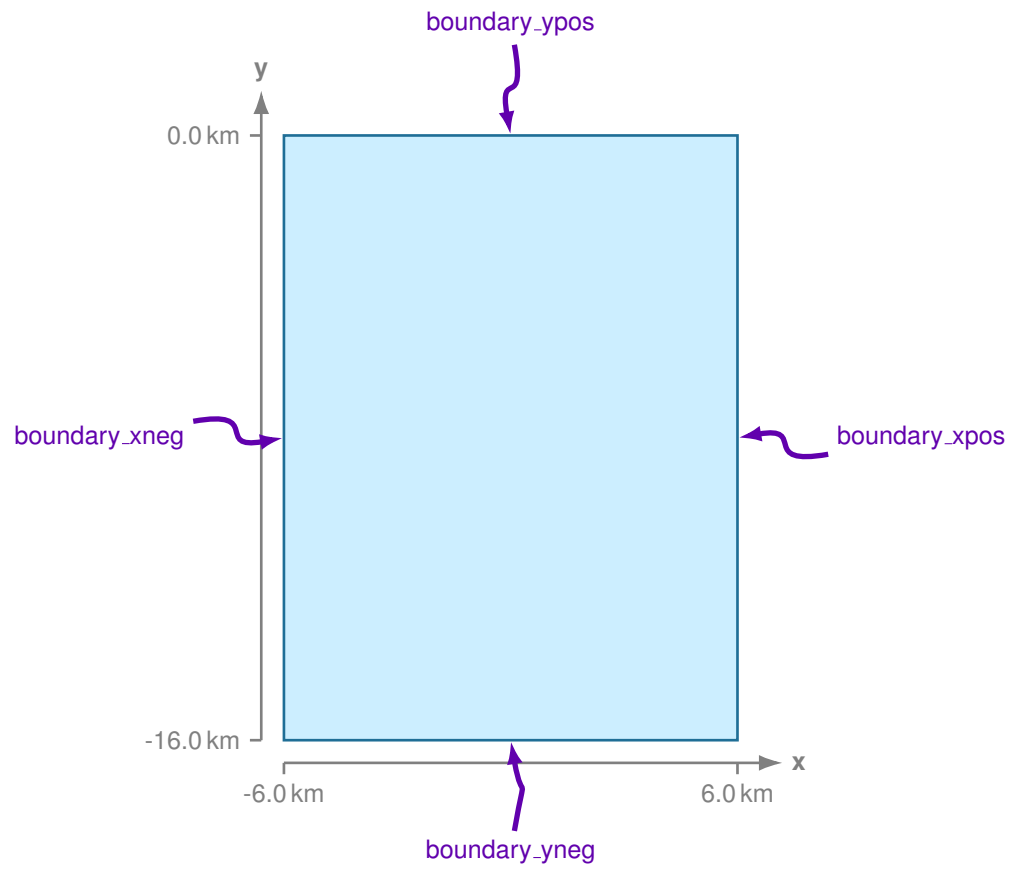


Fig. 4.25: Diagram of geometry for 2D box that extends from -6 km to $+6\text{ km}$ in the x direction and from -16 km to 0 km in the y direction. We refer to the domain boundaries using the names shown in the diagram.

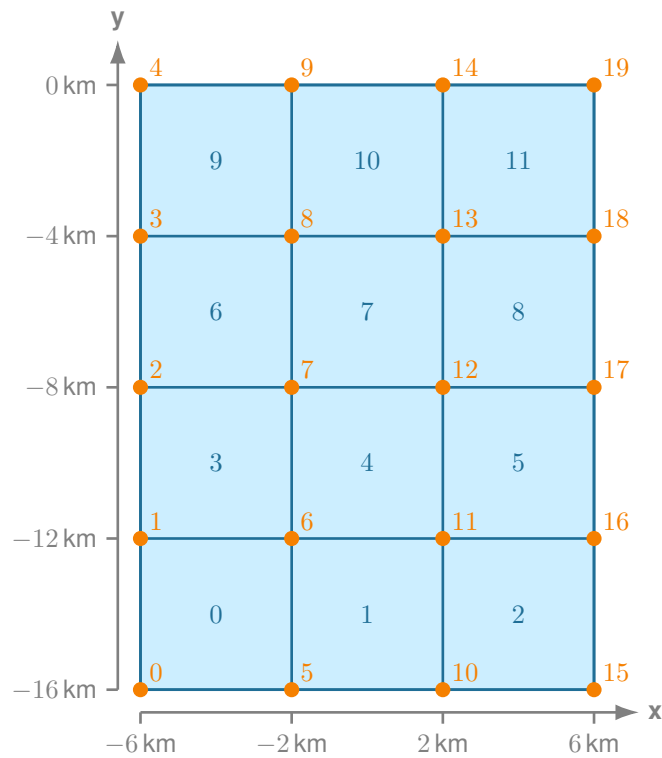


Fig. 4.26: Uniform resolution finite-element mesh with quadrilateral cells. Vertices are numbered sequentially from zero with the labels adjacent to each vertex. Cells are numbered sequentially from zero with the labels at the center of each cell.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`. This limits duplicate information in the parameter files for each simulation.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh in `pylithapp.mesh_generator`.

Physics

All of the simulations in this example suite solve the elasticity equation without inertia,

$$\vec{s} = \left(\vec{u} \right)^T \quad (4.465)$$

$$\nabla \cdot \sigma(\vec{u}) = \vec{0} \quad (4.466)$$

The default PyLith settings target solving the quasi-static elasticity equation, so we can use the default `TimeDependent` problem and solution field with a single displacement subfield of basis order 1.

Listing 4.32: Default PyLith problem and solution field settings appropriate for quasi-static elasticity. We do not include these in `pylithapp.cfg`.

```
[pylithapp]
# Default problem and solution field.
problem = pylith.problems.TimeDependent
problem.solution = pylith.problems.SolnDisp

[pylithapp.problem.solution.subfields]
# The default basis order for solution subfields is 1 (linear variation within cells).
displacement.basis_order = 1

[pylithapp.problem]
# Default output for the solution is over the domain.
solution_observers = [domain]
```

The default material and bulk rheology also target solving the quasi-static elasticity equation. The default bulk rheology is isotropic linear elasticity.

Listing 4.33: Default PyLith material and bulk rheology settings appropriate for quasi-static elasticity. We do not include these in `pylithapp.cfg`.

```
[pylithapp.problem]
materials = [elastic]
materials.elastic = pylith.materials.Elasticity

[pylithapp.problem.materials.elastic]
bulk_rheology = pylith.materials.IsotropicLinearElasticity
```

In this suite of examples we have a single material, and we must specify a description for the material and the label value. In this case the label value corresponds to the `material-id` in the finite-element mesh file. The physical properties for each material are specified in a spatial database. The material properties are uniform, so we use a single `UniformDB` to specify the material properties in `pylithapp.cfg`. When we have a spatial variation in the material

properties, we specify them in a spatial database file. With uniform properties we use a basis order of 0 for the auxiliary subfields.

Listing 4.34: Material settings in `pylithapp.cfg`.

```
[pylithapp.problem]
materials = [elastic]

[pylithapp.problem.materials.elastic]
label_value = 0

# We will use uniform material properties, so we use the UniformDB
# spatial database.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Elastic properties
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500*kg/m**3, 3.0*km/s, 5.2915026*km/s]

# Set the discretization of the material auxiliary fields (properties).
# We have uniform material properties, so we can use a basis order of 0.
auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0
```

Step 1: Axial Extension

Features

- Quadrilateral cells
- `pylith.meshio.MeshIOAscii`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- LU preconditioner
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.ZeroDB`

Simulation parameters

This example corresponds to axial extension in the x direction. We apply Dirichlet (displacement) boundary conditions for the x displacement on the +x (`boundary_xpos`) and -x (`boundary_xneg`) boundaries. We apply roller Dirichlet boundary conditions on the -y (`boundary_yneg`) boundary. [Fig. 4.27](#) shows the boundary conditions on the domain. The parameters specific to this example are in `step01_axialdisp.cfg`.

We create an array of 3 `DirichletTimeDependent` boundary conditions. For each of these boundary conditions we must specify which degrees of freedom are constrained, the name of the label marking the boundary (name of the group of vertices in the finite-element mesh file), and the values for the Dirichlet boundary condition.

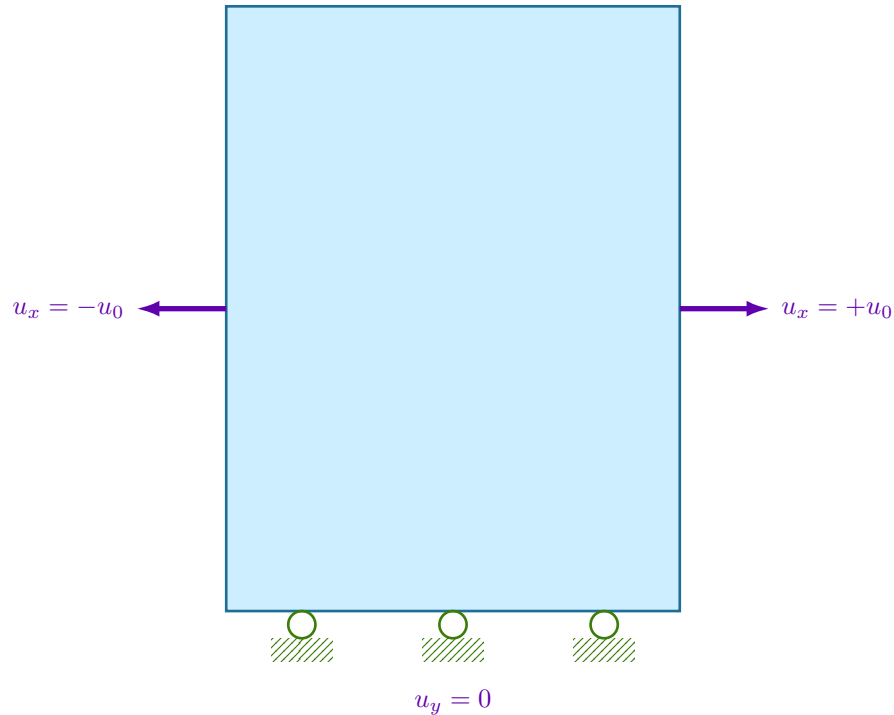


Fig. 4.27: Boundary conditions for axial extension in the x-direction. We constrain the x displacement on the +x and -x boundaries and set the y displacement to zero on the -y boundary.

Listing 4.35: Specifying the boundary conditions for Step 1. We only show the detailed settings for the +x boundary.

```
[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xpos]
# Set Ux=+2.0*m on the +x boundary.
# Degree of freedom (dof) 0 corresponds to x displacement.
constrained_dof = [0]
label = boundary_xpos

# The spatial database must contain both components even though we do
# not constrain the y component.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Dirichlet BC on +x boundary
db_auxiliary_field.values = [initial_amplitude_x, initial_amplitude_y]
db_auxiliary_field.data = [+2.0*m, 0*m]
```

Running the simulation

Listing 4.36: Run Step 1 simulation

```

$ pylith step01_axialdisp.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOAscii.cc:175:virtual void
↳ pylith::meshio::MeshIOAscii::_read()
-- info (application-flow)
-- Component 'meshioascii.reader': Reading finite-element mesh from 'quad.mesh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOAscii.cc:290:virtual void
↳ pylith::meshio::MeshIOAscii::_read()
-- info (application-flow)
-- Component 'meshioascii.reader': Read 12 cells and 20 vertices.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshioascii.reader': Domain bounding box:
    (-6000, 6000)
    (-16000, -0)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
↳ pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
↳ printInfo

```

(continues on next page)

(continued from previous page)

```

-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 100000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↳pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 3.737646285937e+01
      Linear solve converged due to CONVERGED_ATOL iterations 4
    1 SNES Function norm 2.117345506899e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/Utils/PetscManager.
↳py:60:finalize
-- info (application-flow)
-- Finalizing PETSc.

```

At the beginning of the output written to the terminal, we see the PETSc options PyLith selected based on the governing equations and formulation as discussed in *PETSc Options*. Next, we see that PyLith is reading the mesh using the MeshIOAscii reader and that it found the domain to extend from -6000 m to +6000 m in the x direction and from -16000 m to 0 m in the y direction. We also see the scales used to nondimensionalize the problem.

Near the end of the output, we see that the solver advanced the solution one time step (static simulation). The linear solve converged in 4 iterations. The norm of the residual met the absolute tolerance convergence criterion (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`). For this small problem, the multi-grid preconditioner has fewer levels, so we get a warning about an unused PETSc option.

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (.h5) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (.xmf) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In Fig. 4.28 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.37: Visualize PyLith output using pylith_viz.

```
pylith_viz --filenames=output/step01_axialdisp-domain.h5 warp_grid --component=x
```

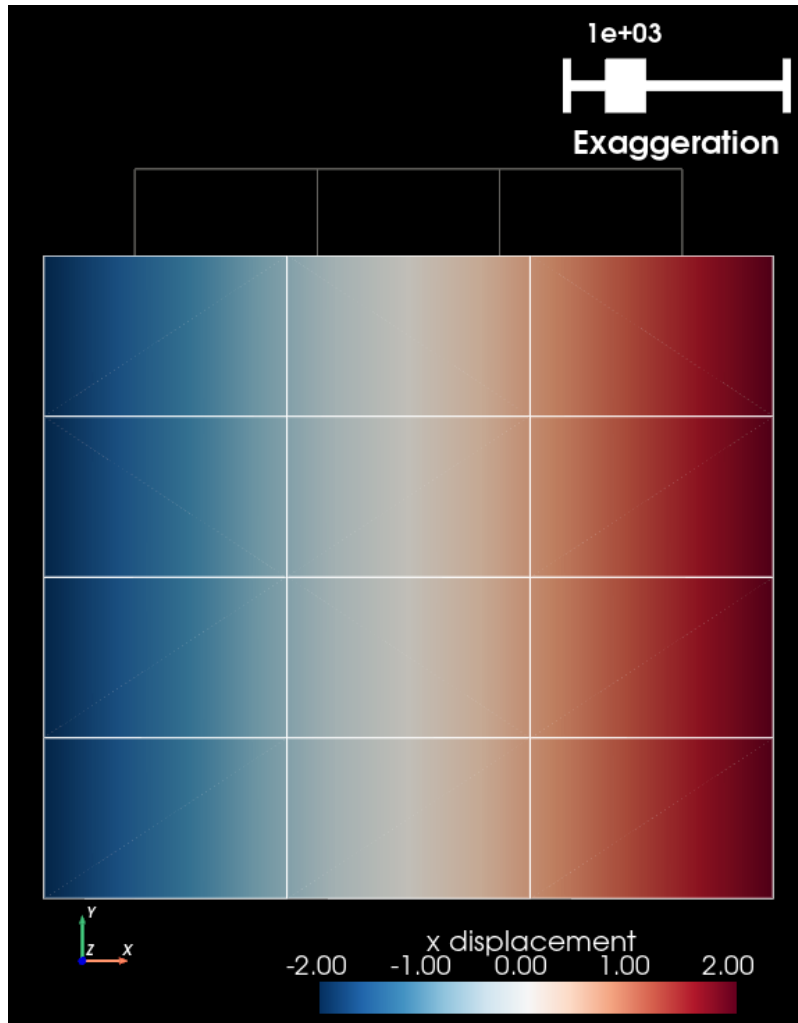


Fig. 4.28: Solution for Step 1. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

Step 2: Shear Displacement

Features

- Quadrilateral cells
- `pylith.meshio.MeshIOAscii`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.UniformDB`

- `pylith.meshio.DataWriterHDF5`
- Static simulation
- LU preconditioner
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`

Simulation parameters

This example corresponds to shear deformation due to Dirichlet (displacement) boundary conditions. We apply Dirichlet (displacement) boundary conditions for the y displacement on the +x (`boundary_xpos`) and -x (`boundary_xneg`) boundaries and for the x displacement on the +y (`boundary_ypos`) and -y (`boundary_yneg`) boundaries. Fig. 4.27 shows the boundary conditions on the domain. The parameters specific to this example are in `step02_sheardisp.cfg`.

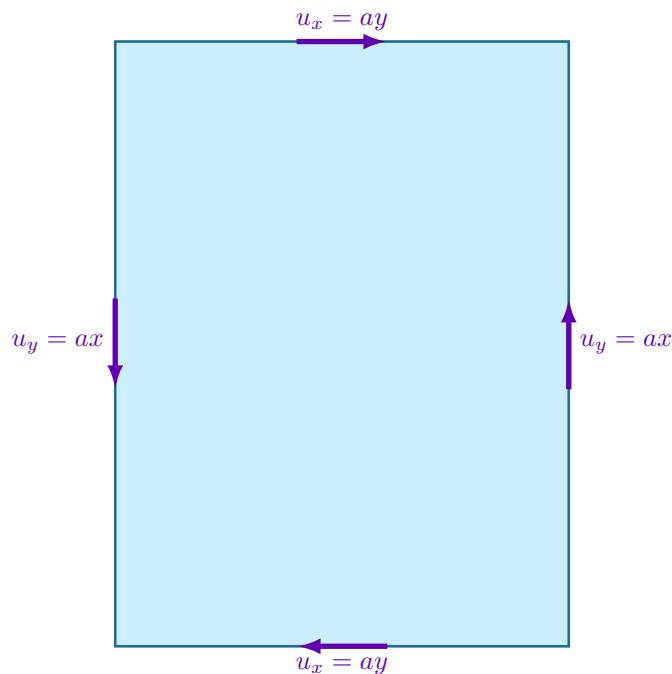


Fig. 4.29: Boundary conditions for shear deformation. We constrain the y displacement on the +x and -x boundaries and the x displacement on the +y and -y boundaries.

We create an array of 4 `DirichletTimeDependent` boundary conditions. For each of these boundary conditions we must specify which degrees of freedom are constrained, the name of the label marking the boundary (name of the group of vertices in the finite-element mesh file), and the values for the Dirichlet boundary condition. The displacement field varies along each boundary, so we use a `SimpleDB` spatial database and the `linear` query type.

Listing 4.38: Specifying the boundary conditions for Step 2. We only show the detailed settings for the -x boundary.

```
[pylithapp.problem]
bc = [bc_xneg, bc_yneg, bc_xpos, bc_ypos]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.DirichletTimeDependent
```

(continues on next page)

(continued from previous page)

```

bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_ypos = pylith.bc.DirichletTimeDependent

# Degree of freedom (dof) 1 corresponds to y displacement.
constrained_dof = [1]
label = boundary_xneg
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC -x edge
db_auxiliary_field.iohandler.filename = sheardisp_bc_xneg.spatialdb
db_auxiliary_field.query_type = linear

```

Running the simulation

Listing 4.39: Run Step 2 simulation

```

$ pylith step02_sheardisp.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines removed --

0 TS dt 0.001 time 0.
  0 SNES Function norm 6.719933035381e+00
    Linear solve converged due to CONVERGED_ATOL iterations 3
  1 SNES Function norm 1.640603910123e-07
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
  ↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The output written to the terminal is nearly identical to what we saw for Step 1. We omit the middle portion of the output which shows that the domain, the scales for nondimensionalization, and PETSc options all remain the same.

Visualizing the results

In Fig. 4.30 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.40: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step02_sheardisp-domain.h5 warp_grid --component=x
```

Step 3: Shear Displacement and Traction

Features

- Quadrilateral cells
- `pylith.meshio.MeshIOAscii`

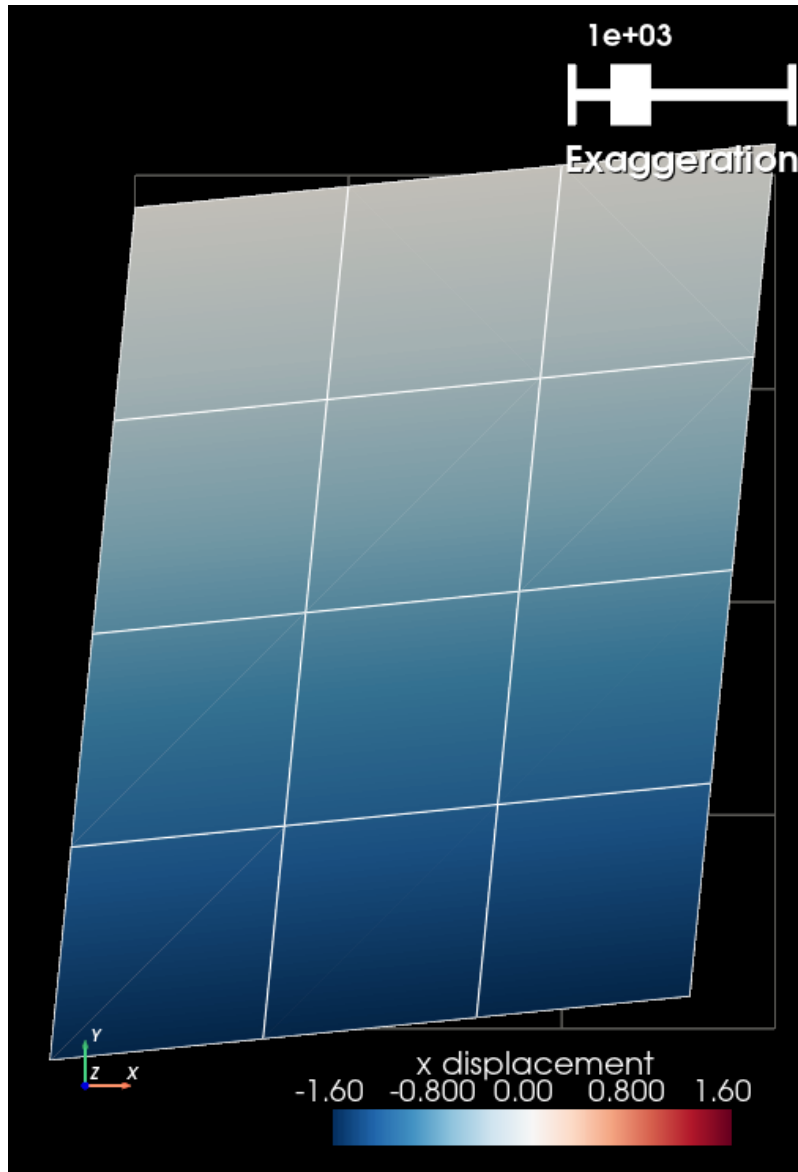


Fig. 4.30: Solution for Step 2. The colors of the shaded surface indicate the magnitude of the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.UniformDB
- pylith.meshio.DataWriterHDF5
- Static simulation
- LU preconditioner
- pylith.bc.DirichletTimeDependent
- pylith.bc.NeumannTimeDependent
- spatialdata.spatialdb.SimpleDB

Simulation parameters

In Step 3 we replace the Dirichlet (displacement) boundary conditions on the +y and -y boundaries with equivalent Neumann (traction) boundary conditions. In order to maintain symmetry and prevent rigid body motion, we constrain both the x and y displacements on the +x and -x boundaries. The solution matches that in Step 2. Fig. 4.31 shows the boundary conditions on the domain. The parameters specific to this example are in `step03_sheardispract.cfg`.

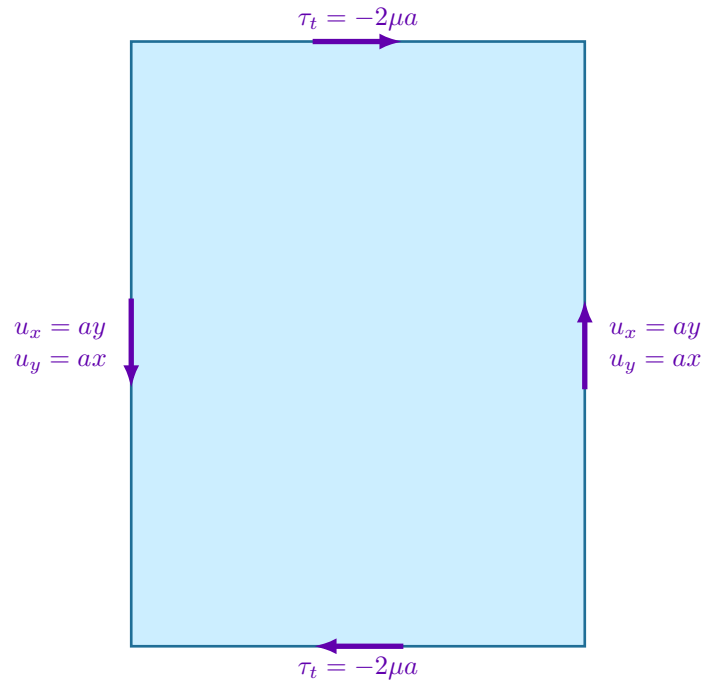


Fig. 4.31: Boundary conditions for shear deformation. We constrain the x and y displacements on the +x and -x boundaries. We apply tangential (shear) tractions on the +y and -y boundaries.

The tractions are uniform on each of the two boundaries, so we use a `UniformDB`. In PyLith the direction of the tangential tractions in 2D is defined by the cross product of the +z direction and the outward normal on the boundary. On the +y boundary a positive tangential traction is in the -x direction, and on the -y boundary a positive tangential traction is in the +x direction. We want tractions in the opposite direction as shown by the arrows in Fig. 4.31, so we apply negative tangential tractions.

Listing 4.41: Specifying the boundary conditions for Step 3. We only show the detailed settings for the -x and -y boundaries.

```
[pylithapp.problem]
bc = [bc_xneg, bc_yneg, bc_xpos, bc_ypos]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.NeumannTimeDependent
bc.bc_ypos = pylith.bc.NeumannTimeDependent

[pylithapp.problem.bc.bc_xneg]
# Degrees of freedom (dof) 0 and 1 correspond to the x and y displacements.
constrained_dof = [0, 1]
label = boundary_xneg
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC -x boundary
db_auxiliary_field.iohandler.filename = sheardisp_bc_xneg.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.bc.bc_yneg]
label = boundary_yneg
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Neumann BC -y boundary
db_auxiliary_field.values = [initial_amplitude_tangential, initial_amplitude_normal]
db_auxiliary_field.data = [-4.5*MPa, 0*MPa]
```

Running the simulation

Listing 4.42: Run Step 3 simulation

```
$ pylith step03_sheardispract.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 1.817939142477e+01
      Linear solve converged due to CONVERGED_ATOL iterations 3
    1 SNES Function norm 8.280473801872e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

As expected, the output written to the terminal is nearly identical to what we saw for Steps 1 and 2.

Visualizing the results

In Fig. 4.32 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.43: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step03_sheardispttract-domain.h5 warp_grid --component=x
```

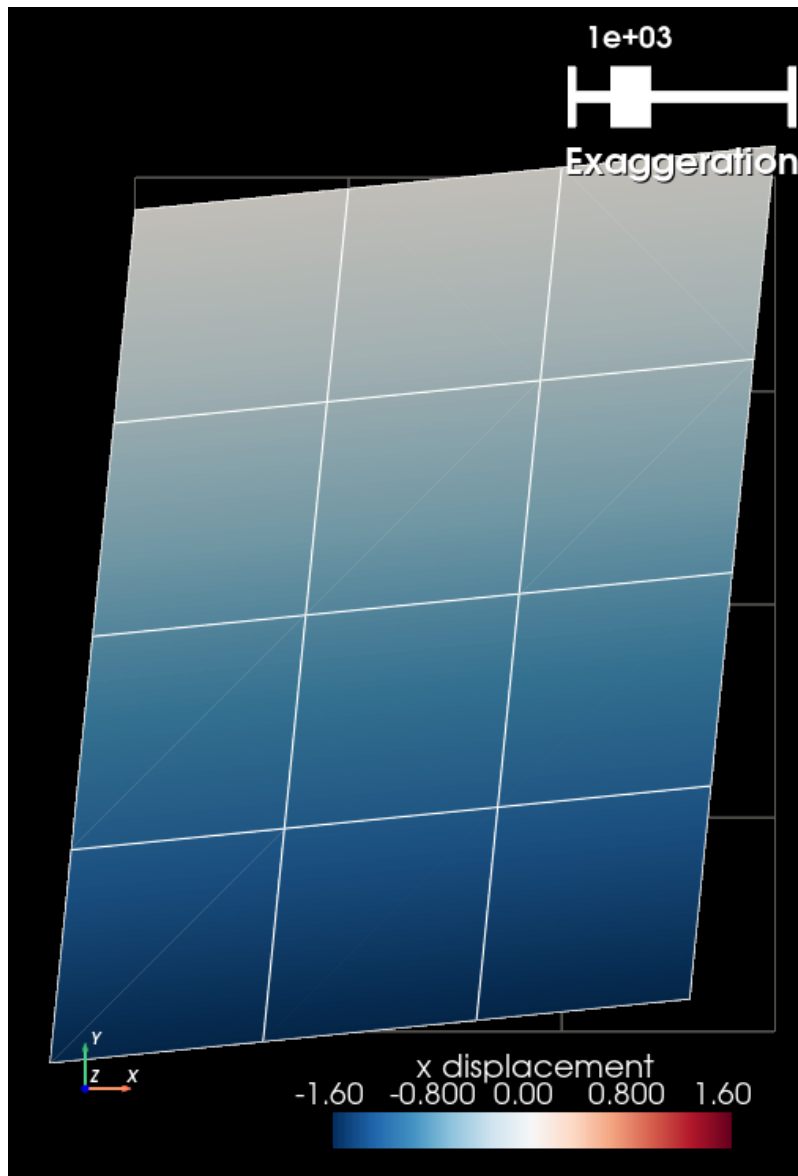


Fig. 4.32: Solution for Step 3. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The solution matches the one from Step 2.

Step 4: Shear Displacement and Initial Conditions

Features

- Quadrilateral cells
- pylith.meshio.MeshIOAscii
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.UniformDB
- pylith.meshio.DataWriterHDF5
- Static simulation
- LU preconditioner
- pylith.problems.InitialConditionDomain
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.SimpleGridDB

Simulation parameters

In this example we demonstrate the use of initial conditions for the boundary value problem in Step 2. We set the displacement field over the domain to the analytical solution as an initial condition.

The parameters specific to this example are in `step04_sheardispic.cfg`. The only difference with respect to Step 2 is the addition of the initial condition. From our boundary conditions we can see that the analytical solution to our boundary value problem is $\vec{u}(x, y) = (ay, ax)$. Because we are specifying the displacement field over the domain, we use the `SimpleGridDB`, which specifies the values on a logically rectangular grid aligned with the coordinate axes. The grid layout of the values allows queries for values at points to be much more efficient than a `SimpleDB` which can have points at arbitrary locations.

Listing 4.44: Initial conditions for Step 4.

```
[pylithapp.problem]
ic = [domain]
ic.domain = pylith.problems.InitialConditionDomain

[pylithapp.problem.ic.domain]
db = spatialdata.spatialdb.SimpleGridDB
db.description = Initial conditions over domain
db.filename = sheardisp_ic.spatialdb
```

Running the simulation

Listing 4.45: Run Step 4 simulation

```
$ pylith step04_sheardispic.cfg

# The output should look something like the following.
```

(continues on next page)

(continued from previous page)

```

>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 1.468687011488e-15
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.
WARNING! There are options you set that were not used!
WARNING! could be spelling mistake, etc!
There is one unused database option. It is:
Option left: name:-mg_fine_ksp_max_it value: 5 source: code

```

By design we set the initial condition so that it satisfies the elasticity equation. As a result, the first nonlinear solver residual evaluation meets the convergence criteria. The linear solver is not used; this is why PETSc reports an unused option at the end of the simulation.

Visualizing the results

In Fig. 4.33 we use the `pylith_viz` utility to visualize the `x` displacement field.

Listing 4.46: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step04_sheardispic-domain.h5 warp_grid --component=x
```

Step 5: Time-Dependent Shear Displacement and Traction

Features

- Quadrilateral cells
- `pylith.meshio.MeshIOAscii`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.DataWriterHDF5`
- Quasi-static simulation
- backward Euler time stepping
- LU preconditioner

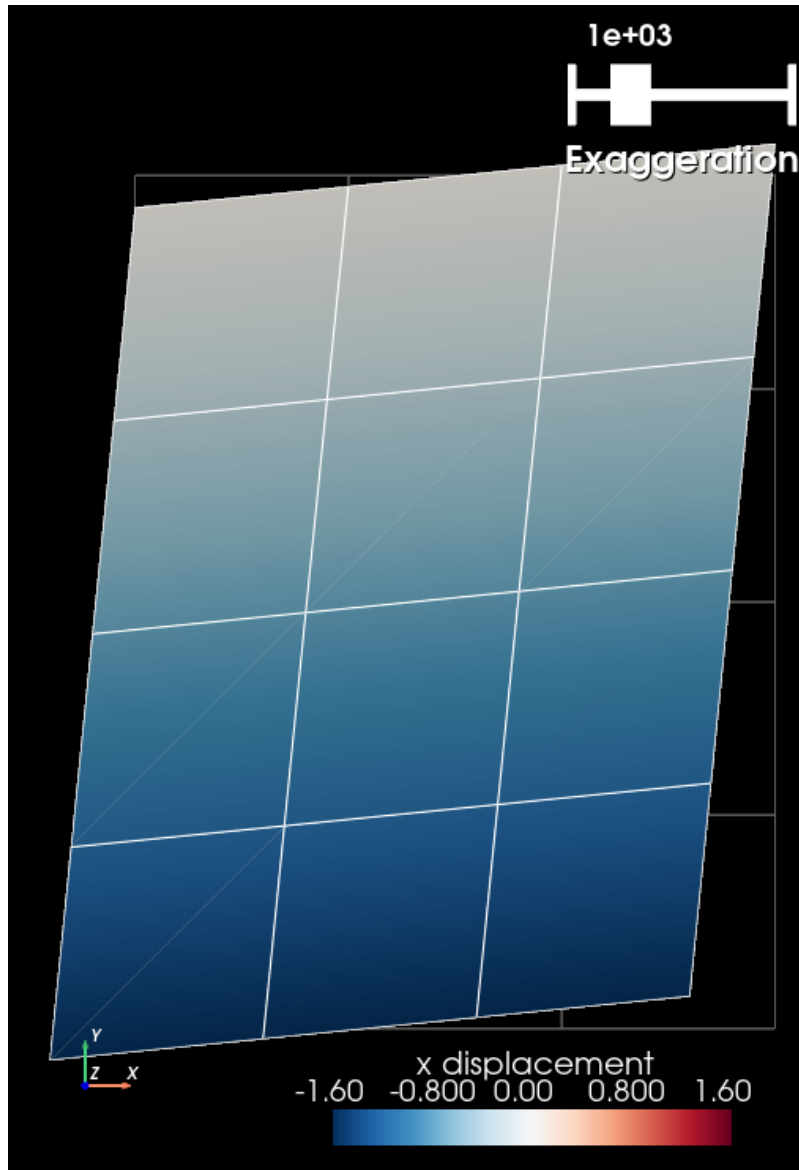


Fig. 4.33: Solution for Step 4. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The solution matches the one in Step 2.

- pylith.bc.DirichletTimeDependent
- pylith.bc.NeumannTimeDependent
- spatialdata.spatialdb.SimpleDB

Simulation parameters

In this example we build on Step 3 and make the Dirichlet (displacement) and Neumann (traction) boundary conditions a bit more complicated by adding variation in time. The simulation has a duration of 5 years with a time step of 1 year. The time-dependent boundary conditions use the same initial amplitude values for the first time step before adding in a constant rate increase at a time of 1 year. Fig. 4.34 shows the boundary conditions on the domain. The parameters specific to this example are in `step05_sheardisptractrate.cfg`.

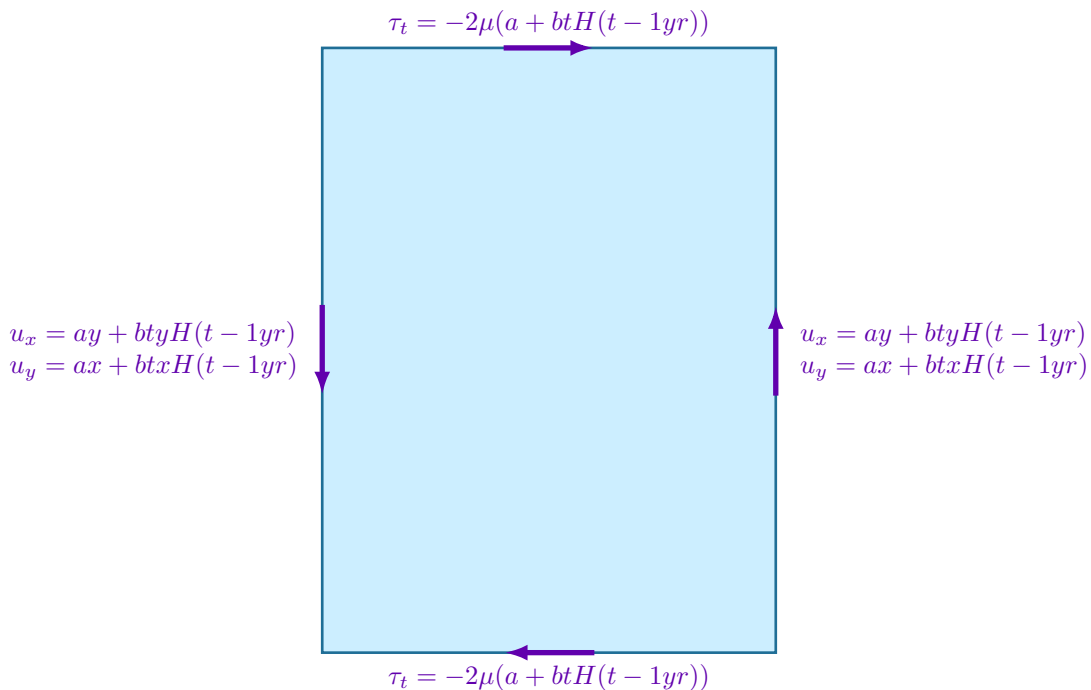


Fig. 4.34: Boundary conditions for shear deformation. We constrain the x and y displacements on the +x and -x boundaries. We apply tangential (shear) tractions on the +y and -y boundaries. At a time of 1 year we increase the amplitude at a constant rate b ($H(t)$ corresponds to the heavyside step function).

This is a time-dependent problem, so we must specify the start and end times of the simulation along with the initial time step. With an initial time step of 1 year, we start the simulation at -1 year so that the first solve will advance the simulation to a time of 0. We also specify a relaxation time on the order of the time scale of the simulation to allow for reasonable nondimensionalization of time.

Listing 4.47: Time stepping parameters for Step 5.

```
[pylithapp.problem]
start_time = -1.0*year
end_time = 5.0*year
initial_dt = 1.0*year

[pylithapp.problem.scales]
relaxation_time = 10.0*year
```

For the time-dependent Dirichlet and Neumann boundary conditions, we specify both the initial displacement and a constant rate; the constant rate begins at $t=1$ year.

Listing 4.48: Time-dependent boundary conditions for Step 5. We show the details for the -x and -y boundaries.

```
[pylithapp.problem.bc.bc_xneg]
# Degrees of freedom (dof) 0 and 1 correspond to the x and y displacements.
constrained_dof = [0, 1]
label = boundary_xneg
use_initial = True
use_rate = True
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC -x boundary
db_auxiliary_field.iohandler.filename = sheardisprate_bc_xneg.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.bc.bc_yneg]
label = boundary_yneg
use_initial = True
use_rate = True
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Neumann BC -y boundary
db_auxiliary_field.values = [initial_amplitude_tangential, initial_amplitude_normal,
↪rate_start_time, rate_amplitude_tangential, rate_amplitude_normal]
db_auxiliary_field.data = [-4.5*MPa, 0.0*MPa, 1.0*year, -1.125*MPa/year, 0.0]
```

Running the simulation

Listing 4.49: Run Step 5 simulation

```
$ pylith step05_sheardispratractrate.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.1 time -0.1
    0 SNES Function norm 1.817939142477e+01
      Linear solve converged due to CONVERGED_ATOL iterations 3
    1 SNES Function norm 8.280473801872e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --

5 TS dt 0.1 time 0.4
```

(continues on next page)

(continued from previous page)

```

0 SNES Function norm 4.544847874296e+00
  Linear solve converged due to CONVERGED_ATOL iterations 0
1 SNES Function norm 1.606117666093e-07
  Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
6 TS dt 0.1 time 0.5
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The output written to the terminal now contains multiple time steps. The PETSc TS (time stepping) monitor shows the time step number followed by the time step and time in nondimensional units.

Visualizing the results

In Fig. 4.35 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.50: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step05_sheardispractrate-domain.h5 warp_grid --component=x
```

You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Suggested Exercises

1. Adjust the material properties in Step 1 and verify the effect on the deformation and stress field.
2. Use the `tri.mesh` file in any of the steps. Can you change the mesh file using one additional command line argument?
3. Change the boundary conditions in Steps 2-5 to generate a different shear strain.
4. Set the basis order and quadrature order to 2. Do you get the same solution? Why?

Axial and Shear Deformation (3D Box)

This example is the 3D version of the 2D box axial shear and deformation example (`examples/box-2d`). The files are in the directory `examples/box-3d` and include:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

generate_gmsh.py

Python script to generate the finite-element mesh using Gmsh.

generate_cubit.py

Python script to generate the finite-element mesh using Cubit.

***.msh**

Gmsh finite-element mesh files generated by Gmsh.

***.exo**

Exodus II finite-element mesh files generated by Cubit.

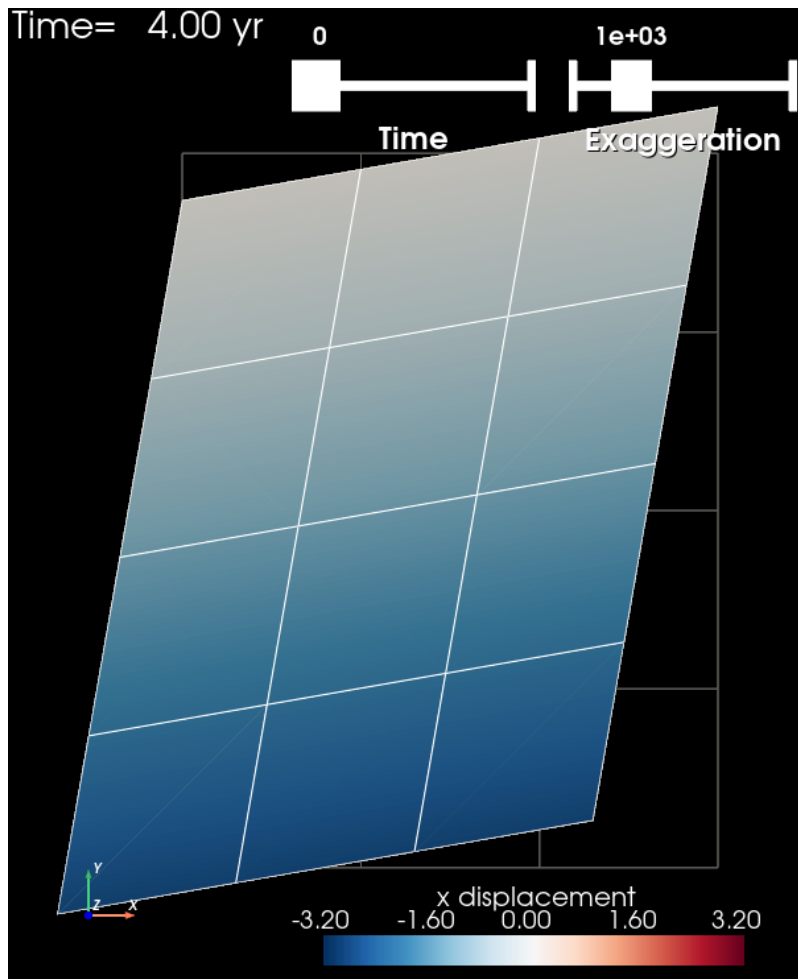


Fig. 4.35: Solution for Step 5 at a time of 4.0 years. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

***.spatialdb**

Spatial database filesFiles associated with the spatial databases.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This suite of examples demonstrates some basic concepts of using PyLith to solve the static and quasistatic boundary elasticity equation in a 3D box (Fig. 4.36) with uniform material properties. The examples incrementally add complexity through a series of steps:

Step 1

Axial extension with Dirichlet (displacement) boundary conditions.

Step 2

Shear deformation with Dirichlet (displacement) boundary conditions.

Step 3

Shear deformation with Dirichlet (displacement) and Neumann (traction) boundary conditions.

Step 4

Same as Step 2 but with initial conditions equal to the analytical solution.

Step 5

Shear deformation with time-dependent Dirichlet (displacement) and Neumann (traction) boundary conditions.

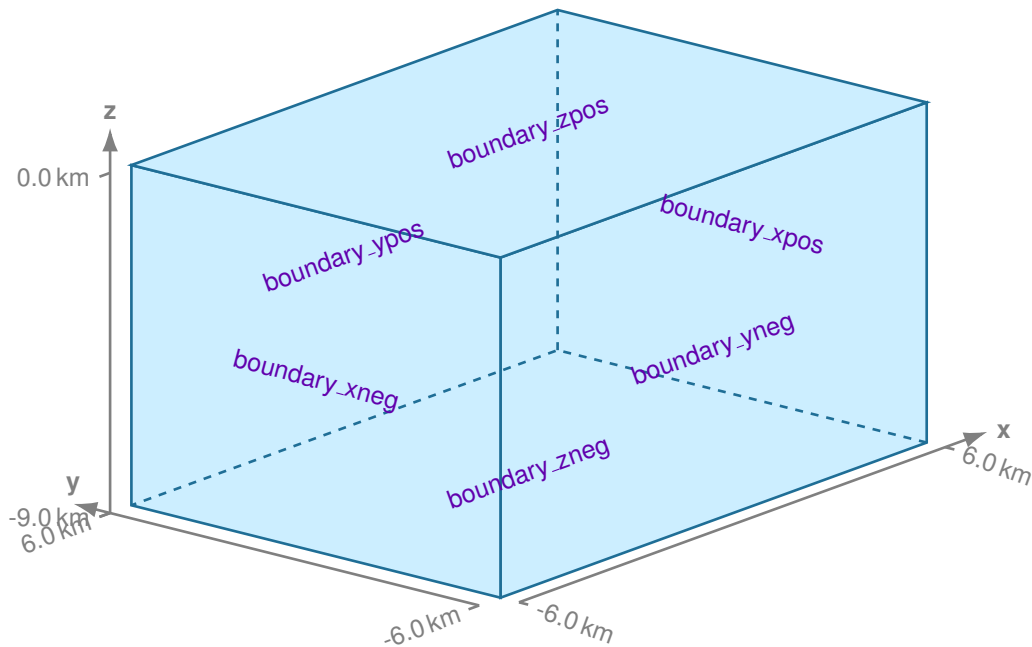


Fig. 4.36: Diagram of geometry for 3D box that extends from -6 km to +6 km in the x and y directions and from -9 km to 0 km in the z direction. We refer to the domain boundaries using the names shown in the diagram.

Example Workflow

Gmsh Mesh

Geometry

We create the geometry directly from the box primitive provided by the OpenCASCADE geometry engine in Gmsh.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. The script makes use of the `gmsh_utils.GenerateMesh` class (discussed in [gmsh_utils](#)), which provides the command line arguments and boiler-plate methods. In our `generate_gmsh.py` Python script, we create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class. We use the Gmsh `MeshSize` options to define a uniform discretization size of 3 km.

Listing 4.51: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with hexahedral cells and save it to `mesh_hex.msh` (default filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui

# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui

# Show available command line arguments.
$ ./generate_gmsh.py --help
```

By default the Python script will generate a finite-element mesh with hexahedral cells and save it to the file `mesh_hex.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.52: View the Gmsh mesh file `mesh_hex.msh` using Gmsh.

```
gmsh -open mesh_hex.msh
```

Cubit Mesh

Overview

We create the geometry, set the mesh parameters, generate the mesh, mark the materials and boundaries, and then write the mesh to a file.

Meshing using Python Script

New in v4.1.0

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

Open the Python script `generate_cubit.py` in the Cubit journal editor. Play the selected script or play the lines,

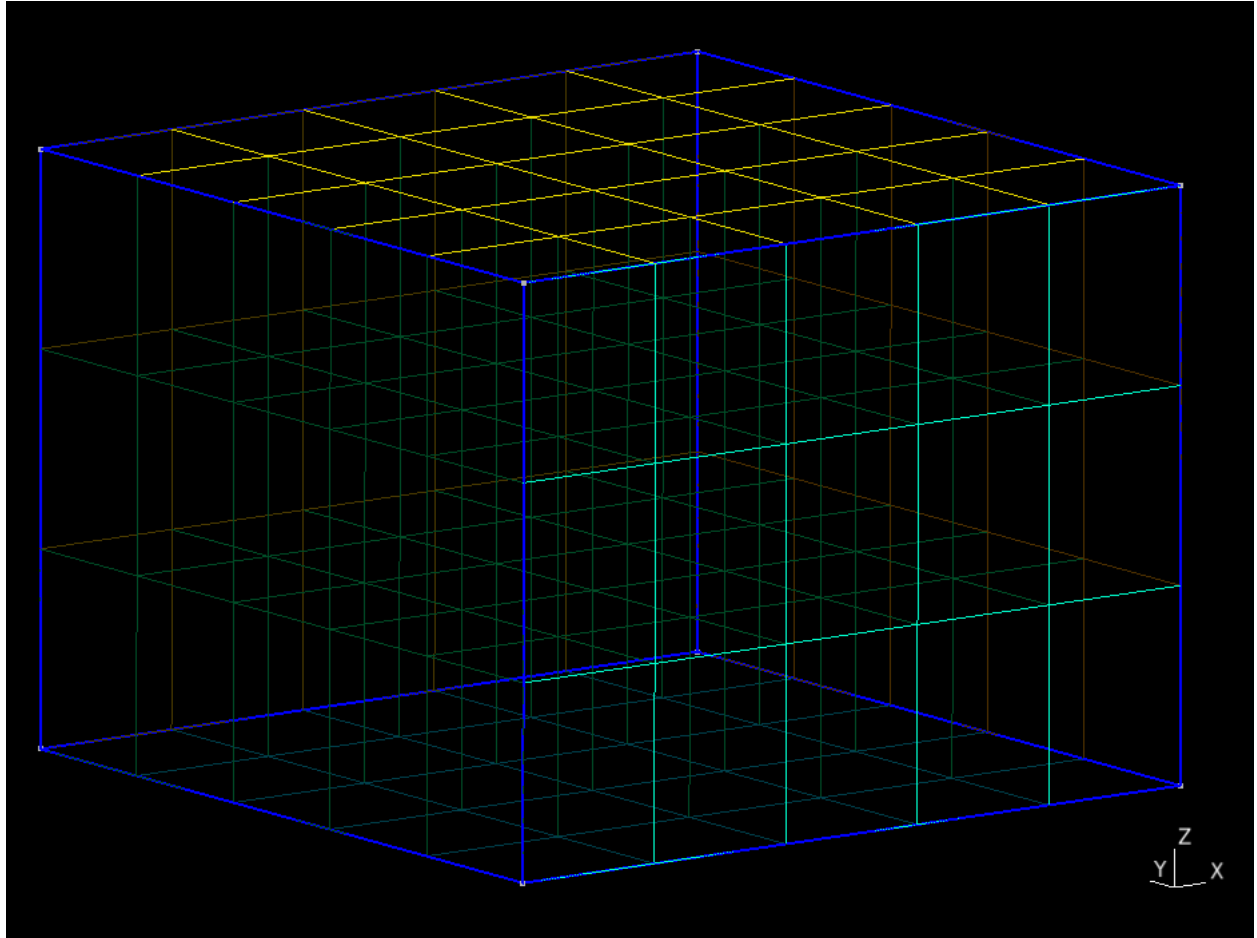


Fig. 4.37: Finite-element mesh with hexahedral cells generated by Gmsh.

making sure you play the first line so that Cubit uses the Python interpreter when running the script. We specify the parameters controlling the geometry, mesh size, and cell shape near the top of the script.

Important

The numbering of geometric entities, such as points, curves, and surfaces can depend upon the version of the ACIS geometry library used in Cubit. Strategies to make scripts independent of the version of the ACIS geometry library being used include:

1. Use a variable to store the geometry id.
2. Use “idless” scripts that refer to entities by location rather than id. Refer to the Cubit documentation for more information.
3. Name entities when they are created to minimize use of the entity id in scripts. We often make use of all three of these methods.

Once you have run the Python script to construct the geometry and generate the mesh, you will have a corresponding Exodus-II file (`mesh_tet.exo` or `mesh_hex.exo`). These are NetCDF files, and they can be loaded into ParaView.

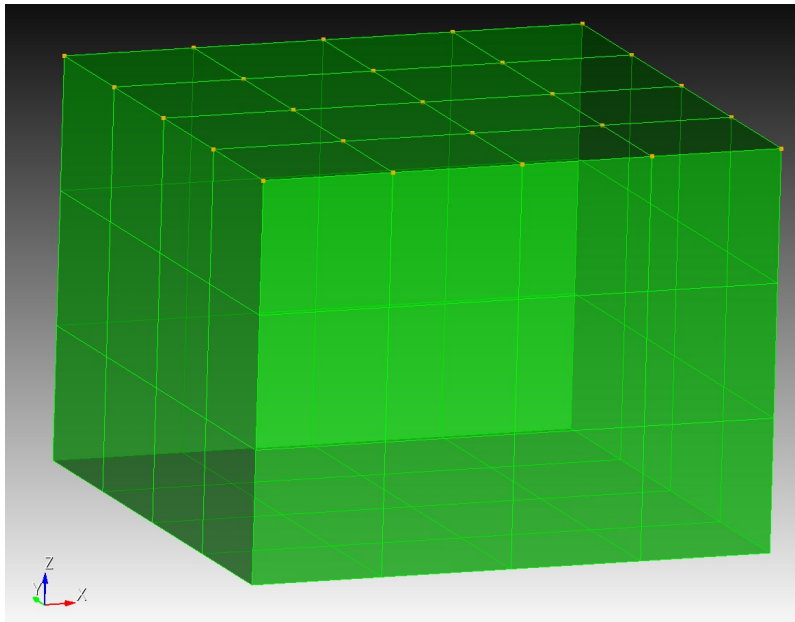


Fig. 4.38: Finite-element mesh with hexahedral cells generated by Cubit. We show the nodeset on the boundary_zpos boundary.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`. This limits duplicate information in the parameter files for each simulation.

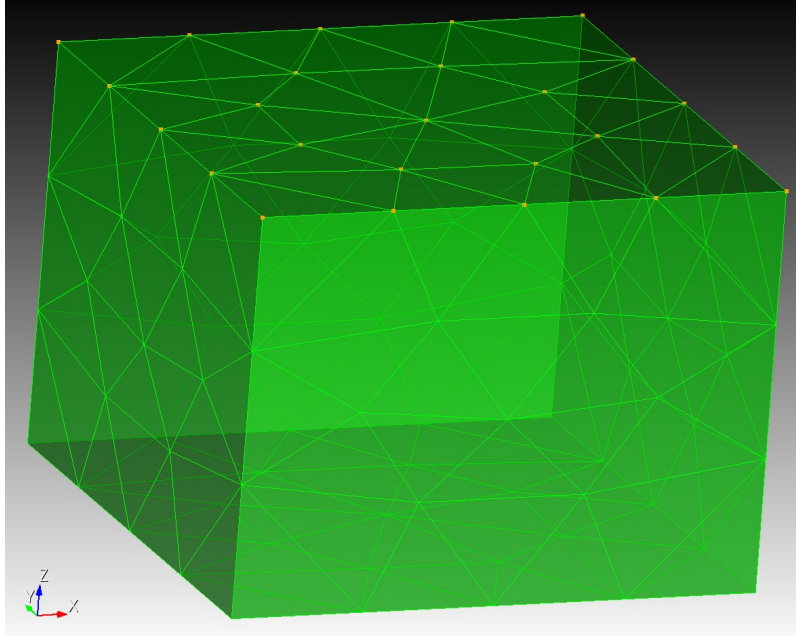


Fig. 4.39: Finite-element mesh with tetrahedral cells generated by Cubit. We show the nodeset on the boundary_zpos boundary.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We use `MeshIOPetsc` to import finite-element mesh from Gmsh.

Listing 4.53: Parameters for importing the finite-element mesh from Gmsh.

```
[pylithapp.mesh_generator]
reader = pylith.meshio.MeshIOPetsc

# Set the filename and the dimension of the default Cartesian coordinate system.
reader.filename = mesh_hex.msh
reader.coordsys.space_dim = 3
```

In addition to output of the solution over the domain (default), we also output the solution on the ground surface (+z boundary).

Listing 4.54: Parameters for output over the domain and ground surface (+z boundary).

```
[pylithapp.problem]
solution_observers = [domain, ground_surface]
solution_observers.ground_surface = pylith.meshio.OutputSolnBoundary

[pylithapp.problem.solution_observers.ground_surface]
# The `label` and `label_value` match the name and tag in the Gmsh Python script.
label = boundary_zpos
label_value = 15
```

Physics

All of the simulations in this example suite solve the elasticity equation without inertia,

$$\vec{s} = \left(\vec{u} \right)^T \quad (4.467)$$

$$\nabla \cdot \sigma(\vec{u}) = \vec{0} \quad (4.468)$$

The default PyLith settings target solving the quasi-static elasticity equation, so we can use the default `TimeDependent` problem and solution field with a single displacement subfield of basis order 1.

In this suite of examples we have a single material, and we must specify a description for the material and the label value. In this case the label value corresponds to the tag of the physical group in the Gmsh file. The material properties are uniform, so we use a single `UniformDB` to specify the material properties in `pylithapp.cfg`. When we have a spatial variation in the material properties, we specify them in a spatial database file. With uniform properties we use a basis order of 0 for the auxiliary subfields.

Listing 4.55: Material settings in pylithapp.cfg.

```
[pylithapp.problem]
materials = [elastic]

[pylithapp.problem.materials.elastic]
label_value = 0

# We will use uniform material properties, so we use the UniformDB
# spatial database.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Elastic properties
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500*kg/m**3, 3.0*km/s, 5.2915026*km/s]

# Set the discretization of the material auxiliary fields (properties).
# We have uniform material properties, so we can use a basis order of 0.
auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0
```

Step 1: Axial Extension

Features

- Tetrahedral cells
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- LU preconditioner
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.ZeroDB

Simulation parameters

This example corresponds to axial extension in the x direction and axial compression in the y direction. We apply Dirichlet (displacement) boundary conditions for the x displacement on the +x (boundary_xpos) and -x (boundary_xneg) boundaries and for the y displacement on the +y (boundary_ypos) and -y (boundary_yneg) boundaries. We apply roller Dirichlet boundary conditions on the -z (boundary_zneg) boundary. Fig. 4.40 shows the boundary conditions on the domain. The parameters specific to this example are in step01_axialdisp.cfg.

We create an array of 5 DirichletTimeDependent boundary conditions. For each of these boundary conditions we must specify which degrees of freedom are constrained, the name of the label marking the boundary (name of the group

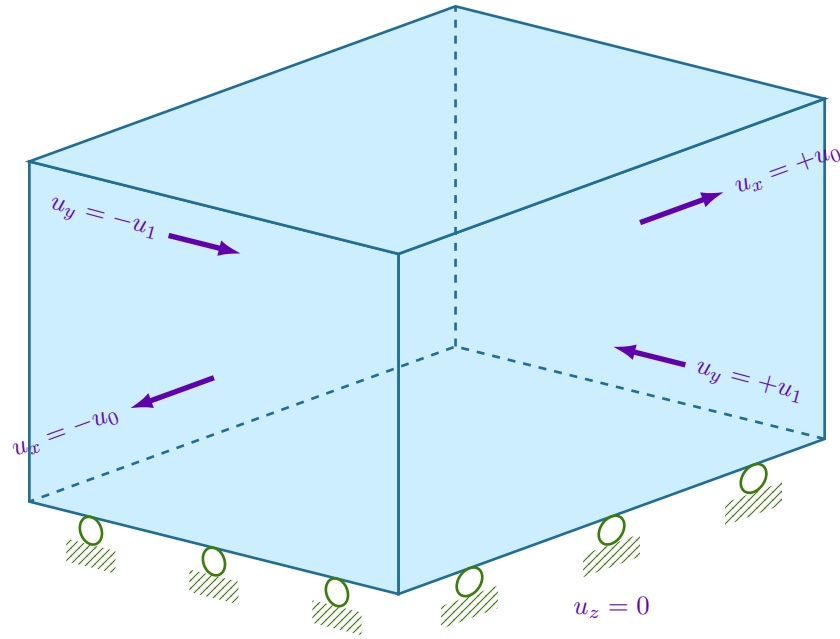


Fig. 4.40: Boundary conditions for axial extension in the x direction and axial compression in the y direction. We constrain the x displacement on the $+x$ and $-x$ boundaries, the y displacement on the $+y$ and $-y$ boundaries, and set the z displacement to zero on the $-z$ boundary.

of vertices in the finite-element mesh file), and the values for the Dirichlet boundary condition.

Listing 4.56: Specifying the boundary conditions for Step 1. We only show the detailed settings for the $+x$ boundary.

```
[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos, bc_zneg]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.DirichletTimeDependent
bc.bc_ypos = pylith.bc.DirichletTimeDependent
bc.bc_zneg = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xpos]
label = boundary_xpos
label_value = 11
constrained_dof = [0]

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Dirichlet BC on +x boundary
db_auxiliary_field.values = [initial_amplitude_x, initial_amplitude_y, initial_amplitude_
↪ z]
db_auxiliary_field.data = [+2.0*m, 0*m, 0*m]
```

Running the simulation

Listing 4.57: Run Step 1 simulation

```

$ pylith step01_axialdisp.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void _
↳ pylith::meshio::MeshIOPetsc::_read()
-- info (application-flow)
-- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_hex.msh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void _
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-6000, 6000)
    (-6000, 6000)
    (-9000, 0)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void _
↳ pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
↳ printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 1000000*m

```

(continues on next page)

(continued from previous page)

```

Displacement scale: 1*m
Time scale: 3.15576e+09*s
Rigidity scale: 1e+10*m**-1*kg*s**-2
Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↳ pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
  0 SNES Function norm 2.045040813383e+00
    Linear solve converged due to CONVERGED_ATOL iterations 4
  1 SNES Function norm 3.894554358059e-09
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see the PETSc options PyLith selected based on the governing equations and formulation as discussed in [PETSc Options](#). Next, we see that PyLith is reading the mesh using the MeshIOPetsc reader and that it reports the bounding box of the domain. We also see the scales used to nondimensionalize the problem.

Near the end of the output written to the terminal, we see the solver advanced the solution one time step (static simulation). The linear solve converged in 4 iterations. The norm of the residual met the absolute tolerance convergence criterion (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snestol`). For this small problem, the multi-grid preconditioner has fewer levels, so we get a warning about an unused PETSc option.

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (.h5) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (.xmf) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In [Fig. 4.41](#) we use the `pylith_viz` utility to visualize the `x` displacement field.

Listing 4.58: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step01_axialdisp-domain.h5 warp_grid --component=x
```

Step 2: Shear Displacement

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOPetsc`

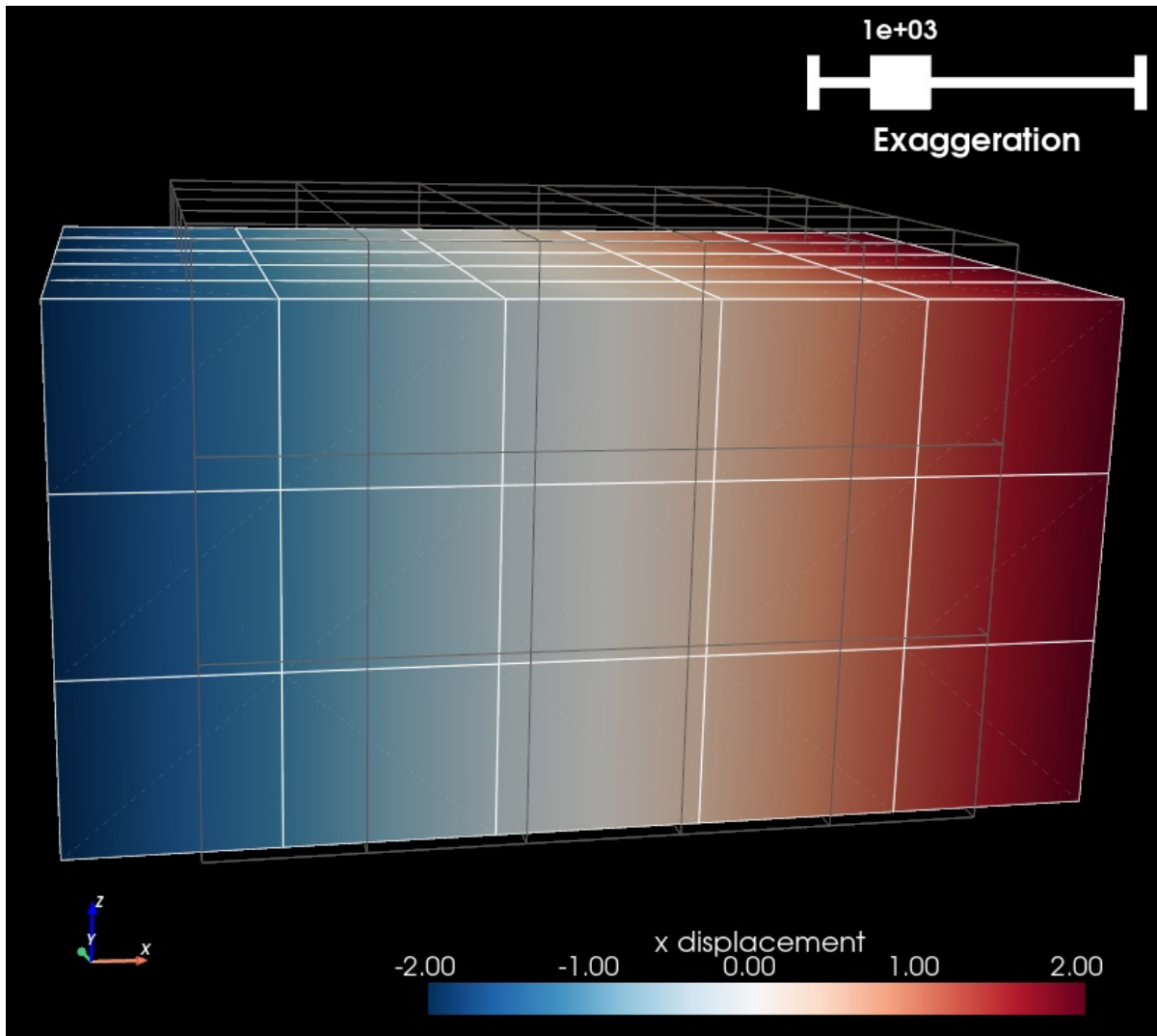


Fig. 4.41: Solution for Step 1. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- LU preconditioner
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`

Simulation parameters

This example corresponds to shear deformation due to Dirichlet (displacement) boundary conditions. We apply Dirichlet (displacement) boundary conditions for the y displacement on the $+x$ (`boundary_xpos`) and $-x$ (`boundary_xneg`) boundaries and for the x displacement on the $+y$ (`boundary_ypos`) and $-y$ (`boundary_yneg`) boundaries. Fig. 4.40 shows the boundary conditions on the domain. The parameters specific to this example are in `step02_sheardisp.cfg`.

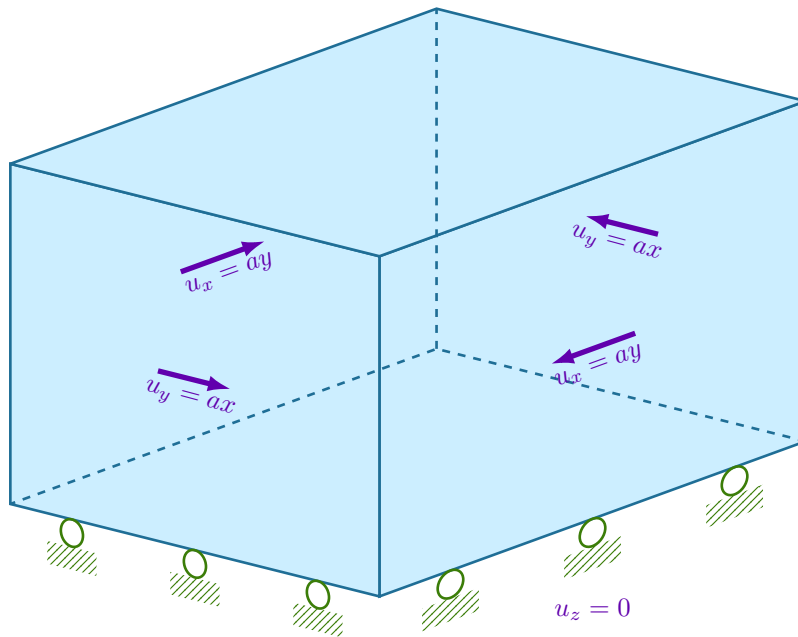


Fig. 4.42: Boundary conditions for shear deformation. We constrain the y displacement on the $+x$ and $-x$ boundaries and the x displacement on the $+y$ and $-y$ boundaries.

We create an array of 5 Dirichlet boundary conditions. On the $-x$, $+x$, $-y$, and $+y$ boundaries we impose shear displacement.

Listing 4.59: Boundary conditions for Step 2. We only show the details for the -x boundary.

```
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos, bc_zneg]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.DirichletTimeDependent
bc.bc_ypos = pylith.bc.DirichletTimeDependent
bc.bc_zneg = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xneg]
label = boundary_xneg
label_value = 10
constrained_dof = [1]

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC -x boundary
db_auxiliary_field.iohandler.filename = sheardisp_bc_xneg.spatialdb
db_auxiliary_field.query_type = linear
```

Running the simulation

Listing 4.60: Run Step 2 simulation

```
$ pylith step02_sheardisp.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pylith/problems/
↳ TimeDependent.py:139:run
-- timedependent(info)
-- Solving problem.
0 TS dt 0.001 time 0.
    0 SNES Function norm 5.433645185324e-01
      Linear solve converged due to CONVERGED_ATOL iterations 4
    1 SNES Function norm 4.280274905934e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/unix/py39-venv/pylith-debug/lib/python3.9/site-packages/pylith/problems/
↳ Problem.py:201:finalize
-- timedependent(info)
-- Finalizing problem.
WARNING! There are options you set that were not used!
WARNING! could be spelling mistake, etc!
There is one unused database option. It is:
Option left: name=-mg_levels_pc_type value: pbjacobi source: code
```

The output written to the terminal is nearly identical to what we saw for Step 1.

Visualizing the results

In Fig. 4.43 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.61: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step02_sheardisp-domain.h5 warp_grid --component=x
```

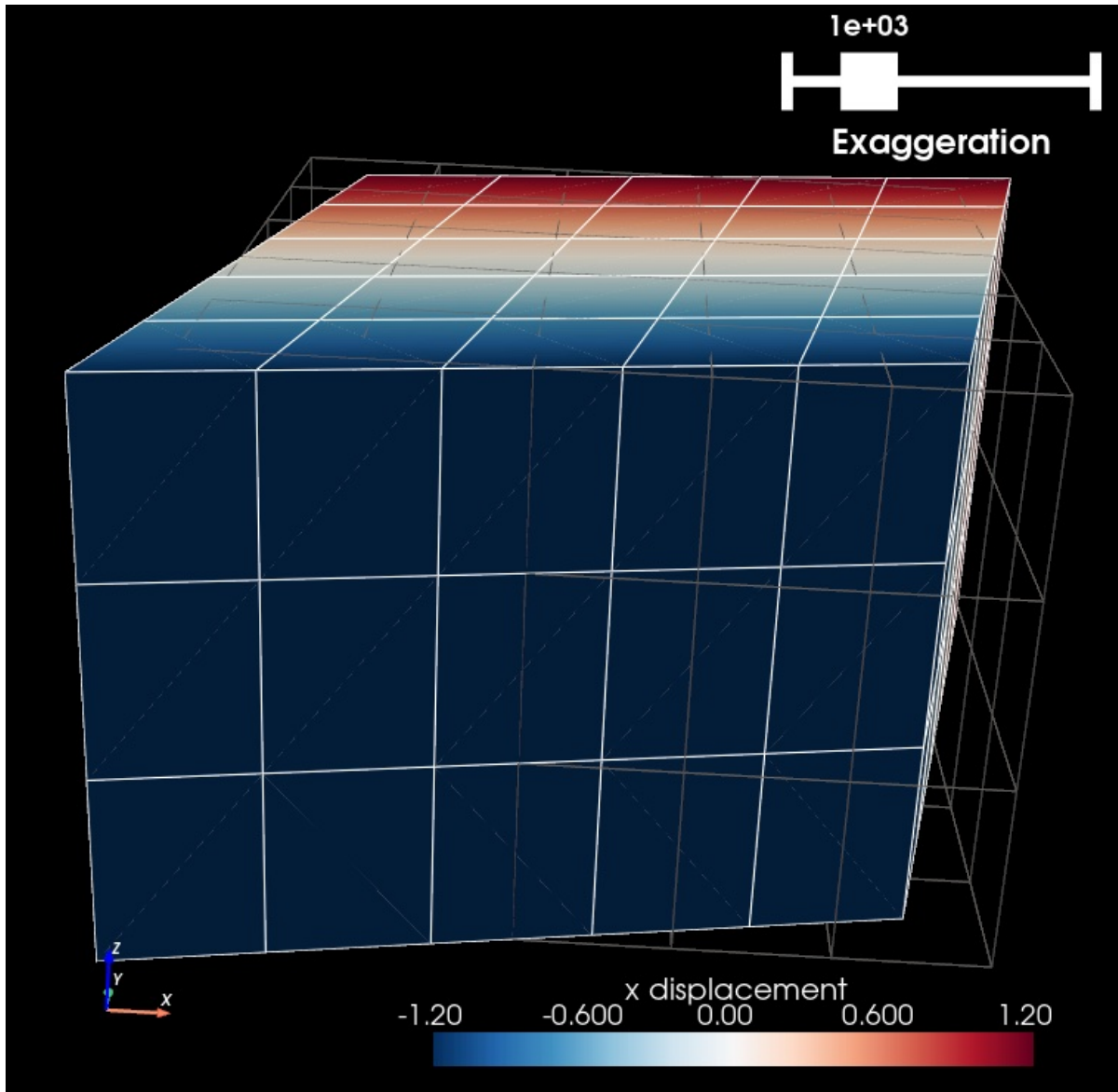


Fig. 4.43: Solution for Step 2. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

Step 3: Shear Displacement and Traction

Features

- Tetrahedral cells
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- LU preconditioner
- pylith.bc.DirichletTimeDependent
- pylith.bc.NeumannTimeDependent
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.ZeroDB

Simulation parameters

In Step 3 we replace the Dirichlet (displacement) boundary conditions on the +y and -y boundaries with equivalent Neumann (traction) boundary conditions. In order to maintain symmetry and prevent rigid body motion, we constrain both the x and y displacements on the +x and -x boundaries. The solution matches that in Step 2. Fig. 4.44 shows the boundary conditions on the domain. The parameters specific to this example are in `step03_sheardispract.cfg`.

The tractions are uniform on each of the two boundaries, so we use a `UniformDB`. In PyLith the direction of the horizontal tangential tractions in 3D are defined by the cross product of the +z direction and the outward normal on the boundary. On the +y boundary a positive tangential traction is in the -x direction, and on the -y boundary a positive tangential traction is in the +x direction. We want tractions in the opposite direction as shown by the arrows in Fig. 4.31, so we apply negative tangential tractions.

Listing 4.62: Specifying the boundary conditions for Step 3. We only show the detailed settings for the -x and -y boundaries.

```
[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos, bc_zneg]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.NeumannTimeDependent
bc.bc_ypos = pylith.bc.NeumannTimeDependent
bc.bc_zneg = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xneg]
label = boundary_xneg
label_value = 10
constrained_dof = [0, 1]
```

(continues on next page)

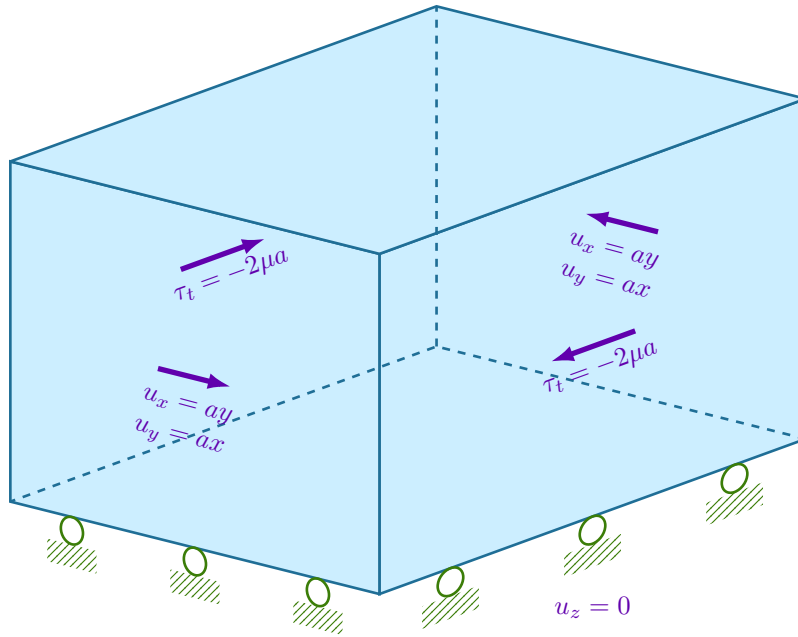


Fig. 4.44: Boundary conditions for shear deformation. We constrain the x and y displacements on the +x and -x boundaries. We apply tangential (shear) tractions on the +y and -y boundaries.

(continued from previous page)

```
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC -x edge
db_auxiliary_field.iohandler.filename = sheardisp_bc_xneg.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.bc.bc_yneg]
label = boundary_yneg
label_value = 12

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Neumann BC -y edge
db_auxiliary_field.values = [initial_amplitude_tangential_1, initial_amplitude_
    ↪ tangential_2, initial_amplitude_normal]
db_auxiliary_field.data = [-9.0*MPa, 0*MPa, 0*MPa]
```

Running the simulation

Listing 4.63: Run Step 3 simulation

```
$ pylith step03_sheardisptract.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
```

(continues on next page)

(continued from previous page)

```

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 8.562738880728e-01
      Linear solve converged due to CONVERGED_ATOL iterations 4
    1 SNES Function norm 3.584100837616e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The output written to the terminal is nearly identical to what we saw for Step 2. The linear solve did require only 4 iteration to converge.

Visualizing the results

In Fig. 4.45 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.64: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step03_sheardisptract-domain.h5 warp_grid --component=x
```

Step 4: Shear Displacement and Initial Conditions

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- LU preconditioner
- `pylith.problems.InitialConditionDomain`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.SimpleGridDB`
- `spatialdata.spatialdb.ZeroDB`

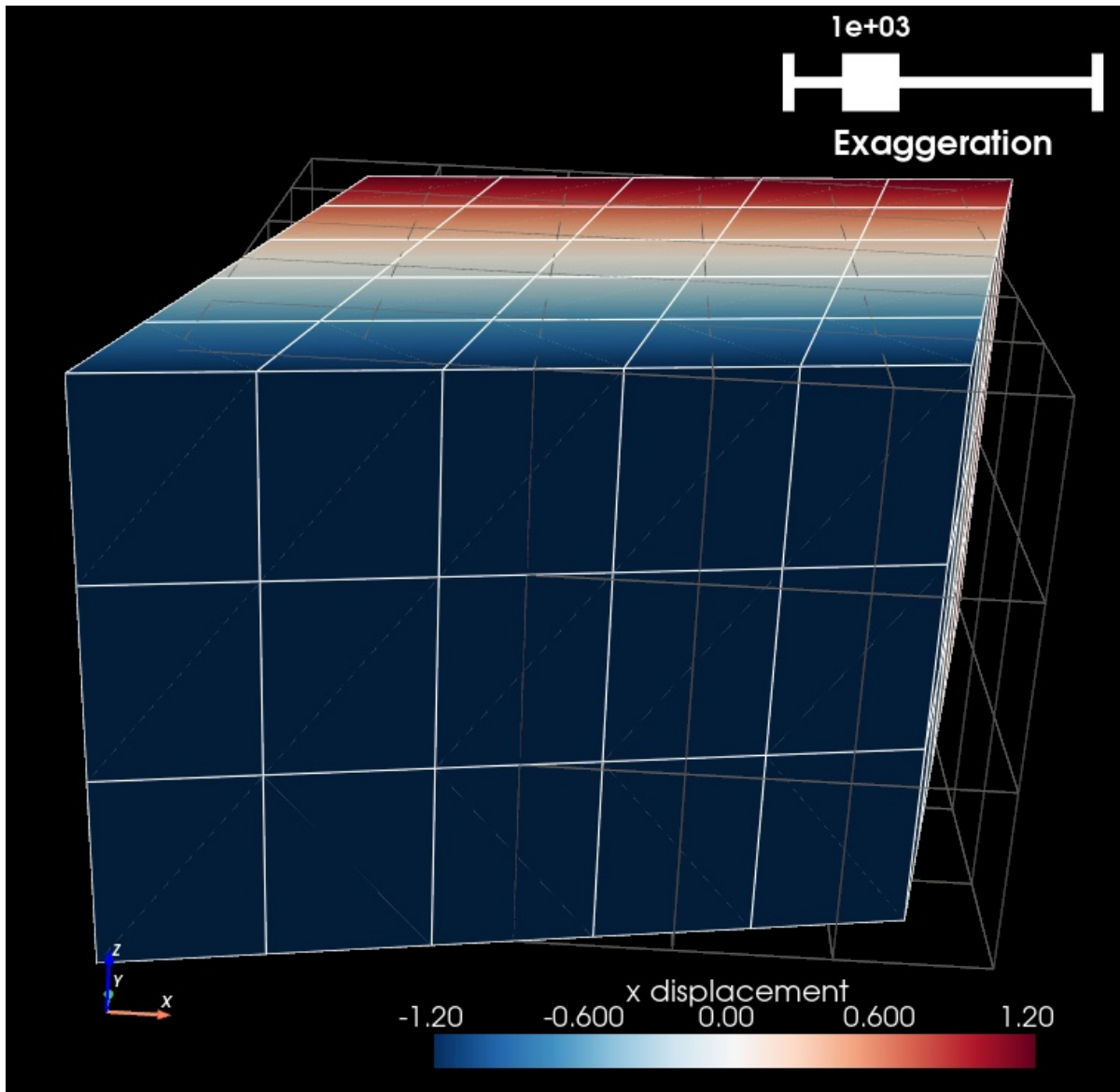


Fig. 4.45: Solution for Step 3. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The solution matches the one from Step 2.

Simulation parameters

In this example we demonstrate the use of initial conditions for the boundary value problem in Step 2. We set the displacement field over the domain to the analytical solution as an initial condition.

The parameters specific to this example are in `step04_sheardispic.cfg`. The only difference with respect to Step 2 is the addition of the initial condition. From our boundary conditions we can see that the analytical solution to our boundary value problem is $\vec{u}(x, y, z) = (ay, ax, 0)$. Because we are specifying the displacement field over the domain, we use the `SimpleGridDB`, which specifies the values on a logically rectangular grid aligned with the coordinate axes. The grid layout of the values allows queries for values at points to be much more efficient than a `SimpleDB` which can have points at arbitrary locations.

Listing 4.65: Initial conditions for Step 4.

```
[pylithapp.problem]
ic = [domain]
ic.domain = pylith.problems.InitialConditionDomain

[pylithapp.problem.ic.domain]
db = spatialdata.spatialdb.SimpleGridDB
db.description = Initial conditions over domain
db.filename = sheardisp_ic.spatialdb
db.query_type = linear
```

Running the simulation

Listing 4.66: Run Step 4 simulation

```
$ pylith step04_sheardispic.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 2.323383610201e-16
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.
WARNING! There are options you set that were not used!
WARNING! could be spelling mistake, etc!
There is one unused database option. It is:
Option left: name=-mg_fine_ksp_max_it value: 5 source: code
```

By design we set the initial condition so that it satisfies the elasticity equation. As a result, the first nonlinear solver

residual evaluation meets the convergence criteria. The linear solver is not used; this is why PETSc reports an unused option at the end of the simulation.

Visualizing the results

In Fig. 4.46 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.67: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step04_sheardispic-domain.h5 warp_grid --component=x
```

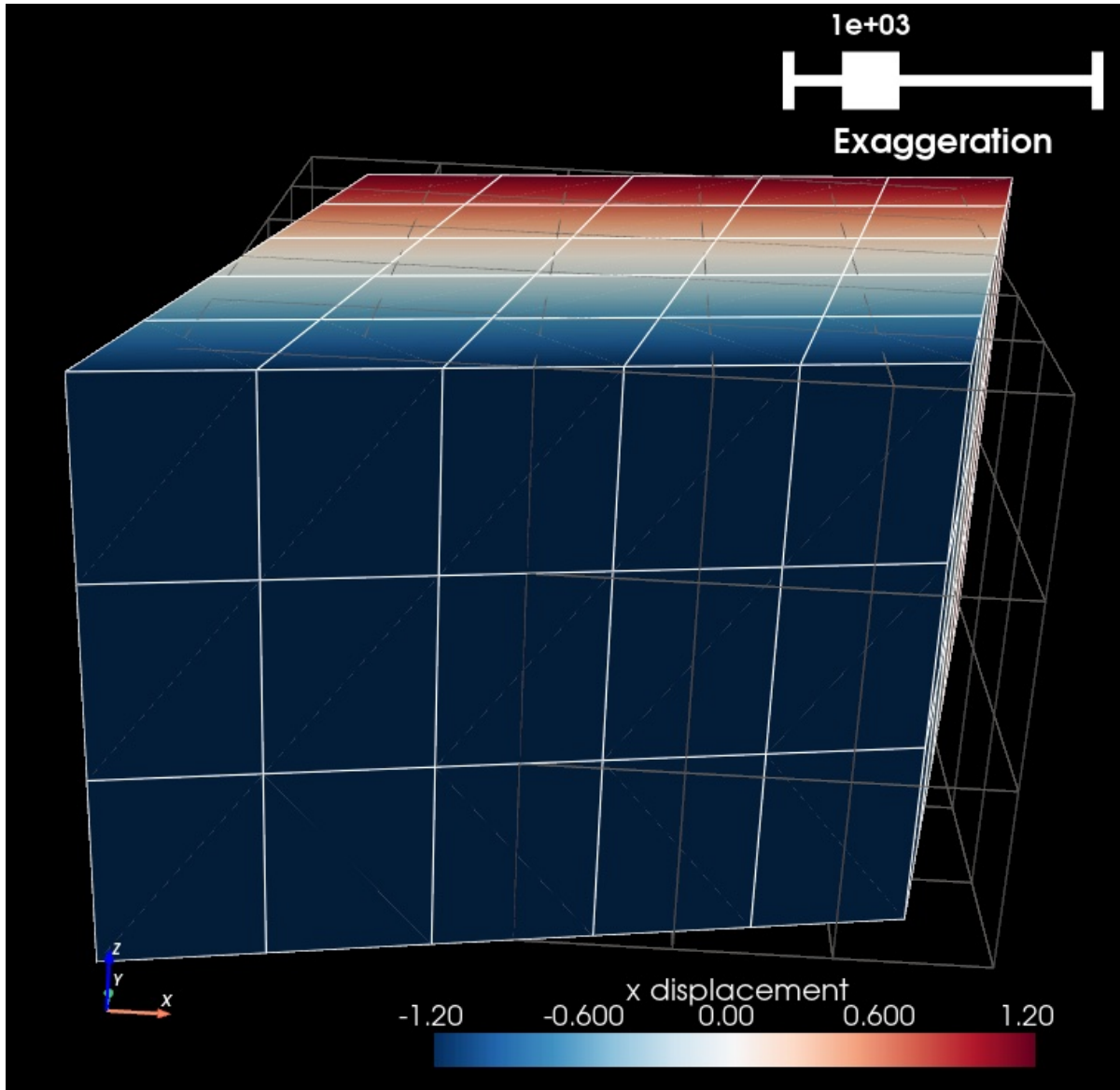


Fig. 4.46: Solution for Step 4. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The solution matches the one in Step 2.

Step 5: Time-Dependent Shear Displacement and Traction

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Quasi-static simulation
- backward Euler time stepping
- LU preconditioner
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.NeumannTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`

Simulation parameters

In this example we build on Step 3 and make the Dirichlet (displacement) and Neumann (traction) boundary conditions a bit more complicated by adding variation in time. The simulation has a duration of 5 years with a time step of 1 year. The time-dependent boundary conditions use the same initial amplitude values for the first time step before adding in a constant rate increase at a time of 1 year. [Fig. 4.47](#) shows the boundary conditions on the domain. The parameters specific to this example are in `step05_sheardisptractrate.cfg`.

This is a time-dependent problem, so we must specify the start and end times of the simulation along with the initial time step. With an initial time step of 1 year, we start the simulation at -1 year so that the first solve will advance the simulation to a time of 0. We also specify a relaxation time on the order of the time scale of the simulation to allow for reasonable nondimensionalization of time.

Listing 4.68: Time stepping parameters for Step 5.

```
[pylithapp.problem]
initial_dt = 1.0*year
start_time = -1.0*year
end_time = 5.0*year

[pylithapp.problem.scales]
relaxation_time = 10.0*year
```

For the time-dependent Dirichlet and Neumann boundary conditions, we specify both the initial displacement and a constant rate; the constant rate begins at $t=1$ year.

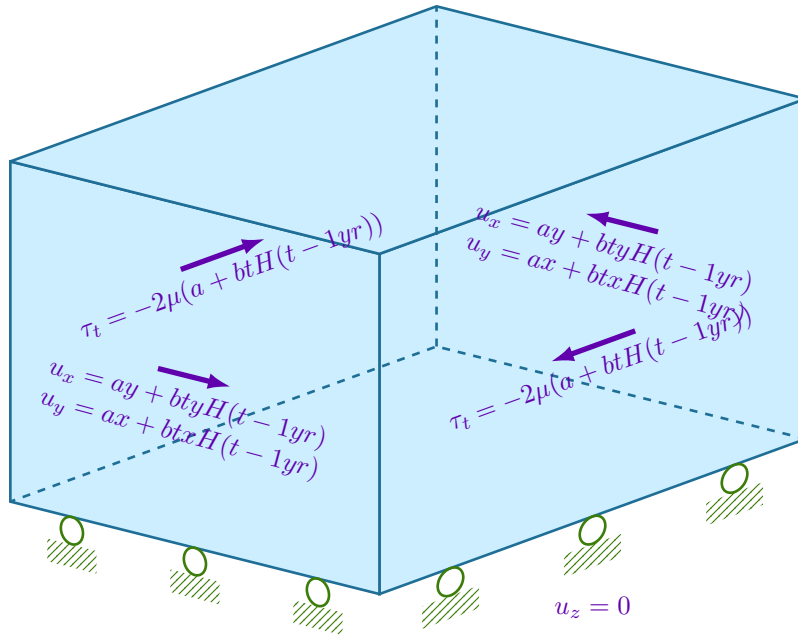


Fig. 4.47: Boundary conditions for shear deformation. We constrain the x and y displacements on the +x and -x boundaries. We apply tangential (shear) tractions on the +y and -y boundaries. At a time of 1 year we increase the amplitude at a constant rate b ($H(t)$ corresponds to the heavyside step function).

Listing 4.69: Time-dependent boundary conditions for Step 5. We show the details for the -x and -y boundaries.

```
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos, bc_zneg]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.NeumannTimeDependent
bc.bc_ypos = pylith.bc.NeumannTimeDependent
bc.bc_zneg = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xneg]
label = boundary_xneg
label_value = 10
constrained_dof = [0, 1]
use_initial = True
use_rate = True

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC -x edge
db_auxiliary_field.iohandler.filename = sheardisprate_bc_xneg.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.bc.bc_yneg]
label = boundary_yneg
label_value = 12
use_initial = True
use_rate = True
```

(continues on next page)

(continued from previous page)

```

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Neumann BC +x edge
db_auxiliary_field.values = [initial_amplitude_tangential_1, initial_amplitude_
↪ tangential_2, initial_amplitude_normal, rate_start_time, rate_amplitude_tangential_1,
↪ rate_amplitude_tangential_2, rate_amplitude_normal]
db_auxiliary_field.data = [-9.0*MPa, 0.0*MPa, 0.0*MPa, 1.0*year, -2.25*MPa/year, 0.0*MPa/
↪ year, 0.0*MPa/year]

```

Running the simulation

Listing 4.70: Run Step 5 simulation

```

$ pylith step05_sheardispractrate.cfg

# The output should look something like the following.
>> /Users/baagaard/software/unix/micromamba/envs/pylith-debug/lib/python3.12/site-
↪ packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> /Users/baagaard/src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.1 time -0.1
    0 SNES Function norm 8.562738880728e-01
      Linear solve converged due to CONVERGED_ATOL iterations 4
    1 SNES Function norm 3.584100837616e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --

5 TS dt 0.1 time 0.4
    0 SNES Function norm 2.140684720394e-01
      Linear solve converged due to CONVERGED_ATOL iterations 0
    1 SNES Function norm 4.123676062187e-10
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
6 TS dt 0.1 time 0.5
>> /Users/baagaard/software/unix/micromamba/envs/pylith-debug/lib/python3.12/site-
↪ packages/pylith/problems/Problem.py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The output written to the terminal now contains multiple time steps. The PETSc TS (time stepping) monitor shows the time step and time in nondimensional units.

Visualizing the results

In Fig. 4.48 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.71: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step05_sheardispractrate-domain.h5 warp_grid --component=x
```

You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Suggested Exercises

1. Adjust the material properties in Step 1 and verify the effect on the deformation and stress field.
2. Use the `tet.gmsh` file in any of the steps. Can you change the mesh file using one additional command line argument?
3. Use the Exodus files from Cubit rather than the files from Gmsh.
4. Change the boundary conditions in Steps 2-5 to generate a different shear strain.
5. Set the basis order and quadrature order to 2. Do you get the same solution? Why?
6. In Step 4 change the initial conditions so that only one of the components is equal to the solution?
7. In Step 5 adjust the rate or time when the rate dependence starts.

Horizontal Cross-Section of Strike-Slip Fault (2D)

The files are in the directory `examples/strikeslip-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

generate_gmsh.py

Python script to generate the finite-element mesh using Gmsh.

generate_cubit.py

Python script to generate the finite-element mesh using Cubit.

***.msh**

Gmsh finite-element mesh files generated by Gmsh.

***.exo**

Exodus II finite-element mesh files generated by Cubit.

***.spatialdb**

Spatial database files associated with the spatial databases.

viz

Directory containing Matplotlib Python scripts for visualizing results.

output

Directory containing simulation output. It is created automatically when running the simulations.

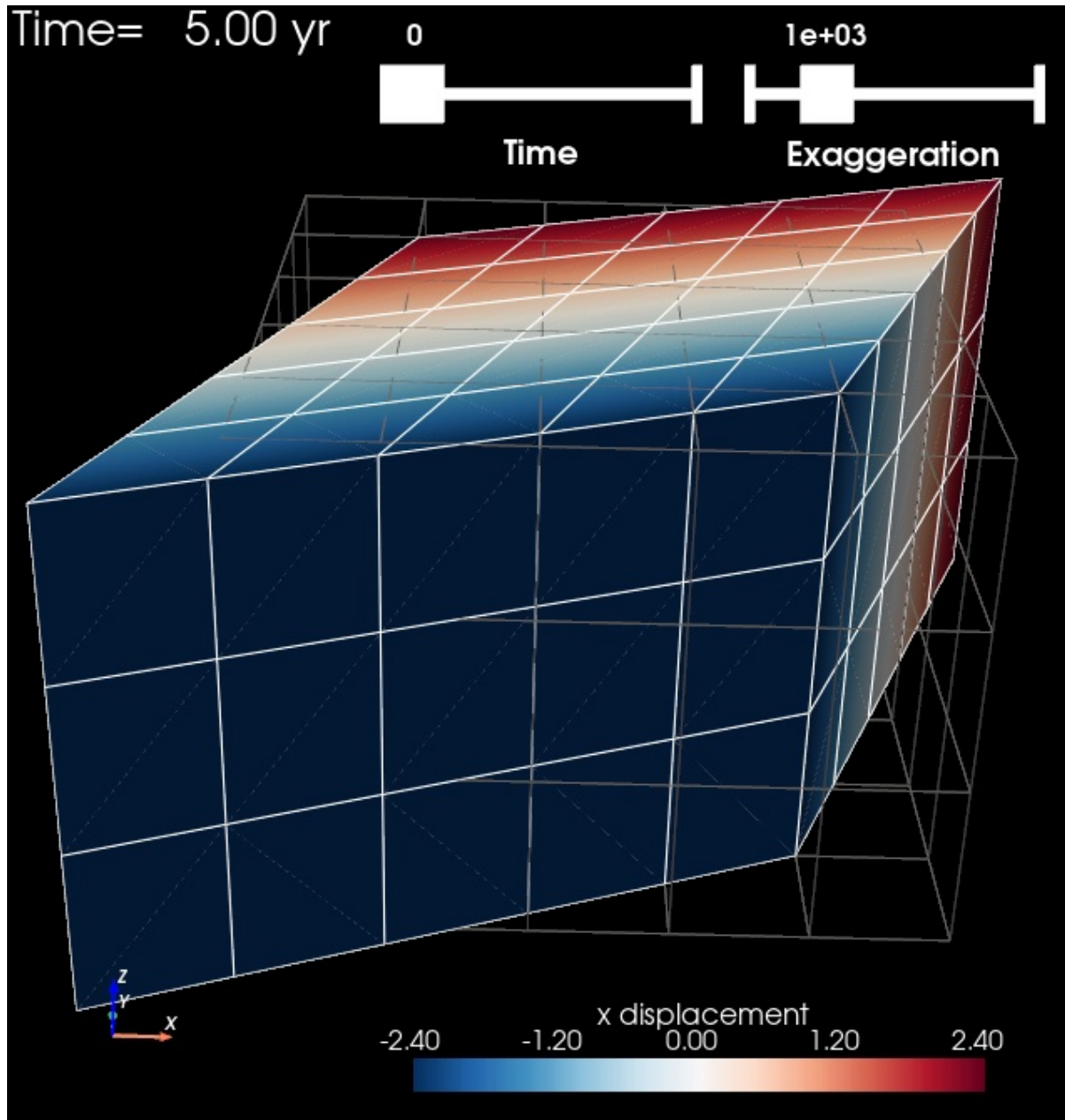


Fig. 4.48: Solution for Step 5 at a time of 5.0 years. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

Overview

This suite of examples demonstrates some basic concepts of using PyLith to solve the static and quasistatic boundary elasticity equation for a horizontal cross-section of a strike-slip fault (Fig. 4.49) with nonuniform material properties. The fault extends the entire length of the domain. The shear modulus is larger on the +x side of the fault. This example builds on the previous examples and adds complexity through a series of steps:

Step 1

Static coseismic slip with Dirichlet (displacement) boundary conditions.

Step 2

Quasistatic coseismic slip with time-dependent Dirichlet (displacement) boundary conditions.

Step 3

Quasistatic slip with two ruptures and time-dependent Dirichlet (displacement) boundary conditions.

Step 4

Variable slip and Dirichlet (displacement) boundary conditions.

Step 5

Static Green's functions with Dirichlet (displacement) boundary conditions.

Step 6

Invert for slip in Step 4 using Green's functions from Step 5 and least squares.

Step 7

Invert for slip in Step 4 using Green's functions from Step 5 and the CATMIP Bayesian framework.

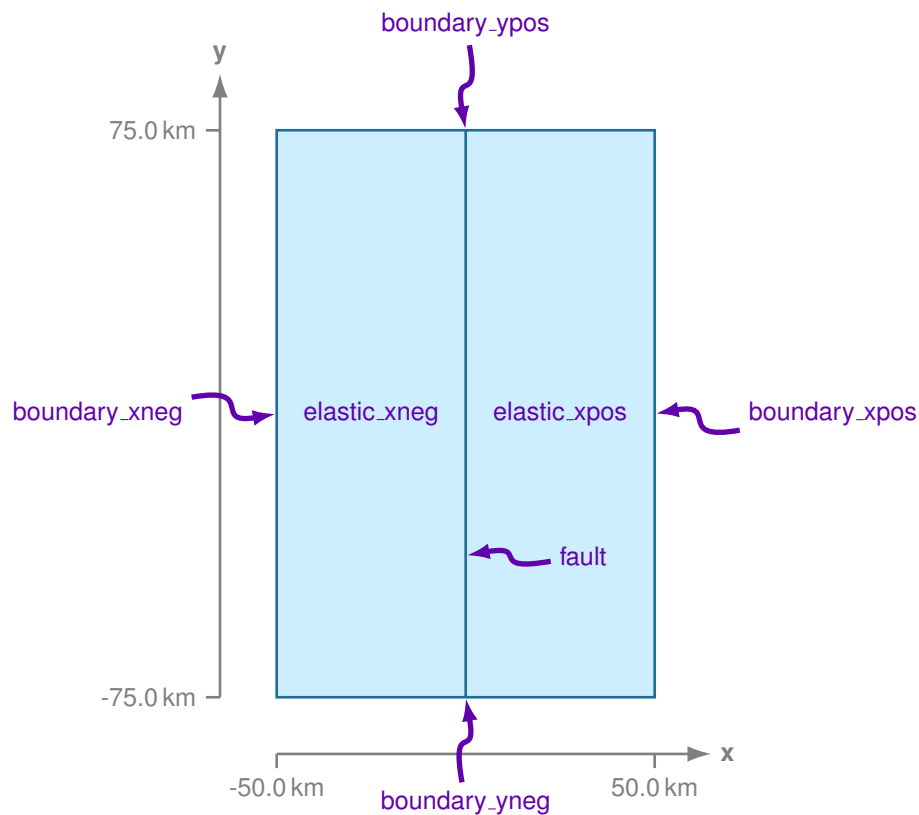


Fig. 4.49: Diagram of geometry for domain with a strike-slip fault. The domain extends from -50 km to +50 km in the x direction and from -75 km to +75 km in the y direction. We refer to the domain boundaries using the names shown in the diagram.

Important

We describe how to generate the finite-element mesh using both Gmsh and Cubit. The files for both methods are included. For Step 1 we provide PyLith parameter files for both meshes; for Steps 2 and 3 we only provide the Parameter files that use the Gmsh file.

Example Workflow**Gmsh Mesh****Geometry**

We construct the geometry by first creating points, then connecting the points into curves, and finally the curves into surfaces. Fig. 4.50 shows the geometry and variables names of the vertices and curves.

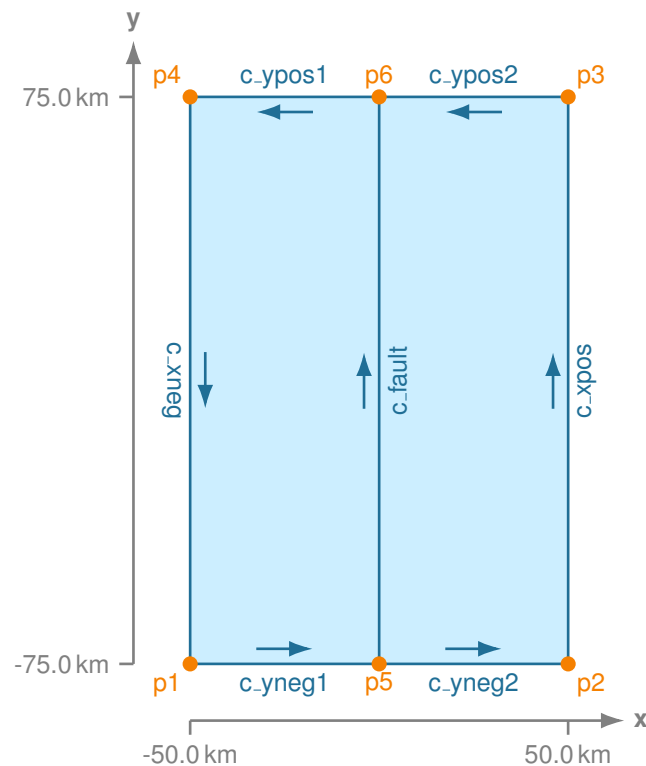


Fig. 4.50: Geometry created in Gmsh for generating the finite-element mesh. We construct curves from points (p1, ..., p6) and surfaces from the curves (for example, c_yneg1). The arrows indicate the direction (orientation) of the curves.

Important

Each curve in Gmsh has a direction (orientation). The direction is from the starting point to the ending point. When connecting curves into surfaces, you must connect the curves in a consistent direction. We connect the curves in a counter-clockwise direction. To reverse the direction of a curve, use the negative tag.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. The script makes use of the `gmsh_utils.GenerateMesh` class (discussed in [gmsh_utils](#)), which provides the command line arguments and boilerplate methods. In our `generate_gmsh.py` Python script, we create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class.

We use the Gmsh `MeshSize` options to define a discretization size that grows slowly at a geometric rate with distance from the fault. See [6.3.1 Specifying mesh element sizes in the Gmsh documentation](#) for more information.

Listing 4.72: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with triangular cells and save it to `mesh_tri.msh` (default filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui

# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui

# Show available command line arguments.
$ ./generate_gmsh.py --help
```

By default the Python script will generate a finite-element mesh with triangular cells and save it to the file `mesh_tri.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.73: View the Gmsh mesh file `mesh_tri.msh` using Gmsh.

```
gmsh -open mesh_tri.msh
```

Cubit Mesh**Geometry**

We construct the geometry by taking a horizontal cross-section of a 3D block. Alternatively, we could have constructed the geometry by building it up from points and curves like we did with Gmsh. [Fig. 4.52](#) shows the geometry and variables names of the vertices and curves.

Meshing using Python Script

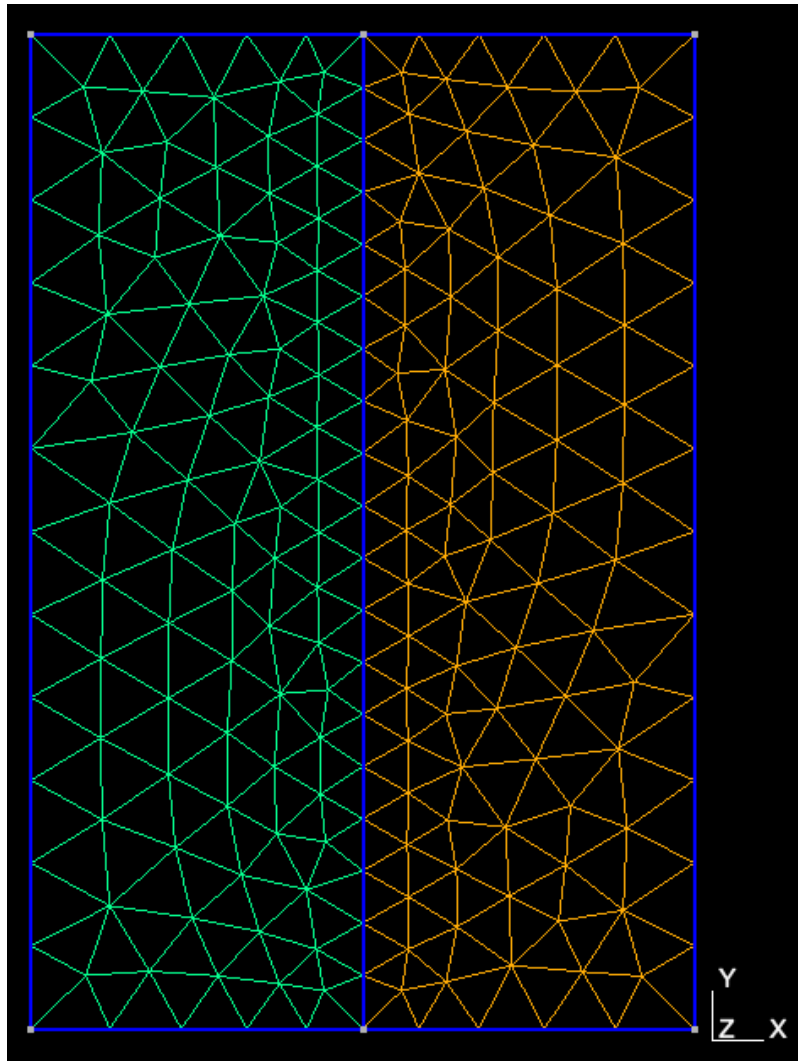


Fig. 4.51: Finite-element mesh with triangular cells generated by Gmsh.

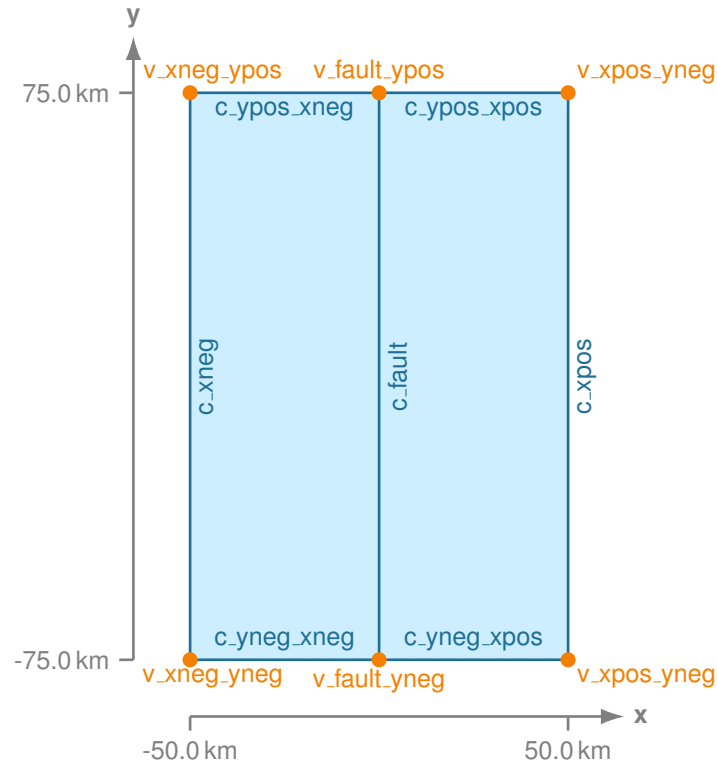


Fig. 4.52: Geometry created in Cubit for generating the finite-element mesh. The names of the vertices and curves match the ones we use in the Cubit journal files.

Note

New in v4.1.0.

We use a Python script to generate the mesh rather than Journal files. We also use the skeleton sizing function for the surface.

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

Open the Python script `generate_cubit.py` in the Cubit journal editor. Play the selected script or play the lines, making sure you play the first line so that Cubit uses the Python interpreter when running the script. We specify the parameters controlling the geometry, mesh size, and cell shape near the top of the script.

Important

We reference objects by their centroid so the resulting script should be independent of the version of Cubit that is being used.

Note

We use the skeleton sizing function to gradually increase the cell size with distance from the fault. This is much

simpler than applying the bias sizing function to the curves and surfaces. We set the minimum size equal to the cell size on the fault and the maximum gradient equal to the “bias”.

After running the Python script in Cubit, you will have a corresponding Exodus-II file (`mesh_tri.exo` or `mesh_quad.exo`).

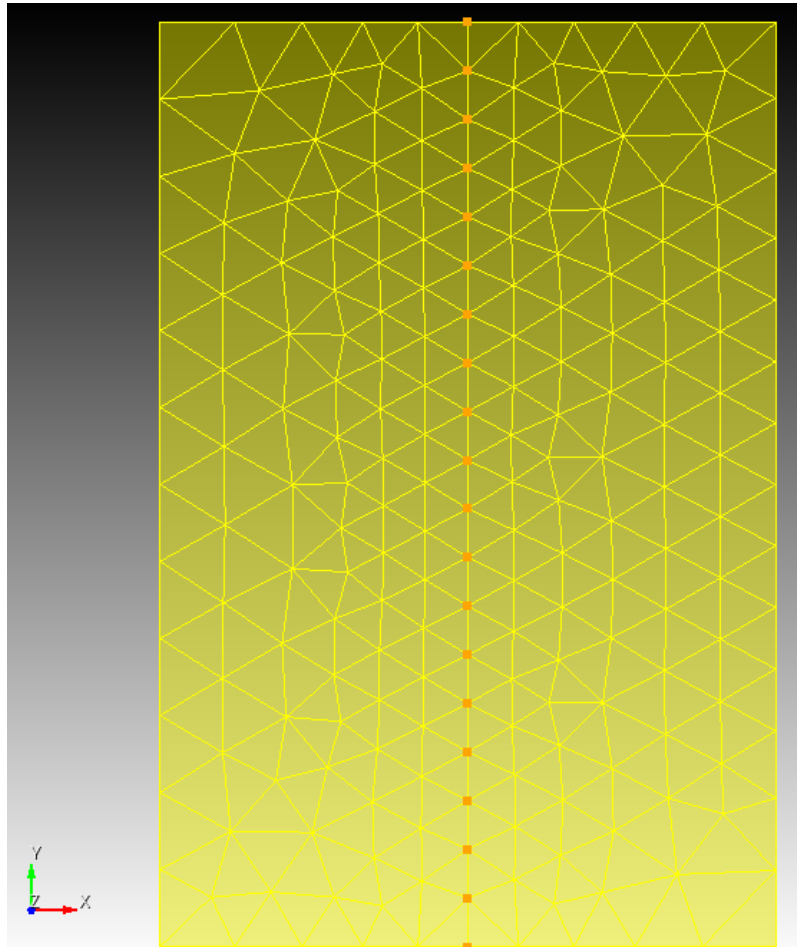


Fig. 4.53: Finite-element mesh with triangular cells generated by Cubit.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`, which contains numerous comments, so we only summarize the parameters here.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh in `pylithapp.mesh_generator`.

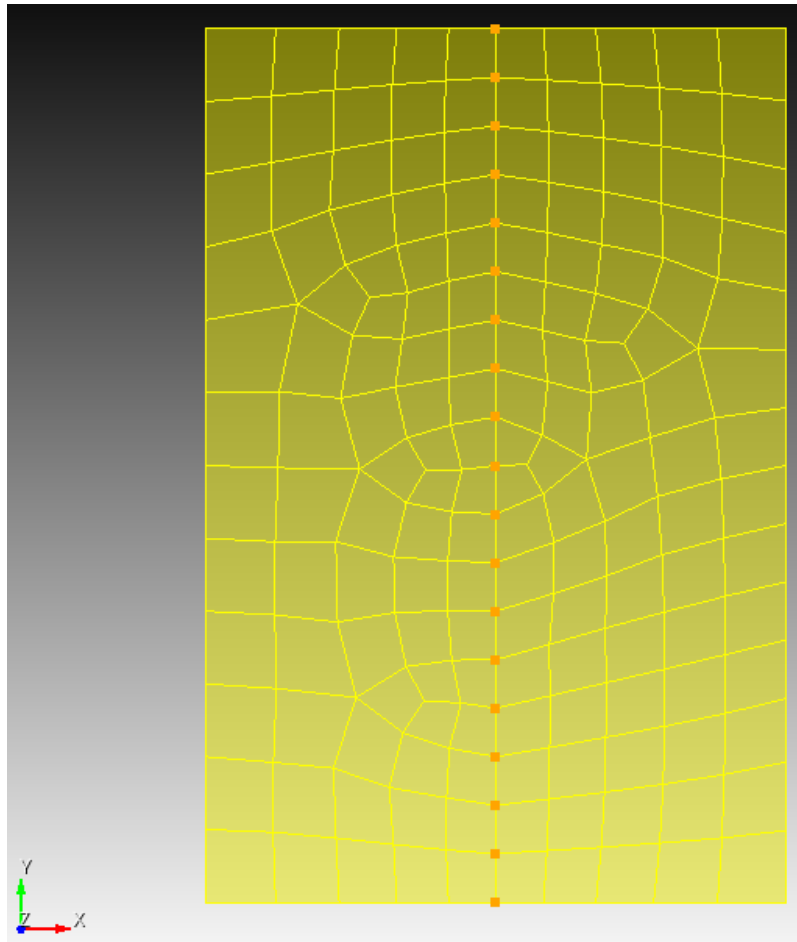


Fig. 4.54: Finite-element mesh with quadrilateral cells generated by Cubit.

Physics

These quasi-static simulations solve the elasticity equation and include a fault, so we have a solution field with both displacement and Lagrange multiplier subfields.

$$\vec{s} = \begin{pmatrix} \vec{u} & \vec{\lambda} \end{pmatrix}^T \quad (4.469)$$

$$\nabla \cdot \sigma(\vec{u}) = \vec{0} \quad (4.470)$$

We use the default TimeDependent problem and solution field with a single displacement subfield of basis order 1. In addition to output of the solution over the domain, we output the solution over the -y and +y boundaries.

Listing 4.74: Parameters for the solution and output of the solution on the -y and +y boundaries.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange

[pylithapp.problem]
solution_observers = [domain, top_boundary, bot_boundary]
solution_observers.top_boundary = pylith.meshio.OutputSolnBoundary
solution_observers.bot_boundary = pylith.meshio.OutputSolnBoundary

[pylithapp.problem.solution_observers.top_boundary]
label = boundary_ypos
label_value = 13

[pylithapp.problem.solution_observers.bot_boundary]
label = boundary_yneg
label_value = 12
```

We use the same material properties in all of the simulations in this directory, so we specify them in pylithapp.cfg to avoid repeating the information in the file with parameters for each simulation. We have two materials with a contrast in the shear modulus across the fault.

Listing 4.75: Material parameters common to all simulations in this directory.

```
[pylithapp.problem]
materials = [elastic_xneg, elastic_xpos]

[pylithapp.problem.materials.elastic_xneg]
label_value = 1

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Elastic properties xneg
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500.0*kg/m**3, 3.00*km/s, 5.29*km/s]

auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0

derived_subfields.cauchy_strain.basis_order = 0
derived_subfields.cauchy_stress.basis_order = 0
```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.materials.elastic_xpos]
label_value = 2

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Elastic properties xpos
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500.0*kg/m**3, 4.24*km/s, 5.29*km/s]

auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0

derived_subfields.cauchy_strain.basis_order = 0
derived_subfields.cauchy_stress.basis_order = 0
```

Similarly, we set the general fault parameters common to all simulations in the directory.

Listing 4.76: General fault parameters common to all simulations in this directory.

```
[pylithapp.problem]
interfaces = [fault]

[pylithapp.problem.interfaces.fault]
label = fault
label_value = 20

# Output `slip` and the change in fault traction on the fault.
observers.observer.data_fields = [slip, traction_change]
```

All simulations in this directory use Dirichlet boundary conditions on the -x and +x boundaries with zero displacements,

$$u_x(-50km, y) = 0, \quad (4.471)$$

$$u_y(-50km, y) = 0, \quad (4.472)$$

$$u_x(+50km, y) = 0, \quad (4.473)$$

$$u_y(+50km, y) = 0. \quad (4.474)$$

Listing 4.77: Dirichlet boundary conditions common to all simulations in this directory.

```
[pylithapp.problem]
bc = [bc_xneg, bc_xpos]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xpos]
label = boundary_xpos
label_value = 11
constrained_dof = [0, 1]
db_auxiliary_field = pylith.bc.ZeroDB
```

(continues on next page)

(continued from previous page)

```

db_auxiliary_field.description = Dirichlet BC +x boundary

[pylithapp.problem.bc.bc_xneg]
label = boundary_xneg
label_value = 10
constrained_dof = [0, 1]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC -x boundary

```

Step 1: Static Coseismic Slip

Features

- Triangular cells
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- field split preconditioner
- Schur complement preconditioner
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation

This example involves a static simulation that solves for the deformation from prescribed coseismic slip on the fault. Fig. 4.55 shows the boundary conditions on the domain.

Step 1a: Coarse Mesh

Note

New in v4.1.0.

We start with a coarse resolution mesh and increase the resolution of the simulation by using uniform refinement or increasing the basis order of the solution fields.

Simulation parameters

The parameters specific to this example are in `step01a_slip.cfg`.

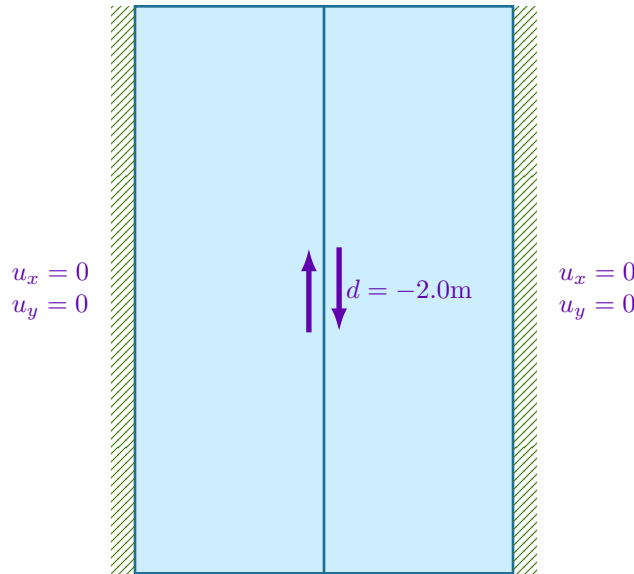


Fig. 4.55: Boundary conditions for static coseismic slip. We set the x and y displacement to zero on the +x and -x boundaries and prescribe 2 meters of right-lateral slip.

Listing 4.78: Prescribed slip parameters for Step 1.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -2.0*m, 0.0*m]
```

Running the simulation

Listing 4.79: Run Step 1a simulation

```
$ pylith step01a_slip.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/utils/PetscOptions.cc:251:static void pylith::utils::_
↪ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    dm_reorder_section = true
    dm_reorder_section_type = cohesive
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
```

(continues on next page)

(continued from previous page)

```

    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    mg_fine_pc_type = vpbjacobi
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:170:virtual void
↳ pylith::meshio::MeshIOCubit::_read()
-- info (application-flow)
-- Component 'meshiocubit.reader': Reading finite-element mesh from 'mesh_tri.exo'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:207:virtual void
↳ pylith::meshio::MeshIOCubit::_read()
-- info (application-flow)
-- Component 'meshiocubit.reader': Read 378 cells and 213 vertices.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiocubit.reader': Domain bounding box:
    (-50000, 50000)
    (-75000, 75000)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
↳ pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:
↳ printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 8000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↳ pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh
↳ *pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const
↳ char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed.

```

(continues on next page)

(continued from previous page)

```

↳ in v6.0. In the future, you will need to mark boundaries instead of vertices.
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
↳ *pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
↳ char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
↳ in v6.0. In the future, you will need to mark boundaries instead of vertices.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 8.489136776687e+00
      Linear solve converged due to CONVERGED_ATOL iterations 9
    1 SNES Function norm 3.930434823490e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
↳ *pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
↳ char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
↳ in v6.0. In the future, you will need to mark boundaries instead of vertices.
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
↳ *pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
↳ char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
↳ in v6.0. In the future, you will need to mark boundaries instead of vertices.
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the `MeshIOPetsc` reader and that it found the domain to extend from -50,000 m to +50,000 m in the x direction and from -75,000 m to +75,000 m in the y direction. The scales for nondimensionalization remain the default values for a quasistatic problem. PyLith detects the presence of a fault based on the Lagrange multiplier for the fault in the solution field and selects appropriate preconditioning options as discussed in [PETSc Options](#).

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 9 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snas_atol`).

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (.h5) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (.xmf) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In [Fig. 4.56](#) and [Fig. 4.57](#) we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.80: Visualize PyLith output using pylith_viz.

```

pylith_viz --filename=output/step01a_slip-domain.h5 warp_grid --component=y
pylith_viz --filename=output/step01a_slip-elastic_xneg.h5,output/step01a_slip-elastic_
xpos.h5 plot_field --field=cauchy_stress --component=xy

```

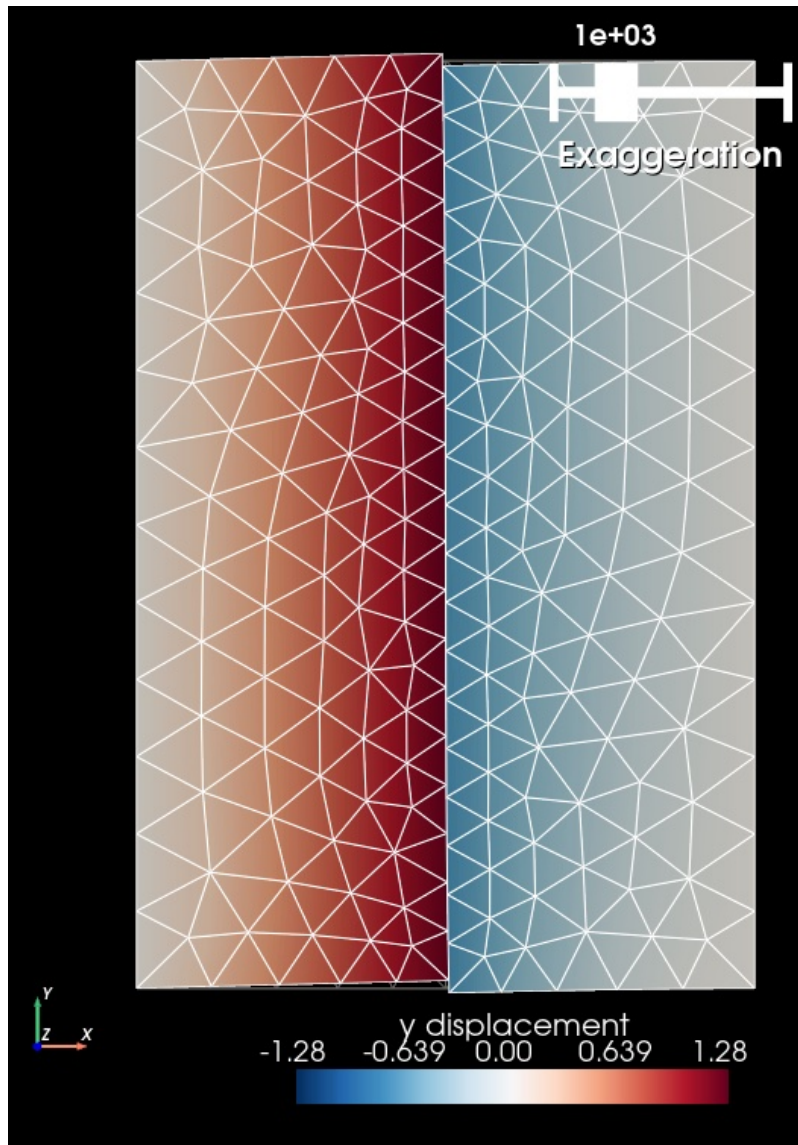


Fig. 4.56: Solution for Step 1a. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The contrast in material properties across the faults causes the asymmetry in the y displacement field.

Step 1a with Cubit Mesh

Using the Cubit mesh rather than the Gmsh mesh involves two changes:

1. Use the `MeshIOCubit` reader instead of the `MeshIOPetsc` reader and change the filename of the mesh file.
2. Set the `label_value` to 1 for boundary conditions and faults.

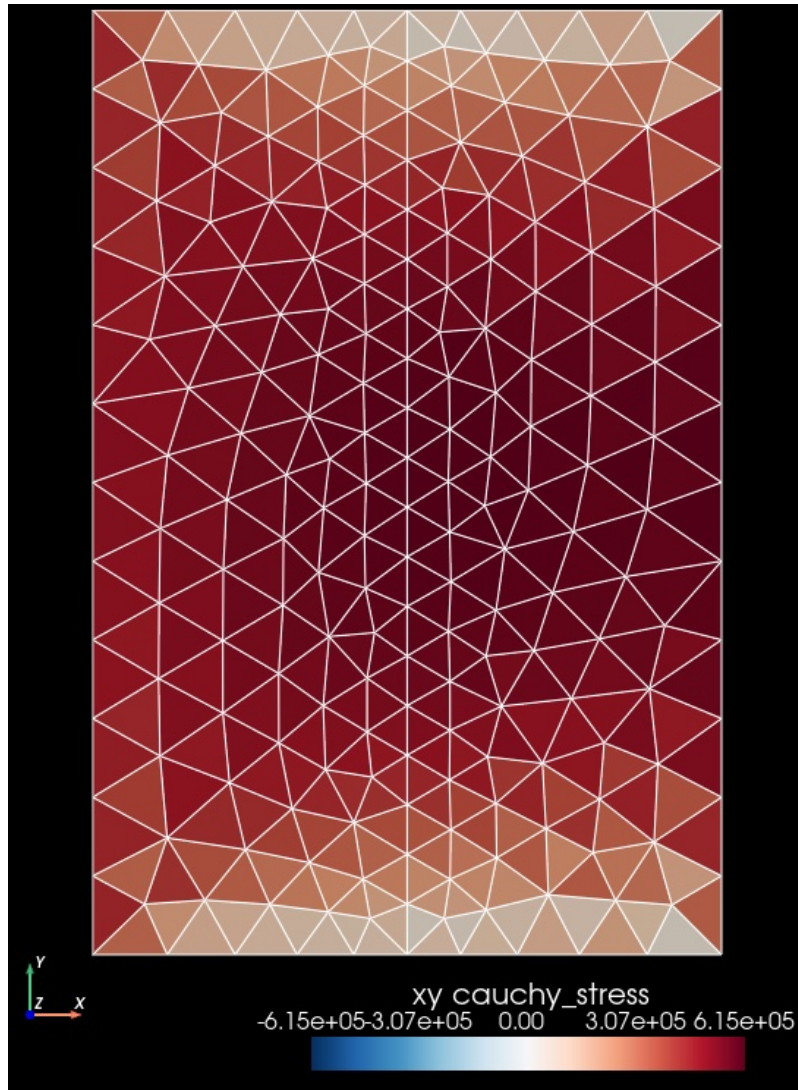


Fig. 4.57: Cauchy stress component xy for for Step 1a. The colors of the shaded surface indicate the xy stress component. The stress field is uniform within each cell, and the coarse resolution shows a checkerboard pattern near the top and bottom boundaries, which is indicative of poor resolution of the stress field in those regions.

We must override the nondefault `label_value` settings in `pylithapp.cfg` that were appropriate for our Gmsh reader but are incorrect for the Cubit reader.

The file `step01_slip_cubit.cfg` provides these changes and updates the names for output.

Listing 4.81: Run Step 1 simulation with the Cubit mesh

```
$ pylith step01_slip_cubit.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    dm_reorder_section = true
    dm_reorder_section_type = cohesive
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    mg_fine_pc_type = vpbjacobi
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOcubit.cc:170:virtual void
↳ pylith::meshio::MeshIOcubit::_read()
-- info (application-flow)
-- Component 'meshiocubit.reader': Reading finite-element mesh from 'mesh_tri.exo'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOcubit.cc:207:virtual void
↳ pylith::meshio::MeshIOcubit::_read()
-- info (application-flow)
-- Component 'meshiocubit.reader': Read 378 cells and 213 vertices.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiocubit.reader': Domain bounding box:
    (-50000, 50000)
    (-75000, 75000)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
```

(continues on next page)

(continued from previous page)

```

→pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
→printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 8000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void_
→pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
→*pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
→char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
→in v6.0. In the future, you will need to mark boundaries instead of vertices.
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
→*pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
→char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
→in v6.0. In the future, you will need to mark boundaries instead of vertices.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
→pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 8.489136776687e+00
      Linear solve converged due to CONVERGED_ATOL iterations 9
    1 SNES Function norm 3.930434823490e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
→*pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
→char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
→in v6.0. In the future, you will need to mark boundaries instead of vertices.
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
→*pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
→char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
→in v6.0. In the future, you will need to mark boundaries instead of vertices.
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
→py:222:finalize

```

(continues on next page)

(continued from previous page)

```
-- info (application-flow)
-- Finalizing problem.
```

The MeshIOCubit reader includes diagnostic information in the journal output related to the sizes of the nodesets and material blocks.

Step 1b: Refined Mesh

The parameters specific to this example are in `step01b_slip.cfg`.

Simulation parameters

In an attempt to better resolve the stress field, we use the same coarse resolution input mesh but add uniform refinement after PyLith reads in the mesh to reduce the discretization size by a factor of 2 in the. During uniform refinement, each triangle is subdivided into four triangles.

Listing 4.82: Uniform refinement parameters added in Step 1b.

```
[pylithapp.mesh_generator]
refiner = pylith.topology.RefineUniform
```

Running the simulation

Listing 4.83: Run Step 1b simulation

```
$ pylith step01b_slip.cfg
# The output will look almost identical to Step 1a.
```

Visualizing the results

In Fig. 4.58 and Fig. 4.59 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.84: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step01b_slip-domain.h5 warp_grid --component=y
pylith_viz --filename=output/step01b_slip-elastic_xneg.h5,output/step01b_slip-elastic_
↪xpos.h5 plot_field --field=cauchy_stress --component=xy
```

Step 1c: Higher Order Discretization

The parameters specific to this example are in `step01c_slip.cfg`.

Simulation parameters

Increasing the basis order of the solution subfields provides an alternative approach to increasing the resolution of the mesh using uniform refinement.

The accuracy of the stress and strain will be 1 order lower than the basis order of the displacement field. Consequently, we use a basis order of 1 (rather than 0) for the output of the Cauchy stress and strain. We continue to output the solution fields using a basis order of 1, because many visualization tools do not know how to display fields with a basis order of 2. This means the solution subfields in the output are at a reduced resolution compared to the simulation and correspond to projection of each subfield from a basis order of 2 to a basis order of 1.

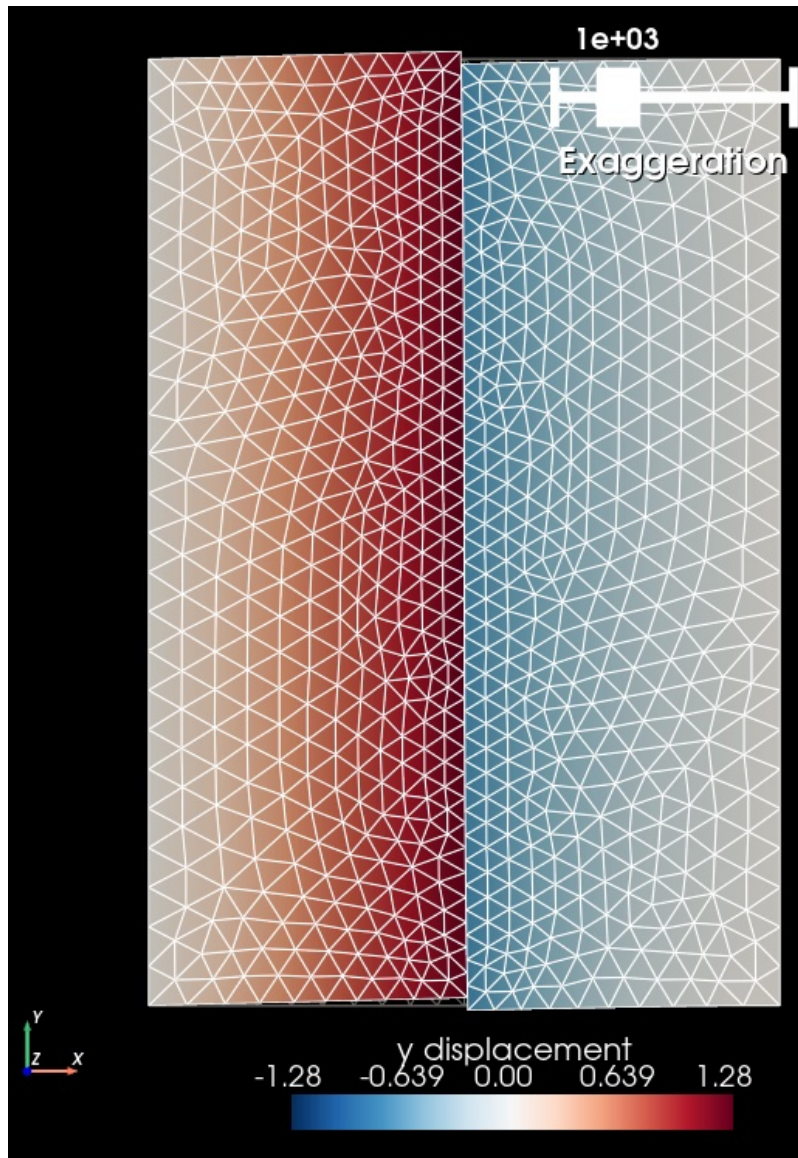


Fig. 4.58: Solution for Step 1b. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. Uniform refinement reduces the discretization size by a factor of 2.

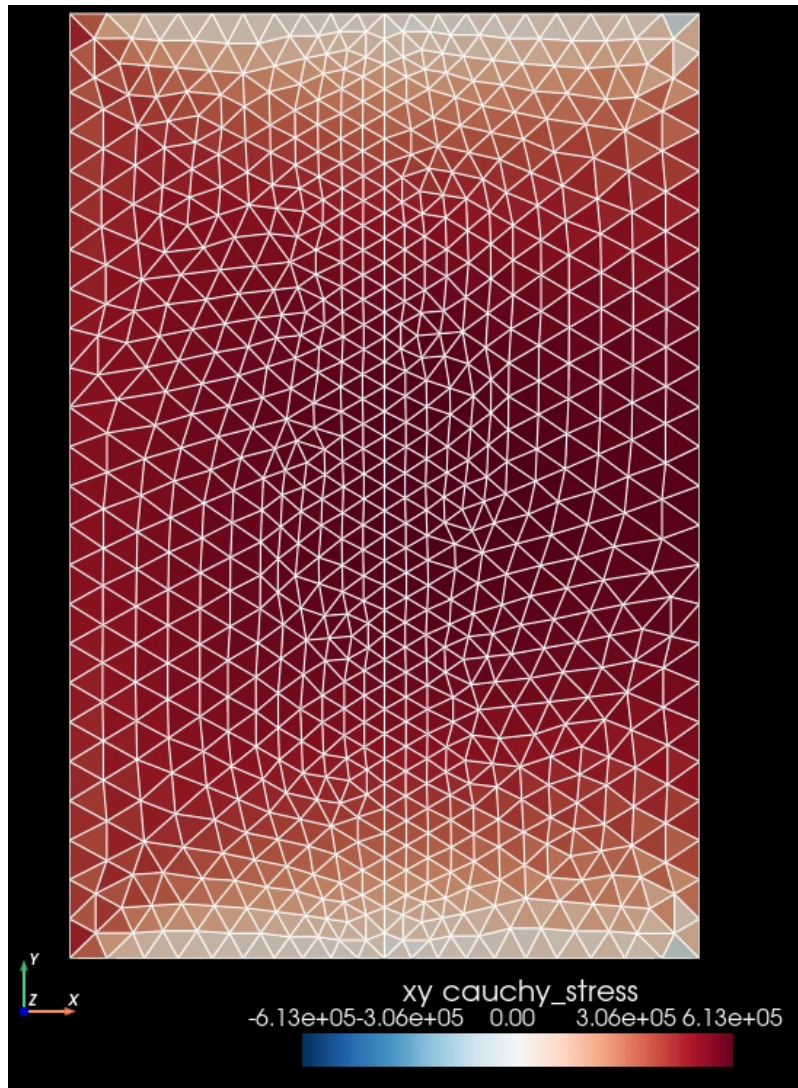


Fig. 4.59: Cauchy stress component xy for for Step 1b. The colors of the shaded surface indicate the xy stress component. The reduced discretization size results in better resolution of the stress changes near the top and bottom boundaries. However, the stress field is still uniform within each cell, so the checkerboard pattern still persists, albeit with less variation between cells.

Listing 4.85: Parameters related to increasing the basis order of the solution subfields to 2.

```
[pylithapp.problem]
defaults.quadrature_order = 2

[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
lagrange_multiplier_fault.basis_order = 2

[pylithapp.problem.materials.elastic_xneg]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.elastic_xpos]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1
```

Running the simulation

Listing 4.86: Run Step 1c simulation

```
$ pylith step01c_slip.cfg
# The output will look almost identical to Steps 1a and 1b.
```

Visualizing the results

In Fig. 4.60 and Fig. 4.61 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.87: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step01c_slip-domain.h5 warp_grid --component=y
pylith_viz --filename=output/step01c_slip-elastic_xneg.h5,output/step01c_slip-elastic_
↪xpos.h5 plot_field --field=cauchy_stress --component=xy
```

Note

We focus on the displacement field in the subsequent steps in the example, which has sufficient accuracy with a basis order of 1 if we use uniform refinement. Consequently, in the subsequent steps we adopt the uniform refinement parameters that we used in Step 1b.

Key Points

1. We can generate a mesh at coarse resolution that captures the geometry and use uniform refinement and high order discretizations of the solution subfields to achieve sufficient resolution.
2. Using uniform mesh refinement in PyLith requires only two parameter settings: enabling uniform refinement by setting the mesh generator `refiner` facility to `pylith.topology.RefineUniform` and setting the number of levels of refinement.

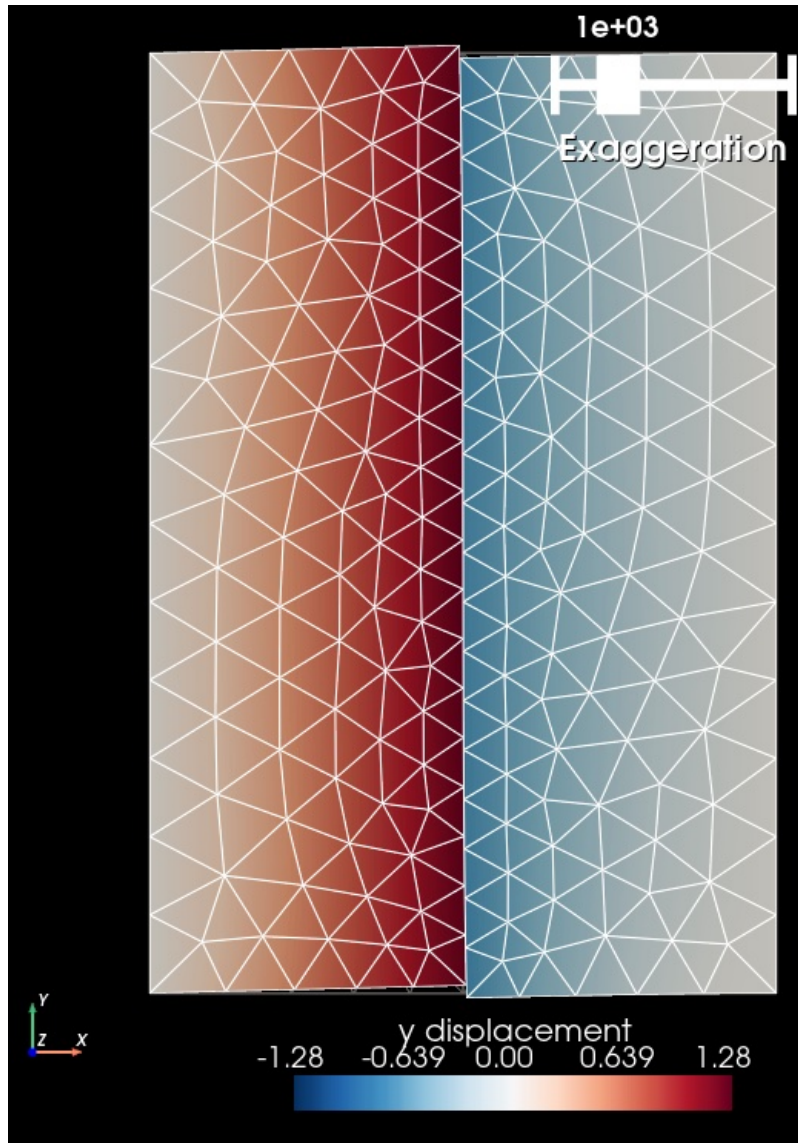


Fig. 4.60: Solution for Step 1a. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The displacement field shows little difference from Step 1a.

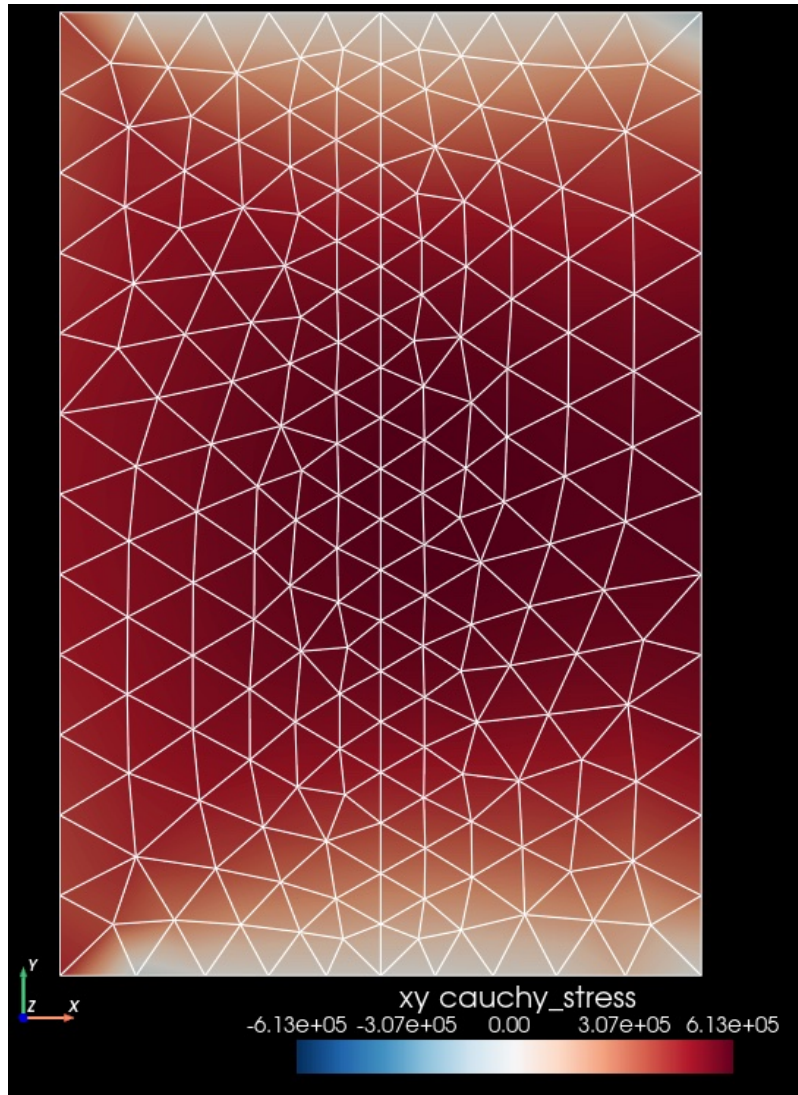


Fig. 4.61: Cauchy stress component xy for for Step 1a. The colors of the shaded surface indicate the xy stress component. Output with a basis order of 1 shows much better resolution of the shear stress near the top and bottom of the domain with no noticeable checkboard pattern.

3. Using higher order discretizations of the solution subfields requires adjusting more parameters than uniform refinement. We adjust the default quadrature order, basis order of the solution fields, and basis order of relevant output fields.
4. Because uniform mesh refinement requires only a couple of additional parameters, we recommend first using uniform refinement to assess the sensitivity of the results to the discretization size before increasing the basis order of the solution subfields. However, if you know your results are sensitive to the stress field, then we recommend first increasing the basis order.
5. In some cases, you may find that uniform refinement and higher order discretizations indicate that some portions of the mesh need higher resolution than others, and you might need to adjust the spatial variation of the discretization size in the initial coarse mesh.

Step 2: Single Earthquake Rupture and Velocity Boundary Conditions

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.faults.FaultCohesiveKin`
- `pylith.faults.KinSrcStep`
- field split preconditioner
- Schur complement preconditioner
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Quasi-static simulation
- `spatialdata.spatialdb.SimpleDB`

Simulation parameters

This example involves a quasi-static simulation that solves for the deformation from velocity boundary conditions and prescribed coseismic slip on the fault. We let strain accumulate due to the motion of the boundaries and then release the strain by prescribing 2 meters of right-lateral slip at $t=100$ years. [Fig. 4.62](#) shows the boundary conditions on the domain. The parameters specific to this example are in `step02_slip_velbc.cfg`.

Listing 4.88: Time stepping parameters for Step 2.

```
[pylithapp.problem]
initial_dt = 5.0*year
start_time = -5.0*year
end_time = 120.0*year
```

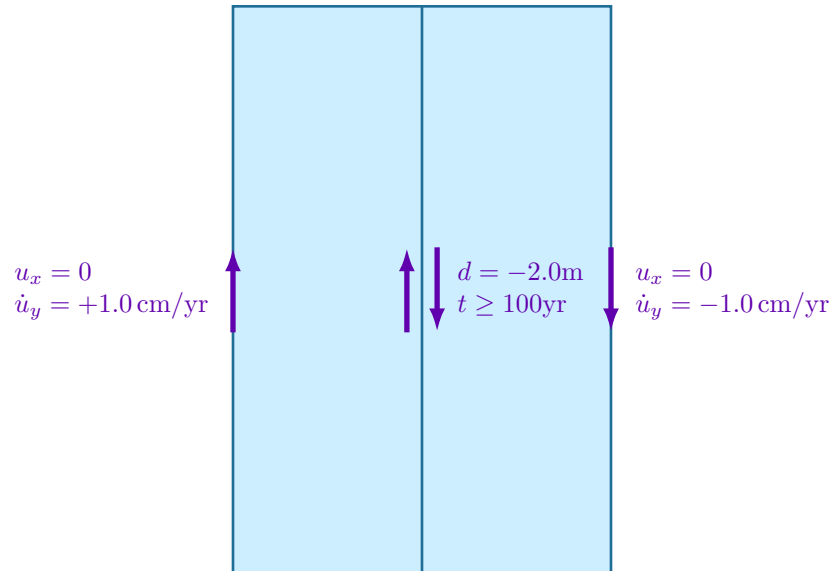



Fig. 4.62: Boundary conditions for quasistatic simulation with velocity boundary conditions and coseismic slip. We set the x displacement to zero on the +x and -x boundaries. We set the y velocity to -1 cm/yr on the +x boundary and +1 cm/yr on the -x boundary. We prescribe 2 meters of right-lateral slip to occur at 100 years to release the accumulated strain energy.

Boundary conditions

We switch the Dirichlet boundary conditions from specifying an initial amplitude to specifying a constant velocity,

$$\dot{u}_x(-50km, y) = 0, \quad (4.475)$$

$$\dot{u}_y(-50km, y) = +1cm/yr, \quad (4.476)$$

$$\dot{u}_x(+50km, y) = 0, \quad (4.477)$$

$$\dot{u}_y(+50km, y) = -1cm/yr. \quad (4.478)$$

Listing 4.89: Dirichlet boundary conditions for Step 2. We only show the details for the +x boundary.

```
[pylithapp.problem.bc.bc_xpos]
use_initial = False
use_rate = True

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC +x boundary
db_auxiliary_field.iohandler.filename = disprate_bc_xpos.spatialdb
```

As in Step 1 we prescribe 2.0 meters of right-lateral slip, but in this case we set slip to occur at t=100 years.

Listing 4.90: Prescribed slip parameters for Step 2.

```
[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
```

(continues on next page)

(continued from previous page)

```
db_auxiliary_field.data = [100.0*year, -2.0*m, 0.0*m]
```

Running the simulation

Listing 4.91: Run Step 2 simulation

```
$ pylith step02_slip_velbc.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.05 time -0.05
  0 SNES Function norm 0.000000000000e+00
  Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
1 TS dt 0.05 time 0.
  0 SNES Function norm 1.587863174294e+00
  Linear solve converged due to CONVERGED_ATOL iterations 9
  1 SNES Function norm 4.122054854127e-08
  Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --
24 TS dt 0.05 time 1.15
  0 SNES Function norm 1.587863188128e+00
  Linear solve converged due to CONVERGED_ATOL iterations 1
  1 SNES Function norm 4.146299281838e-08
  Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
25 TS dt 0.05 time 1.2
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

The beginning of the output written to the terminal is identical to that from Step 1. At the end of the output, we see that the simulation advanced the solution 25 time steps. Remember that the PETSc TS monitor shows the nondimensionalized time and time step values.

Visualizing the results

In Fig. 4.63 we use the `pylith_viz` utility to visualize the y displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.92: Visualize PyLith output using pylith_viz.

```
pylith_viz --filename=output/step02_slip_velbc-domain.h5 warp_grid --component=y
```

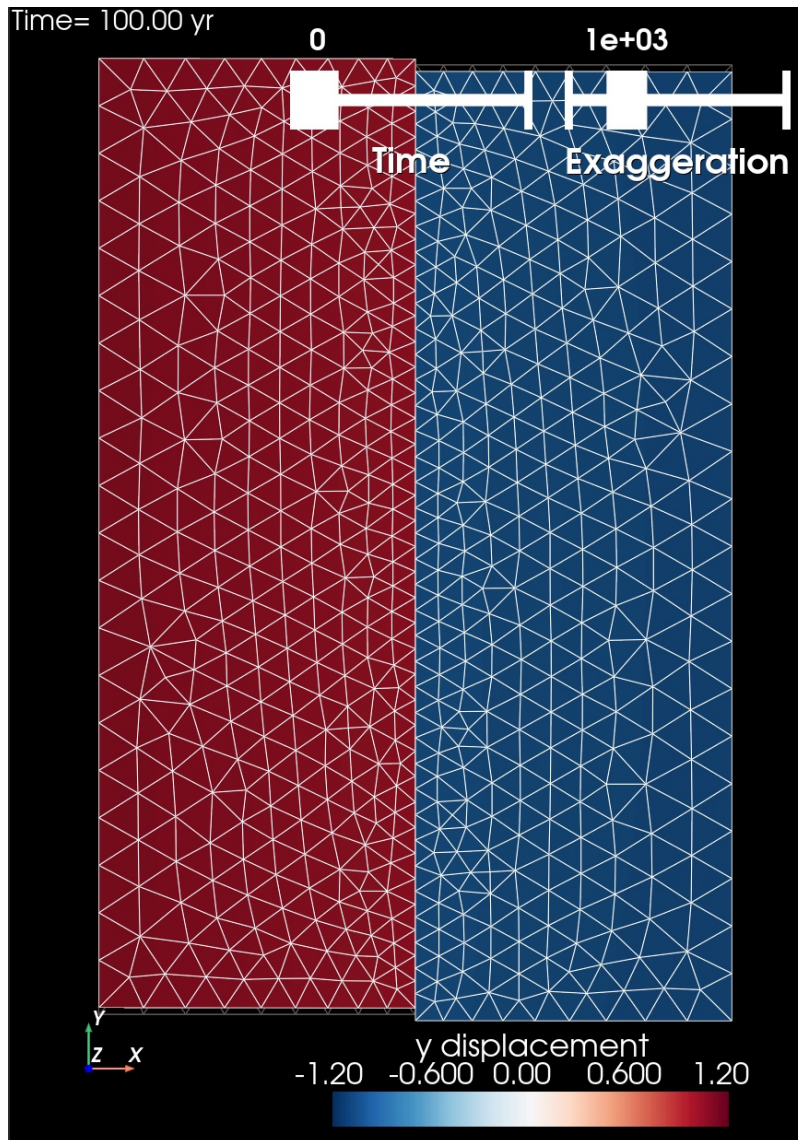


Fig. 4.63: Solution for Step 2 at $t=100$ yr. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The coseismic fault slip at 100 years releases all of the accumulated strain energy.

Step 3: Multiple Earthquake Ruptures and Velocity Boundary Conditions

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`

- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- field split preconditioner
- Schur complement preconditioner
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Quasi-static simulation
- spatialdata.spatialdb.SimpleDB

Simulation parameters

This example involves a quasistatic simulation that solves for the deformation from velocity boundary conditions and multiple earthquake ruptures on the fault. The velocity boundary conditions match those in Step 2. We prescribe the first earthquake rupture to occur at 100 years with 1 meter of right-lateral slip and the second earthquake rupture to occur at 200 years with 3 meters of right-lateral slip. Fig. 4.64 shows the boundary conditions on the domain. The parameters specific to this example are in `step03_multislip_velbc.cfg`.

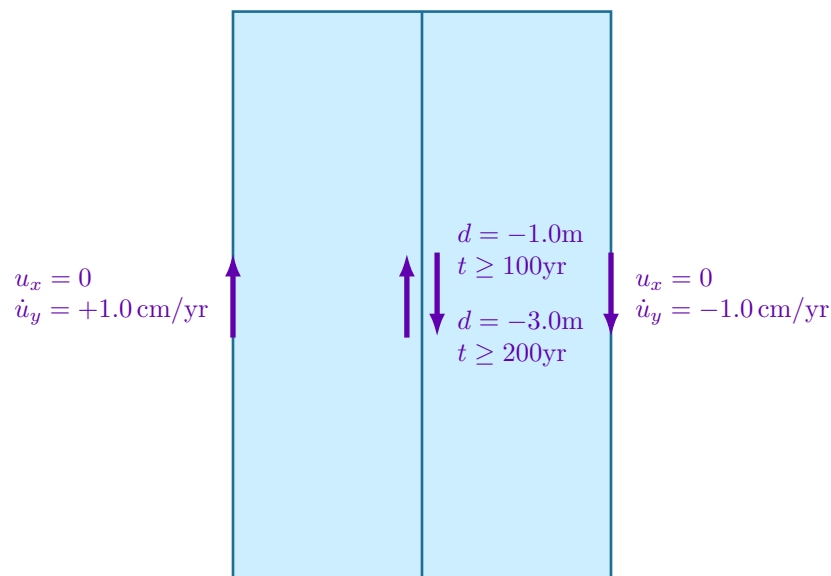


Fig. 4.64: Boundary conditions for quasi-static simulation with velocity boundary conditions and coseismic slip. We set the x displacement to zero on the +x and -x boundaries. We set the y velocity to -1 cm/yr on the +x boundary and +1 cm/yr on the -x boundary. We prescribe 1 meter of right-lateral slip to occur at 100 years and 3 meters of right-lateral slip to occur at 200 years.

Listing 4.93: Prescribed slip rupture parameters for Step 3 with two earthquake ruptures on the strike-slip fault.

```
[pylithapp.problem.interfaces.fault]
eq_ruptures = [one, two]

[pylithapp.problem.interfaces.fault.eq_ruptures.one]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture one
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪opening]
db_auxiliary_field.data = [100.0*year, -1.0*m, 0.0*m]

[pylithapp.problem.interfaces.fault.eq_ruptures.two]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture two
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪opening]
db_auxiliary_field.data = [200.0*year, -3.0*m, 0.0*m]
```

Running the simulation

Listing 4.94: Run Step 3 simulation

```
$ pylith step03_multislip_velbc.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --

25 TS dt 0.1 time 2.4
    0 SNES Function norm 3.175726344664e+00
      Linear solve converged due to CONVERGED_ATOL iterations 0
    1 SNES Function norm 1.372645307731e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
26 TS dt 0.1 time 2.5
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

The beginning of the output written to the terminal is identical to that from Step 1. At the end of the output, we see that the simulation advanced the solution 26 time steps. Remember that the PETSc TS monitor shows the nondimensionalized time and time step values.

Visualizing the results

In Fig. 4.65 we use the `pylith_viz` utility to visualize the y displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.95: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step03_multislip_velbc-domain.h5 warp_grid --component=y
```

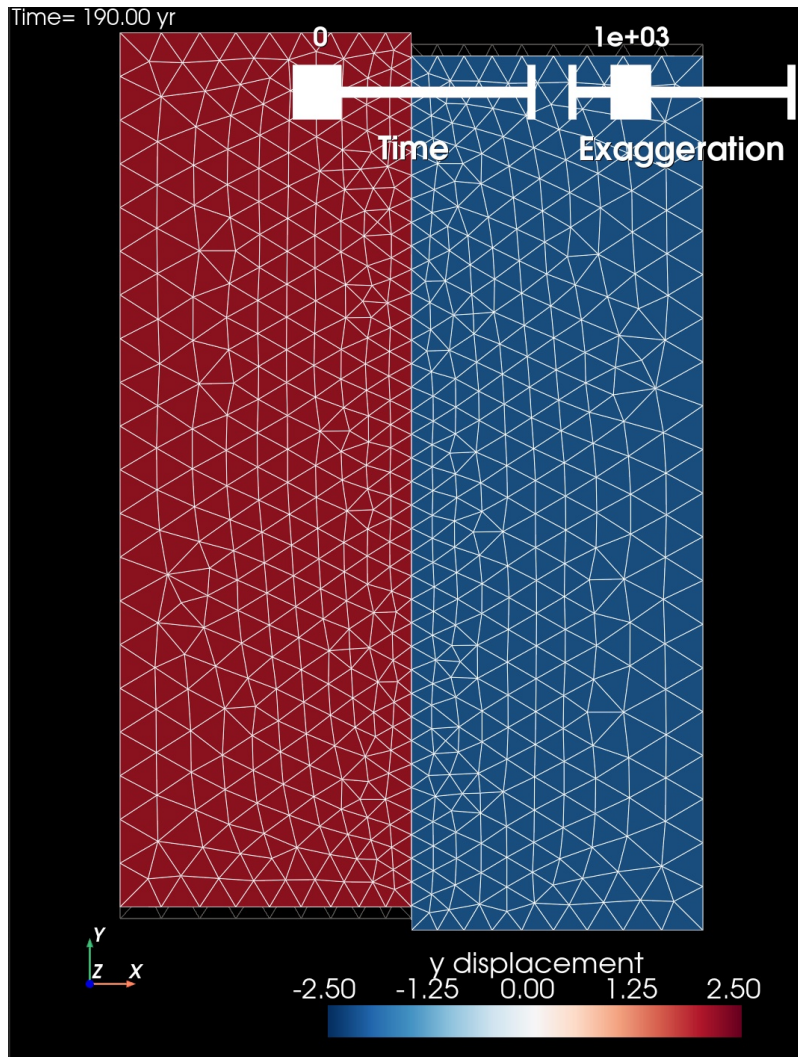


Fig. 4.65: Solution for Step 3 at $t=190$ yr. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

Step 4: Variable Coseismic Slip

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`

- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.faults.FaultCohesiveKin`
- `pylith.faults.KinSrcStep`
- field split preconditioner
- Schur complement preconditioner
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation

Simulation parameters

We use this example to illustrate prescribing slip that varies along the strike of the fault. This example also serves as a means to generate coseismic displacements at fake GNSS stations. In Step 6 we will use the displacements at these stations along with static Green's functions computed in Step 5 to invert for the slip on the fault.

We prescribe left-lateral slip that varies along the strike of the fault with fixed displacements on the $+x$ and $-x$ boundaries (Fig. 4.66), similar to what we had in Step 1. The slip is nonzero over the region $-20 \text{ km} \leq y \leq +20 \text{ km}$ with a peak slip of 80 cm at $y = -0.5 \text{ km}$ (Fig. 4.68).

This example involves a static simulation that solves for the deformation from prescribed coseismic slip on the fault. Fig. 4.66 shows the boundary conditions on the domain. The parameters specific to this example are in `step04_varsip.cfg`.

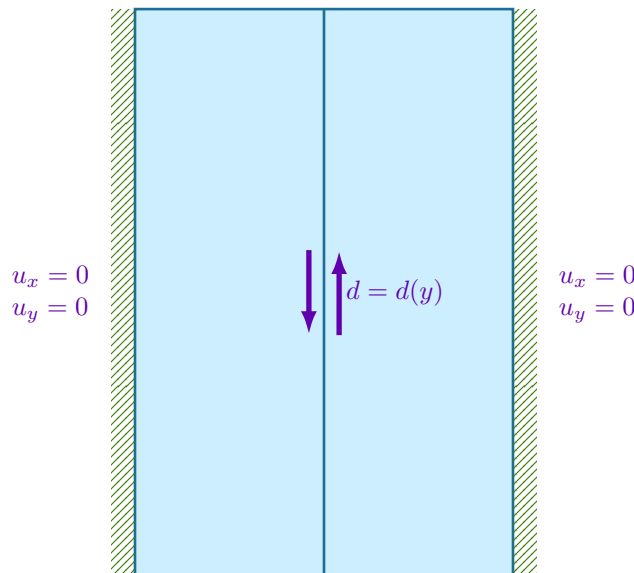


Fig. 4.66: Boundary conditions for static coseismic slip. We set the x and y displacement to zero on the $+x$ and $-x$ boundaries and prescribe left-lateral slip that varies along strike.

For greater accuracy in modeling the spatial variation in slip, we refine the mesh by a factor of 2 and use a basis order of 2 for the solution subfields.

Listing 4.96: Parameters related to increasing the basis order of the solution subfields to 2.

```
[pylithapp.problem]
defaults.quadrature_order = 2

[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
lagrange_multiplier_fault.basis_order = 2

[pylithapp.problem.materials.elastic_xneg]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.elastic_xpos]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1
```

We also add output of the solution at fake GNSS stations given in the file `gnss_stations.txt`. You can use the Python script `generate_gnssstations.py` to generate a different random set of stations; the default parameters will generate the provided `gnss_stations.txt` file.

Listing 4.97: Solution and output parameters for Step 4. We add output of the solution at fake GNSS stations.

```
[pylithapp.problem]
solution_observers = [domain, top_boundary, bot_boundary, gnss_stations]
solution_observers.gnss_stations = pylith.meshio.OutputSolnPoints

[pylithapp.problem.solution_observers.gnss_stations]
label = gnss_stations
reader.filename = gnss_stations.txt
reader.coordsys.space_dim = 2
```

Listing 4.98: Solution and output parameters for Step 4. We add output of the solution at fake GNSS stations.

```
[pylithapp.problem]
solution_observers = [domain, top_boundary, bot_boundary, gnss_stations]
solution_observers.gnss_stations = pylith.meshio.OutputSolnPoints

[pylithapp.problem.solution_observers.gnss_stations]
label = gnss_stations
reader.filename = gnss_stations.txt
reader.coordsys.space_dim = 2
```

The earthquake rupture occurs along the central portion of the fault with spatially variable slip.

We use a `SimpleGridDB` to define the spatial variation in slip.

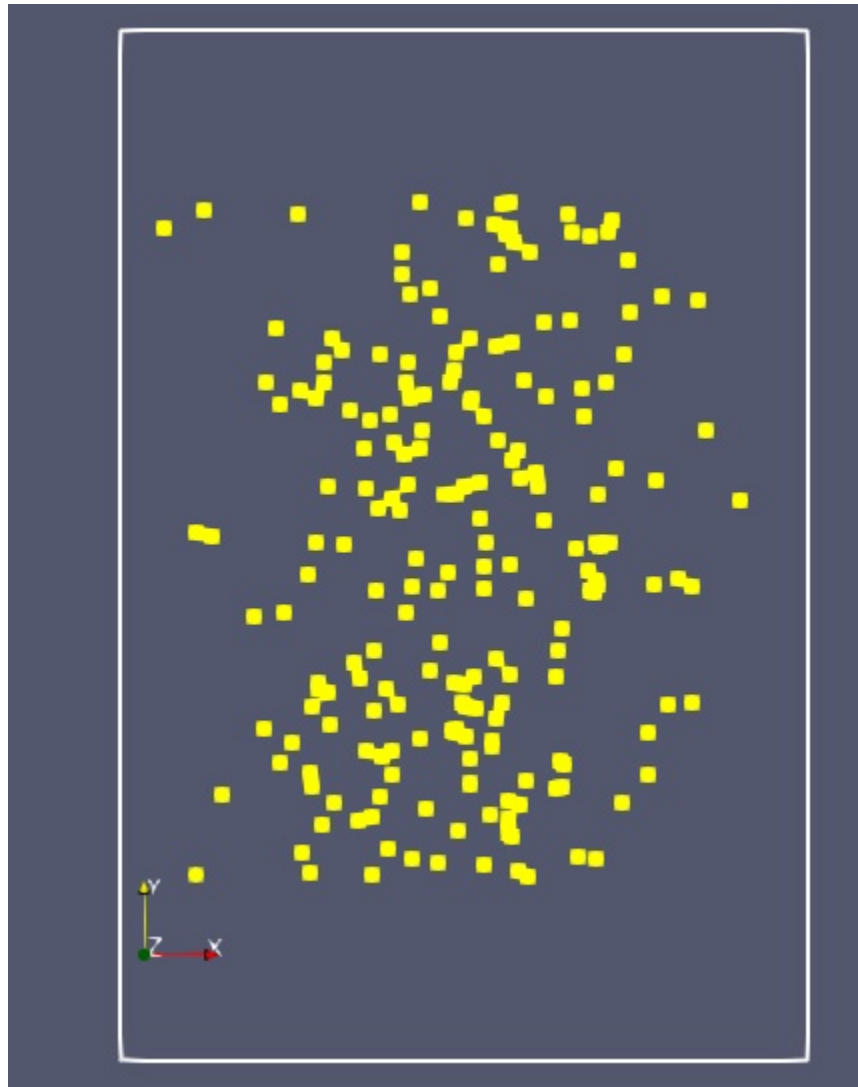


Fig. 4.67: Location of randomly distributed fake GNSS stations in `gnss_stations.txt`.

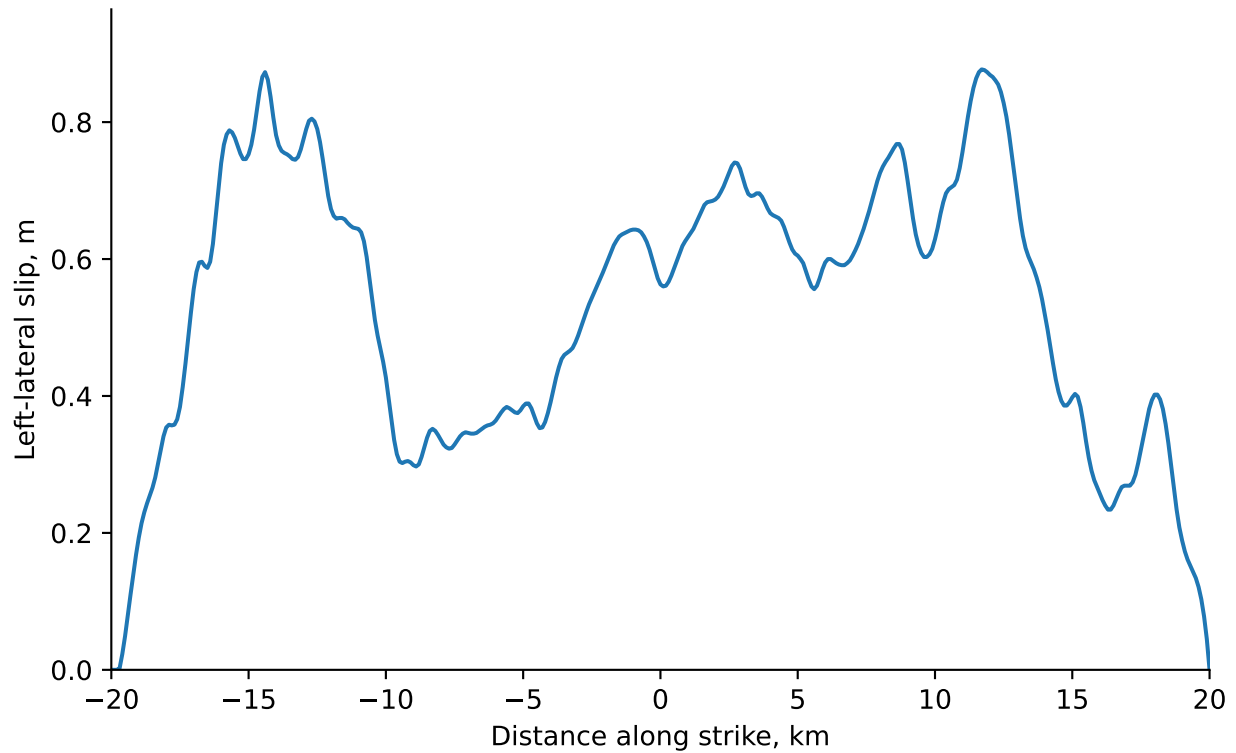


Fig. 4.68: Prescribed left-lateral slip that varies along the strike of the fault. A strike of 0 corresponds to $y=0$.

Listing 4.99: Prescribed slip parameters for Step 4. We refine the fault mesh by a factor of 8 (3 levels of refinement by a factor of 2) so that the output, which uses a basis order of 1, better captures the discretization of slip, which uses a basis order of 2.

```
[pylithapp.problem.interfaces.fault]
observers.observer.refine_levels = 3

[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.iohandler.filename = slip_variable.spatialdb
db_auxiliary_field.query_type = linear
```

Running the simulation

Listing 4.100: Run Step 4 simulation

```
$ pylith step04_varslip.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
```

(continues on next page)

(continued from previous page)

```
# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 2.567358905251e+00
      Linear solve converged due to CONVERGED_ATOL iterations 17
    1 SNES Function norm 6.879508633532e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

The beginning of the output written to the terminal matches that in our previous simulations. At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 17 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snese_atol`).

Earthquake rupture parameters

We use the `pylith_eqinfo` utility to compute rupture information, such as earthquake magnitude, seismic moment, seismic potency, and average slip. For 2D simulations, the average slip provides the most useful information. `pylith_eqinfo` is a Pyre application and you specify parameters using `cfg` files and the command line. It writes results to a Python script for use in post-processing of simulation results.

The file `eqinfoapp.cfg` holds the parameters for `pylith_eqinfo` in this example. By default, `pylith_eqinfo` extracts information for the final time step in the output file. In order to compute the seismic moment and moment magnitude, we need the shear modulus for the fault. We account for the contrast in rigidity across the fault by using the effective shear modulus from Wu and Chen [2003] and set the shear wave speed to 3.46 km/s.

Listing 4.101: Run `pylith_eqinfo` for the Step 4 simulation.

```
$ pylith_eqinfo
```

Listing 4.102: Contents of `output/step04_varslip-eqinfo.py` generated by `pylith_eqinfo`. The `all` object includes earthquake rupture information combined from all of the individual faults.

```
class RuptureStats(object):
    pass
all = RuptureStats()
all.timestamp = [ 3.155760e+07]
all.ruparea = [ 6.300870e+04]
all.potency = [ 1.306180e+04]
all.moment = [ 3.909267e+14]
all.avgslip = [ 2.073016e-01]
all.mommag = [ 3.694730e+00]
fault = RuptureStats()
```

(continues on next page)

(continued from previous page)

```

fault.timestamp = [ 3.155760e+07]
fault.ruparea = [ 6.300870e+04]
fault.potency = [ 1.306180e+04]
fault.moment = [ 3.909267e+14]
fault.avgslip = [ 2.073016e-01]
fault.mommag = [ 3.694730e+00]

```

Visualizing the results

In Fig. 4.69 we use the `pylith_viz` utility to visualize the y displacement field.

Listing 4.103: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step04_varslip-domain.h5 warp_grid --component=y
```

Step 5: Green's Functions

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.faults.FaultCohesiveKin`
- `pylith.faults.KinSrcStep`
- field split preconditioner
- Schur complement preconditioner
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.UniformDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- “Green’s functions”
- Fault slip impulses

Simulation parameters

In this example we compute static Green’s functions for fault slip and use then in Step 6 to invert for fault slip. We generated the “observations” for the slip inversion in Step 4.

We impose fault slip impulses over the central portion of the strike-slip fault ($-25 \text{ km} \leq y \leq +25 \text{ km}$), which is slightly larger than where we specified coseismic in Step 4. Fig. 4.70 summarizes the boundary conditions and fault slip. The parameters specific to this example are in `step05_greensfns.cfg`.

We use the `GreensFns` problem and specify the fault on which to impose fault slip impulses. As in Step 4, we include output at the fake GNSS stations using `OutputSolnPoints`. In the fault interfaces section we set the fault type to `FaultCohesiveImpulses` for our fault where we want to impose fault slip impulses for the Green’s functions. We

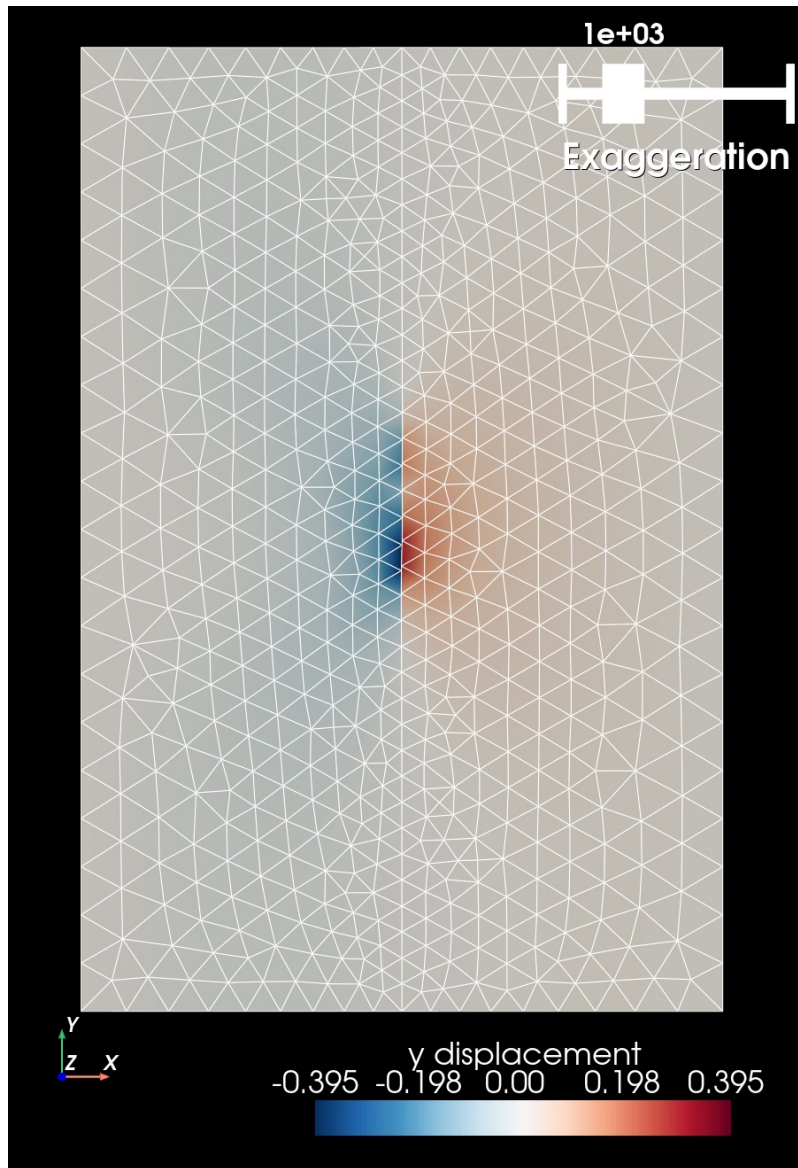


Fig. 4.69: Solution for Step 4. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000.

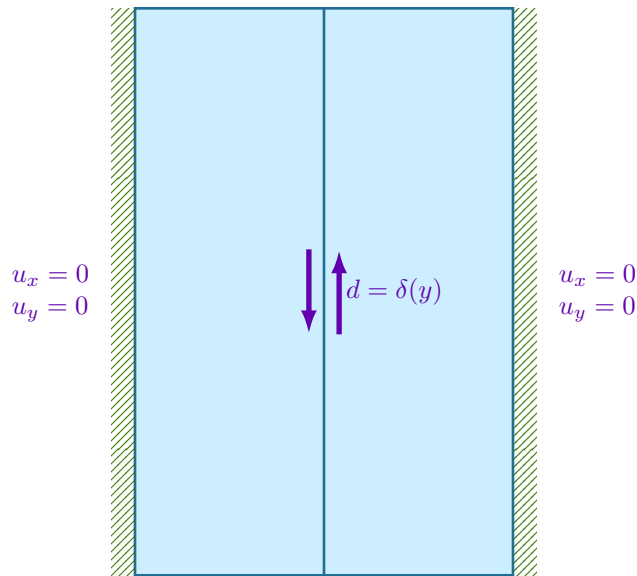


Fig. 4.70: Boundary conditions for static Green's functions. We set the x and y displacement to zero on the +x and -x boundaries and prescribe left-lateral slip impulses.

also use a spatial database to limit the section of the fault where we impose the fault slip impulses to $-25 \text{ km} \leq y \leq +25 \text{ km}$.

Important

Currently, we recommend a basis order of 1 (default) for the slip auxiliary subfield.

The basis order for the slip auxiliary subfield controls the representation of the slip field for the impulses. For a given impulse, a basis order of 1 will impose unit slip at a vertex with zero slip at all other vertices. A basis order of 0 will impose unit slip over a cell with zero slip in all other cells; however, this creates jumps in the slip at the cell boundaries and reduces the accuracy of the slip inversion for a given mesh resolution. A basis order of 2 will impose slip at vertices as well as edge degrees of freedom in the cell; regularization becomes trickier with the higher order discretization, and a basis order of 2 does not always give more accurate results for a given number of impulses.

Listing 4.104: Parameters for computing static Green's functions for fault slip impulses. We change the problem type and specify the fault on which we apply the slip impulses.

```
[pylithapp]
problem = pylith.problems.GreensFns

[pylithapp.greensfns]
label = fault
label_value = 20
```

Listing 4.105: Parameters for the slip impulses. We change the fault type and limit the impulses to the lateral slip component.

```
[pylithapp.problem.interfaces]
```

(continues on next page)

(continued from previous page)

```

fault = pylith.faults.FaultCohesiveImpulses

[pylithapp.problem.interfaces.fault]
impulse_dof = [1]

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.iohandler.filename = slip_impulses.spatialdb

auxiliary_subfields.slip.basis_order = 1

```

Running the simulation

Listing 4.106: Run Step 5 simulation

```

$ pylith step05_greensfns.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/GreensFns.cc:291:void_
↳ pylith::problems::GreensFns::solve()
-- info (application-flow)
-- Component 'greensfns.problem': Computing Green's functions.
>> src/cig/pylith/libsrc/pylith/problems/GreensFns.cc:318:void_
↳ pylith::problems::GreensFns::solve()
-- info (application-flow)
-- Component 'greensfns.problem': Computing Green's function 1 of 13.
  0 SNES Function norm 6.961541624902e-01
    Linear solve converged due to CONVERGED_ATOL iterations 9
  1 SNES Function norm 6.826069208403e-08
    Nonlinear solve converged due to CONVERGED_ITS iterations 1

# -- many lines omitted --
>> /Users/baagaard/src/cig/pylith/libsrc/pylith/problems/GreensFns.cc:318:void_
↳ pylith::problems::GreensFns::solve()
-- info (application-flow)
-- Component 'greensfns.problem': Computing Green's function 13 of 13.
  0 SNES Function norm 9.567723812071e-01
    Linear solve converged due to CONVERGED_ATOL iterations 10
  1 SNES Function norm 2.578211184664e-08
    Nonlinear solve converged due to CONVERGED_ITS iterations 1
>> /Users/baagaard/software/unix/micromamba/envs/pylith-debug/lib/python3.12/site-
↳ packages/pylith/problems/Problem.py:222:finalize
-- info (application-flow)
-- Finalizing problem.
WARNING! There are options you set that were not used!
WARNING! could be spelling mistake, etc!

```

(continues on next page)

(continued from previous page)

```

There are 4 unused database options. They are:
Option left: name:-ts_error_if_step_fails (no value) source: code
Option left: name:-ts_exact_final_time value: matchstep source: code
Option left: name:-ts_monitor (no value) source: code
Option left: name:-ts_type value: beuler source: code

```

The beginning of the output written to the terminal matches that in our previous simulations. The second half of the output written to the terminal resembles the output from time-dependent problems, but with the time step information replaced by the impulse information. We get warnings about unused PETSc options because we do not use time stepping.

Visualizing the results

In Fig. 4.71 we use the `viz/plot_slip_impulses.py` Python script to visualize the slip impulses. In Fig. 4.72 we use the `pylith_viz` utility to visualize the *y* displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement slip impulses (shown as different time stamps).

Listing 4.107: Plot the slip impulses and visualize PyLith output using `pylith_viz`.

```

viz/plot_slip_impulses.py
pylith_viz --filename=output/step05_greensfns-domain.h5 warp_grid --component=y

```

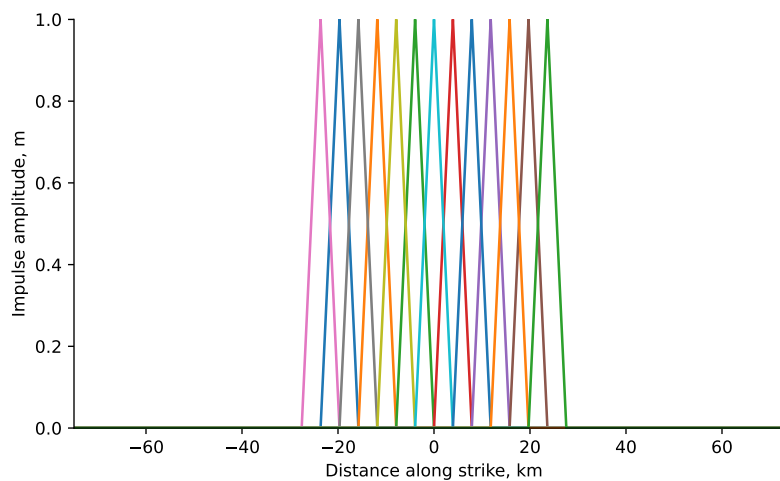


Fig. 4.71: Slip impulses for the Green's functions in Step 5.

Step 6: Least Squares Fault Slip Inversion

In this example we do a simple static slip inversion using least squares. We treat the displacements at the fake GNSS stations in Step 4 as the “observations” and use the Green's functions from Step 5 to invert for the fault slip that we prescribed in Step 4.

We use simple generalized inversion method with penalties to minimize the seismic moment. The Python script `invert_slip.py` will load the observations from Step 4 and Green's functions and responses from Step 5 and invert for the slip.

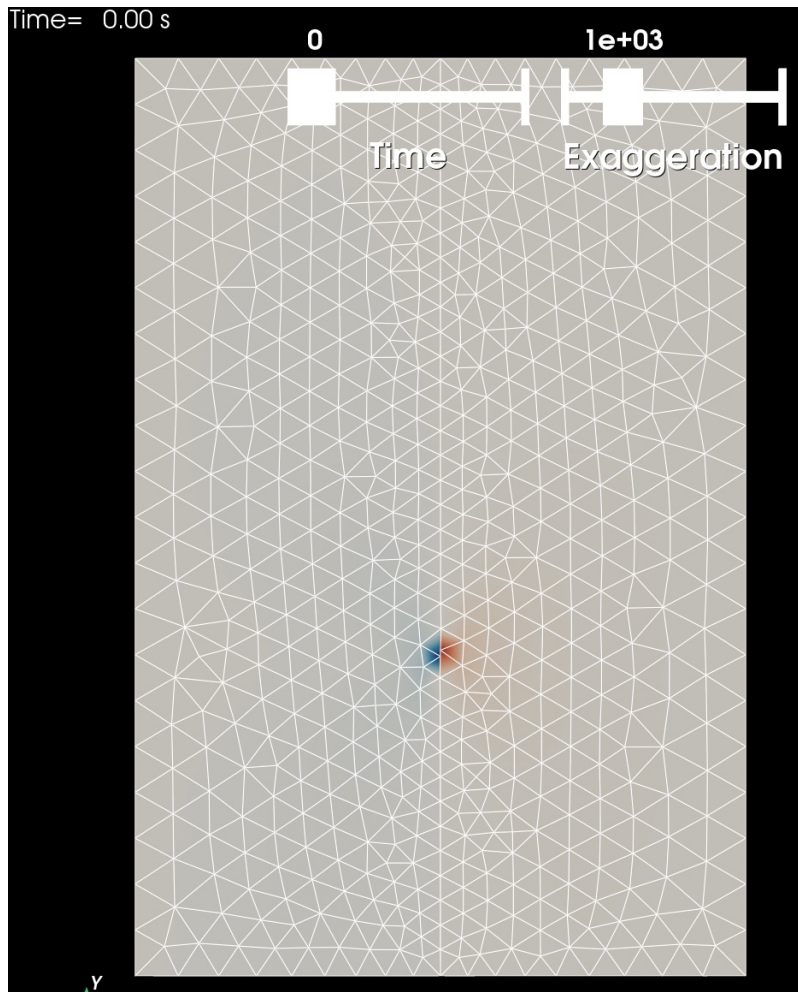


Fig. 4.72: Solution for Step 4. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The time value corresponds to the zero-based index of the slip impulses.

Listing 4.108: Run the fault slip inversion code.

```
$ ./invert_slip.py
# Show command line options for the inversion code
$ ./invert_slip.py --help
```

By default, the inversion code will write the results of the inversion to `output/step06_inversion-results.txt`.

Plotting the results

If you are using the PyLith binary, which includes the `matplotlib` Python module, or have it installed, then you can plot the results of the simulation using the `viz/plot_inversion_results.py` Python script.

Listing 4.109: Plot the inversion results using the `matplotlib` Python module.

```
$ viz/plot_inversion_results.py
```

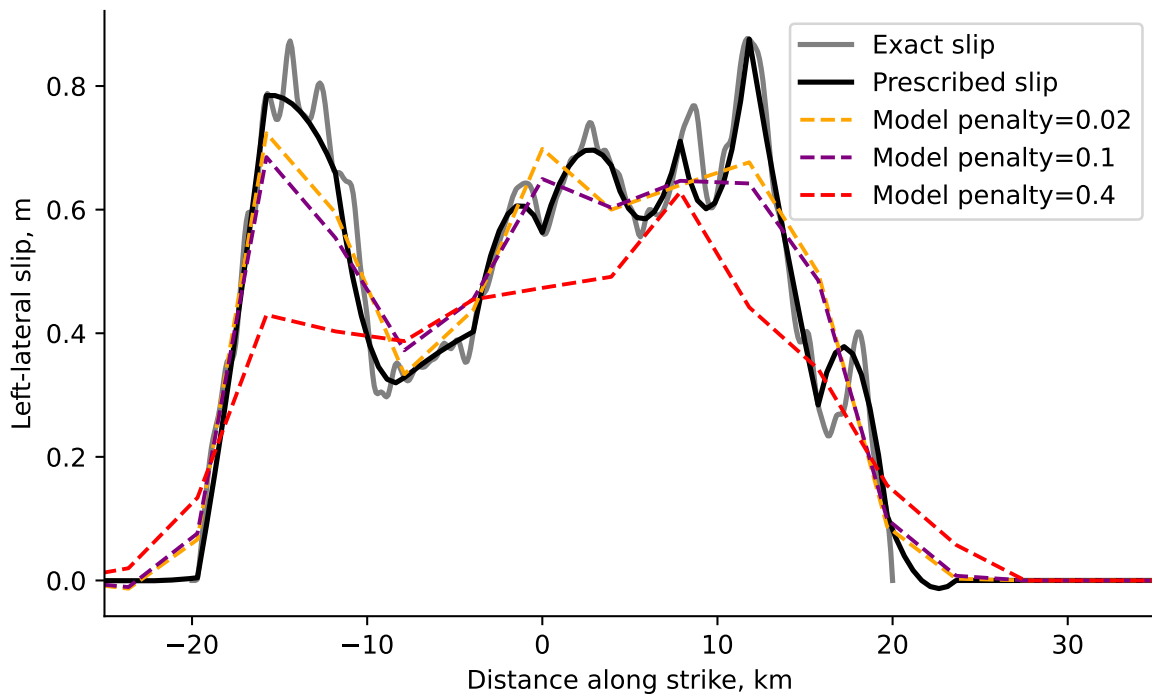


Fig. 4.73: Results of slip inversion in Step 6. The thick black line shows the prescribed slip in Step 4, and the thick gray line shows the raw (exact) slip. The thin colored lines show the slip from the inversion with different penalty factors.

`::{tip}` You can pass `--no-gui` as a command line argument to the plotting script to turn off displaying the plot window. This is useful if you do not have a `matplotlib` GUI backend.

Step 7: Bayesian Fault Slip Inversion

Danger

This examples requires the CATMIP Bayesian inversion framework which is not yet publicly available.

In this example we perform the same inversion as in Step 6, but replace the least squares inversion with the CATMIP Bayesian inversion framework. We demonstrate inverting for fault slip using both the original CATMIP algorithm (Step 7a) and the crossfade CATMIP algorithm (Step 7b).

We assume a uniform distribution with positive slip values but use a logistic (sigmoid) distribution to compute the probability. This helps prevent back slip.

Inversion using original CATMIP algorithm

The `catmip_pylith_staticslip` executable uses the original CATMIP algorithm and must be run using at least 2 processes (1 manager process and at least 1 worker process). The model parameters for the inversion are specified in `step07a_catmip.in` (general CATMIP parameters) and `catmip_parameters.txt` (specific to the PyLith static slip model). Currently, the `catmip_parameters.txt` filename is hardwired in the PyLith static slip model code.

Listing 4.110: General CATMIP parameters in `step07a_catmip.in`.

```
# Use 100 samples and Markov chains with a length of 50
N                100
Nsteps           50

# Use the current directory for user input and write output files as output/step07a-
→catmip-
data_directory   .
output_directory ./output
output_prefix    step07a_catmip-
```

Listing 4.111: Parameters in `catmip_parameters.txt` for static slip inversion using the original CATMIP algorithm.

```
# Sample parameter file for pylith_catmip model.

# Data files
filename_observations = output/step04_varslip-gnss_stations.h5
filename_greens_fns   = output/step05_greensfns-gnss_stations.h5

# Model parameters
rake_parallel_prior = logistic
rake_parallel_prior_k = 25.0
rake_parallel_prior_min_sample_value = 0.01
rake_parallel_prior_max_sample_value = 2.0

rake_perpendicular_prior = gaussian

# For a 2D problem we only have 1 component of slip.
num_impulse_components = 1
```

Listing 4.112: Run the fault slip inversion using the original CATMIP algorithm.

```
mpiexec -n 2 catmip_pylith_staticslip step07a_catmip.in
```

By default, the CATMIP code will write the results of the inversion to `output/step07a_catmip-*`. The `.gsl` files are raw binary files that can be read using Python.

Step 7a: Plotting the results

If you are using the PyLith binary, which includes the `matplotlib` Python module, or have it installed, then you can plot the results of the simulation using the `viz/plot_catmip_results.py` Python script.

Listing 4.113: Plot the CATMIP inversion results using the `matplotlib` Python module.

```
viz/plot_catmip_results.py --catmip-theta=output/step07a_catmip-theta20.bin
```

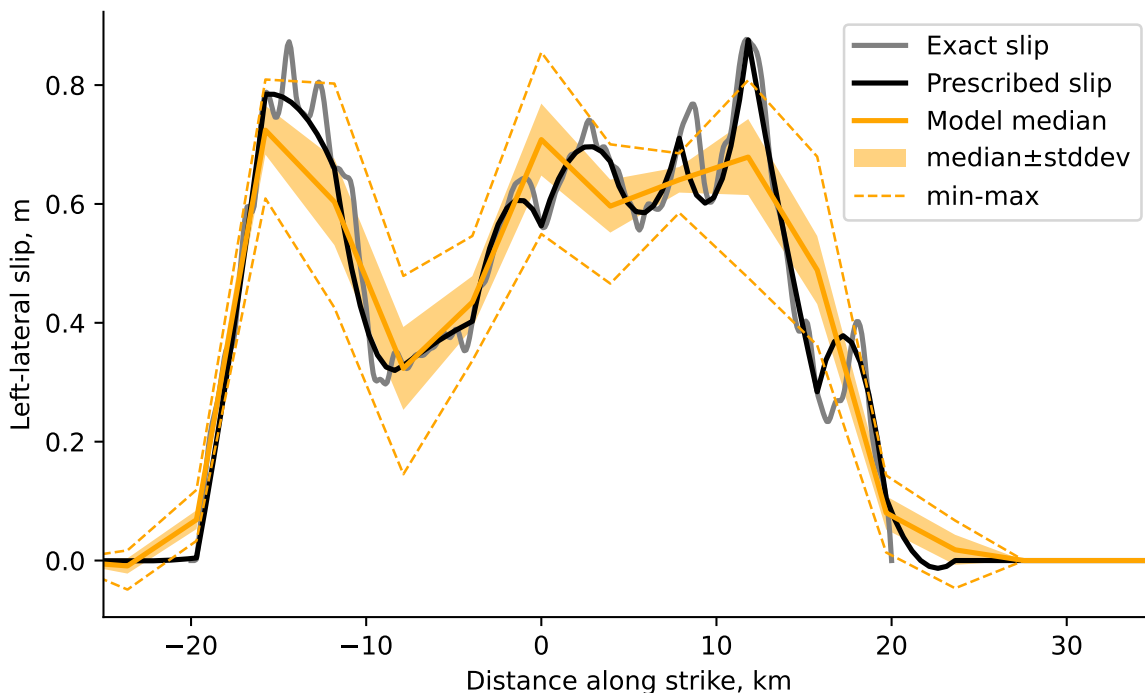


Fig. 4.74: Results of slip inversion in Step 7a. The thick black line shows the prescribed slip in Step 4, and the thick gray line shows the raw (exact) slip. The solid orange line shows the median slip, the shaded orange regions shows the median plus and minus one standard deviation, and the dashed lines show the minimum and maximum values. The median values are almost identical to the results with minimal smoothing in Step 6.

tip You can pass `--no-gui` as a command line argument to the plotting script turn off displaying the plot window. This is useful if you do not have a `matplotlib` GUI backend.

Inversion using CF-CATMIP algorithm

The `cfcattmip_pylith_staticslip` executable uses the CF-CATMIP algorithm and must be run using at least 2 processes (1 manager process and at least 1 worker process). The model parameters for the inversion are specified in `step07b_cfcattmip.in` (general CATMIP parameters) and `cfcattmip_parameters.txt` (specific to the PyLith static slip model). Currently, the `cfcattmip_parameters.txt` filename is hardwired in the PyLith static slip model code. We use the same parameters for the prior distribution of model parameters. For the conjugate posterior distribution, we use a Gaussian distribution with broad enough parameters to sample the parameter space; the CF-CATMIP algorithm is not particularly sensitive to the median and standard deviation of the conjugate posterior Gaussian distribution.

Listing 4.114: General CATMIP parameters in `step07b_catmip.in`.

```
# Use 100 samples and Markov chains with a length of 50
N                100
Nsteps           50

# Use the current directory for user input and write output files as output/step07b-
↪cfcattmip-
data_directory   .
output_directory ./output
output_prefix    step07b_cfcattmip-
```

Listing 4.115: Parameters in `cfcattmip_parameters.txt` for static slip inversion using the CF-CATMIP algorithm.

```
# Sample parameter file for pylith_catmip model.

# Data files
filename_observations = output/step04_varslip-gnss_stations.h5
filename_greens_fns   = output/step05_greensfns-gnss_stations.h5

# Model parameters
rake_parallel_prior = logistic
rake_parallel_prior_k = 25.0
rake_parallel_prior_min_sample_value = 0.01
rake_parallel_prior_max_sample_value = 2.0

rake_perpendicular_prior = gaussian

conjugate_posterior = gaussian
conjugate_posterior_median = 0.5
conjugate_posterior_stddev = 0.2

num_impulse_components = 1
```

Listing 4.116: Run the fault slip inversion using the CF-CATMIP algorithm.

```
mpirun -n 2 cfcattmip_pylith_staticslip step07b_cfcattmip.in
```

By default, the CATMIP code will write the results of the inversion to `output/step07b_catmip-*`. The `.gsl` files are raw binary files that can be read using Python.

Step 7b: Plotting the results

If you are using the PyLith binary, which includes the `matplotlib` Python module, or have it installed, then you can plot the results of the simulation using the `viz/plot_catmip_results.py` Python script.

Listing 4.117: Plot the CATMIP inversion results using the `matplotlib` Python module.

```
viz/plot_catmip_results.py --catmip-theta=output/step07b_cfcattmip-theta1.bin
```

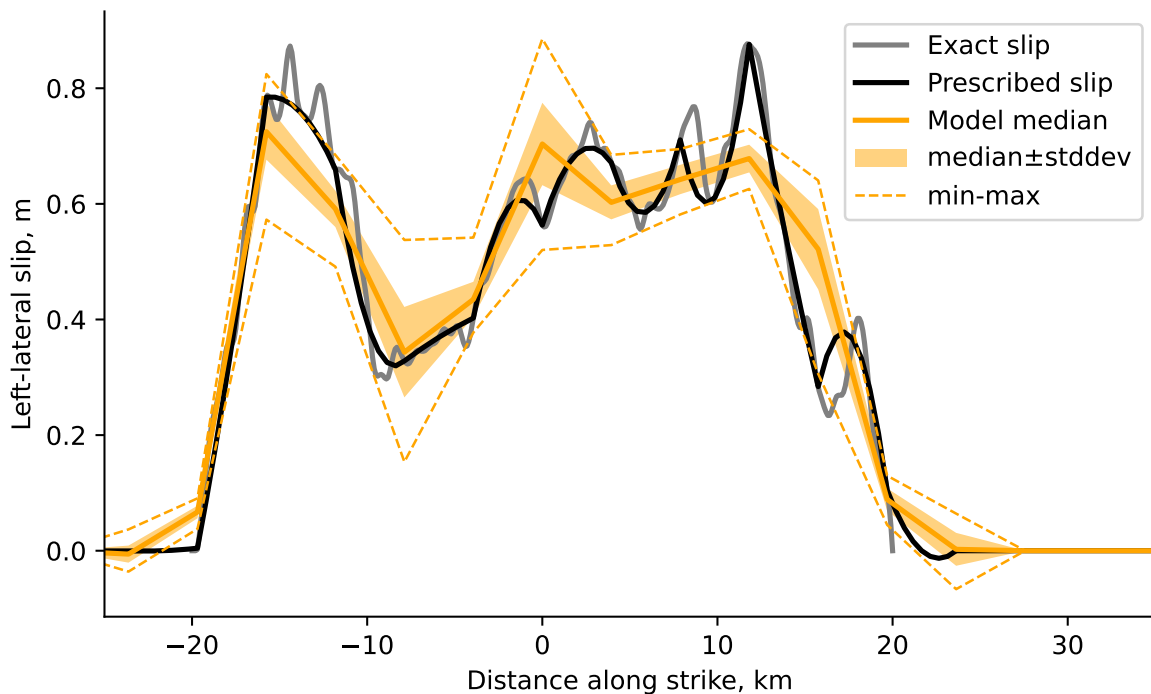


Fig. 4.75: Results of slip inversion in Step 7b. The thick black line shows the prescribed slip in Step 4, and the thick gray line shows the raw (exact) slip. The solid orange line shows the median slip, the shaded orange regions shows the median plus and minus one standard deviation, and the dashed lines show the minimum and maximum values. The results are almost identical to those in Step 7a.

Suggested Exercises

1. Increase the contrast in the shear modulus across the fault. How does this affect the solution?
2. Create `.cfg` files to run Steps 2 and 3 with a Cubit mesh.

3. Use uniform refinement with 2 and 3 levels of refinement. How does this affect the resulting stress and strain field?
4. Change Step 3 to include multiple ruptures at different time intervals, while still accounting for the accumulated slip deficit.
5. Change the distribution and number of fake GNSS stations in Steps 4 and 5. How does this affect the inversion results?
6. Change the distribution of slip in Step 4. How does this affect the inversion results?
7. Create Cubit Journal files that build up the geometry from points and curves, analogous to the Gmsh Python script.
8. Create a Gmsh Python script that generates the geometry from a cross-section of a 3D block, analogous to the provided Cubit Journal files.

Vertical Cross-Section of a Reverse Fault with Splay (2D)

The files are in the directory `examples/reverse-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

generate_gmsh.py

Python script to generate mesh using Gmsh.

generate_cubit.py

Python script to generate mesh using Cubit.

***.msh**

Gmsh mesh files generated by Gmsh.

***.exo**

Exodus II mesh files generated by Cubit.

***.spatialdb**

Spatial database files associated with the spatial databases.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This suite of examples demonstrates use of a number of features for a vertical cross section of a reverse fault accompanied by a splay fault (Fig. 4.76) with elastic and viscoelastic material properties. We separately consider loading from gravitational body forces, surface tractions, and coseismic slip. We build on the previous examples and add complexity through a series of steps:

Step 1

Gravitational body forces and linear isotropic elasticity.

Step 2

Gravitational body forces and linear isotropic elasticity with a reference stress state.

Step 3

Gravitational body forces and linear isotropic incompressible elasticity.

Step 4

Surface tractions and linear isotropic linear elasticity.

Step 5

Earthquake rupture on one fault and linear isotropic linear elasticity.

Step 6

Earthquake rupture on two faults and linear isotropic linear elasticity.

Step 7

Earthquake rupture on two faults and linear isotropic Maxwell viscoelastic rheology.

Step 8

Earthquake rupture on two faults and linear isotropic powerlaw viscoelastic rheology.

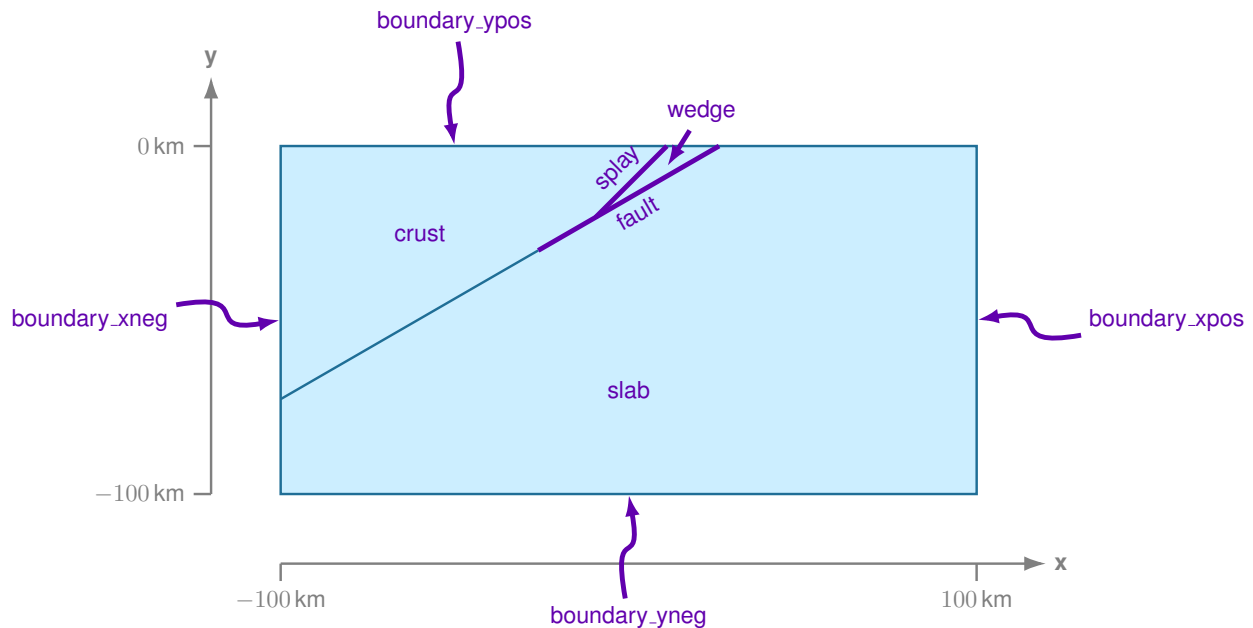


Fig. 4.76: Diagram of geometry for domain with reverse and splay faults and three materials (crust, slab, and wedge). The domain extends from -100 km to +100 km in the x direction and from -100 km to 0 in the y direction. We refer to the domain boundaries using the names shown in the diagram.

Important

We describe how to generate the finite-element mesh using both Gmsh and Cubit. The mesh files for both methods are included; however, we only use the Gmsh file in the PyLith parameter files. See [examples/strikeslip-2d/step01_slip_cubit.cfg](#) for a description of how to modify the parameter files to switch from using mesh files from Gmsh to mesh files from Cubit.

Example Workflow

Gmsh Mesh

Geometry

We construct the geometry by first creating points, then connecting the points into curves, and finally the curves into surfaces. Fig. 4.77 shows the geometry and variables names of the vertices and curves.

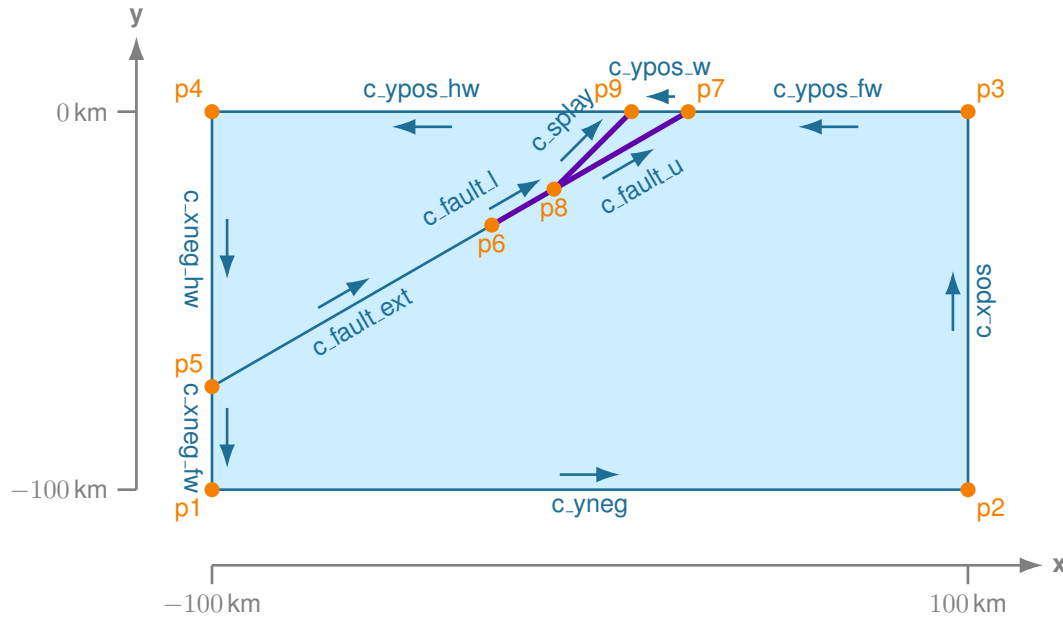


Fig. 4.77: Geometry created in Gmsh for generating the finite-element mesh. We construct curves from points (p1, ..., p9) and surfaces from the curves (for example, c_yneg). The arrows indicate the direction (orientation) of the curves.

Important

Each curve in Gmsh has a direction (orientation). The direction is from the starting point to the ending point. When connecting curves into surfaces, you must connect the curves in a consistent direction. We connect the curves in a counter-clockwise direction. To reverse the direction of a curve, use the negative tag.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. The script is structured identically to the one we used in `examples/strikeslip-2d`. We create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class. The main difference is that the geometry is slightly more complex, and we calculate the location of the points on the fault using trigonometry.

We use the Gmsh `MeshSize` options to define a discretization size that grows slowly at a geometric rate with distance from the main fault.

Listing 4.118: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with triangular cells and save it to `mesh_tri.msh` (default filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui

# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui
```

(continues on next page)

(continued from previous page)

```
# Show available command line arguments.  
$ ./generate_gmsh.py --help
```

By default the Python script will generate a finite-element mesh with triangular cells and save it to the file `mesh_tri.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.119: View the Gmsh mesh file `mesh_tri.msh` using Gmsh.

```
gmsh -open mesh_tri.msh
```

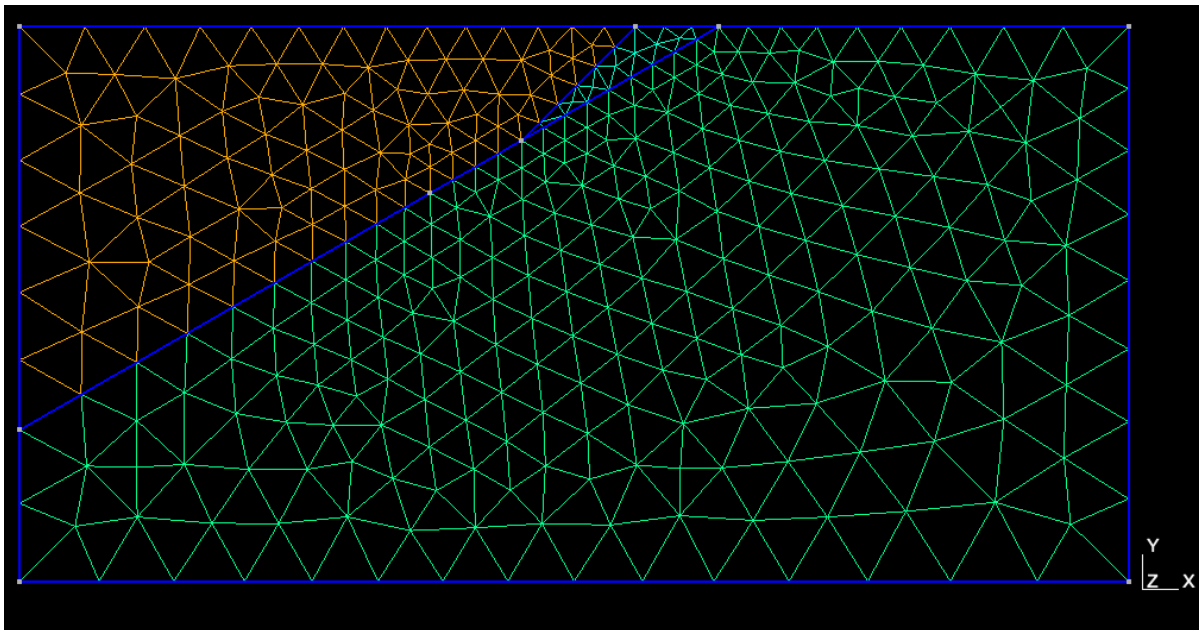


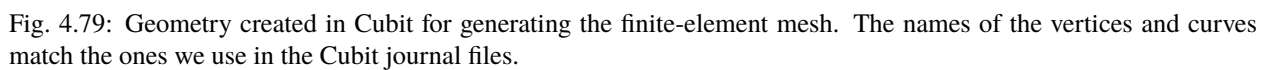
Fig. 4.78: Finite-element mesh with triangular cells generated by Gmsh.

Cubit Mesh

Geometry

We construct the geometry by taking a vertical cross-section of a 3D block. Alternatively, we could have constructed the geometry by building it up from points and curves like we did with Gmsh. [Fig. 4.79](#) shows the geometry and variables names of the vertices and curves.

Meshing using Python Script



New in v4.1.0.

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

We create a brick, extracting a midsurface from it, and then splitting the remaining surface with an extended fault and a splay surface. We assign names to the surfaces, curves, and important vertices that we use when we specify the mesh sizing information and defining blocks and nodesets.

We use IDless journaling in Cubit. This allows us to reference objects in a manner that should be independent of the version of Cubit that is being used. In the Python script the original command used is typically commented out, and the following command is the equivalent IDless command.

In addition to providing sidesets for the fault and splay, it is also important to provide nodesets defining the buried

edges of these two surfaces. In 2D this consists of a single vertex for each surface. This information is required by PyLith to form the corresponding cohesive cells defining fault surfaces.

Note

We use the skeleton sizing function to gradually increase the cell size with distance from the fault. This is much simpler than applying the bias sizing function to the curves and surfaces. We set the minimum size equal to the cell size on the fault and the maximum gradient equal to the “bias”.

Once you have run the Python script to construct the geometry and generate the mesh, you will have a corresponding Exodus-II file (`mesh_tri.exo` or `mesh_quad.exo`).

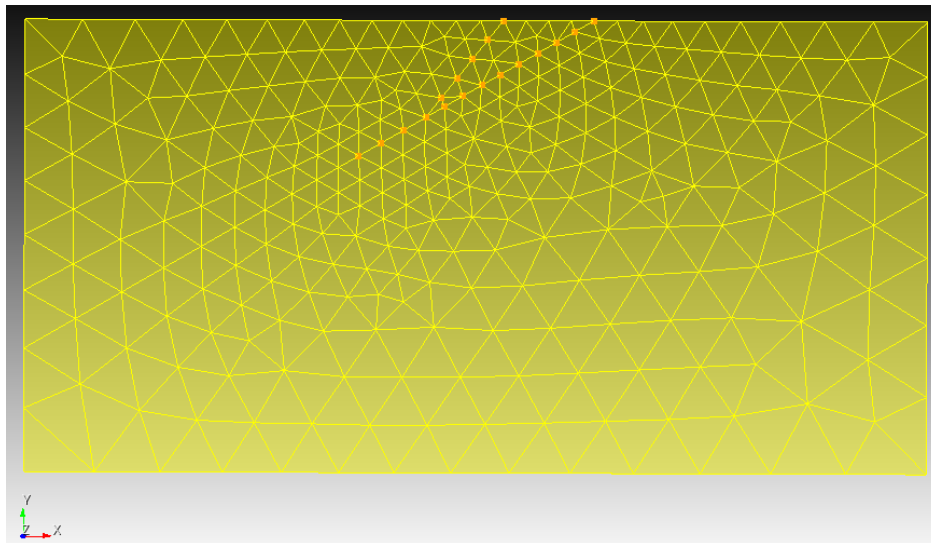


Fig. 4.80: Finite-element mesh with triangular cells generated by Cubit.

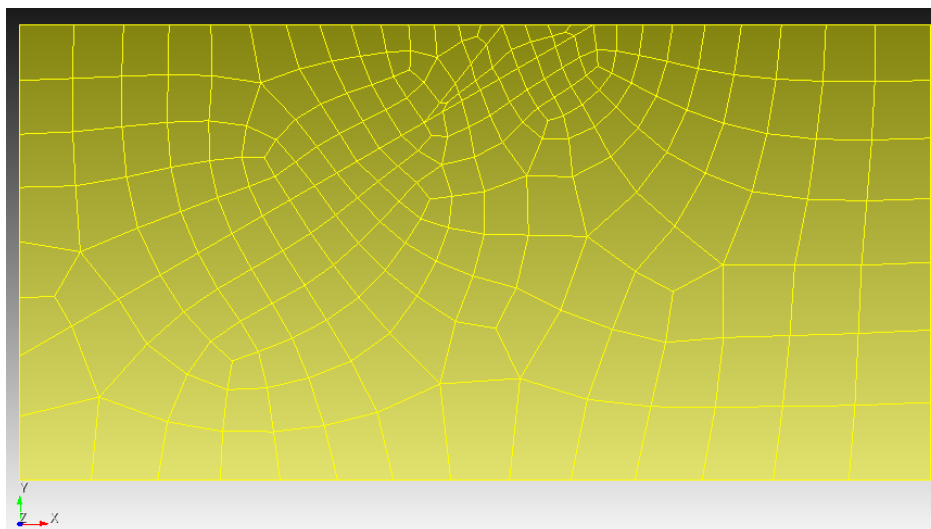


Fig. 4.81: Finite-element mesh with quadrilateral cells generated by Cubit.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`, which contains numerous comments, so we only summarize the parameters here.

We output the solution over the domain and the ground surface (+y boundary).

Listing 4.120: Parameters for output of the solution over the domain and ground surface (+y boundary).

```
[pylithapp.problem]
solution_observers = [domain, boundary]
solution_observers.boundary = pylith.meshio.OutputSolnBoundary

[pylithapp.problem.solution_observers.boundary]
label = boundary_ypos
label_value = 13
```

These static and quasi-static simulations solve the elasticity equation. We use the same material properties for several simulations in this directory, so we specify them in `pylithapp.cfg` to avoid repeating the information in the file with parameters for each simulation. We use a SimpleDB spatial database so that we can simply use a different spatial database file when we change the bulk rheology.

Listing 4.121: Material parameters for isotropic, linear elasticity. We only show the details for the slab material.

```
[pylithapp.problem]
materials = [slab, crust, wedge]

[pylithapp.problem.materials.slab]
label_value = 1

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Elastic properties for slab
db_auxiliary_field.iohandler.filename = mat_elastic.spatialdb

auxiliary_subfields.density.basis_order = 0
auxiliary_subfields.gravitational_acceleration.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0
```

Similarly, for all of the simulations in this directory we use Dirichlet (displacement) boundary conditions on the +x, -x, and -y boundaries that constrain the displacement component perpendicular to the fault.

Listing 4.122: Dirichlet boundary condition parameters common to all simulations in this directory. We only show the details for the +x boundary.

```
[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg]
bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.DirichletTimeDependent
```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.bc.bc_xpos]
label = boundary_xpos
label_value = 11
constrained_dof = [0]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC +x edge

auxiliary_subfields.initial_amplitude.basis_order = 0
```

Step 1: Gravitational Body Forces

Features

- Triangular cells
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.ZeroDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.GravityField

This example involves a static simulation that solves for the deformation from loading by gravitational body forces. Fig. 4.82 shows the boundary conditions on the domain.

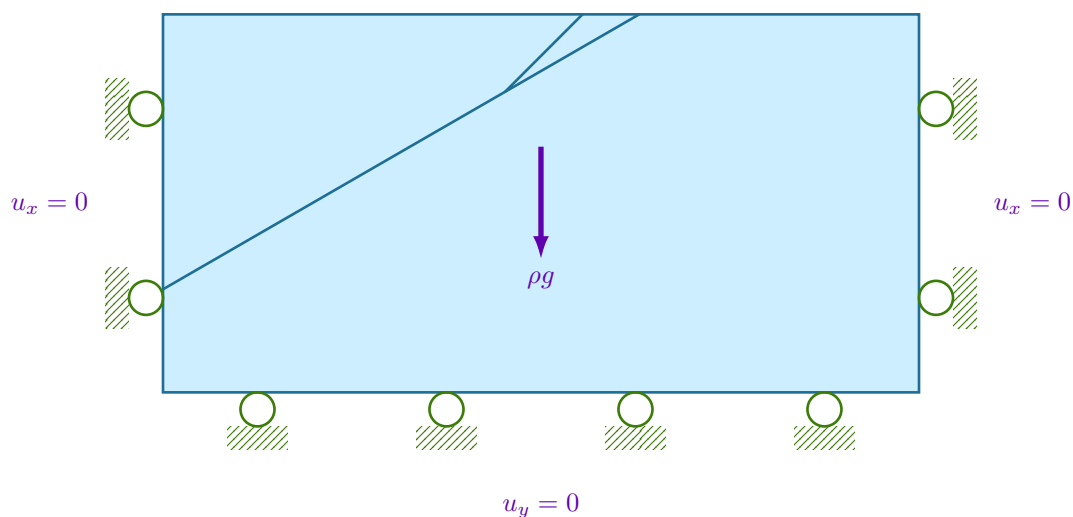


Fig. 4.82: We apply roller boundary conditions on the lateral sides and bottom of the domain.

We solve the static elasticity equation with gravitational body forces,

$$\vec{s} = \left(\vec{u} \right)^T \quad (4.479)$$

$$\rho(\vec{x})\vec{g} + \nabla \cdot \boldsymbol{\sigma}(\vec{u}) = \vec{0}. \quad (4.480)$$

Step 1a: Coarse Mesh

Note

New in v4.1.0.

We start with a coarse resolution mesh that captures the geometry and increase the resolution of the simulation by using uniform refinement or increasing the basis order of the solution fields.

Simulation parameters

The parameters specific to this example are in `step01a_gravity.cfg`. In 2D with gravitational body forces acting in the -y direction, we need to set the direction.

Listing 4.123: Parameters for gravitational body forces for Step 1a.

```
[pylithapp.problem]
gravity_field = spatialdata.spatialdb.GravityField
gravity_field.gravity_dir = [0.0, -1.0, 0.0]
```

Running the simulation

Listing 4.124: Run Step 1a simulation

```
$ pylith step01a_gravity.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
```

(continues on next page)

(continued from previous page)

```

    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void
↳ pylith::meshio::MeshIOPetsc::_read()
-- info (application-flow)
-- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-100000, 100000)
    (-100000, 0)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
↳ pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
↳ printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 5000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↳ pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 3.007829881319e+03
      Linear solve converged due to CONVERGED_ATOL iterations 11
    1 SNES Function norm 3.524316172860e-07
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the MeshIOPetsc reader and that it found the domain to extend from -100,000 m to +100,000 m in the x direction and from -100,000 m to 0 in the y direction. The output also shows the scales for nondimensionalization and the PETSc options selected by PyLith.

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 11 iterations and the norm of the residual met the relative convergence tolerance (`ksp_rtol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`).

Visualizing the results

The `output` directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (`.h5`) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (`.xmf`) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In Fig. 4.83 and Fig. 4.84 we use the `pylith_viz` utility to visualize the simulation results. Because we apply the gravitational body forces to an undeformed, stress-free domain, the vertical deformation is about 2 kilometers.

Listing 4.125: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step01a_gravity-domain.h5 warp_grid --exaggeration=5
pylith_viz --filenames=output/step01a_gravity-crust.h5,output/step01a_gravity-slab.h5,
↪output/step01a_gravity-wedge.h5 warp_grid --field=cauchy_stress --component=xy --
↪exaggeration=5
```

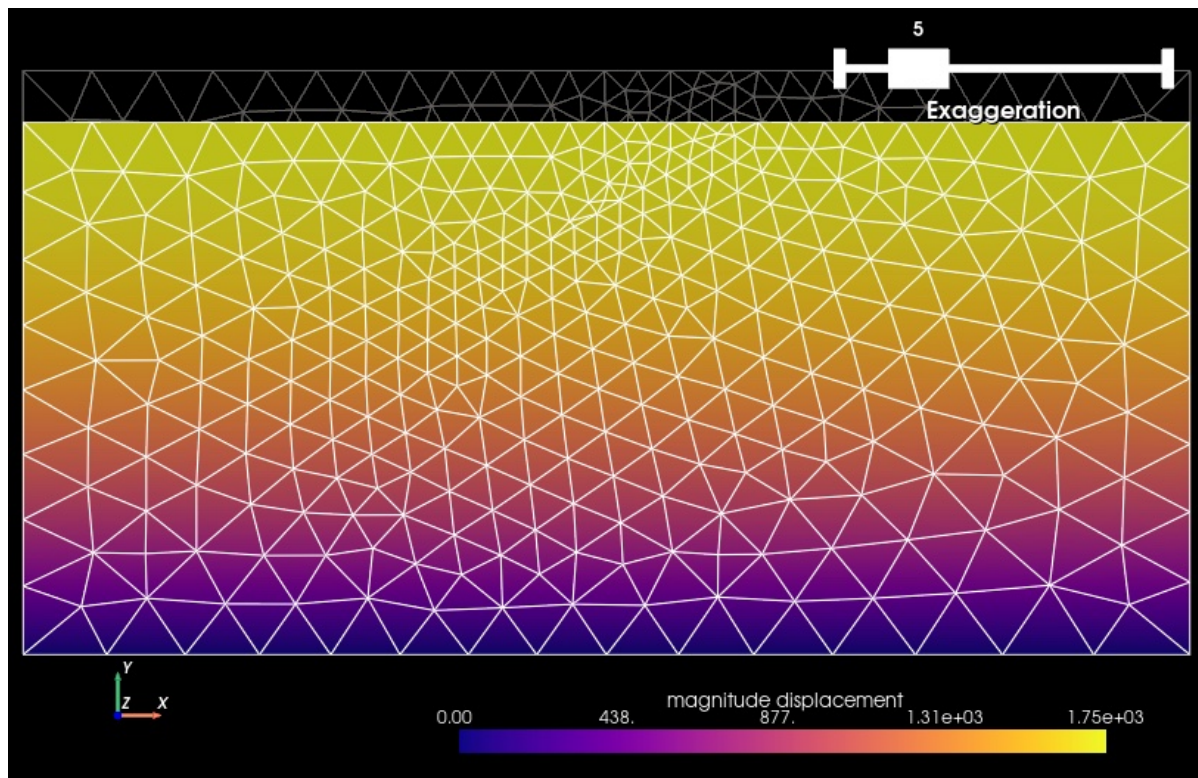


Fig. 4.83: Solution for Step 1a. The colors of the shaded surface indicate the magnitude of the displacement, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe.

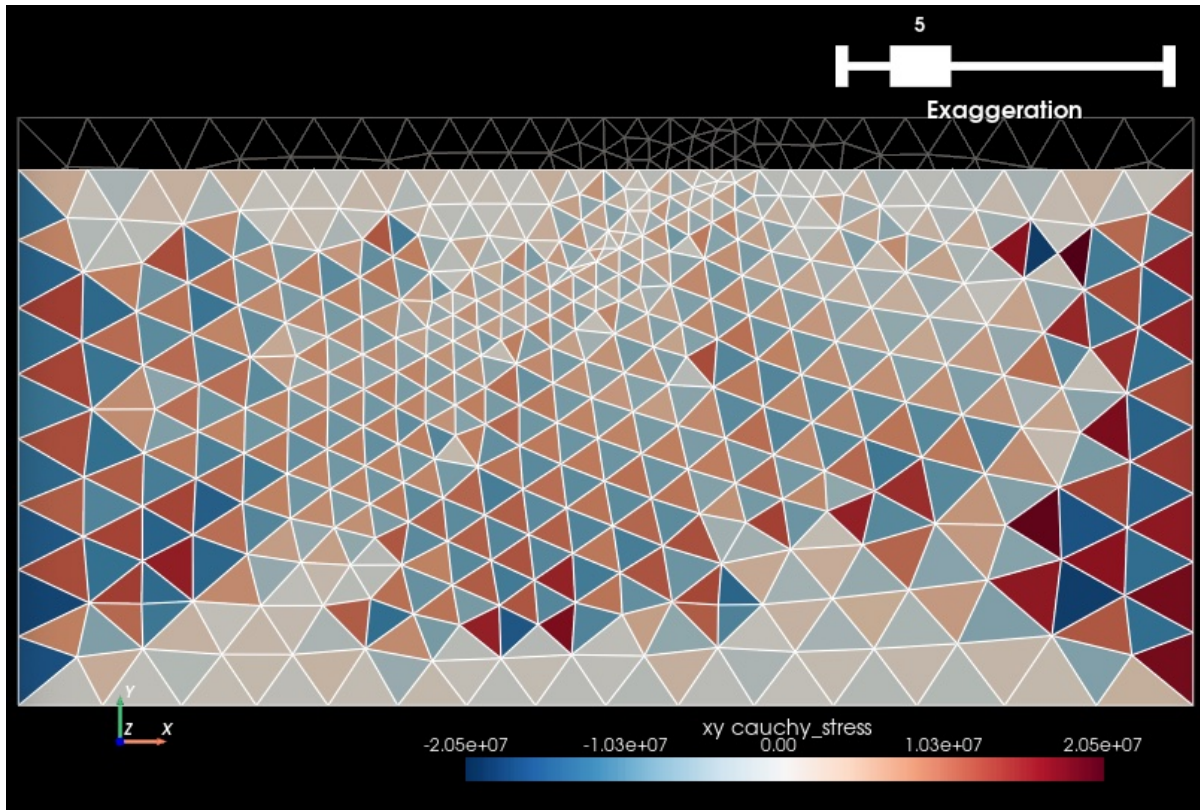


Fig. 4.84: Cauchy stress tensor component xy for Step 1a. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe. We expect the shear stress (xy component) to be zero. The checkerboard pattern shows that it is close to zero on average, but there are substantial variations.

Step 1b: Refined Mesh

To reduce the numerical errors in the shear stress, we refine the mesh to decrease the discretization size by a factor of 2. The parameters specific to this example are in `step01b_gravity.cfg`.

Simulation parameters

We refine the mesh by setting the `refiner` component of the `mesh_generator` to `pylith.topology.RefineUniform`. The default refinement is one level of refinement, which decreases the discretization size by a factor of two. Selecting two levels of refinement would decrease the discretization size by a factor of four.

Listing 4.126: Parameters for refining the mesh in Step 1b.

```
[pylithapp.mesh_generator]
refiner = pylith.topology.RefineUniform
```

Running the simulation

Listing 4.127: Run Step 1b simulation

```
$ pylith step01b_gravity.cfg
# The output should look very similar to that in Step 1a.
```

Visualizing the results

In Fig. 4.85 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.128: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step01b_gravity-domain.h5 warp_grid --exaggeration=5
pylith_viz --filenames=output/step01b_gravity-crust.h5,output/step01b_gravity-slab.h5,
↪output/step01b_gravity-wedge.h5 warp_grid --field=cauchy_stress --component=xy --
↪exaggeration=5
```

Step 1c: Higher Order Discretization

Rather than refining the mesh further, we use a basis order of 2 for the displacement subfield in the solution. Applying gravitational body forces without a reference stress state will result in stresses and strains that increase linearly with depth. Consequently, the displacement field will increase with depth squared, which we can model very accurately with a basis order 2. The parameters specific to this example are in `step01c_gravity.cfg`.

Simulation parameters

Most visualization programs do not support a solution specified with coefficients for second order basis functions. The workaround is to project the solution to a basis order of 1 when we write the solution field. PyLith does this automatically. For fields that are uniform, we use a basis order of 0 to minimize memory use.

Listing 4.129: Parameters for using higher order discretization in Step 1c.

```
[pylithapp.problem]
defaults.quadrature_order = 2
```

(continues on next page)

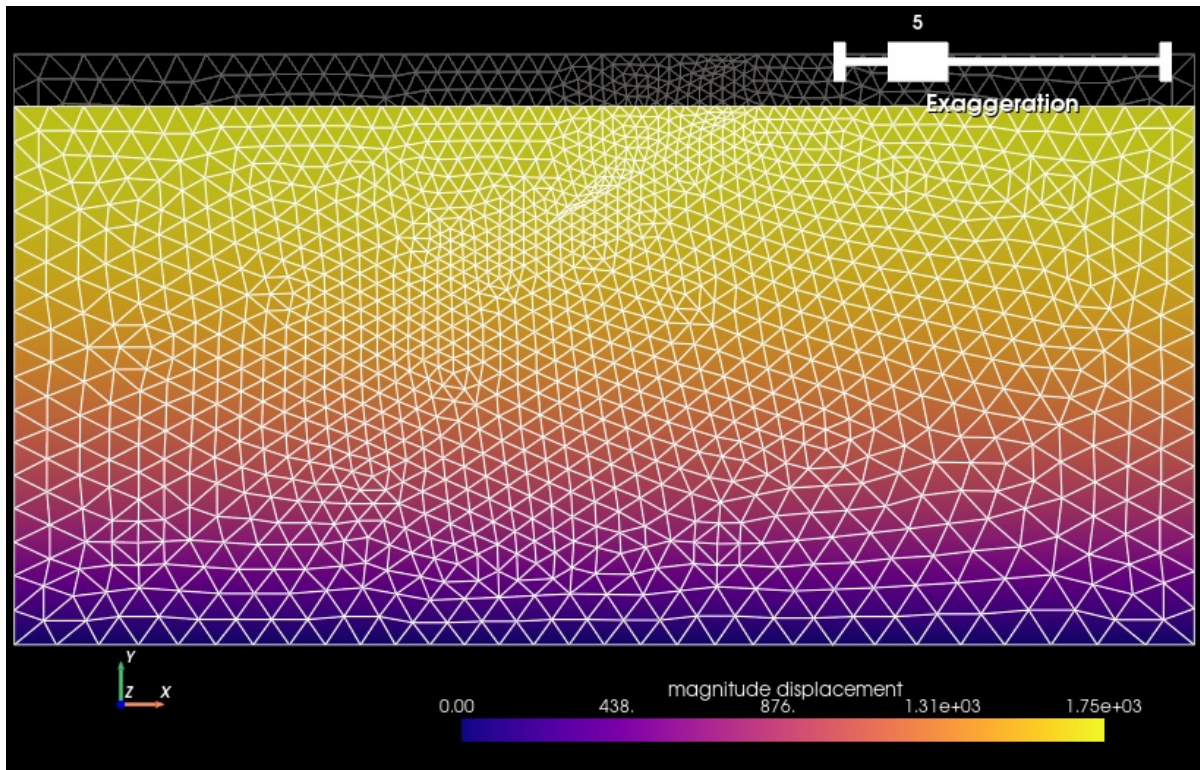


Fig. 4.85: Solution for Step 1b. The colors of the shaded surface indicate the magnitude of the displacement, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe.

(continued from previous page)

```
[pylithapp.problem.solution.subfields.displacement]
basis_order = 2

[pylithapp.problem.materials.slabs]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.crust]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.wedge]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1
```

Running the simulation

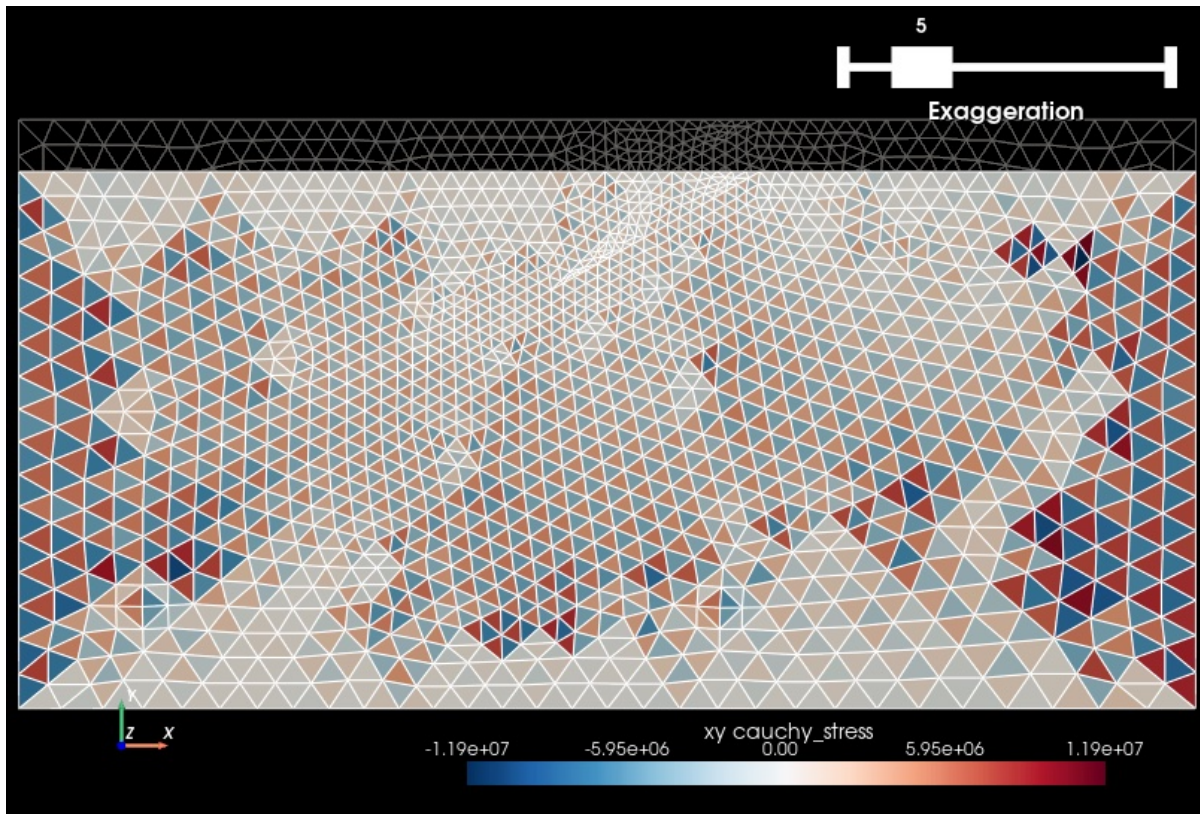


Fig. 4.86: Cauchy stress tensor component xy for Step 1b. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe. Refining the mesh reduced the maximum shear stress by about a factor of 2.

Listing 4.130: Run Step 1c simulation

```
$ pylith step01c_gravity.cfg
# The output should look very similar to that in Steps 1a and 1b.
```

Visualizing the results

In Fig. 4.87 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.131: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step01c_gravity-domain.h5 warp_grid --exaggeration=5
pylith_viz --filenames=output/step01c_gravity-crust.h5,output/step01c_gravity-slab.h5,
↪ output/step01c_gravity-wedge.h5 warp_grid --field=cauchy_stress --component=xy --
↪ exaggeration=5
```

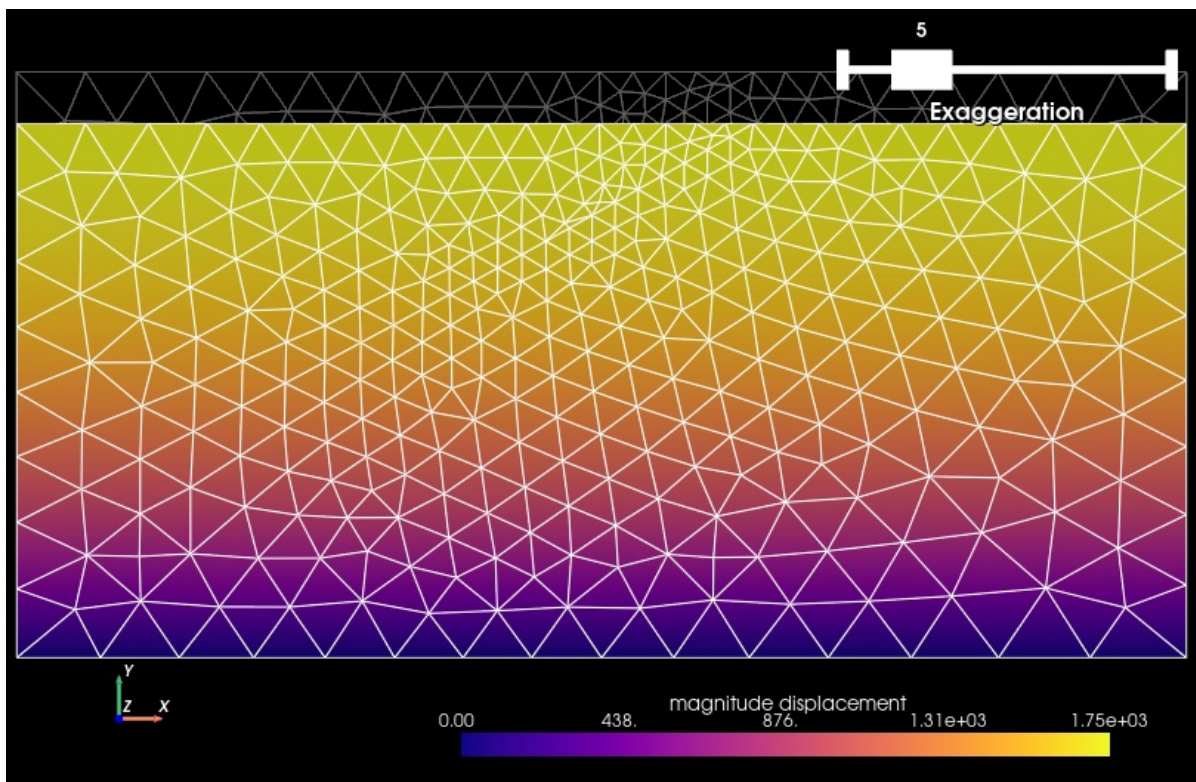


Fig. 4.87: Solution for Step 1c. The colors of the shaded surface indicate the magnitude of the displacement, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe.

Key Points

1. The displacement field is very similar for the three different discretizations.
2. In this case, we know the vertical displacement in the exact solution depends on the square of the depth, so we expect the numerical accuracy to be limited when we use a basis order of 1 for the displacement solution subfield.
3. In the exact solution, the axial components of the Cauchy stress tensor increase linearly with depth, and the xy

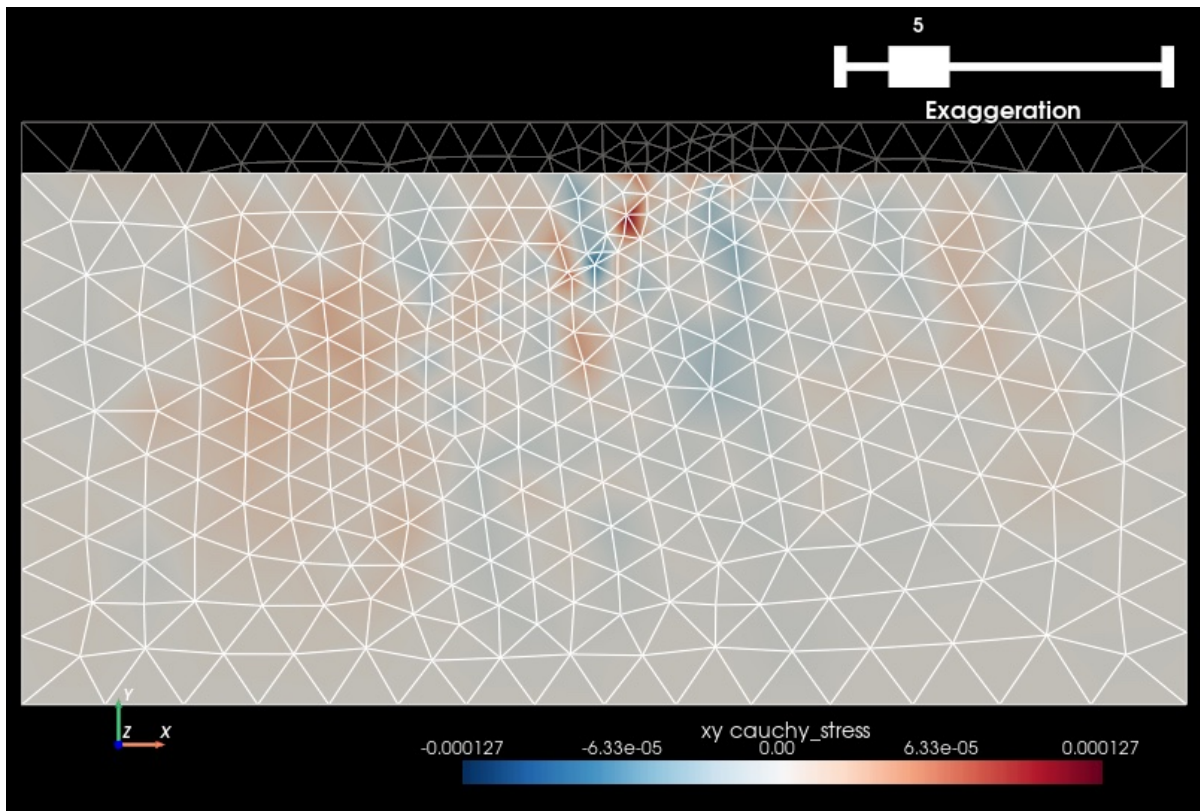


Fig. 4.88: Cauchy stress tensor component xy for Step 1c. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe. Increasing the basis order reduces the maximum shear stress by about 10 orders of magnitude!

component of the Cauchy stress is zero.

4. With the coarse mesh, the shear stress is close to 10 MPa throughout the mesh.
5. Refining the mesh decreases the magnitude shear stress by about a factor of 2.
6. Using a basis order of 2 for the solution subfields reduces the magnitude of the shear stress by nearly 10 orders of magnitude.
7. When resolving the stress field components is important, you will likely want to use a basis order of 2 for the displacement solution subfield.

Step 2: Gravitational Body Forces with Reference Stress

Features

- `Triangular cells`
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `Static simulation`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.GravityField`

Simulation parameters

This example involves using a reference stress state to minimize the deformation when we apply the gravitational body forces. The solution will be the perturbation from the reference state with zero displacements. This is one method for obtaining an initial stress state associated with gravitational body forces. We use the same roller boundary conditions that we used in Step 1. The parameters specific to this example are in `step02_gravity_refstate.cfg`.

We use a reference stress state that matches the overburden (lithostatic) pressure. We have uniform material properties, so the overburden is

$$\sigma_{xx} = \sigma_{yy} = \sigma_{zz} = \int_0^z \rho g \, dz = \rho g z, \quad (4.481)$$

where compressive stress is negative.

Listing 4.132: Parameters for reference stresses for Step 2. We only show the details for the slab material.

```
[pylithapp.problem.materials.slab]
db_auxiliary_field.iohandler.filename = mat_gravity_refstate.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.materials.slab.bulk_rheology]
use_reference_state = True
```

(continues on next page)

(continued from previous page)

```
auxiliary_subfields.reference_stress.basis_order = 1
auxiliary_subfields.reference_strain.basis_order = 0
```

Running the simulation

Listing 4.133: Run Step 2 simulation

```
$ pylith step02_gravity_refstate.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 6.916025621683e-12
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.
WARNING! There are options you set that were not used!
WARNING! could be spelling mistake, etc!
There is one unused database option. It is:
Option left: name:-mg_fine_ksp_max_it value: 5 source: code
```

By design we set the reference stress state so that it matches the loading from gravitational body forces in our domain with uniform material properties. As a result, the first nonlinear solver residual evaluation meets the convergence criteria. The linear solver is not used; this is why PETSc reports an unused option at the end of the simulation.

Visualizing the results

In Fig. 4.89 and Fig. 4.90 we use the `pylith_viz` utility to visualize the simulation results.

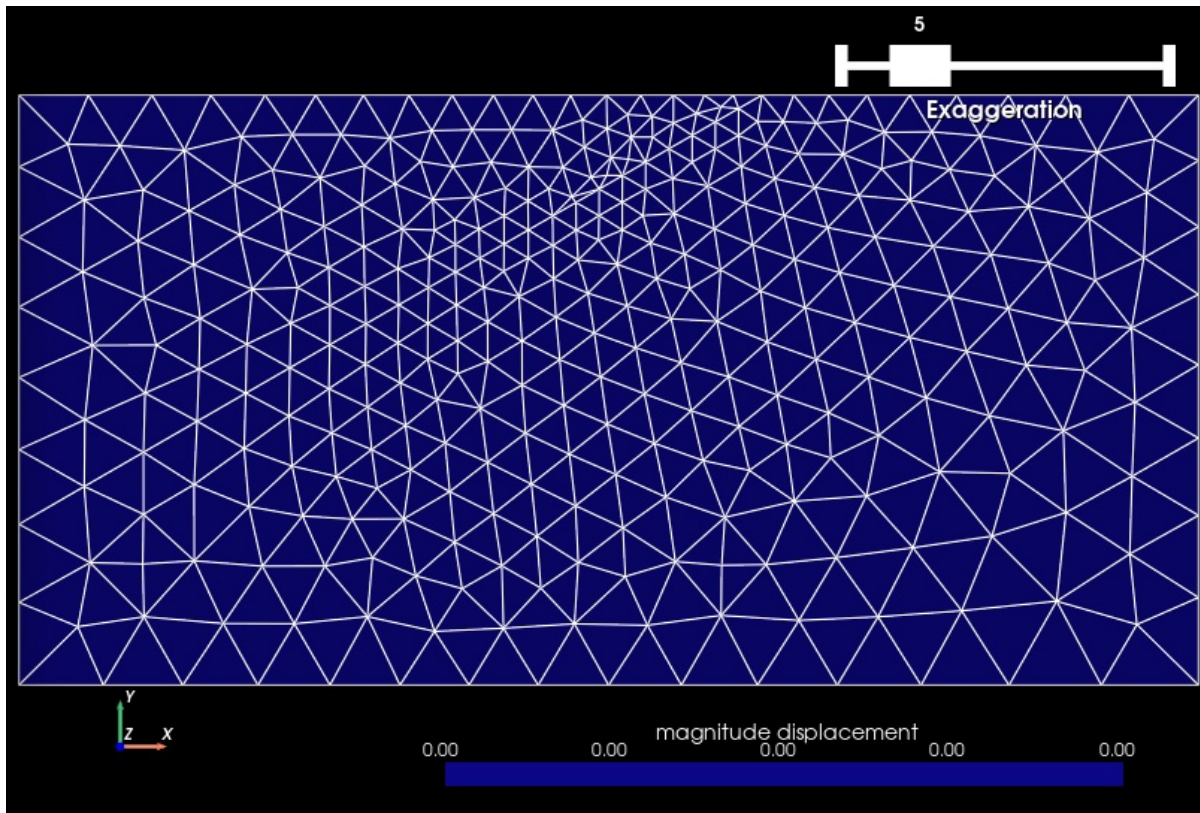


Fig. 4.89: Solution for Step 2. The colors of the shaded surface indicate the magnitude of the displacement, which is zero. The undeformed configuration is shown by the gray wireframe.

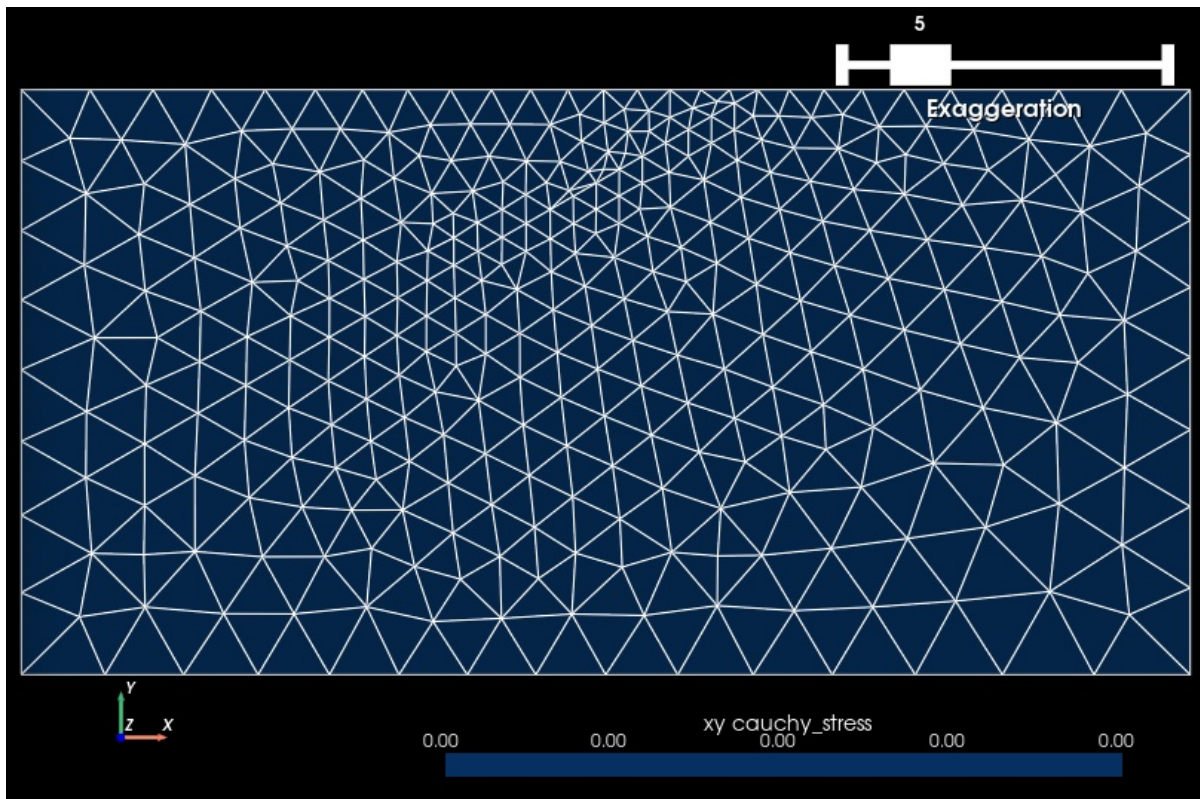


Fig. 4.90: Cauchy stress tensor component xy for Step 2. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe. The shear stress is zero.

Listing 4.134: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step02_gravity_refstate-domain.h5 warp_grid --
↳exaggeration=5
pylith_viz --filenames=output/step02_gravity_refstate-crust.h5,output/step02_gravity_
↳refstate-slab.h5,output/step02_gravity_refstate-wedge.h5 warp_grid --field=cauchy_
↳stress --component=xy --exaggeration=5
```

Step 3: Gravitational Body Forces with Incompressible Elasticity

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- `pylith.materials.IncompressibleElasticity`
- `spatialdata.spatialdb.GravityField`
- field split preconditioner
- Schur complement preconditioner

Simulation parameters

In this example we use incompressible elasticity (see *Incompressible Elasticity* for the finite-element formulation) to obtain the stress field associated with gravitational body forces,

$$\vec{s} = (\vec{u} \quad p)^T, \quad (4.482)$$

$$\rho(\vec{x})\vec{g} + \nabla \cdot (\boldsymbol{\sigma}^{dev}(\vec{u}) - p\mathbf{I}) = \vec{0}, \quad (4.483)$$

$$\vec{\nabla} \cdot \vec{u} + \frac{p}{K} = 0. \quad (4.484)$$

Because the material is incompressible and the material is confined on the lateral boundaries and bottom, we do not expect any deformation. In general, this is a more robust way to determine an initial stress state for gravitational body forces compared to using a reference stress state, especially when the material properties are not uniform. We use the same roller boundary conditions that we used in Steps 1 and 2. The parameters specific to this example are in `step03_gravity_incompressible.cfg`.

Listing 4.135: Parameters for incompressible elasticity in Step 3.

```
solution = pylith.problems.SolnDispPres
[pylithapp.problem.materials]
```

(continues on next page)

(continued from previous page)

```

slab = pylith.materials.IncompressibleElasticity
crust = pylith.materials.IncompressibleElasticity
wedge = pylith.materials.IncompressibleElasticity

[pylithapp.problem.materials.slab]
db_auxiliary_field.iohandler.filename = mat_elastic_incompressible.spatialdb

[pylithapp.problem.materials.crust]
db_auxiliary_field.iohandler.filename = mat_elastic_incompressible.spatialdb

[pylithapp.problem.materials.wedge]
db_auxiliary_field.iohandler.filename = mat_elastic_incompressible.spatialdb

```

With pressure as a solution subfield, we add a Dirichlet boundary condition to set the confining pressure to 0 on the ground surface (+y boundary).

Listing 4.136: Adjustments to the Dirichlet boundary condition parameters for Step 3.

```

[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos]
bc.bc_ypos = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_ypos]
label = boundary_ypos
label_value = 13
constrained_dof = [0]
field = pressure
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC for pressure on +y edge

auxiliary_subfields.initial_amplitude.basis_order = 0

observers.observer.data_fields = [pressure]

```

Running the simulation

Listing 4.137: Run Step 3 simulation

```

$ pylith step03_gravity_incompressible.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.

```

(continues on next page)

(continued from previous page)

```

0 SNES Function norm 3.007829881319e+03
  Linear solve converged due to CONVERGED_ATOL iterations 1
1 SNES Function norm 3.268174844829e-10
  Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:239:static pylith::topology::Mesh_
↳ *pylith::topology::MeshOps::createLowerDimMesh(const pylith::topology::Mesh &, const_
↳ char *, const int, const char *)
-- warning (deprecation)
-- Creating lower dimension mesh from label with vertices. This feature will be removed_
↳ in v6.0. In the future, you will need to mark boundaries instead of vertices.
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

PyLith detected use of incompressible elasticity, so it selected a field split preconditioner with an LU preconditioner for each of the solution subfields as described in *PETSc Options*. As a result, the linear solve converged in 1 iteration.

Visualizing the results

In Fig. 4.91 and Fig. 4.92 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.138: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filename=output/step03_gravity_incompressible-domain.h5 warp_grid --
↳ exaggeration=5
pylith_viz --filenames=output/step03_gravity_incompressible-crust.h5,output/step03_
↳ gravity_incompressible-slab.h5,output/step03_gravity_incompressible-wedge.h5 warp_grid_
↳ --field=cauchy_stress --component=xy --exaggeration=5

```

Step 4: Surface Traction

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.bc.NeumannTimeDependent`

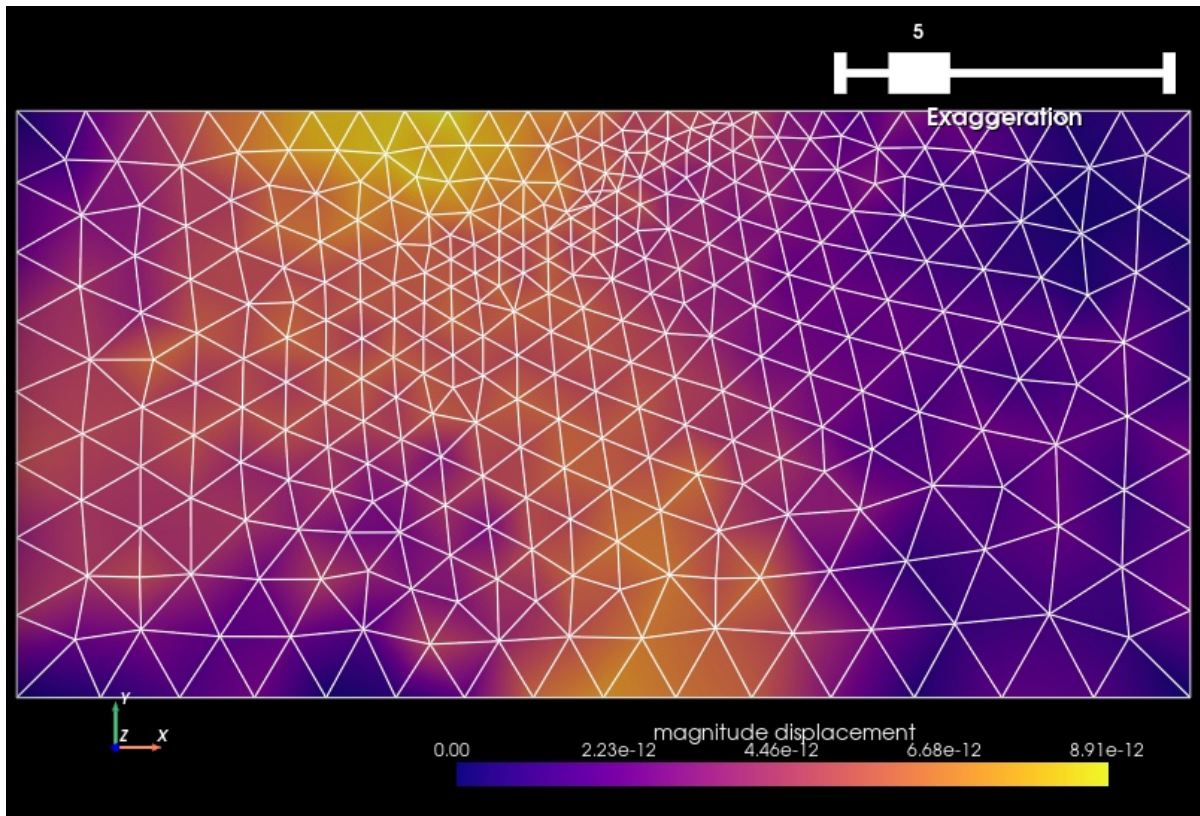


Fig. 4.91: Solution for Step 3. The colors of the shaded surface indicate the magnitude of the displacement. The undeformed configuration is shown by the gray wireframe. There is negligible deformation and the stress state (not shown) matches the one in Step 2.

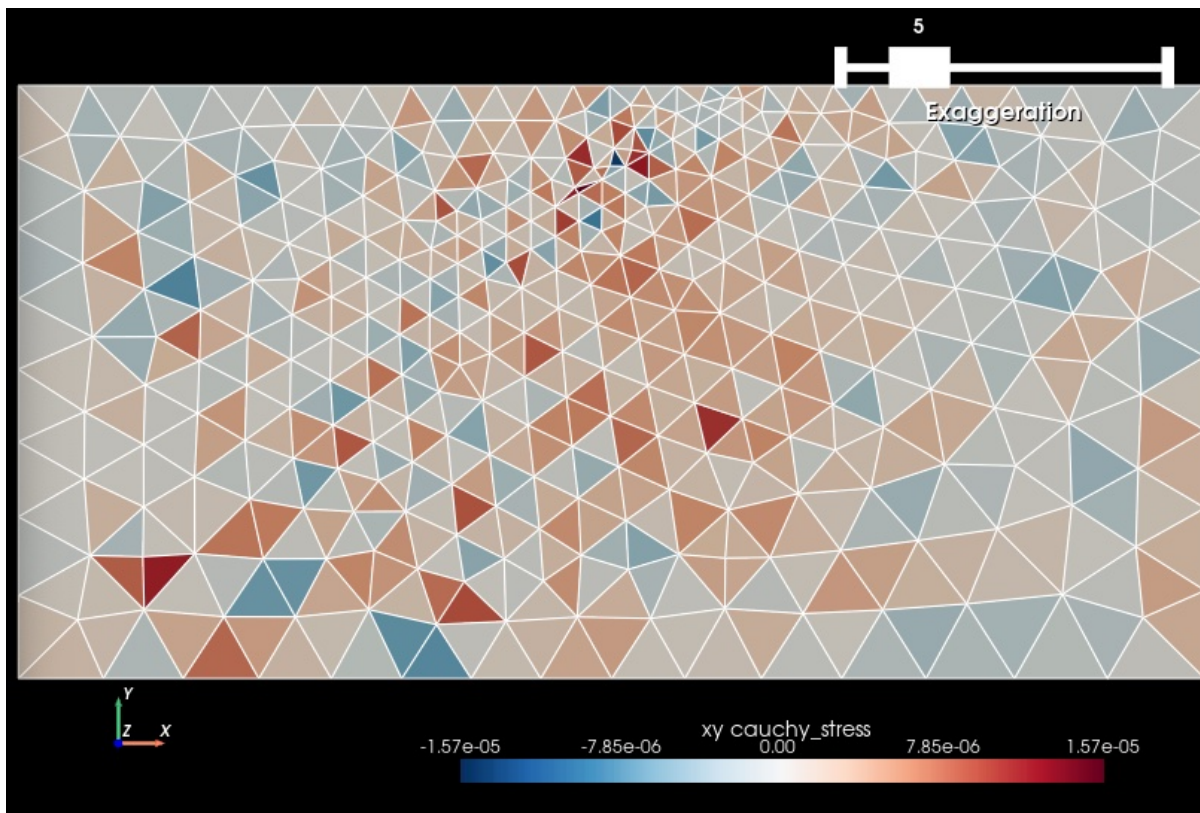


Fig. 4.92: Cauchy stress tensor component xy for Step 3. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 5. The undeformed configuration is shown by the gray wireframe. The shear stress is negligible.

This example focuses on loading via surface tractions on the +y boundary. We apply tractions normal to the boundary with a trapezoidal distribution as shown in Fig. 4.93. We use the same roller boundary conditions that we used in Steps 1-3.

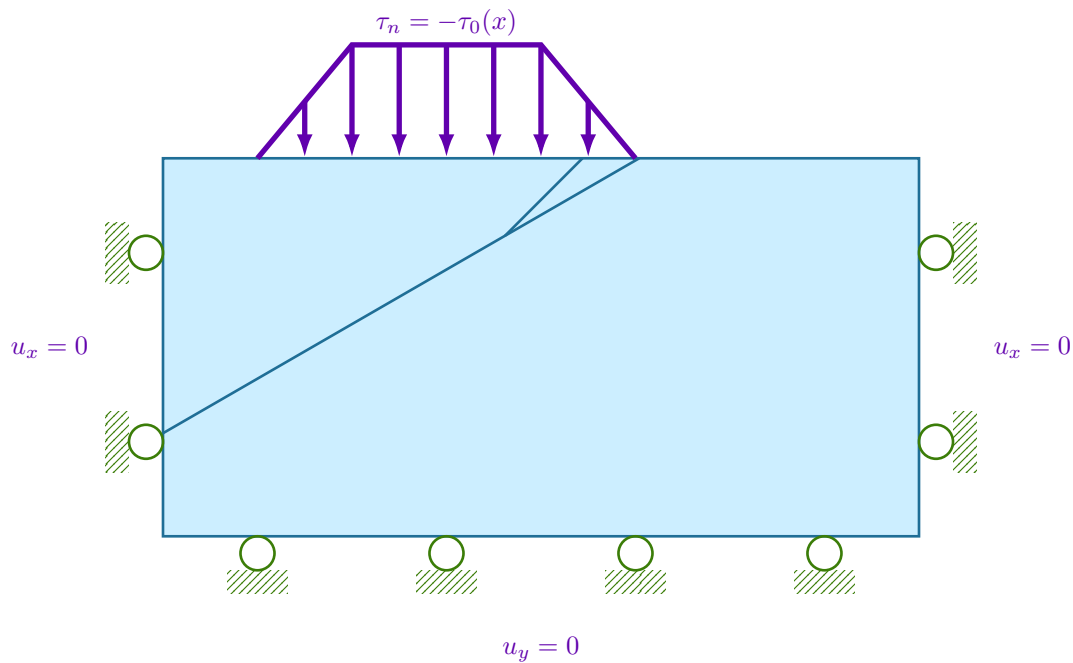


Fig. 4.93: We add a Neumann (traction) boundary condition on the +y boundary with roller boundary conditions on the lateral sides and bottom of the domain.

Step 4a: Coarse Mesh

Note

New in v4.1.0.

We start with a coarse resolution mesh that captures the geometry and increase the resolution of the simulation by using uniform refinement or increasing the basis order of the solution fields.

Simulation parameters

The parameters specific to this example are in `step04a_surfload.cfg`.

Listing 4.139: Surface load parameters for Step 4a.

```
[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos]
bc.bc_ypos = pylith.bc.NeumannTimeDependent

[pylithapp.problem.bc.bc_ypos]
label = boundary_ypos
label_value = 13

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
```

(continues on next page)

(continued from previous page)

```

db_auxiliary_field.description = Neumann BC +y edge
db_auxiliary_field.iohandler.filename = traction_surflowd.spatialdb

db_auxiliary_field.query_type = linear

auxiliary_subfields.initial_amplitude.basis_order = 1

```

Running the simulation

Listing 4.140: Run Step 4a simulation

```

$ pylith step04a_surflowd.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 5.420547826416e+01
      Linear solve converged due to CONVERGED_ATOL iterations 10
    1 SNES Function norm 6.164978944375e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

Visualizing the results

In Fig. 4.94 and Fig. 4.95 we use the `pylith_viz` utility to visualize the simulation results.

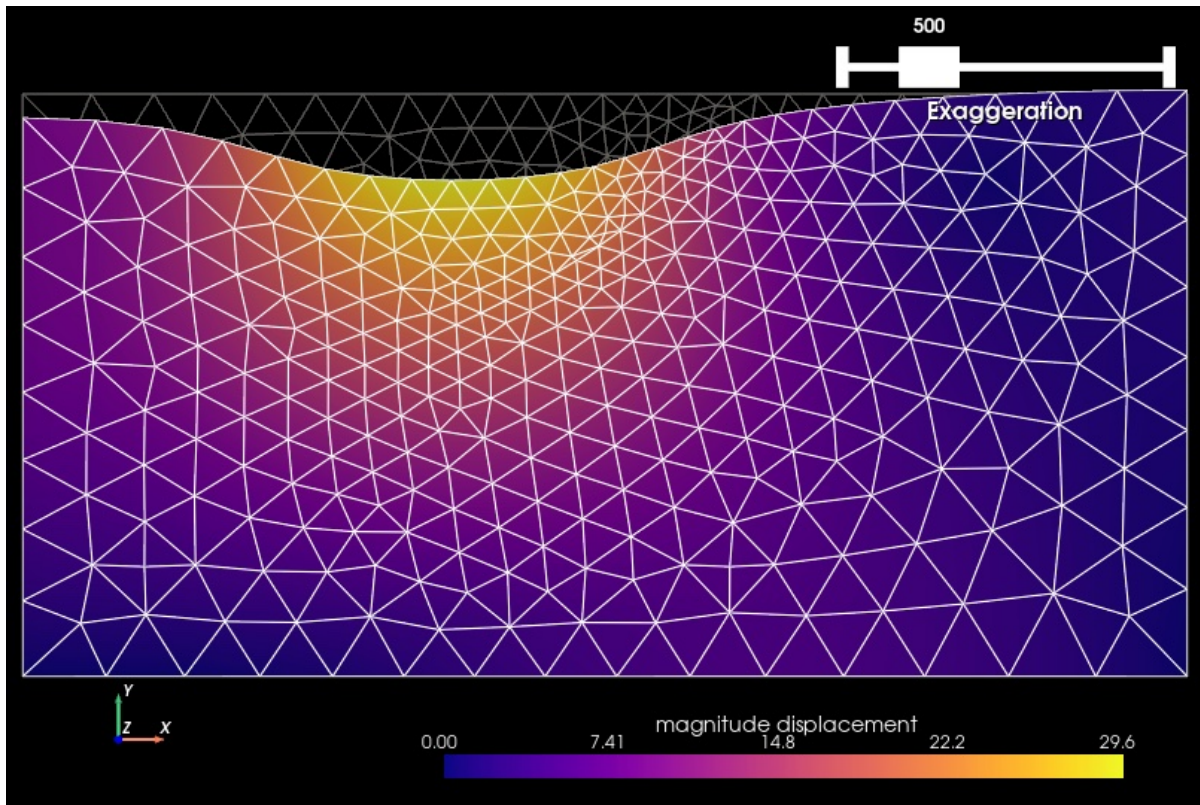


Fig. 4.94: Solution for Step 4a. The colors of the shaded surface indicate the magnitude of the displacement, and the deformation is exaggerated by a factor of 500. The undeformed configuration is shown by the gray wireframe.

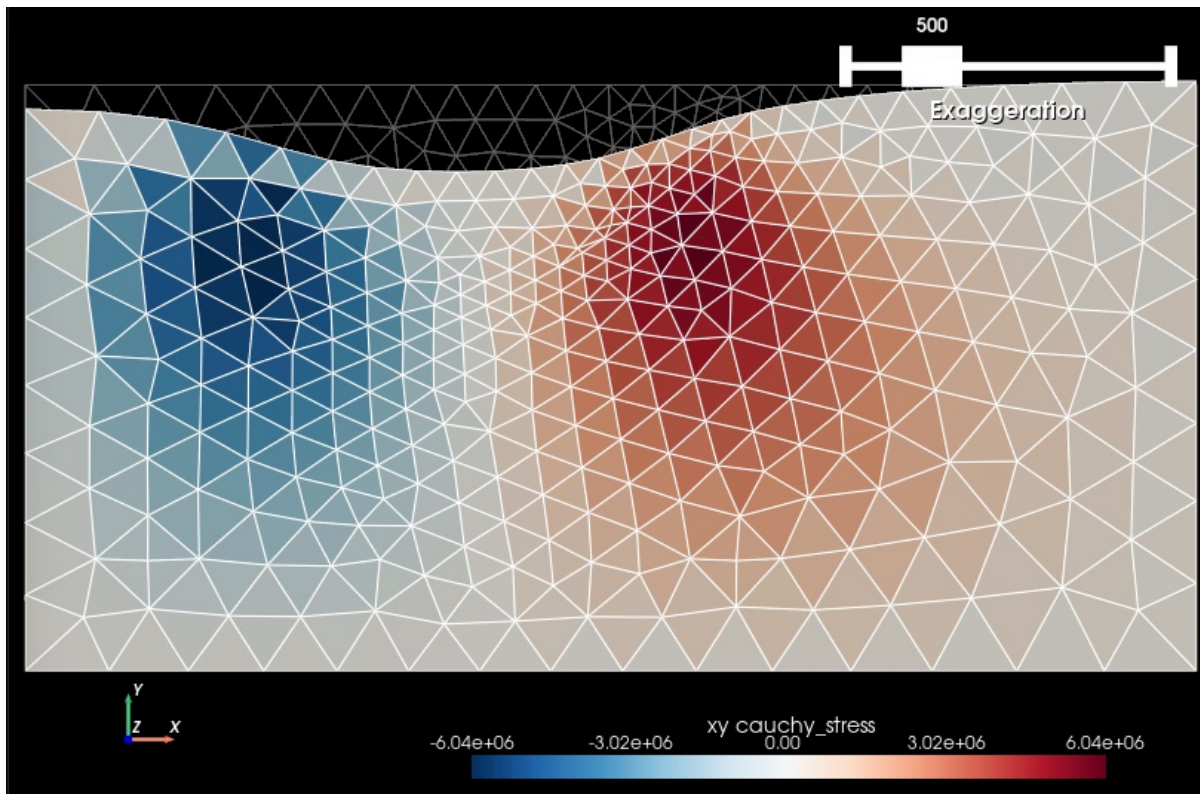


Fig. 4.95: Cauchy stress tensor component xy for Step 4a. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 500. The undeformed configuration is shown by the gray wireframe. We observe that the coarse resolution mesh does not resolve the shear stress very well, and we have moderate jumps in the shear stress field between adjacent cells.

Listing 4.141: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step04a_surfload-domain.h5 warp_grid --exaggeration=500
pylith_viz --filenames=output/step04a_surfload-crust.h5,output/step04a_surfload-slab.h5,
↪output/step04a_surfload-wedge.h5 warp_grid --field=cauchy_stress --component=xy --
↪exaggeration=500
```

Step 4b: Refined Mesh

As in Step 1, we try to improve on the accuracy of the solution by refining the mesh.

Simulation parameters

The parameters specific to this example are in `step04b_surfload.cfg`.

Listing 4.142: Mesh refinement parameters for Step 4b.

```
[pylithapp.mesh_generator]
refiner = pylith.topology.RefineUniform
```

Running the simulation

Listing 4.143: Run Step 4b simulation

```
$ pylith step04b_surfload.cfg
# The output should look very similar to Step 4a.
```

Visualizing the results

In Fig. 4.96 and Fig. 4.97 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.144: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step04b_surfload-domain.h5 warp_grid --exaggeration=500
pylith_viz --filenames=output/step04b_surfload-crust.h5,output/step04b_surfload-slab.h5,
↪output/step04b_surfload-wedge.h5 warp_grid --field=cauchy_stress --component=xy --
↪exaggeration=500
```

Step 4c: Higher Order Discretization

As in Step 1, we again use a higher order discretization of the solution field to improve the accuracy of the model.

Simulation parameters

The parameters specific to this example are in `step04c_surfload.cfg`.

Listing 4.145: Parameters for using higher order discretization in Step 4c.

```
[pylithapp.problem]
defaults.quadrature_order = 2
```

(continues on next page)

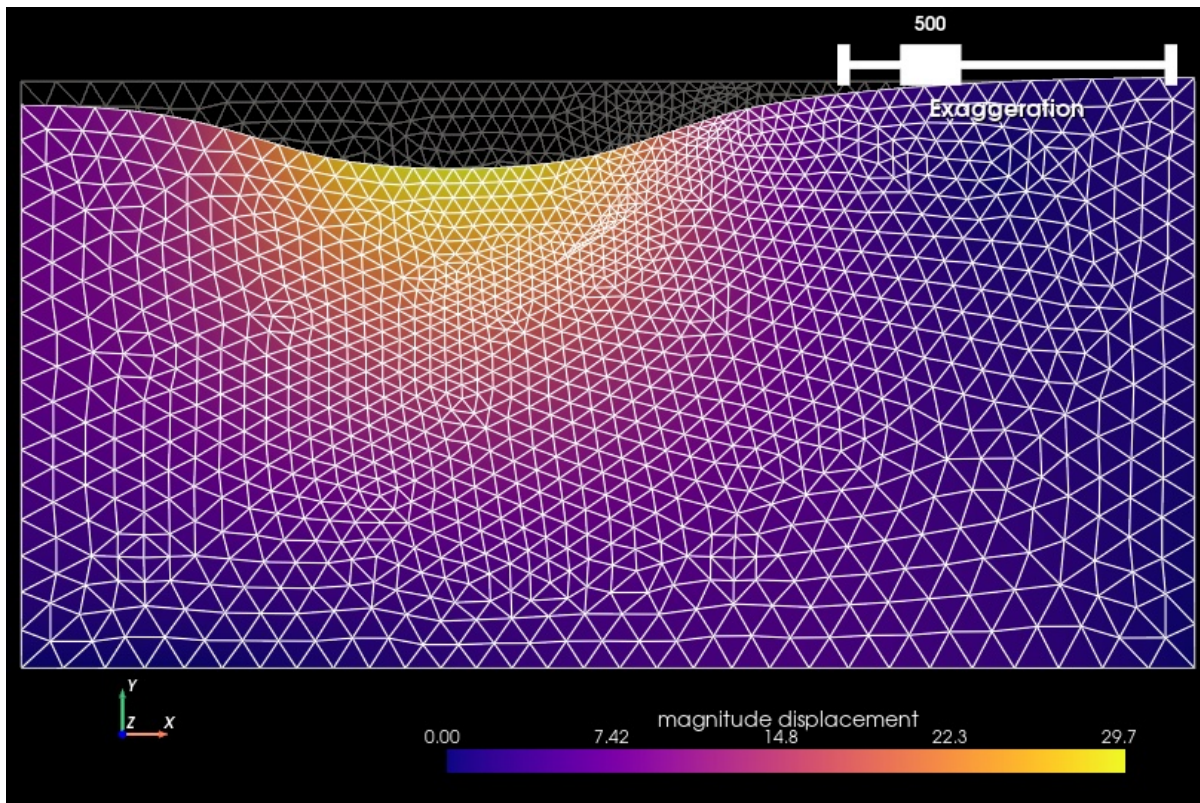


Fig. 4.96: Solution for Step 4b. The colors of the shaded surface indicate the magnitude of the displacement, and the deformation is exaggerated by a factor of 500. The undeformed configuration is shown by the gray wireframe.

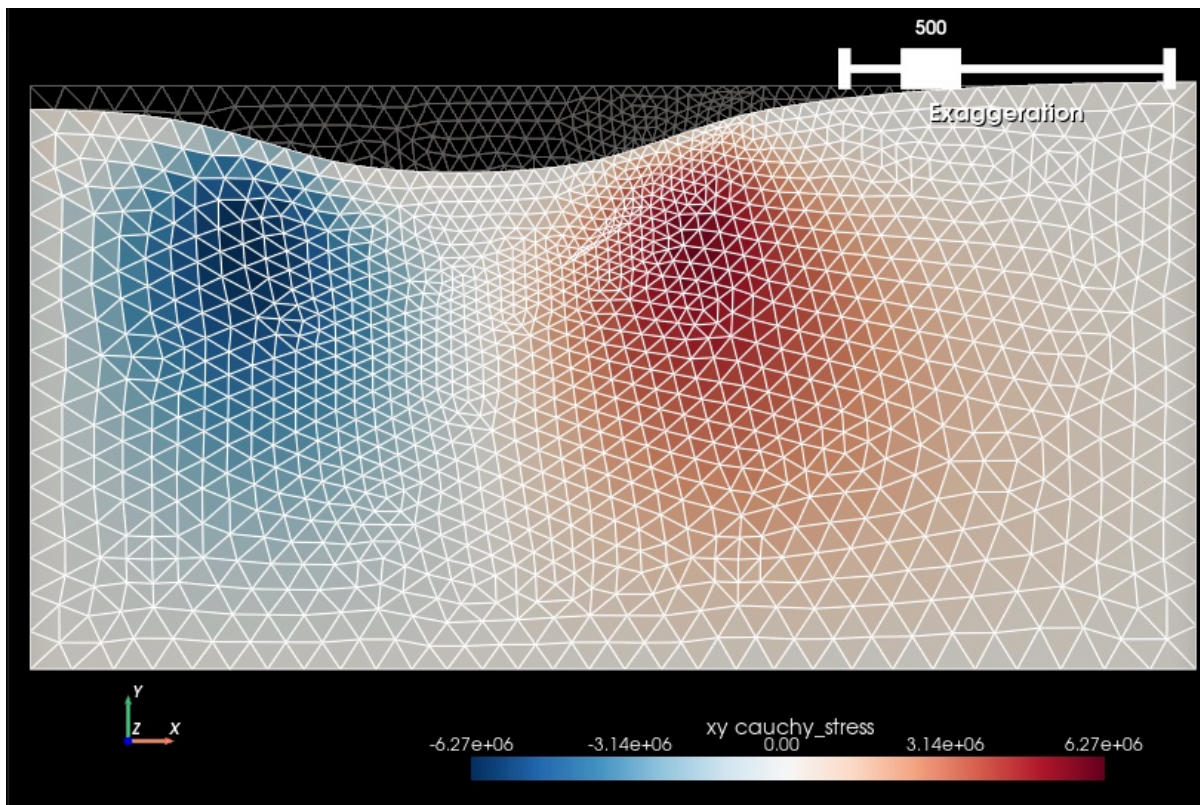


Fig. 4.97: Cauchy stress tensor component xy for Step 4b. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 500. The undeformed configuration is shown by the gray wireframe. Refining the mesh reduces the size of the jumps in the shear stress field between adjacent cells.

(continued from previous page)

```
[pylithapp.problem.solution.subfields.displacement]
basis_order = 2

[pylithapp.problem.materials.slab]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.crust]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.wedge]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1
```

Running the simulation

Listing 4.146: Run Step 4c simulation

```
$ pylith step04c_surflowd.cfg

# The output should look very similar to Steps 4a and 4b.
```

Visualizing the results

In Fig. 4.98 and Fig. 4.99 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.147: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step04c_surflowd-domain.h5 warp_grid --exaggeration=500
pylith_viz --filenames=output/step04c_surflowd-crust.h5,output/step04c_surflowd-slab.h5,
↪output/step04c_surflowd-wedge.h5 warp_grid --field=cauchy_stress --component=xy --
↪exaggeration=500
```

Key Points

1. The displacement field is very similar for the three different discretizations.
2. This stress field, especially the shear stress component, is not well resolved with a basis order of 1 and the coarse mesh.
3. Refining the mesh to decrease the discretization size by a factor of two yields better resolution of the shear stress field.
4. Using a basis order of 2 with the coarse mesh provides better resolution of the stress concentration compared with refining the mesh.
5. When resolving the stress field components is important, you will likely want to use a basis order of 2 for the displacement solution subfield.

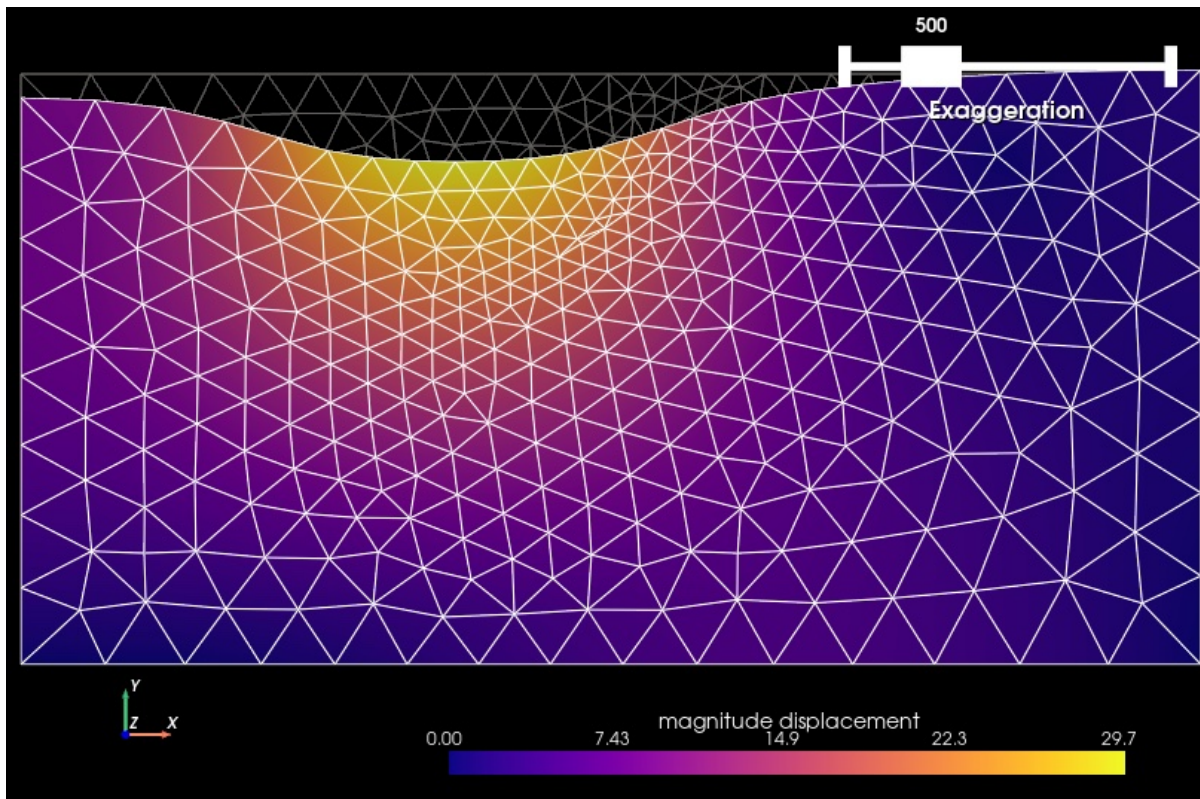


Fig. 4.98: Solution for Step 4c. The colors of the shaded surface indicate the magnitude of the displacement, and the deformation is exaggerated by a factor of 500. The undeformed configuration is shown by the gray wireframe.

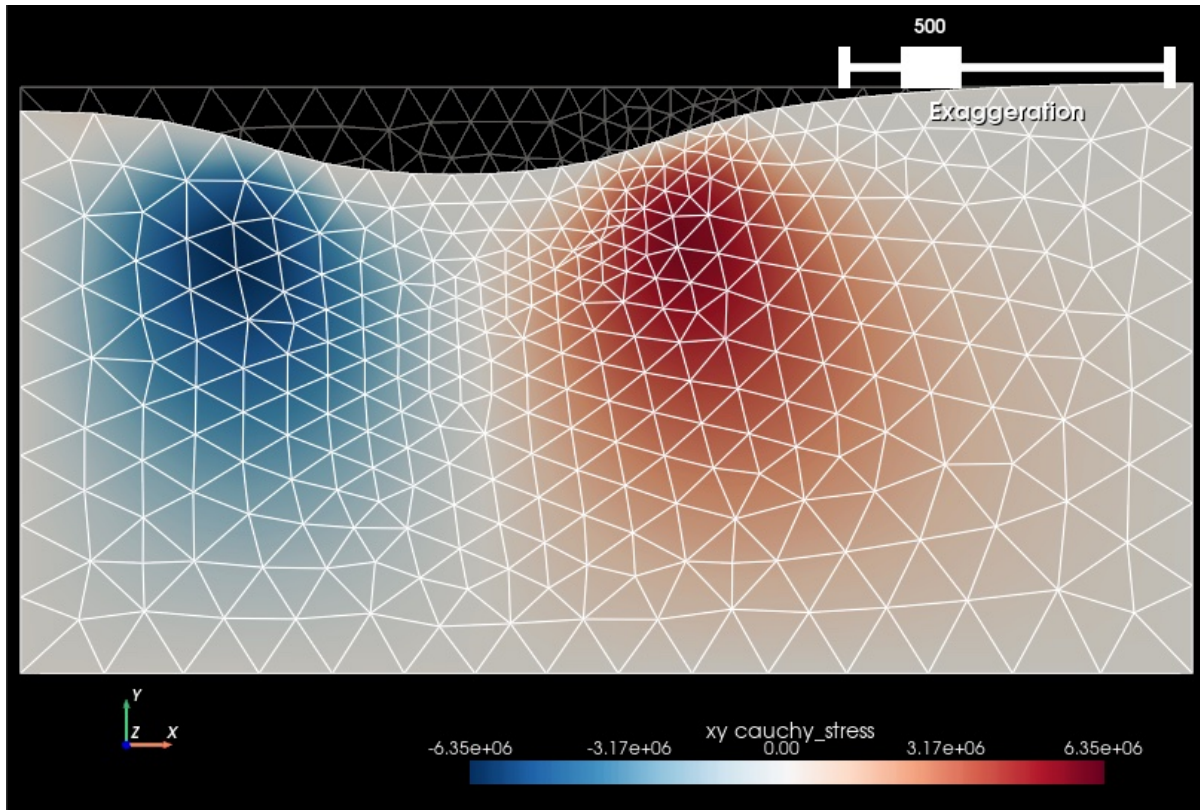


Fig. 4.99: Cauchy stress tensor component xy for Step 4c. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 500. The undeformed configuration is shown by the gray wireframe. The higher order discretization removes the jumps in the shear stress field between adjacent cells.

Step 5: Static Coseismic Slip

This example involves a static simulation that solves for the deformation from prescribed coseismic slip on the main fault. We specify 2 meters of reverse slip. Fig. 4.100 shows the boundary conditions on the domain.

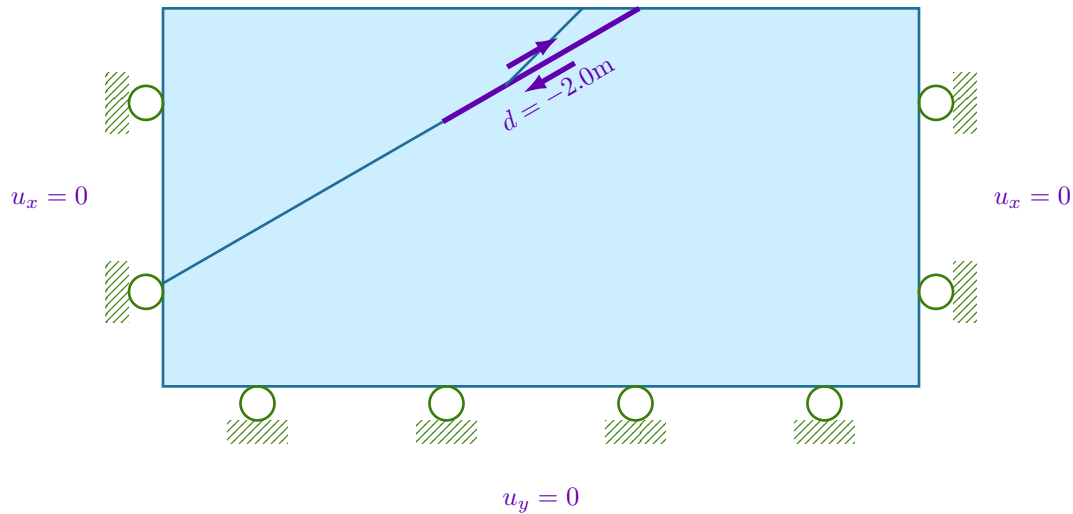


Fig. 4.100: Boundary conditions for static coseismic slip on the main fault. We prescribe 2 meters of reverse slip with roller boundary conditions on the lateral sides and bottom of the domain.

Important

In 2D simulations slip is specified in terms of opening and left-lateral components. This provides a consistent, unique sense of slip that is independent of the fault orientation. For our geometry in this example, right lateral slip corresponds to reverse slip on the dipping fault.

Important

The main fault contains one end that is buried within the domain. When PyLith inserts cohesive cells into a mesh with buried edges (in this case a point), we must identify these buried edges so that PyLith properly adjusts the topology along these edges.

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation

- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- spatialdata.spatialdb.UniformDB

Step 5a: Coarse Mesh

Note

New in v4.1.0.

We start with a coarse resolution mesh that captures the geometry and increase the resolution of the simulation by using uniform refinement or increasing the basis order of the solution fields. We also introduce different origin times for the coseismic slip on the two faults.

Simulation parameters

The parameters specific to this example are in `step05a_onefault.cfg`. We adjust the solution field to include both displacement and the Lagrange multiplier associated with the fault. For uniform prescribed slip we use a `UniformDB`.

Listing 4.148: Parameters for earthquake rupture on the main reverse fault in Step 5a.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange

[pylithapp.problem.interfaces.fault]
label = fault
label_value = 20
observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -2.0*m, 0.0*m]
```

Running the simulation

Listing 4.149: Run Step 5a simulation

```
$ pylith step05a_onefault.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
```

(continues on next page)

(continued from previous page)

```
# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 6.455846400259e+00
      Linear solve converged due to CONVERGED_ATOL iterations 10
    1 SNES Function norm 3.254498049626e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

From the end of the output written to the terminal window, we see that the linear solver converged in 10 iterations and met the absolute convergence tolerance (`ksp_atol`). As we expect for this linear problem, the nonlinear solver converged in 1 iteration.

Visualizing the results

In Fig. 4.101 and Fig. 4.102 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.150: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step05a_onefault-domain.h5 warp_grid --component=x --
↳ exaggeration=3000
pylith_viz --filenames=output/step05a_onefault-crust.h5,output/step05a_onefault-wedge.h5,
↳ output/step05a_onefault-slab.h5 warp_grid --field=cauchy_stress --component=xy --
↳ exaggeration=3000
```

Step 5b: Coarse Mesh

As in some of the previous steps in this example, we refine the mesh to improve the accuracy of the results. The parameters specific to this example are in `step05b_onefault.cfg`.

Simulation parameters

We adjust the solution field to include both displacement and the Lagrange multiplier associated with the fault. For uniform prescribed slip we use a `UniformDB`.

Listing 4.151: Mesh refinement parameters for Step 5b.

```
[pylithapp.problem.mesh_initializer.phases.refine_mesh]
refiner = pylith.topology.RefineUniform
```

Running the simulation

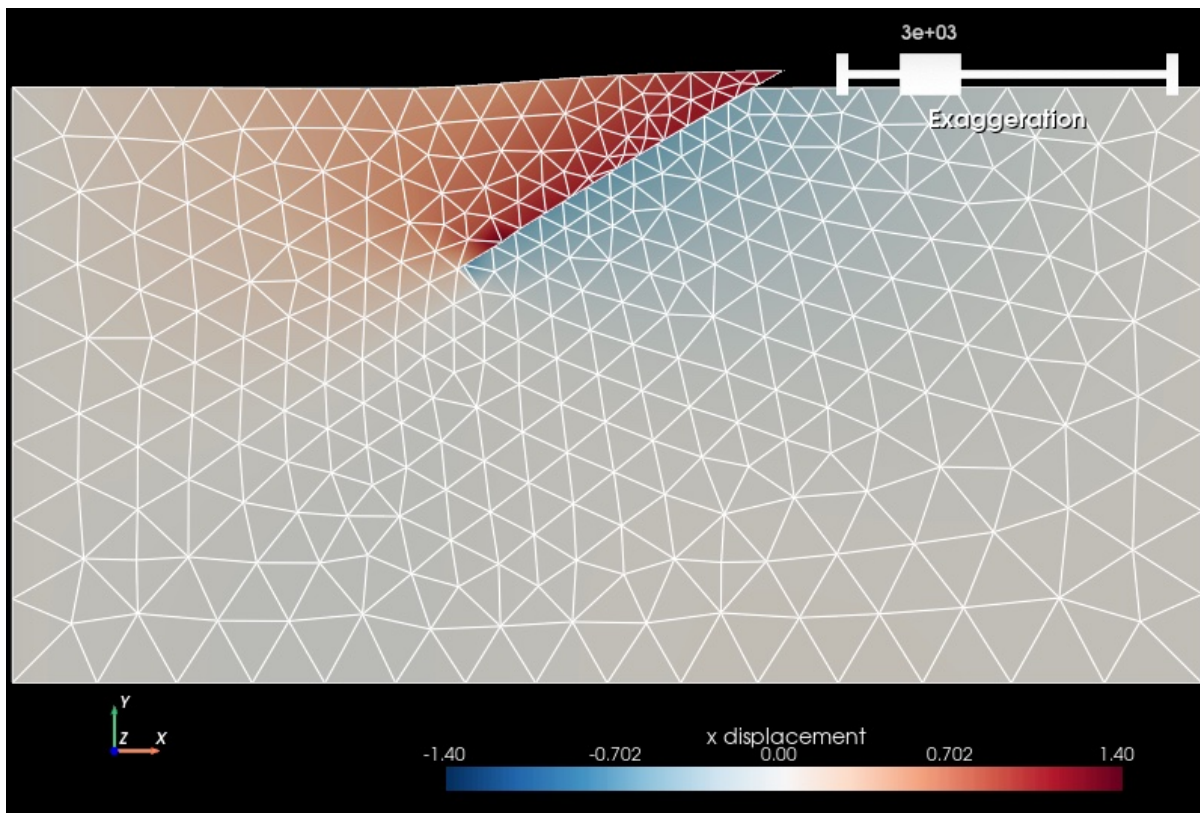


Fig. 4.101: Solution for Step 5a. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 3000. The undeformed configuration is shown by the gray wireframe.

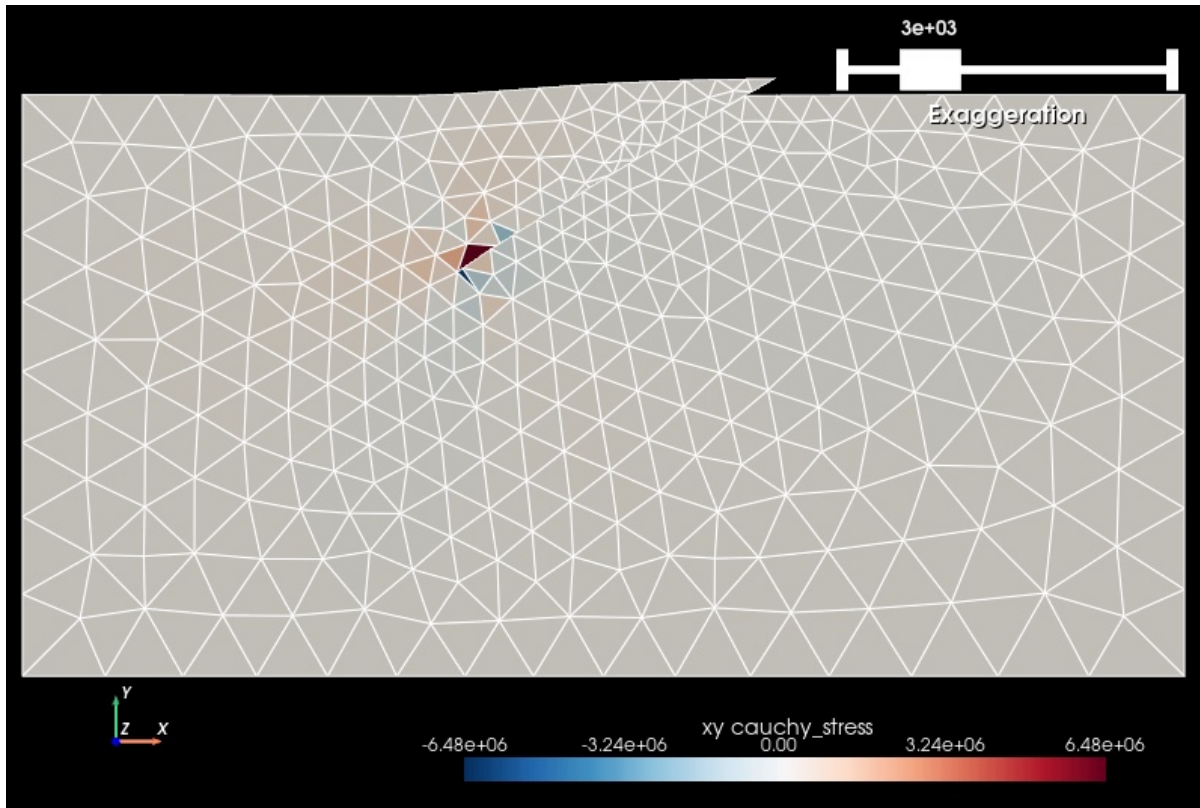


Fig. 4.102: Cauchy stress tensor component xy for Step 5a. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 3000. The undeformed configuration is shown by the gray wireframe. With uniform slip on the fault, we generate a stress concentration at the buried end of the fault that is confined to just one or two cells. Thus, the stress concentration is not well resolved.

Listing 4.152: Run Step 5b simulation

```
$ pylith step05b_onefault.cfg
# The output should look very similar to Step 5a.
```

Visualizing the results

In Fig. 4.103 and Fig. 4.104 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.153: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step05b_onefault-domain.h5 warp_grid --component=x --
↳exaggeration=3000
pylith_viz --filenames=output/step05b_onefault-crust.h5,output/step05b_onefault-wedge.h5,
↳output/step05b_onefault-slab.h5 warp_grid --field=cauchy_stress --component=xy --
↳exaggeration=3000
```

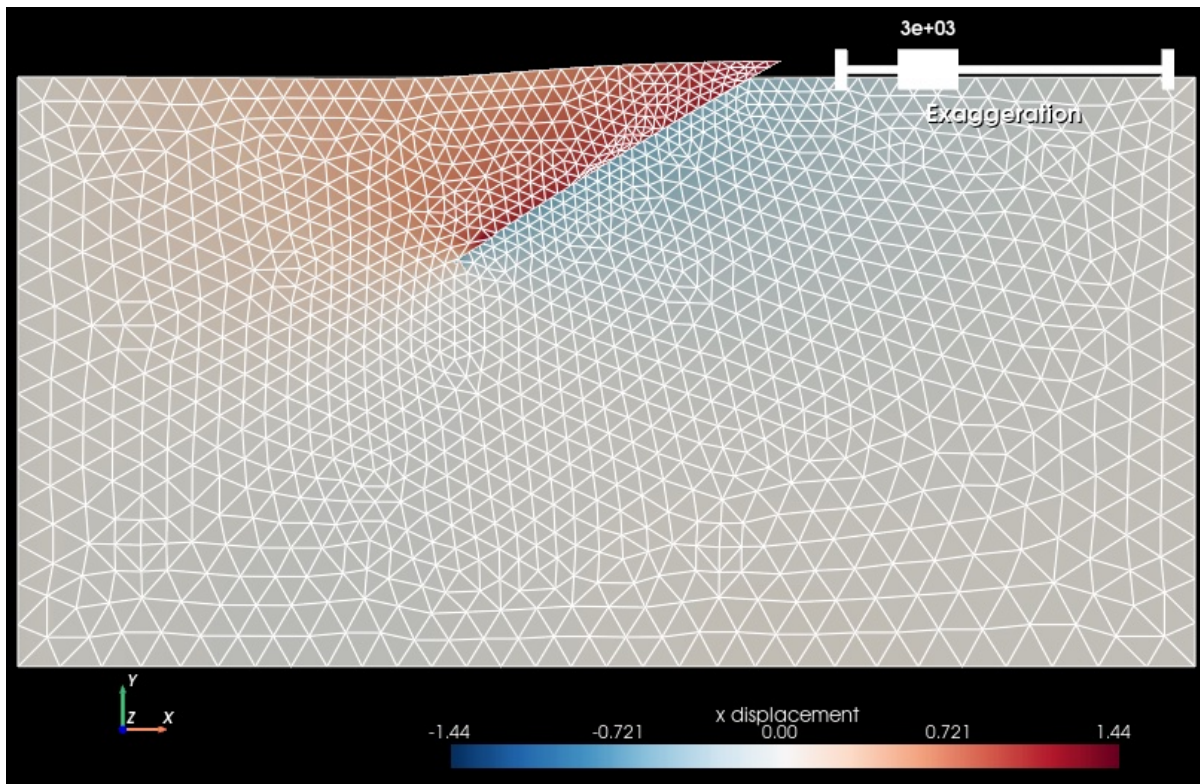


Fig. 4.103: Solution for Step 5b. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 3000. The undeformed configuration is shown by the gray wireframe.

Step 5c: Coarse Mesh

We now use higher order discretization to try to better resolve the stress concentration. The parameters specific to this example are in `step05c_onefault.cfg`.

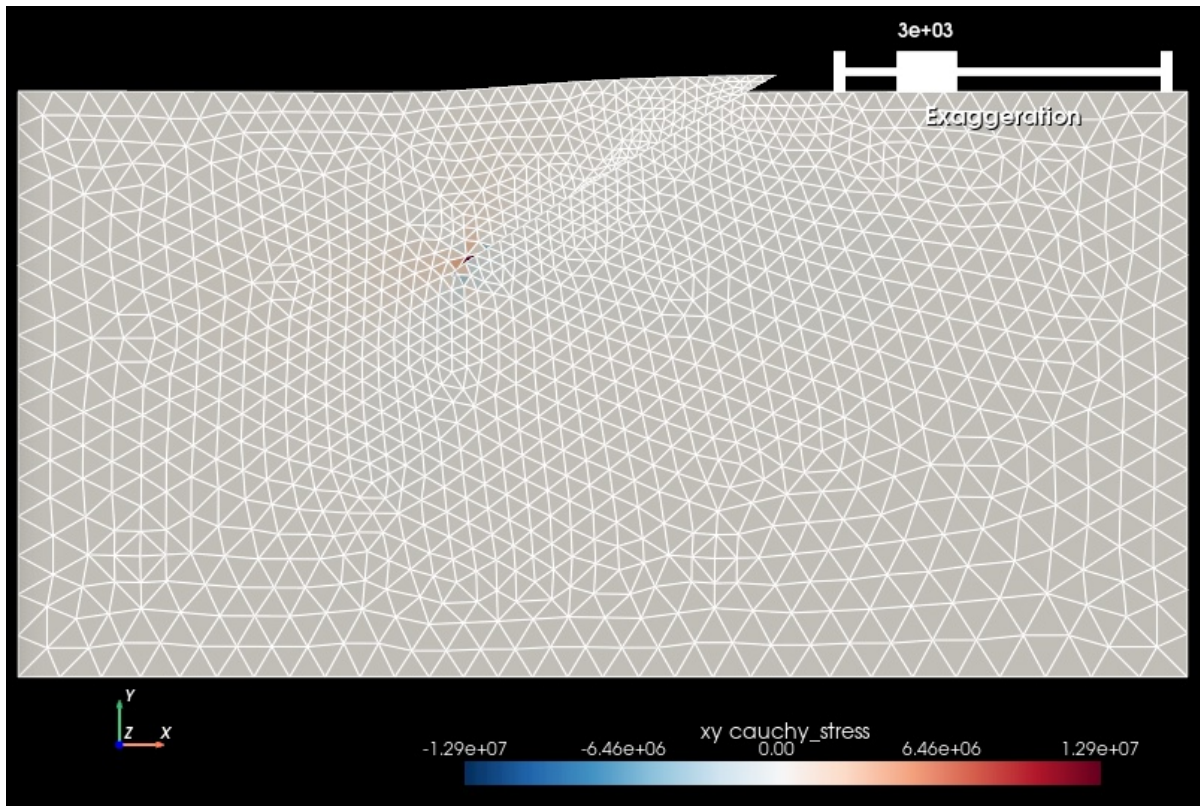


Fig. 4.104: Cauchy stress tensor component xy for Step 5b. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 3000. The undeformed configuration is shown by the gray wireframe. Refining the mesh shows the stress concentration at the buried end of the fault is smaller in spatial extent than we observed in Step 5a. We need a higher resolution mesh to resolve the stress concentration.

Simulation parameters

We adjust the solution field to include both displacement and the Lagrange multiplier associated with the fault. For uniform prescribed slip we use a `UniformDB`.

Listing 4.154: Parameters for using higher order discretization in Step 5c.

```
[pylithapp.problem]
defaults.quadrature_order = 2

[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
lagrange_multiplier_fault.basis_order = 2

[pylithapp.problem.materials.slab]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.crust]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.wedge]
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1
```

Running the simulation

Listing 4.155: Run Step 5c simulation

```
$ pylith step05c_onefault.cfg

# The output should look very similar to Steps 5a and 5b.
```

Visualizing the results

In Fig. 4.105 and Fig. 4.106 we use the `pylith_viz` utility to visualize the simulation results.

Listing 4.156: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step05c_onefault-domain.h5 warp_grid --component=x --
↳exaggeration=3000
pylith_viz --filenames=output/step05c_onefault-crust.h5,output/step05c_onefault-wedge.h5,
↳output/step05c_onefault-slab.h5 warp_grid --field=cauchy_stress --component=xy --
↳exaggeration=3000
```

Key Points

1. The displacement field is very similar for the three different discretizations.
2. Uniform slip generates a stress concentration at the buried end of the fault.
3. This stress concentration is difficult to resolve with a basis order of 1 for the solution subfields, even if we refine the mesh.

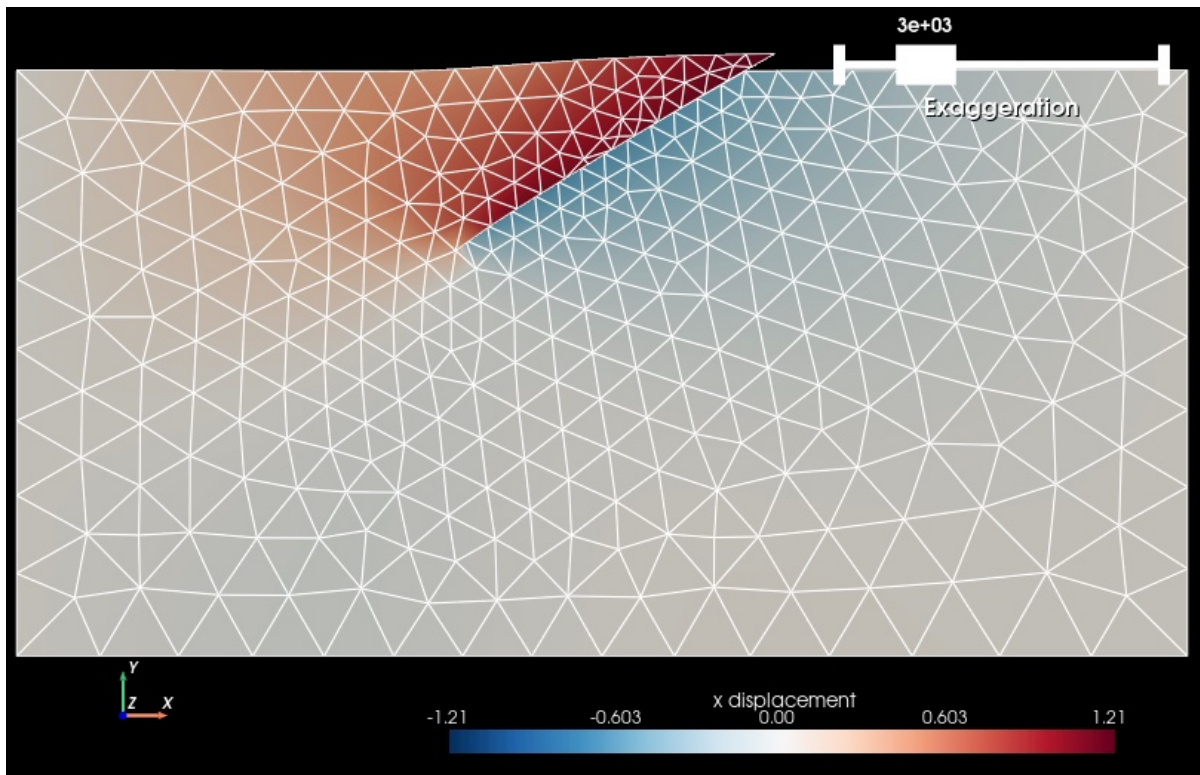


Fig. 4.105: Solution for Step 5c. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 3000. The undeformed configuration is shown by the gray wireframe.

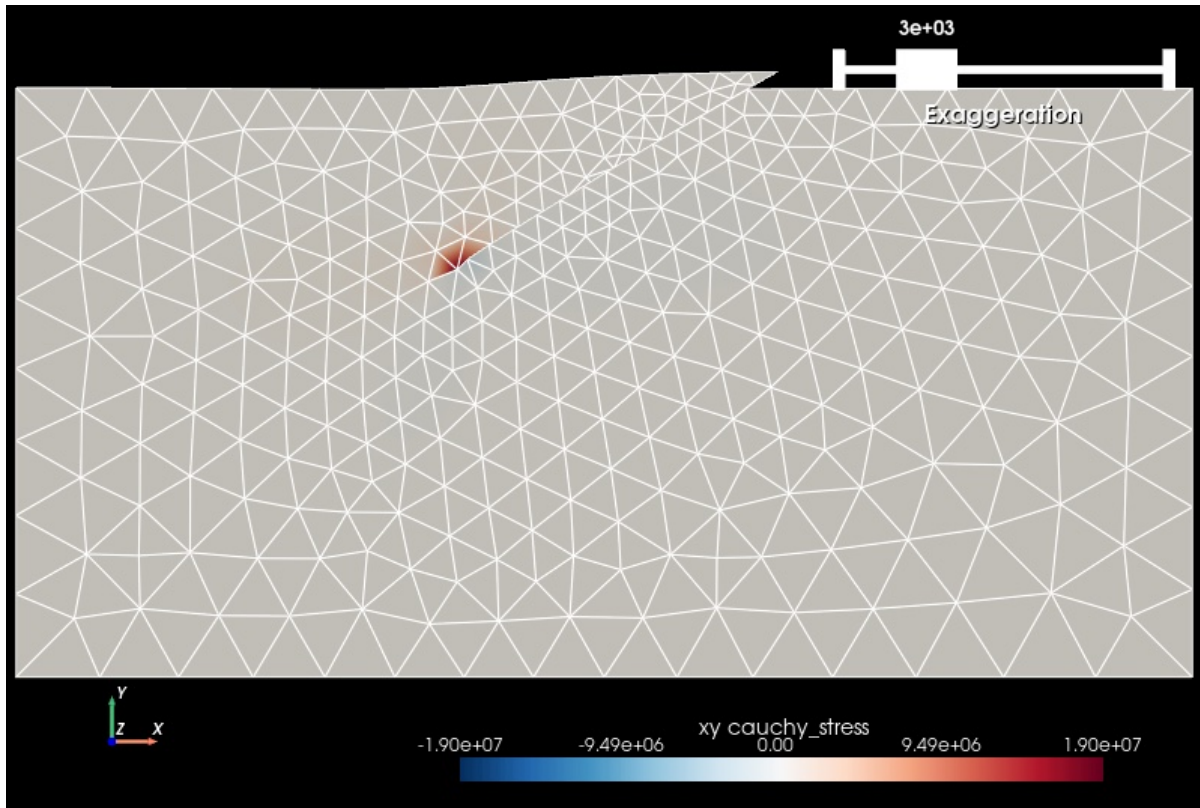


Fig. 4.106: Cauchy stress tensor component xy for Step 5c. The colors of the shaded surface indicate the xy component of the Cauchy stress tensor, and the deformation is exaggerated by a factor of 3000. The undeformed configuration is shown by the gray wireframe. With the higher order discretization of the solution subfields, we find better resolution of the stress concentration, which is now spread across a few cells.

4. Using a basis order of 2 provides better resolution of the stress concentration, but it is poorly resolved without also refining the mesh.
5. Adaptive mesh refinement (not yet available in PyLith) that refines the region just around the stress concentration would help resolve the stress concentration without substantially increasing the problem size.

Step 6: Slip on Two Faults and Elastic Materials

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.faults.FaultCohesiveKin`
- `pylith.faults.KinSrcStep`
- `spatialdata.spatialdb.UniformDB`

In this example we add coseismic slip on the splay fault with an origin time of 40 years. We specify 2 meters of reverse slip on the main fault and 1 meter of reverse slip on the splay fault. [Fig. 4.107](#) shows the boundary conditions on the domain. The parameters specific to this example are in `step06_twofaults-elastic.cfg`.

Simulation parameters

We use uniform refinement to reduce the discretization size by a factor of 2 and increase the numerical resolution. With different origin times for slip on the two faults, we need to add time stepping. With a linearly elastic, quasistatic model (no inertia) and fixed displacements on the boundaries, the displacement field only changes as a result of fault slip, so we can use just a few time steps to resolve the deformation. We impose slip on the main fault in the first time step (advancing the solution to $t=0$), and impose slip on the splay fault in the third time step (advancing the solution to $t=40$ years).

Listing 4.157: Parameters for uniform refinement and time stepping in Step 6.

```
[pylithapp.mesh_generator]
refiner = pylith.topology.RefineUniform

[pylithapp.problem]
initial_dt = 20.0*year
start_time = -20.0*year
end_time = 40.0*year
```

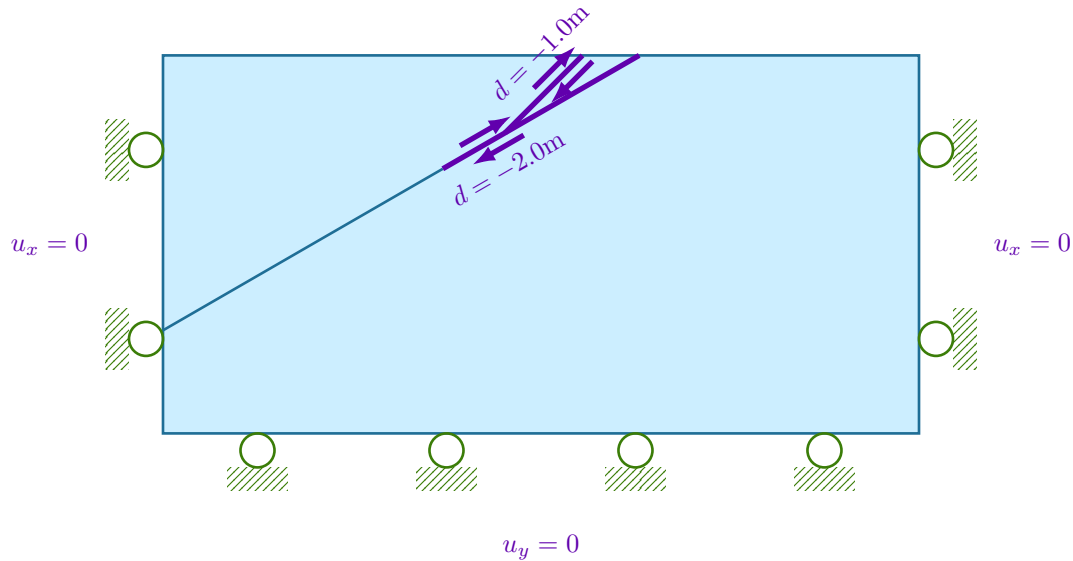


Fig. 4.107: Boundary conditions for static coseismic slip on both the main and splay faults. We prescribe 2 meters of reverse slip on the main fault with 1 meter of reverse slip on the splay fault. We use roller boundary conditions on the lateral sides and bottom of the domain.

Important

In 2D simulations slip is specified in terms of opening and left-lateral components. This provides a consistent, unique sense of slip that is independent of the fault orientation. For our geometry in this example, right lateral slip corresponds to reverse slip on both of the dipping faults.

Important

Both faults contain one end that is buried within the domain. The splay fault ends where it meets the main fault. When PyLith inserts cohesive cells into a mesh with buried edges (in this case a point), we must identify these buried edges so that PyLith properly adjusts the topology along these edges.

For properly topology of the cohesive cells, the main fault *must* be listed first in the array of faults so that it will be created before the splay fault.

We create an array of 2 faults, which are `FaultCohesiveKin` by default, and use `UniformDB` objects to specify uniform reverse slip on each fault.

Listing 4.158: Parameters for prescribed earthquake rupture on the main and splay faults for Step 6.

```
[pylithapp.problem]
interfaces = [fault, splay]

[pylithapp.problem.interfaces.fault]
label = fault
label_value = 20
```

(continues on next page)

(continued from previous page)

```

observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture for main fault
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -2.0*m, 0.0*m]

[pylithapp.problem.interfaces.splay]
label = splay
label_value = 22

observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.splay.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture for splay fault
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -1.0*m, 0.0*m]

```

Running the simulation

Listing 4.159: Run Step 6 simulation

```

$ pylith step06_twofaults_elastic.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.2 time -0.2
  0 SNES Function norm 9.084511105523e+00
    Linear solve converged due to CONVERGED_ATOL iterations 14
  1 SNES Function norm 5.480772341407e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.2 time 0.
  0 SNES Function norm 5.480772341407e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
2 TS dt 0.2 time 0.2
  0 SNES Function norm 3.245094761897e+00
    Linear solve converged due to CONVERGED_ATOL iterations 13
  1 SNES Function norm 3.950115997088e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

```

(continues on next page)

(continued from previous page)

```

3 TS dt 0.2 time 0.4
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

From the end of the output written to the terminal window, we see that the linear solver converged in 13-14 iterations and met the absolute convergence tolerance (`ksp_atol`). As we expect for this linear problem, the nonlinear solver converged in 1 iteration.

Visualizing the results

In Fig. 4.108 we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.160: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filenames=output/step06_twofaults_elastic-domain.h5 warp_grid --component=x
↪ --exaggeration=3000

```

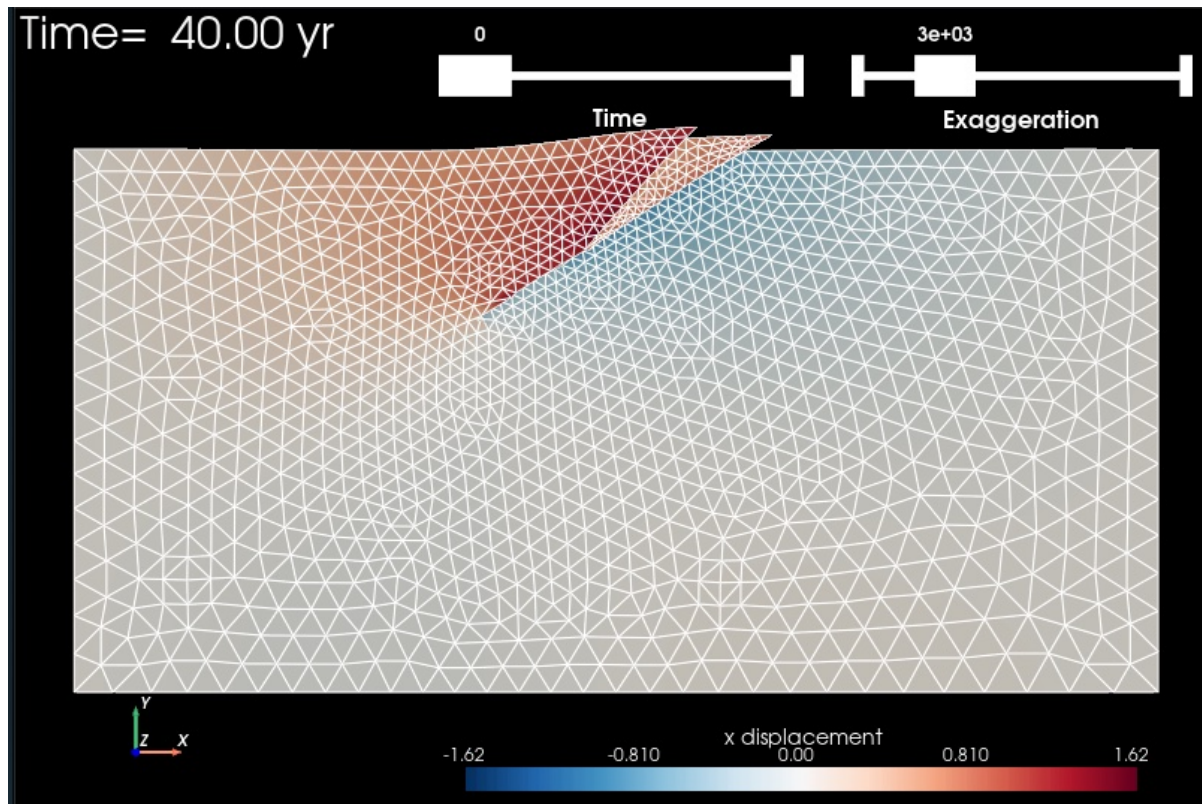


Fig. 4.108: Solution for Step 6 at $t=40$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe.

Step 7: Slip on Two Faults and Maxwell Viscoelastic Materials

Features

- Triangular cells
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.ZeroDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Quasi-static simulation
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearMaxwell
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- spatialdata.spatialdb.UniformDB

Simulation parameters

In this example we replace the linear, isotropic elastic bulk rheology in the slab with a linear, isotropic Maxwell viscoelastic rheology. We use the same boundary conditions as in Step 6 but reduce the time step to resolve the viscoelastic relaxation. The parameters specific to this example are in `step07_twofaults_maxwell.cfg`.

We use a very short relaxation time of 20 years, so we run the simulation for 100 years with a time step of 4 years. We use a starting time of -4 years so that the first time step will advance the solution time to 0 years.

Listing 4.161: Time stepping parameters for Step 7.

```
[pylithapp.problem]
initial_dt = 4.0*year
start_time = -4.0*year
end_time = 100.0*year

scales.relaxation_time = 20.0*year
```

The uniform slip creates a large stress concentration at the end of the fault. We improve the resolution by using uniform global refinement and discretizing the displacement and Lagrange multiplier fields with a basis order of 2.

Listing 4.162: Enabling uniform global refinement and discretizing the displacement and Lagrange multiplier fields with a basis order of 2 in Step 7.

```
[pylithapp.mesh_generator]
refiner = pylith.topology.RefineUniform

[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange
```

(continues on next page)

(continued from previous page)

```
defaults.quadrature_order = 2

[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
lagrange_multiplier_fault.basis_order = 2
```

Listing 4.163: Maxwell viscoelastic bulk rheology parameters for the slab in Step 7.

```
[pylithapp.problem.materials]
slab.bulk_rheology = pylith.materials.IsotropicLinearMaxwell

[pylithapp.problem.materials.slab]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Maxwell viscoelastic properties
db_auxiliary_field.iohandler.filename = mat_maxwell.spatialdb

bulk_rheology.auxiliary_subfields.maxwell_time.basis_order = 0
```

Running the simulation

Listing 4.164: Run Step 7 simulation

```
$ pylith step07_twofaults_maxwell.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --

25 TS dt 0.2 time 4.8
    0 SNES Function norm 3.491930818111e-01
      Linear solve converged due to CONVERGED_ATOL iterations 12
    1 SNES Function norm 6.649621549079e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
26 TS dt 0.2 time 5.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
  ↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

From the end of the output written to the terminal window, we see that the simulation advanced the solution 26 time steps. The PETSc TS display time in the nondimensional units, so a time of 5 corresponds to 100 years.

Visualizing the results

In Fig. 4.109 we use the `pylith_viz` utility to visualize the `x` displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.165: Visualize PyLith output using pylith_viz.

```
pylith_viz --filename=output/step07_twofaults_maxwell-domain.h5 warp_grid --component=x
```

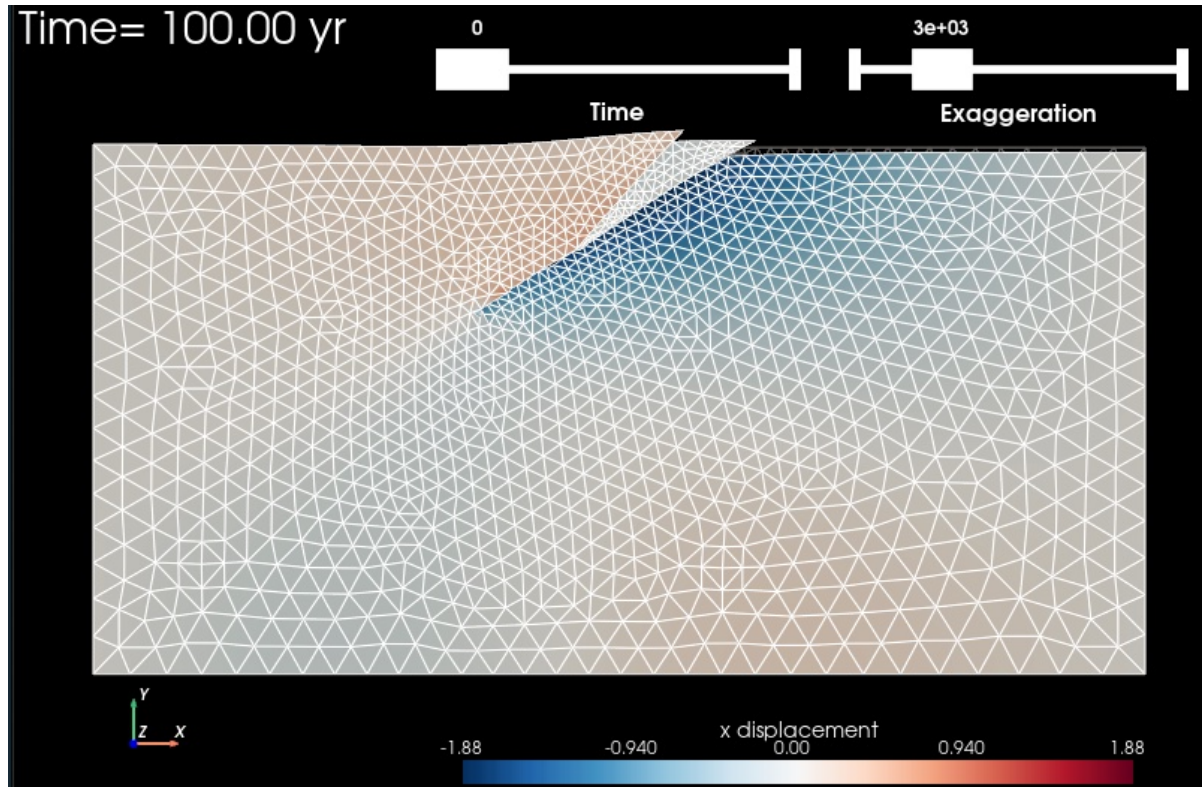


Fig. 4.109: Solution for Step 7 at $t=100$ years. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. Viscoelastic relaxation results in significant deformation in the slab material.

Step 7b: Adaptive Time Stepping

In Step 7b we demonstrate the use adaptive time stepping. We start with an initial time step of 0.2 years and let the adaptive time stepping algorithm increase the time step as the rate of deformation decreases.

Listing 4.166: Run Step 7b simulation

```
$ pylith step07_twofaults_maxwell.cfg step07b_twofaults_maxwell.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:80:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
```

(continues on next page)

(continued from previous page)

```

0 TS dt 0.01 time -0.01
  0 SNES Function norm 9.786738657687e+00
    Linear solve converged due to CONVERGED_ATOL iterations 21
  1 SNES Function norm 4.181718260419e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
  TSAdapt basic beuler 0: step 0 accepted t=-0.01 + 1.000e-02 dt=1.000e-02
1 TS dt 0.01 time 0.
  0 SNES Function norm 6.800013826439e-02
    Linear solve converged due to CONVERGED_ATOL iterations 15
  1 SNES Function norm 4.941010511485e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
  TSAdapt basic beuler 0: step 1 rejected t=0 + 1.000e-02 dt=1.495e-03 wlte=
↪1.79 wltea= -1 wlter= -1
  0 SNES Function norm 3.573809583653e-02
    Linear solve converged due to CONVERGED_ATOL iterations 13
  1 SNES Function norm 3.795363345199e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
  TSAdapt basic beuler 0: step 1 accepted t=0 + 1.495e-03 dt=1.041e-03
↪wlte=0.0824 wltea= -1 wlter= -1

# -- many lines omitted --

30 TS dt 0.363756 time 4.63624
  0 SNES Function norm 5.855825329812e-01
    Linear solve converged due to CONVERGED_ATOL iterations 17
  1 SNES Function norm 7.186563774310e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
  TSAdapt basic beuler 0: step 30 accepted t=4.63624 + 3.638e-01 dt=3.432e-01
↪wlte=0.0449 wltea= -1 wlter= -1
31 TS dt 0.343229 time 5.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

With smaller time steps following the two earthquakes, this simulation takes 5 more time steps than Step 7 with uniform time steps. For simulations covering longer periods of time or a single earthquake, the adaptive time stepping usually results in substantially fewer time steps.

Listing 4.167: Compare the results from Step 7 ad 7b using the `plot_compare.py` Python script.

```
./plot_compare.py --step=7
```

Step 8: Slip on Two Faults and Power-law Viscoelastic Materials

Features

- Triangular cells
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.bc.DirichletTimeDependent`

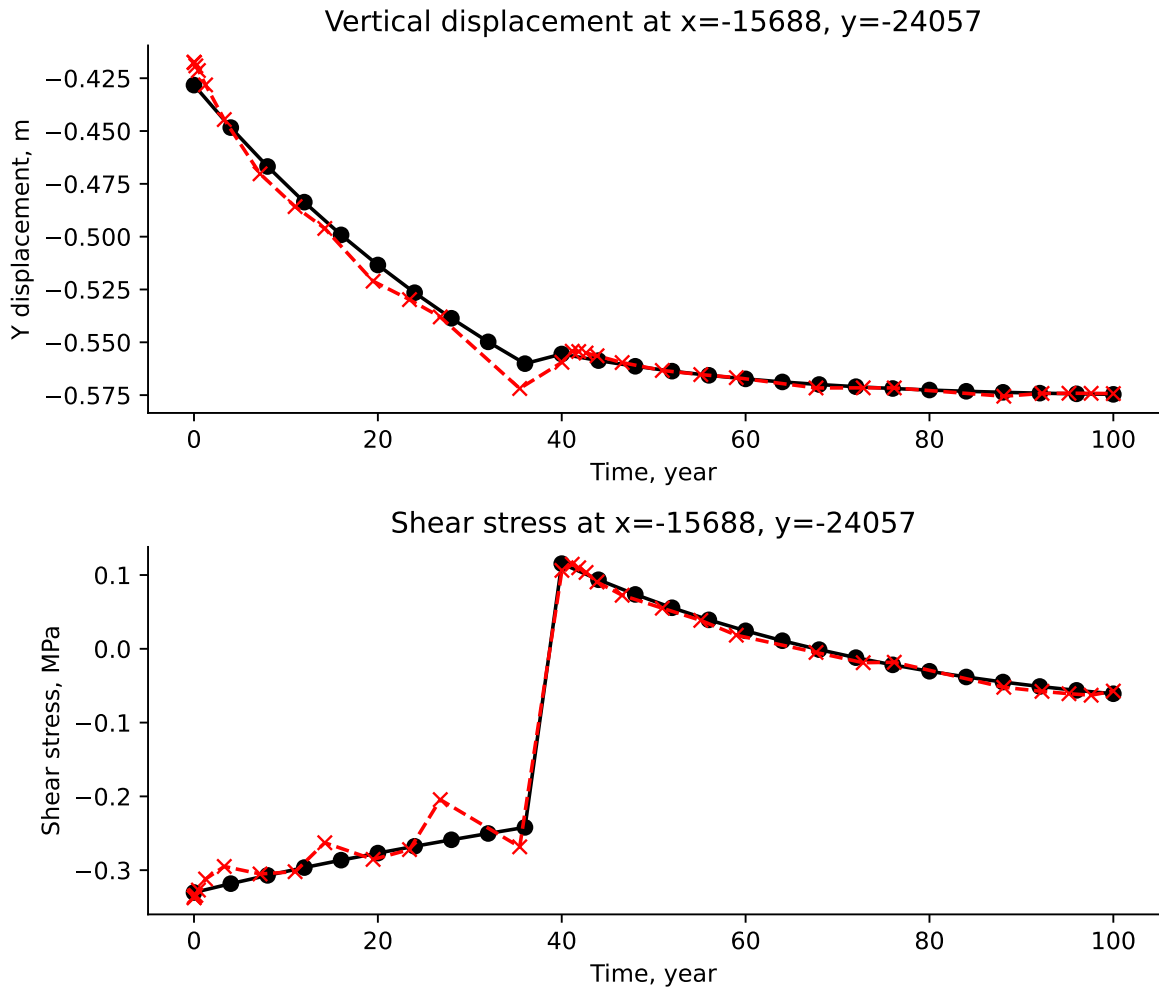


Fig. 4.110: Displacement and shear stress time histories for Step 7 and 7b at a location in the viscoelastic slab below the bottom ends of the main fault and splay fault. The backward Euler time stepping algorithm does a poor job of resolving the rapid deformation following the sudden application of coseismic slip, resulting in some oscillations in the deformation in Step 7b which uses adaptive time stepping.

- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.ZeroDB`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Quasi-static simulation
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicPowerLaw`
- `pylith.faults.FaultCohesiveKin`
- `pylith.faults.KinSrcStep`
- `spatialdata.spatialdb.UniformDB`
- `spatialdata.spatialdb.CompositeDB`

Simulation parameters

In this example we replace the linear, isotropic Maxwell viscoelastic bulk rheology in the slab in Step 7 with an isotropic powerlaw viscoelastic rheology. The other parameters remain the same as those in Step 7. The parameters specific to this example are in `step08_twofaults_powerlaw.cfg`.

Listing 4.168: Power-law viscoelastic bulk rheology parameters for Step 8.

```
[pylithapp.problem.materials]
slab.bulk_rheology = pylith.materials.IsotropicPowerLaw

[pylithapp.problem.materials.slab]
db_auxiliary_field = spatialdata.spatialdb.CompositeDB
db_auxiliary_field.description = Power law material properties

bulk_rheology.auxiliary_subfields.power_law_reference_strain_rate.basis_order = 0
bulk_rheology.auxiliary_subfields.power_law_reference_stress.basis_order = 0
bulk_rheology.auxiliary_subfields.power_law_exponent.basis_order = 0

[pylithapp.problem.materials.slab.db_auxiliary_field]
# Elastic properties
values_A = [density, vs, vp]
db_A = spatialdata.spatialdb.SimpleDB
db_A.description = Elastic properties for slab
db_A.iohandler.filename = mat_elastic.spatialdb

# Power law properties
values_B = [
    power_law_reference_stress, power_law_reference_strain_rate, power_law_
↪exponent,
    viscous_strain_xx, viscous_strain_yy, viscous_strain_zz, viscous_strain_xy,
    reference_stress_xx, reference_stress_yy, reference_stress_zz, reference_stress_
↪xy,
    reference_strain_xx, reference_strain_yy, reference_strain_zz, reference_strain_
↪xy,
    deviatoric_stress_xx, deviatoric_stress_yy, deviatoric_stress_zz, deviatoric_
```

(continues on next page)

(continued from previous page)

```

↪ stress_xy
    ]
db_B = spatialdata.spatialdb.SimpleDB
db_B.description = Material properties specific to power law bulk rheology for the slab
db_B.iohandler.filename = mat_powerlaw.spatialdb
db_B.query_type = linear

```

Power-law spatial database

New in v4.0.0.

We use the utility script `pylith_powerlaw_gendb` (see [pylith_powerlaw_gendb](#)) to generate the spatial database `mat_powerlaw.spatialdb` with the power-law bulk rheology parameters. We provide the parameters for `pylith_powerlaw_gendb` in `powerlaw_gendb.cfg`, which follows the same formatting conventions as the PyLith parameter files.

Listing 4.169: Generate spatial database with power-law viscoelastic material properties.

```
pylith_powerlaw_gendb powerlaw_gendb.cfg
```

Running the simulation

Listing 4.170: Run Step 8 simulation

```

$ pylith step08_twofaults_powerlaw.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.2 time -0.2
  0 SNES Function norm 6.969252188583e+00
    Linear solve converged due to CONVERGED_ATOL iterations 23
  1 SNES Function norm 3.4102000001998e-01
    Linear solve converged due to CONVERGED_ATOL iterations 15
  2 SNES Function norm 1.086370548021e-02
    Linear solve converged due to CONVERGED_ATOL iterations 11
  3 SNES Function norm 3.546734950165e-04
    Linear solve converged due to CONVERGED_ATOL iterations 7
  4 SNES Function norm 1.156742268560e-05
    Linear solve converged due to CONVERGED_ATOL iterations 3
  5 SNES Function norm 3.793910664255e-07
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 5

# -- many lines omitted --

```

(continues on next page)

(continued from previous page)

```

25 TS dt 0.2 time 4.8
   0 SNES Function norm 8.706014661840e-02
     Linear solve converged due to CONVERGED_ATOL iterations 15
   1 SNES Function norm 1.154191339686e-03
     Linear solve converged due to CONVERGED_ATOL iterations 9
   2 SNES Function norm 2.319736370610e-05
     Linear solve converged due to CONVERGED_ATOL iterations 4
   3 SNES Function norm 5.416653473355e-07
     Linear solve converged due to CONVERGED_ATOL iterations 1
   4 SNES Function norm 3.211036236790e-08
     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 4
26 TS dt 0.2 time 5.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

As in Step 7, the simulation advances 26 time steps. With a nonlinear bulk rheology, the nonlinear solver now requires a few iterations to converge at each time step.

Visualizing the results

In Fig. 4.111 we use the `pylith_viz` utility to visualize the `x` displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.171: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step08_twofaults_powerlaw-domain.h5 warp_grid --component=x
```

Step 8b: Adaptive Time Stepping

In Step 8b we demonstrate the use adaptive time stepping. We start with an initial time step of 0.2 years and let the adaptive time stepping algorithm increase the time step as the rate of deformation decreases.

Listing 4.172: Run Step 8b simulation

```

$ pylith step08_twofaults_maxwell.cfg step08b_twofaults_maxwell.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.01 time -0.01
   0 SNES Function norm 6.969252188583e+00
     Linear solve converged due to CONVERGED_ATOL iterations 23

```

(continues on next page)

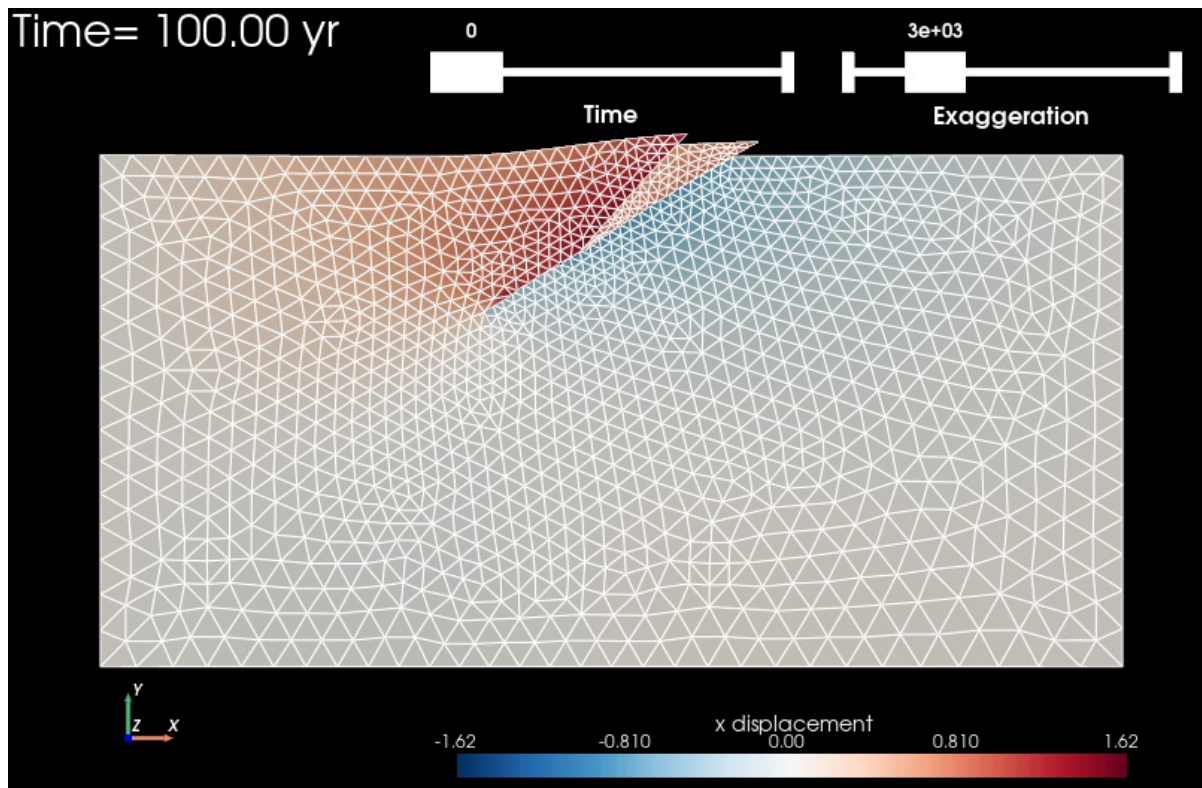


Fig. 4.111: Solution for Step 8 at $t=100$ years. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. Our parameters for the power-law bulk rheology result in much less viscoelastic relaxation in this case compared to Step 7.

(continued from previous page)

```

1 SNES Function norm 2.536646382544e-02
  Linear solve converged due to CONVERGED_ATOL iterations 11
2 SNES Function norm 8.123236908769e-05
  Linear solve converged due to CONVERGED_ATOL iterations 5
3 SNES Function norm 2.954288172419e-07
  Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 3
TSAdapt basic beuler 0: step 0 accepted t=-0.01 + 1.000e-02 dt=1.000e-02

# -- many lines omitted --

19 TS dt 0.776784 time 4.22322
  0 SNES Function norm 3.212286568880e-01
    Linear solve converged due to CONVERGED_ATOL iterations 17
  1 SNES Function norm 8.434433734286e-03
    Linear solve converged due to CONVERGED_ATOL iterations 12
  2 SNES Function norm 2.503978690373e-04
    Linear solve converged due to CONVERGED_ATOL iterations 7
  3 SNES Function norm 9.325515431121e-06
    Linear solve converged due to CONVERGED_ATOL iterations 3
  4 SNES Function norm 3.575321328214e-07
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 4
  TSAdapt basic beuler 0: step 19 accepted t=4.22322 + 7.768e-01 dt=2.634e+00
↳ wlte=0.00348 wltea= -1 wlter= -1
20 TS dt 2.63361 time 5.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The deformation rate is smoother with the power-law rheology yielding more efficient results with the adaptive time stepper which uses 6 fewer time steps compared with the uniform time step of 4 years.

Listing 4.173: Compare the results from Step 8 ad 8b using the `plot_compare.py` Python script.

```
./plot_compare.py --step=8
```

Suggested Exercises

1. Change the reference stress state in Step 2 by editing the spatial database. How does this affect the solution?
2. Create `.cfg` files to run the simulations with a Cubit mesh.
3. Create additional spatial database files so that the material properties for each material are different. Rerun Steps 2 and 3. How does this affect the solution?
4. Adjust the distribution of surface tractions in Step 4.
5. Experiment with refining the mesh and basis orders of 1 or 2 for the solution subfields. What combination gives the highest accuracy with a reasonable simulation runtime?
6. Adjust the timing of the earthquake ruptures in Steps 6-8.

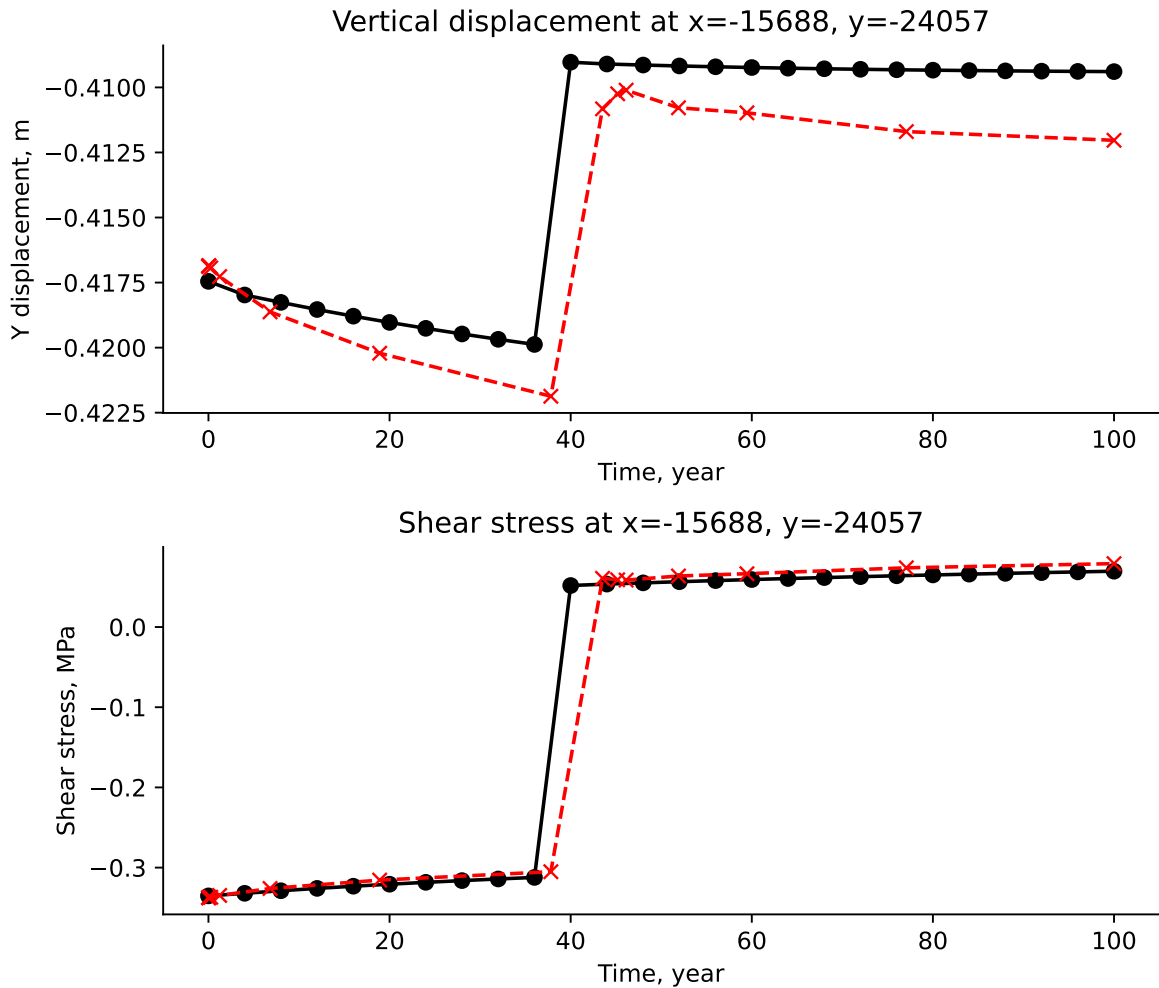


Fig. 4.112: Displacement and shear stress time histories for Step 8 and 8b at a location in the viscoelastic slab below the bottom ends of the main fault and splay fault. The adaptive time stepping does a better job of capturing the variable rate of deformation associated with the power-law viscoelastic rheology; however, the simulation steps over the time of the earthquake on the splay fault before reducing the time step.

Horizontal Cross-Section of Intersecting Strike-Slip Faults (2D)

New in v4.1.0.

The files are in the directory `examples/crustal-strikeslip-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

faulttrace*_latlon.txt

Coordinates of points on fault traces in geographic coordinates.

faulttrace*_utm.txt

Coordinates of points on fault traces in the UTM zone 11 projection.

generate_gmsh.py

Python script to generate the finite-element mesh using Gmsh.

generate_cubit.py

Python script to generate the finite-element mesh using Cubit.

***.msh**

Gmsh finite-element mesh files generated by Gmsh.

***.exo**

Exodus II finite-element mesh files generated by Cubit.

***.spatialdb**

Spatial database filesFiles associated with the spatial databases.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This suite of examples demonstrates using PyLith to model multiple intersecting faults embedded in a 2D georeferenced domain (Fig. 4.113). The geometry is based on studies of the 2019 Ridgecrest earthquake sequence by Uphoff *et al.* [2022] and Nevitt *et al.* [2023].

Step 1

Static uniform coseismic slip with Dirichlet (displacement) boundary conditions.

Step 2

Static spatially varying coseismic slip with Dirichlet (displacement) boundary conditions.

Important

We describe how to generate the finite-element mesh using both Gmsh and Cubit. For Step 1 we provide PyLith parameter files for both meshes; for Step 2 we only provide the parameter files that use the Gmsh file.

Example Workflow

Gmsh Mesh

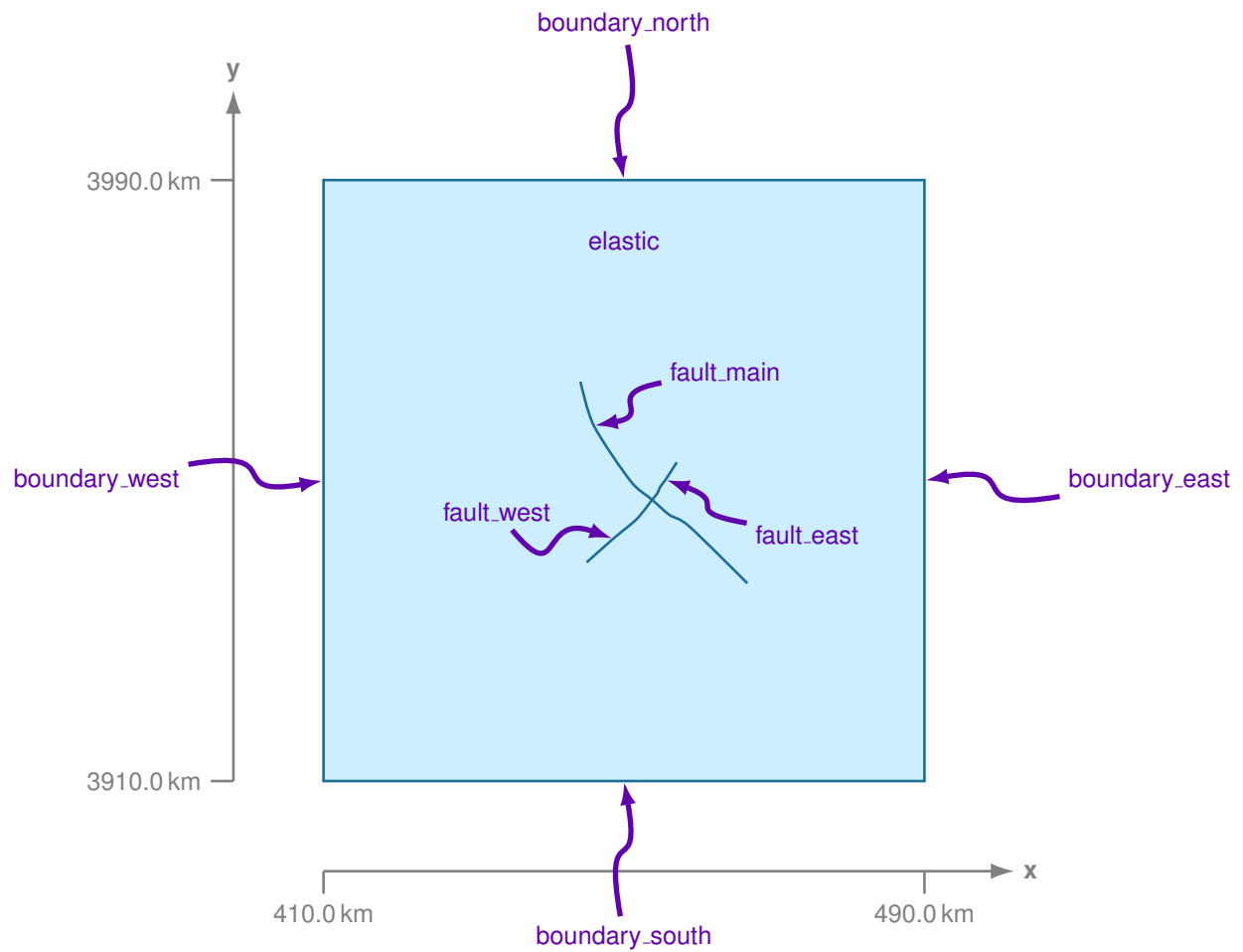


Fig. 4.113: Diagram of geometry for the domain with three intersecting strike-slip faults. The domain extends from 80 km in the east-west direction and 80 km in the north-south direction. We refer to the domain boundaries using the names shown in the diagram.

Geometry

We construct the geometry by first creating points, then connecting the points into curves, and finally the curves into a surface. Fig. 4.114 shows the geometry and variables names of the vertices and curves.

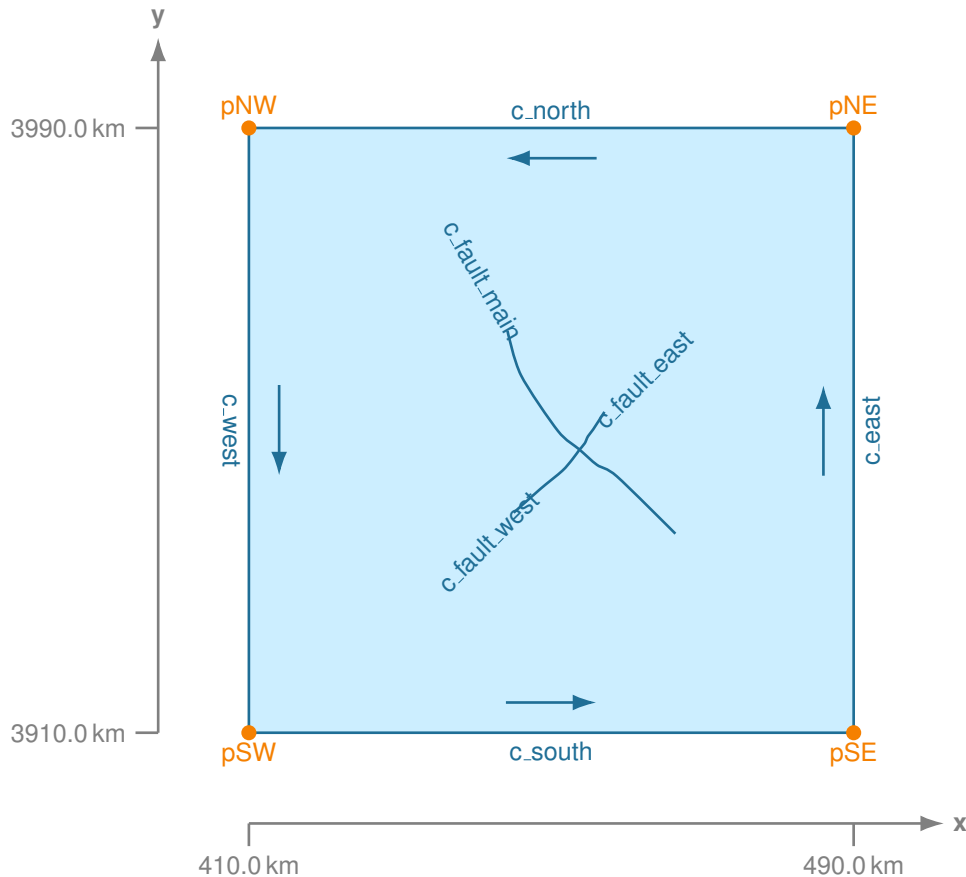


Fig. 4.114: Geometry created in Gmsh for generating the finite-element mesh. We construct curves from points (pSW, ..., pNW) and surfaces from the curves (for example, c_north). The arrows indicate the direction (orientation) of the curves.

Important

Each curve in Gmsh has a direction (orientation). The direction is from the starting point to the ending point. When connecting curves into surfaces, you must connect the curves in a consistent direction. We connect the curves in a counter-clockwise direction. To reverse the direction of a curve, use the negative tag.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. The script makes use of the `gmsh_utils.GenerateMesh` class (discussed in [gmsh_utils](#)), which provides the command line arguments and boilerplate methods. In our `generate_gmsh.py` Python script, we create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class.

Important

In order for Gmsh to create cells with edges that align with the fault traces, we must embed the curves into the surface.

Additionally, we must split the main fault trace into two so that there is a common point among all three fault traces where they intersect.

We use the Gmsh MeshSize options to define a discretization size that grows slowly at a geometric rate with distance from the fault. See 6.3.1 [Specifying mesh element sizes in the Gmsh documentation](#) for more information.

Listing 4.174: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with triangular cells and save it to `mesh_tri.msh` (default filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui

# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui

# Show available command line arguments.
$ ./generate_gmsh.py --help
```

By default the Python script will generate a finite-element mesh with triangular cells and save it to the file `mesh_tri.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.175: View the Gmsh mesh file `mesh_tri.msh` using Gmsh.

```
gmsh -open mesh_tri.msh
```

Cubit Mesh

Geometry

We construct the geometry by first creating points, then connecting the points into curves, and finally the curves into a surface. [Fig. 4.116](#) shows the geometry and variables names of the vertices and curves.

Meshing using Python Script

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

Open the Python script `generate_cubit.py` in the Cubit journal editor. Play the selected script or play the lines, making sure you play the first line so that Cubit uses the Python interpreter when running the script. We specify the parameters controlling the geometry, mesh size, and cell shape near the top of the script.

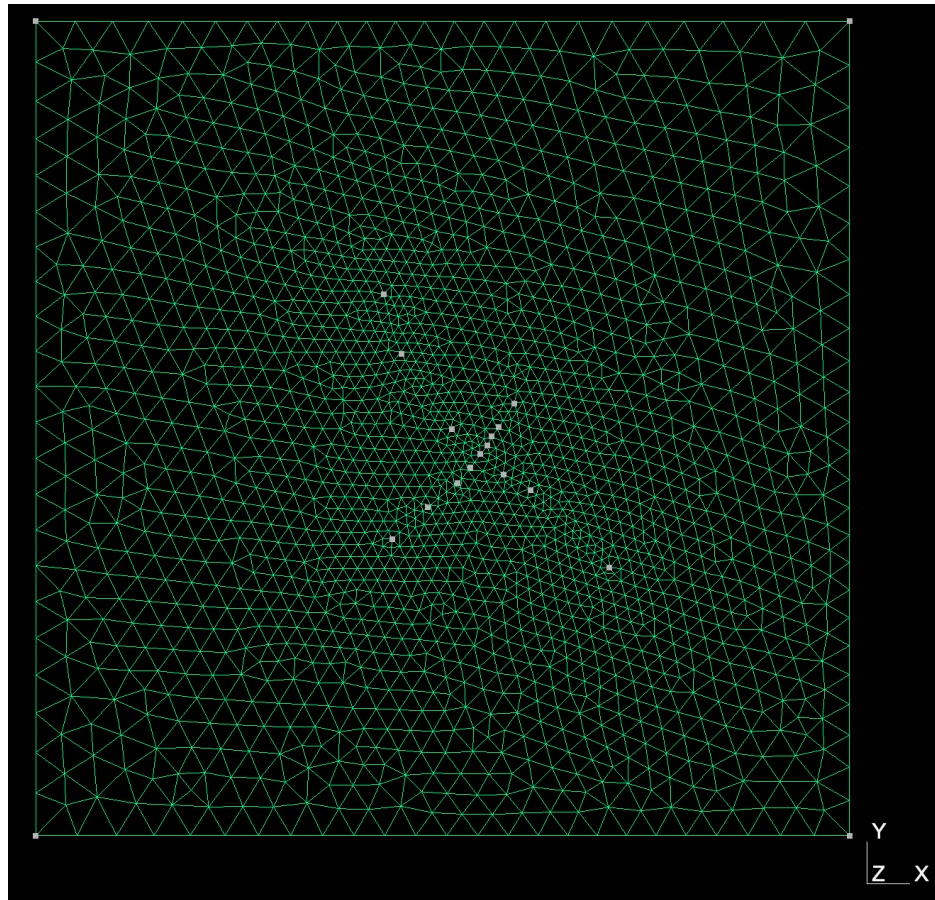


Fig. 4.115: Finite-element mesh with triangular cells generated by Gmsh.

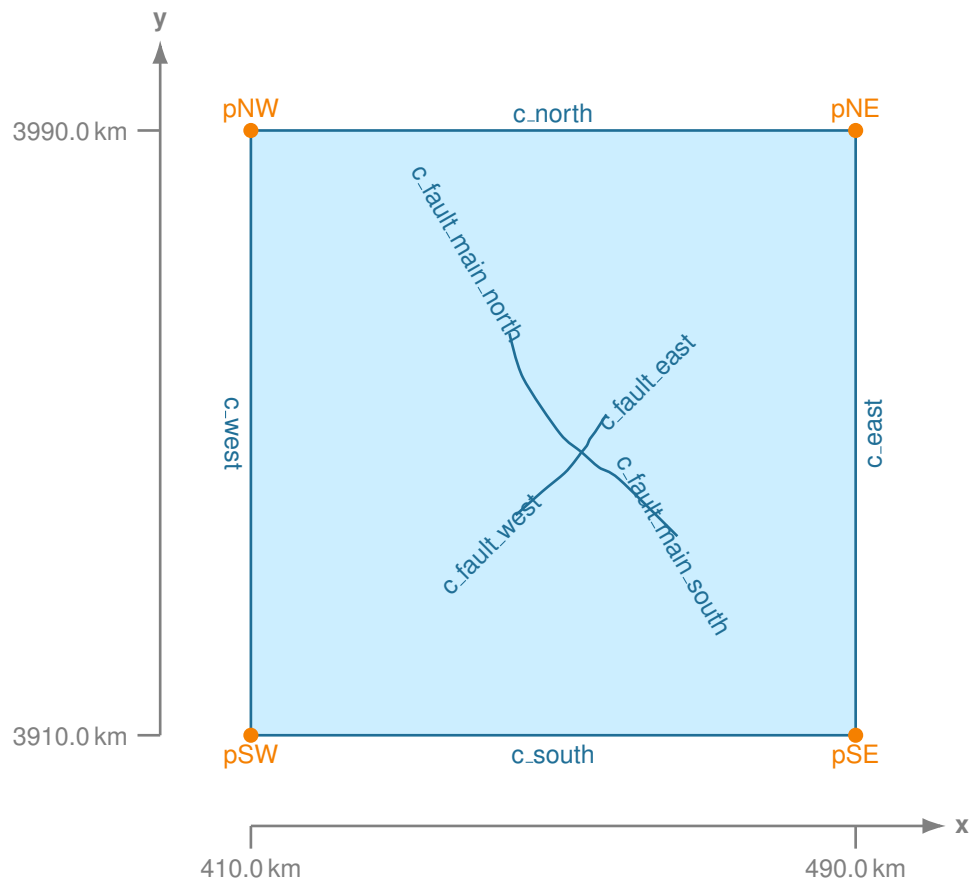


Fig. 4.116: Geometry created in Cubit for generating the finite-element mesh. The names of the vertices and curves match the ones we use in the Cubit Python script.

Important

In most cases, we name the geometric entity immediately after creating it, so that we can refer to it later. There are a few cases, such as splitting curves, in which we do not know the ids of the newly created entities. In these cases we use the GUI to identify the curves and use special Cubit functions that allow us to specify a geometric entity based on location. This results in Cubit commands that are independent of the version of the underlying geometry libraries.

Note

We use the skeleton sizing function to set cell sizes that increases at a geometric rate with distance from the fault. This makes it easy to specify a fine cell size near the fault where the solution has greater spatial variability and a coarse cell size near the boundaries where the solution is very smooth.

After running the Python script in Cubit, you will have a corresponding Exodus-II file (`mesh_tri.exo`). This is a NetCDF file, and it can be loaded into ParaView.

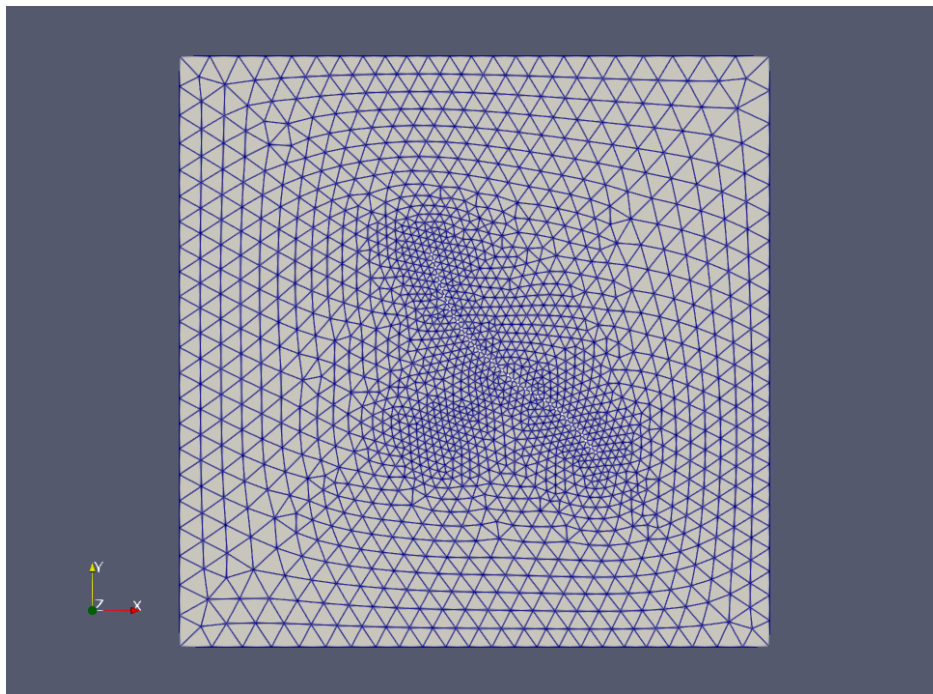


Fig. 4.117: Finite-element mesh with triangular cells generated by Cubit (viewed using Cubit).

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`, which contains numerous comments, so we only summarize the parameters here.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh in `pylithapp.mesh_generator`.

Important

We specify the geographic projection that we used when we constructed the finite-element mesh. This georeferences the model and enables us to specify points in other geographic coordinates system in spatial databases, which may be more convenient.

Listing 4.176: Parameters for the mesh input file and the model coordinate system.

```
[pylithapp.mesh_generator]
reader = pylith.meshio.MeshIO Petsc
reader.filename = mesh_tri.msh

# Model coordinate system is UTM zone 11
reader.coordsys = spatialdata.geocoords.CSGeo
reader.coordsys.crs_string = EPSG:32611
reader.coordsys.space_dim = 2
```

Physics

These quasi-static simulations solve the elasticity equation and include a fault, so we have a solution field with both displacement and Lagrange multiplier subfields.

$$\vec{s} = \begin{pmatrix} \vec{u} & \vec{\lambda} \end{pmatrix}^T \quad (4.485)$$

$$\nabla \cdot \sigma(\vec{u}) = \vec{0} \quad (4.486)$$

We use the default `TimeDependent` problem and solution field with a single displacement subfield of basis order 1.

Listing 4.177: Parameters for the solution.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange
```

We use the same material properties in all of the simulations in this directory, so we specify them in `pylithapp.cfg` to avoid repeating the information in the file with parameters for each simulation. We have one material with uniform material properties.

Listing 4.178: Material parameters common to all simulations in this directory.

```
[pylithapp.problem]
materials = [elastic]

[pylithapp.problem.materials.elastic]
# We use the default material (elasticity) and rheology
# (isotropic, linearly elastic).
#
```

(continues on next page)

(continued from previous page)

```

# `label_value` must match the tag for the physical group in the Gmsh Python script.
label_value = 1

# The properties are uniform within the material, so we use a UniformDB.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Elastic properties xneg
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500.0*kg/m**3, 3.00*km/s, 5.29*km/s]

# The properties are uniform, so we use a basis order of 0, corresponding
# to uniform properties within a cell.
auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0

# We discretize the displacement field with a basis order of 1
# so the stress and strain computed from the displacement field
# will have an accuracy of one order lower. Consequently, we use
# a basis order of 0.
derived_subfields.cauchy_strain.basis_order = 0
derived_subfields.cauchy_stress.basis_order = 0

```

Similarly, we set the fault parameters common to all simulations in the directory. For these simulations, we have three faults that intersect at a common point.

Listing 4.179: General fault parameters common to all simulations in this directory.

```

[pylithapp.problem]
interfaces = [main_fault, west_branch, east_branch]

## main fault
[pylithapp.problem.interfaces.main_fault]
# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
label = fault_main
label_value = 20

# Output `slip` and `traction_changes` on the fault.
observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.west_branch]
# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
label = fault_west
label_value = 21

# Output `slip` and `traction_changes` on the fault.
observers.observer.data_fields = [slip, traction_change]

```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.interfaces.east_branch]
# The `label` and `label_value` correspond to the name and tag of the
## physical group in the Gmsh Python script.
label = fault_east
label_value = 22

# Output `slip` and `traction_changes` on the fault.
observers.observer.data_fields = [slip, traction_change]
```

All simulations in this directory use Dirichlet boundary conditions on the boundaries of the domain with zero displacements in which we constrain the displacement component perpendicular to the boundary.

Listing 4.180: Dirichlet boundary conditions common to all simulations in this directory.

```
[pylithapp.problem]
bc = [bc_south, bc_east, bc_north, bc_west]
bc.bc_south = pylith.bc.DirichletTimeDependent
bc.bc_east = pylith.bc.DirichletTimeDependent
bc.bc_north = pylith.bc.DirichletTimeDependent
bc.bc_south = pylith.bc.DirichletTimeDependent

# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
#
# We constrain the displacement component normal to each boundary.
# We use the specialized `ZeroDB` to specify zero values for the Dirichlet
# BC. We will override this parameter in some of the .cfg files to specify
# nonzero values.
[pylithapp.problem.bc.bc_south]
label = boundary_south
label_value = 10
constrained_dof = [1]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on south boundary

[pylithapp.problem.bc.bc_east]
label = boundary_east
label_value = 11
constrained_dof = [0]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on east boundary

[pylithapp.problem.bc.bc_north]
label = boundary_north
label_value = 12
constrained_dof = [1]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on north boundary

[pylithapp.problem.bc.bc_west]
label = boundary_west
```

(continues on next page)

(continued from previous page)

```
label_value = 13
constrained_dof = [0]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on west boundary
```

Step 1: Static Uniform Coseismic Slip

Features

- Triangular cells
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- pylith.meshio.MeshIOPetsc
- Gmsh

Simulation parameters

This example involves a static simulation that solves for the deformation from prescribed uniform coseismic slip on each of the faults. Fig. 4.118 shows the boundary conditions on the domain. The parameters specific to this example are in `step01_slip.cfg`.

Listing 4.181: Prescribed slip parameters for Step 1.

```
[pylithapp.problem.interfaces.main_fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -4.0*m, 0.0*m]

[pylithapp.problem.interfaces.west_branch.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, 3.0*m, 0.0*m]

[pylithapp.problem.interfaces.east_branch.eq_ruptures.rupture]
```

(continues on next page)

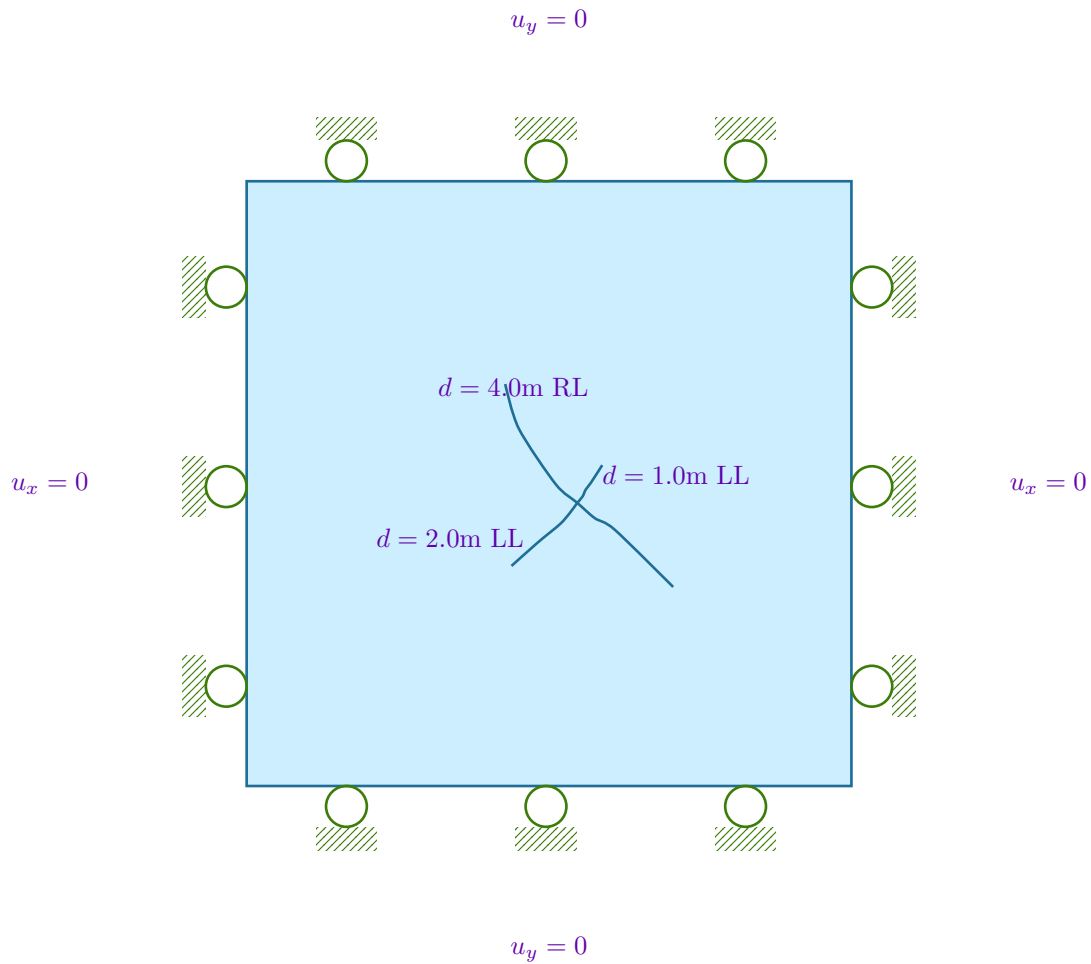


Fig. 4.118: Boundary conditions for static coseismic slip. On the boundary of the domain, we set the displacement component that is perpendicular to the boundary to zero. We prescribe 4.0 m of right-lateral slip on the main fault, 2.0 m of left-lateral slip on the west branch, and 1.0 m of left-lateral slip on the east branch.

(continued from previous page)

```

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, 1.0*m, 0.0*m]

```

Running the simulation

Listing 4.182: Run Step 1 simulation

```

$ pylith step01_slip.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↪ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    dm_reorder_section = true
    dm_reorder_section_type = cohesive
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    mg_fine_pc_type = vpbjacobi
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void_
↪ pylith::meshio::MeshIOPetsc::_read()
-- info (application-flow)
-- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
↪ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (410000, 490000)

```

(continues on next page)

(continued from previous page)

```

(3.91e+06, 3.99e+06)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
↳ pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
↳ printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 1000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↳ pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 2.419639130581e+01
      Linear solve converged due to CONVERGED_ATOL iterations 13
    1 SNES Function norm 6.875321480887e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/Utils/PetscManager.
↳ py:60:finalize
-- info (application-flow)
-- Finalizing PETSc.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the `MeshIOPetsc` reader and that it found the domain to extend from 410000 to 490000 in the x direction and from 3.91e+06 to 3.99e+06 in the y direction. PyLith detects the presence of a fault based on the Lagrange multiplier for the fault in the solution field and selects appropriate preconditioning options as discussed in *PETSc Options*. The scales for nondimensionalization remain the default values for a quasistatic problem.

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 13 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snestol`).

Earthquake rupture parameters

We use the `pylith_eqinfo` utility to compute rupture information, such as earthquake magnitude, seismic moment, seismic potency, and average slip. For 2D simulations, the average slip provides the most useful information. `pylith_eqinfo` is a Pyre application and you specify parameters using `cfg` files and the command line. It writes results to a Python script for use in post-processing of simulation results.

The file `eqinfoapp.cfg` holds the parameters for `pylith_eqinfo` in this example. By default, `pylith_eqinfo` extracts information for the final time step in the output file. In order to compute the seismic moment and moment magnitude, we need the shear modulus, which in this case is uniform over the domain. Consequently, we specify the density and shear wave speed using a `UniformDB` in `eqinfoapp.cfg`.

Listing 4.183: Run `pylith_eqinfo` for the Step 1 simulation.

```
$ pylith_eqinfo
```

Listing 4.184: Contents of `output/step01_slip-eqinfo.py` generated by `pylith_eqinfo`. The `all` object includes earthquake rupture information combined from the three the individual faults. The average slip matches the uniform slip prescribed on each fault.

```
class RuptureStats(object):
    pass
all = RuptureStats()
all.timestamp = [ 3.155760e+07]
all.ruparea = [ 5.387992e+04]
all.potency = [ 1.733080e+05]
all.moment = [ 3.899430e+15]
all.avgslip = [ 3.216560e+00]
all.mommag = [ 4.360667e+00]
main_fault = RuptureStats()
main_fault.timestamp = [ 3.155760e+07]
main_fault.ruparea = [ 3.577375e+04]
main_fault.potency = [ 1.430950e+05]
main_fault.moment = [ 3.219637e+15]
main_fault.avgslip = [ 4.000000e+00]
main_fault.mommag = [ 4.305205e+00]
east_branch = RuptureStats()
east_branch.timestamp = [ 3.155760e+07]
east_branch.ruparea = [ 5.999357e+03]
east_branch.potency = [ 5.999357e+03]
east_branch.moment = [ 1.349855e+14]
east_branch.avgslip = [ 1.000000e+00]
east_branch.mommag = [ 3.386858e+00]
west_branch = RuptureStats()
west_branch.timestamp = [ 3.155760e+07]
west_branch.ruparea = [ 1.210682e+04]
west_branch.potency = [ 2.421364e+04]
west_branch.moment = [ 5.448068e+14]
west_branch.avgslip = [ 2.000000e+00]
west_branch.mommag = [ 3.790828e+00]
```

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (.h5) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (.xmf) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In Fig. 4.119 we use the `pylith_viz` utility to visualize the `y` displacement field.

Listing 4.185: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step01_slip-domain.h5 warp_grid --component=y
```

Step 1 with Cubit Mesh

Using the Cubit mesh rather than the Gmsh mesh involves two changes:

1. Use the `MeshIOCubit` reader instead of the `MeshIOPetsc` reader and change the filename of the mesh file.
2. Set the `label_value` to 1 for boundary conditions and faults.
We must override the `label_value` settings in `pylithapp.cfg` that were appropriate for our Gmsh reader but are incorrect for the Cubit reader.

The file `step01_slip_cubit.cfg` provides these changes and updates the names for output.

Listing 4.186: Run Step 1 simulation with the Cubit mesh

```
$ pylith step01_slip_cubit.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:170:virtual void_
↪pylith::meshio::MeshIOCubit::_read()
-- info (application-flow)
-- Component 'meshiocubit.reader': Reading finite-element mesh from 'mesh_tri.exo'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:207:virtual void_
↪pylith::meshio::MeshIOCubit::_read()
-- info (application-flow)
-- Component 'meshiocubit.reader': Read 3125 cells and 1610 vertices.

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
  0 SNES Function norm 2.428257473425e+01
    Linear solve converged due to CONVERGED_ATOL iterations 14
  1 SNES Function norm 1.922622064928e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
```

(continues on next page)

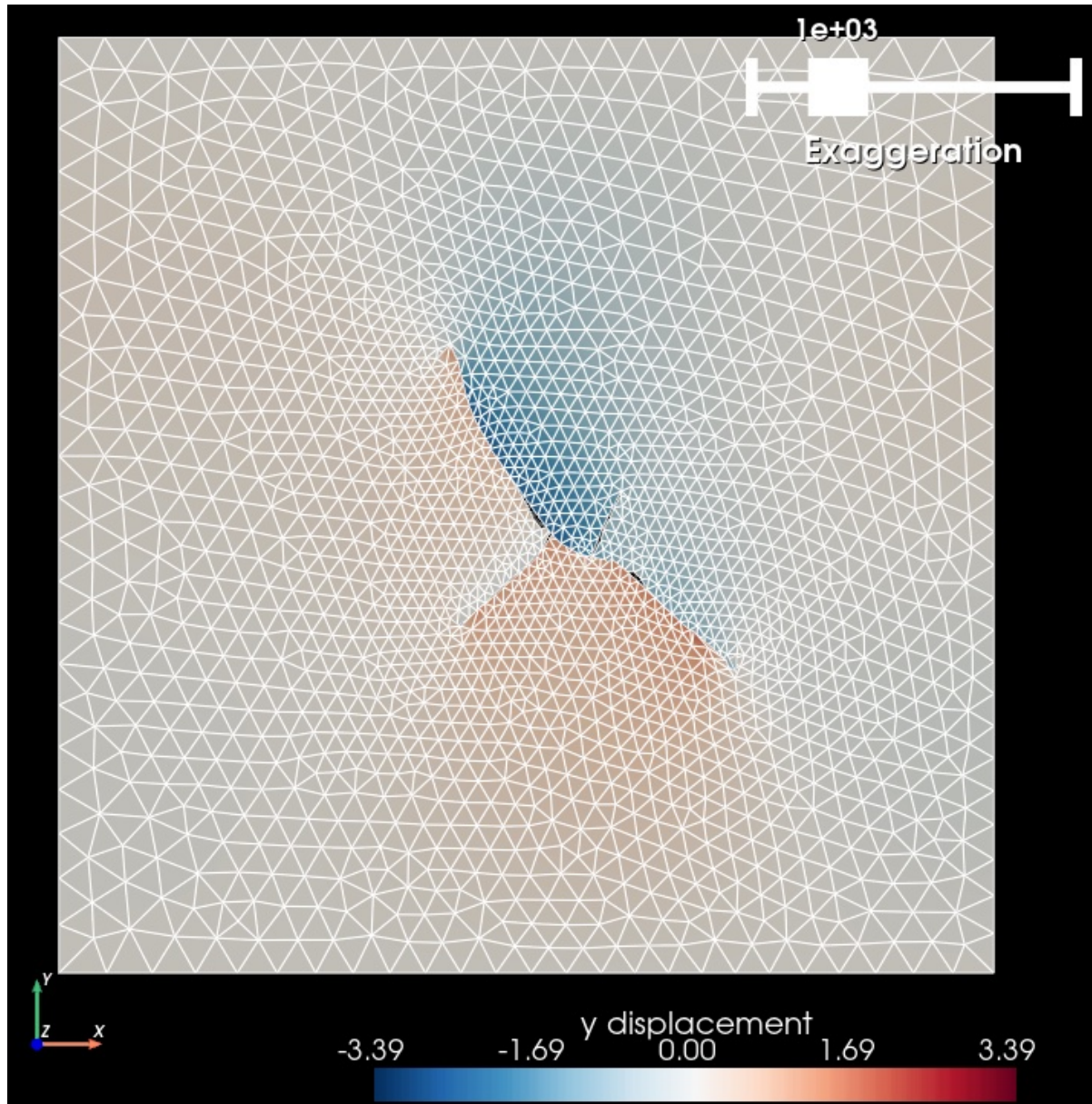


Fig. 4.119: Solution for Step 1. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The contrast in material properties across the faults causes the asymmetry in the y displacement field.

(continued from previous page)

```
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

Step 2: Static Spatially Variable Coseismic Slip

Features

- Triangular cells
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- pylith.meshio.MeshIOPetsc
- Gmsh
- spatially variable slip

Simulation parameters

This example involves a static simulation that solves for the deformation from prescribed spatially variable coseismic slip on each of the faults. Fig. 4.120 shows the boundary conditions on the domain. The parameters specific to this example are in `step02_varslip.cfg`.

We prescribe piecewise linear variations in slip along strike on each of the faults using `SimpleDB` spatial databases. The spatial databases contain coordinates of points in geographic coordinates. PyLith will transform each the coordinates of point on the fault to the geographic coordinates and interpolate the values in the spatial database to the point on the fault.

Listing 4.187: Prescribed slip parameters for Step 2.

```
[pylithapp.problem.interfaces.main_fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Slip parameters for fault 'main'
db_auxiliary_field.iohandler.filename = fault_main_slip.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.interfaces.west_branch.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Slip parameters for fault 'west'
```

(continues on next page)

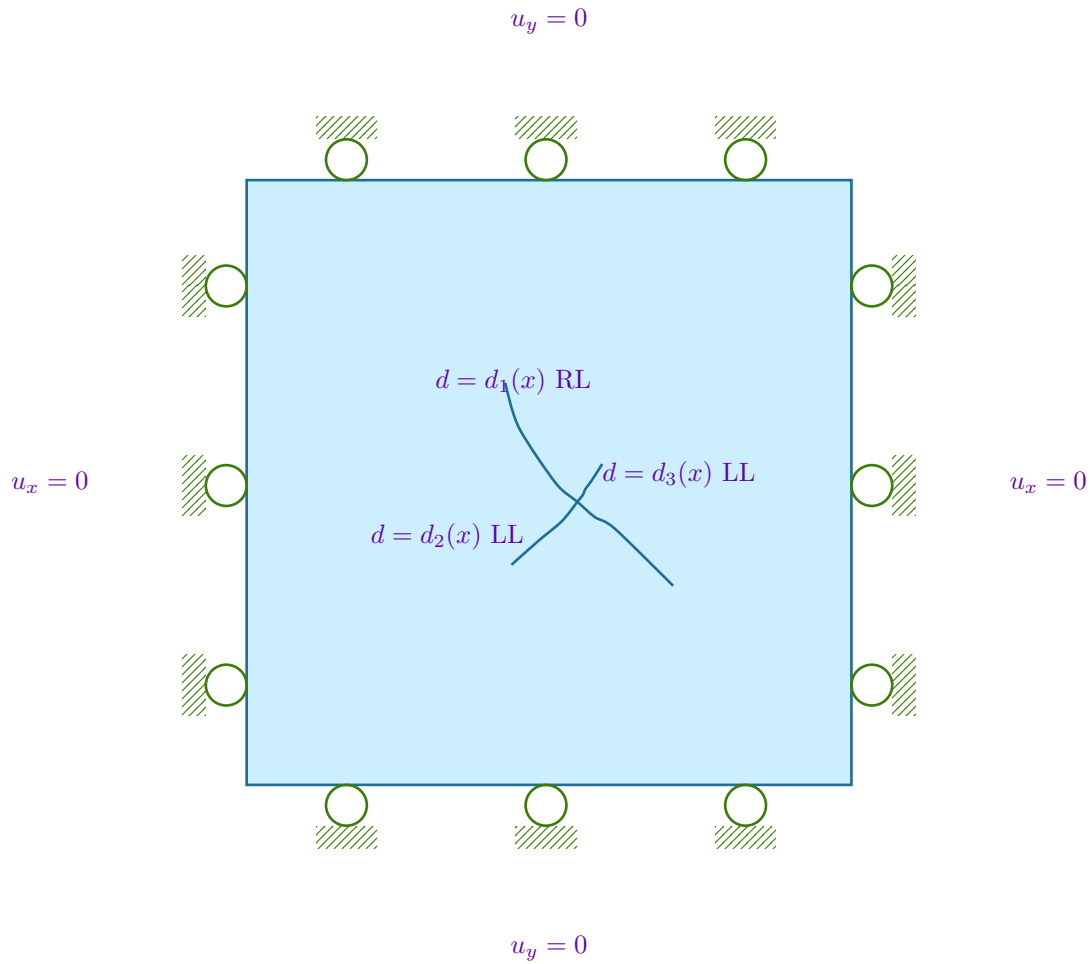


Fig. 4.120: Boundary conditions for static coseismic slip. On the boundary of the domain, we set the displacement component that is perpendicular to the boundary to zero. We prescribe piecewise linear variations in slip along strike on each of the faults.

(continued from previous page)

```

db_auxiliary_field.iohandler.filename = fault_west_slip.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.interfaces.east_branch.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Slip parameters for fault 'east'
db_auxiliary_field.iohandler.filename = fault_east_slip.spatialdb
db_auxiliary_field.query_type = linear

```

Running the simulation

Listing 4.188: Run Step 2 simulation

```

$ pylith step02_varslip.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines removed --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
  0 SNES Function norm 1.139546193525e+01
    Linear solve converged due to CONVERGED_ATOL iterations 13
  1 SNES Function norm 3.278661158667e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 13 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`).

Visualizing the results

In Fig. 4.121 we use the `pylith_viz` utility to visualize the y displacement field.

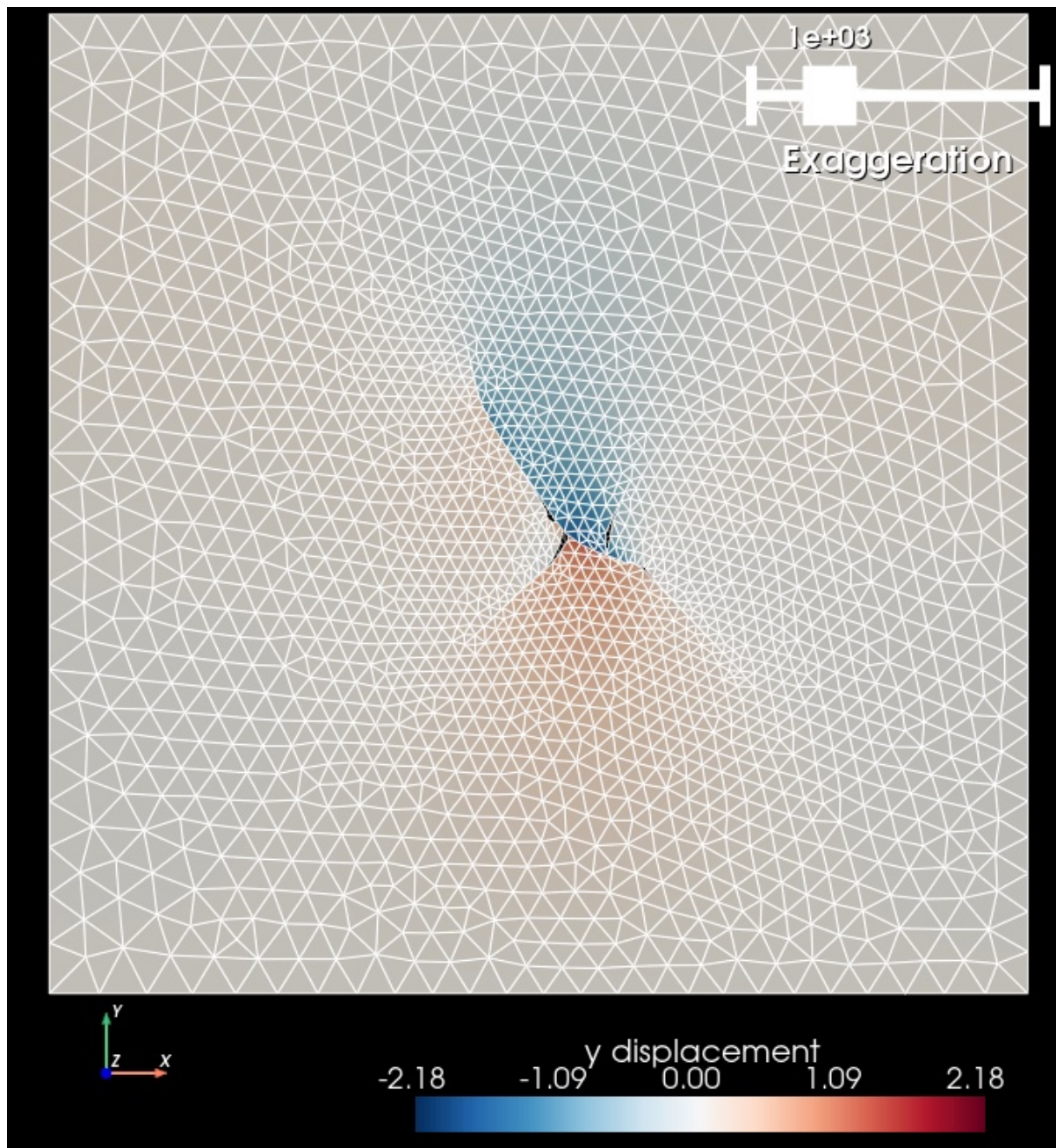


Fig. 4.121: Solution for Step 2. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The contrast in material properties across the faults causes the asymmetry in the y displacement field.

Listing 4.189: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step02_varslip-domain.h5 warp_grid --component=y
```

Suggested Exercises

1. Adjust the discretization size on the fault and the rate at which it increases with distance from the fault.
2. Create a `.cfg` file to run Step 2 with the Cubit mesh.
3. Add additional complexity to the distribution of slip in Step 2.
4. Create a time-dependent problem with earthquake ruptures at different origin times.

Intersecting Strike-Slip Faults (3D)

New in v4.1.0.

The files are in the directory `examples/crustal-strikeslip-3d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

faulttrace*_latlon.txt

Coordinates of points on fault traces in geographic coordinates.

faulttrace*_utm.txt

Coordinates of points on fault traces in the UTM zone 11 projection.

generate_gmsh.py

Python script to generate the finite-element mesh using Gmsh.

generate_cubit.py

Python script to generate the finite-element mesh using Cubit.

***.msh**

Gmsh finite-element mesh files generated by Gmsh.

***.spatialdb**

Spatial database files associated with the spatial databases.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This suite of examples demonstrates using PyLith to model multiple intersecting faults embedded in a 3D georeferenced domain (Fig. 4.122). The geometry is based on studies of the 2019 Ridgecrest earthquake sequence by Uphoff *et al.* [2022] and Nevitt *et al.* [2023].

Step 1

Static uniform coseismic slip with Dirichlet (displacement) boundary conditions.

Step 2

Static spatially varying coseismic slip with Dirichlet (displacement) boundary conditions.

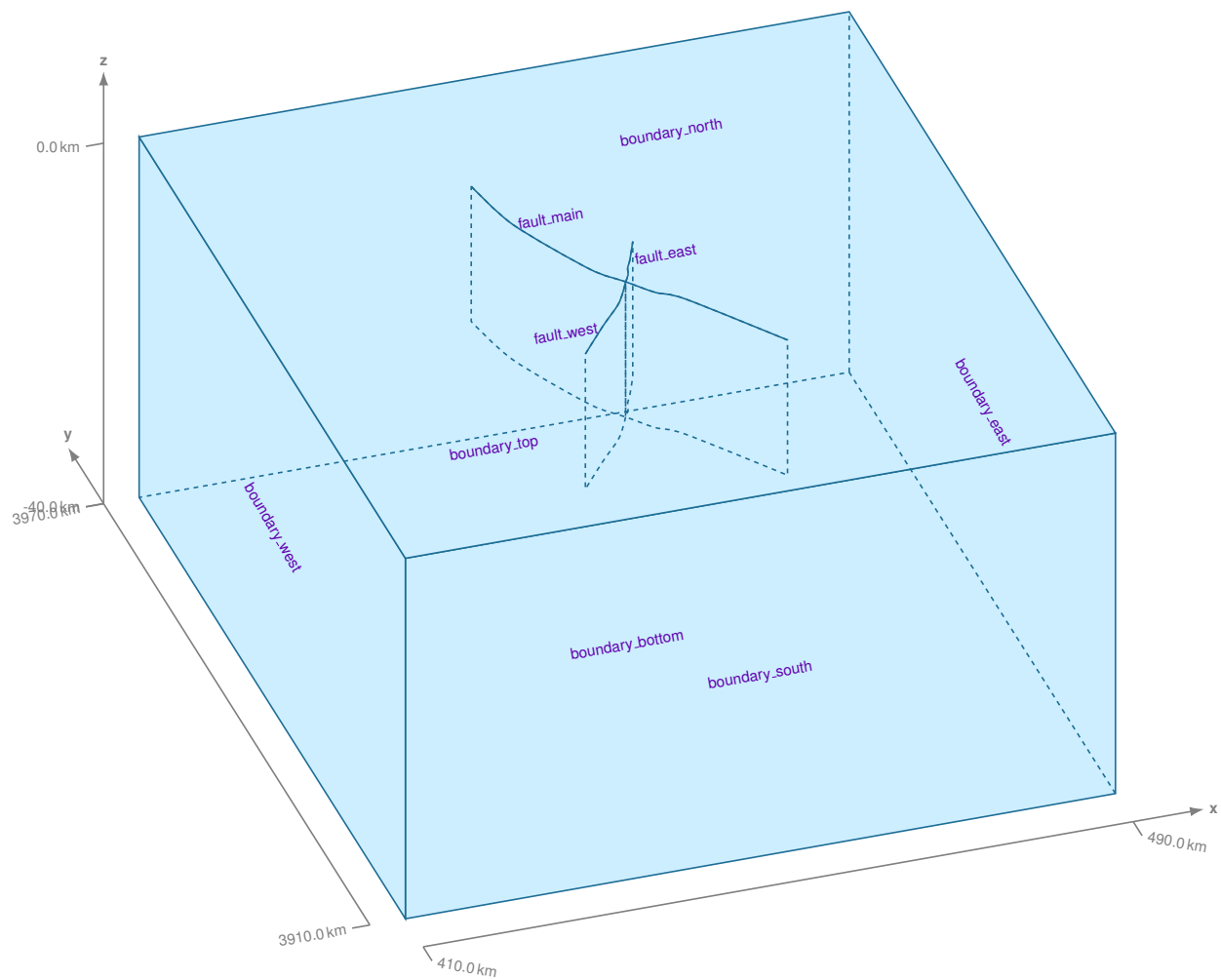


Fig. 4.122: Diagram of geometry for the domain with three intersecting strike-slip faults. The domain extends from 80 km in the east-west direction, 60 km in the north-south direction, and 40 km in the vertical direction. We refer to the domain boundaries using the names shown in the diagram.

Important

We describe how to generate the finite-element mesh using both Gmsh and Cubit. For Step 1 we provide PyLith parameter files for both meshes; for Step 2 we only provide the parameter files that use the Gmsh file.

Example Workflow**Gmsh Mesh****Geometry**

We use the OpenCASCADE geometry engine to create a rectangular prism for the domain and construct vertical fault surfaces from the surface traces.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. The script makes use of the `gmsh_utils.GenerateMesh` class (discussed in [gmsh_utils](#)), which provides the command line arguments and boilerplate methods. In our `generate_gmsh.py` Python script, we create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class.

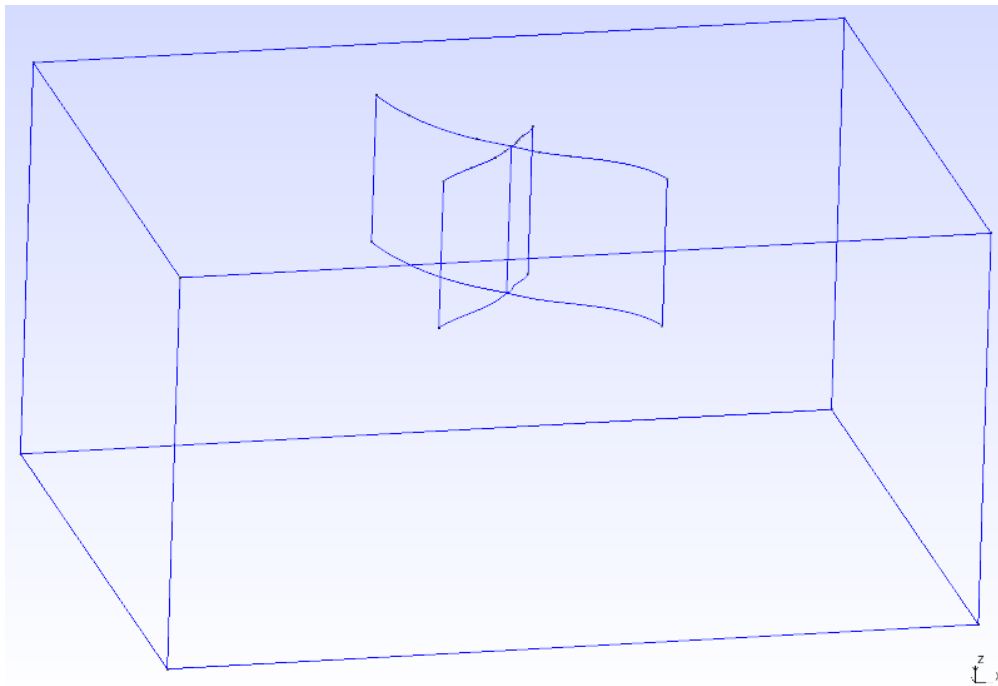


Fig. 4.123: Geometry created in Gmsh for generating the finite-element mesh.

tro

Important

In most cases, we store geometric entities in variables, so that we can refer to them later. However, when we subdivide the domain using the fault surfaces, we do not know the ids of the newly created entities. In these cases

we use the GUI to identify the curves and surfaces and save the ids in variables.

Important

In order for Gmsh to create cells with faces that align with the fault surfaces, we embed the fault surfaces in the domain. We use the Gmsh OpenCASCADE geometry function `fragment()` to embed the fault surfaces.

We use the Gmsh `MeshSize` options to define a discretization size that grows slowly at a geometric rate with distance from the fault. See 6.3.1 [Specifying mesh element sizes in the Gmsh documentation](#) for more information.

Listing 4.190: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with tetrahedral cells and save it to `mesh_tet.msh` (default_
↪ filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui

# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui

# Show available command line arguments.
$ ./generate_gmsh.py --help
```

By default the Python script will generate a finite-element mesh with tetrahedral cells and save it to the file `mesh_tet.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.191: View the Gmsh mesh file `mesh_tet.msh` using Gmsh.

```
gmsh -open mesh_tet.msh
```

Cubit Mesh

Geometry

We use a rectangular prism for the domain and construct vertical fault surfaces from the surface traces. We adjust the dimensions of the domain to avoid small surfaces where the extended fault surfaces meet the domain boundaries.

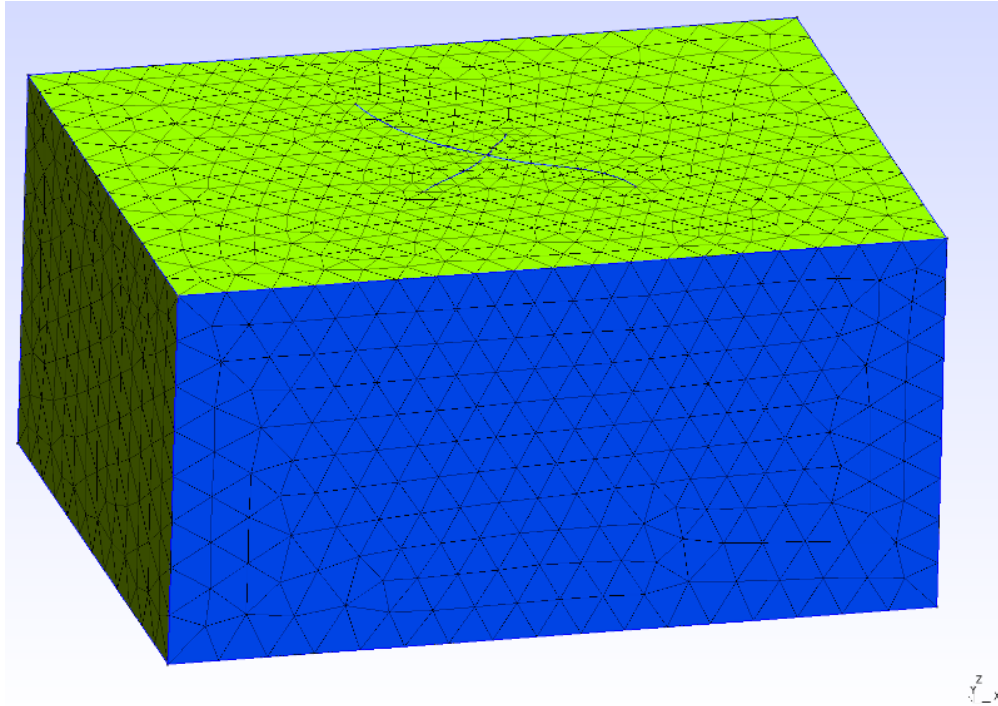


Fig. 4.124: Finite-element mesh with tetrahedral cells generated by Gmsh.

Warning

With Cubit, we can embed the fault surfaces in the domain analogous to what we did using Gmsh. However, in most cases Cubit will not generate a valid mesh when there are embedded surfaces. The workaround is to create closed volumes by extending the fault surfaces to other interior surfaces or the boundaries of the domain. This can result in volumes with small angles when there are multiple fault surfaces with different strikes and dips, and you might need to adjust the geometry to create volumes that will not result in severely distorted cells.

Meshing using Python Script

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

Open the Python script `generate_cubit.py` in the Cubit journal editor. Play the selected script or play the lines, making sure you play the first line so that Cubit uses the Python interpreter when running the script. We specify the parameters controlling the geometry, mesh size, and cell shape near the top of the script.

Important

In most cases, we store geometric entities in variables, so that we can refer to them later. However, when we subdivide the domain using the fault surfaces, we do not know the ids of the newly created entities. For these situations we use the GUI to identify the curves and surfaces and use the `cubit.get_idless_signature()` function to get the string identifying the geometric entity based on location. This results in Cubit commands that are independent of the version of the underlying geometry libraries.

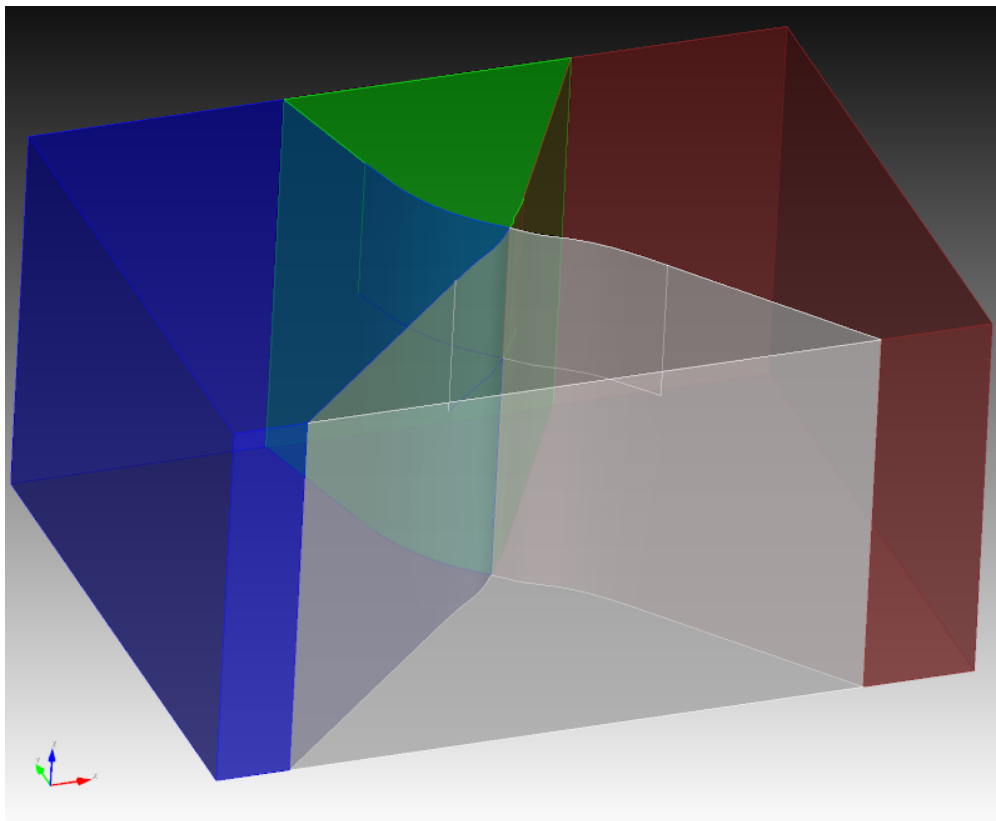


Fig. 4.125: Geometry of domain generated by Cubit.

Note

We use the skeleton sizing function to set cell sizes that increases at a geometric rate with distance from the fault. This makes it easy to specify a fine cell size near the fault where the solution has greater spatial variability and a coarse cell size near the boundaries where the solution is very smooth. We specify the minimum cell size that matches the cell size on the fault surfaces and the gradient in cell size (bias).

After running the Python script in Cubit, you will have a corresponding Exodus-II file (`mesh_tet.exo`). This is a NetCDF file, and it can be loaded into ParaView.

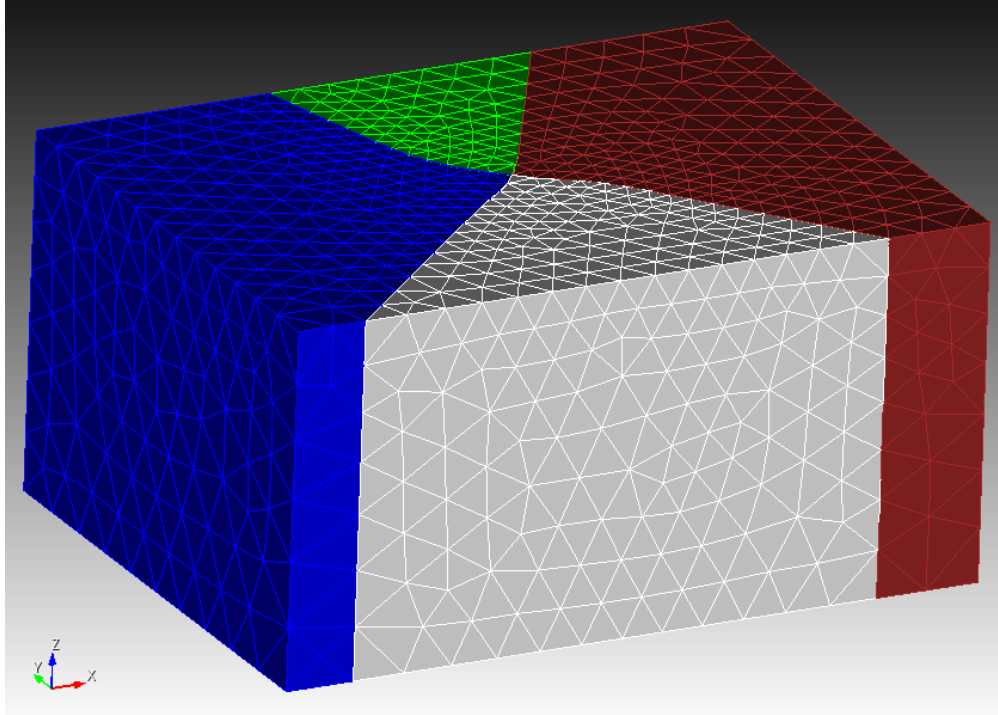


Fig. 4.126: Finite-element mesh with tetrahedral cells generated by Cubit (viewed using Cubit).

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`, which contains numerous comments, so we only summarize the parameters here.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh in `pylithapp.mesh_generator`.

Important

We specify the geographic projection that we used when we constructed the finite-element mesh. This georeferences the model and enables us to specify points in other geographic coordinates system in spatial databases, which may be more convenient.

Listing 4.192: Parameters for the mesh input file and the model coordinate system.

```
[pylithapp.mesh_generator]
reader = pylith.meshio.MeshIOPetsc
reader.filename = mesh_tri.msh

# Model coordinate system is UTM zone 11
reader.coordsys = spatialdata.geocoords.CSGeo
reader.coordsys.crs_string = EPSG:32611
reader.coordsys.space_dim = 3
```

Physics

These quasi-static simulations solve the elasticity equation and include a fault, so we have a solution field with both displacement and Lagrange multiplier subfields.

$$\vec{s} = \begin{pmatrix} \vec{u} & \vec{\lambda} \end{pmatrix}^T \quad (4.487)$$

$$\nabla \cdot \sigma(\vec{u}) = \vec{0} \quad (4.488)$$

We use the default TimeDependent problem and solution field with a single displacement subfield of basis order 1.

Listing 4.193: Parameters for the solution and output of the solution on the +z boundary (ground surface).

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange

[pylithapp.problem]
# Output the solution over the domain and on the +z (groundsurf).
solution_observers = [domain, groundsurf]
solution_observers.groundsurf = pylith.meshio.OutputSolnBoundary

# `label` and `label_value` correspond to the name and tag of the
# physical groups marking the boundaries in the Gmsh script.
[pylithapp.problem.solution_observers.groundsurf]
label = boundary_top
label_value = 15
```

We use the same material properties in all of the simulations in this directory, so we specify them in `pylithapp.cfg` to avoid repeating the information in the file with parameters for each simulation. We have one material with uniform material properties.

Listing 4.194: Material parameters common to all simulations in this directory.

```
[pylithapp.problem]
materials = [elastic]

[pylithapp.problem.materials.elastic]
# We use the default material (elasticity) and rheology
# (isotropic, linearly elastic).
```

(continues on next page)

(continued from previous page)

```

#
# `label_value` must match the tag for the physical group in the Gmsh Python script.
label_value = 1

# The properties are uniform within the material, so we use a UniformDB.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Elastic properties xneg
db_auxiliary_field.values = [density, vs, vp]
db_auxiliary_field.data = [2500.0*kg/m**3, 3.00*km/s, 5.29*km/s]

# The properties are uniform, so we use a basis order of 0, corresponding
# to uniform properties within a cell.
auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0

# We discretize the displacement field with a basis order of 1
# so the stress and strain computed from the displacement field
# will have an accuracy of one order lower. Consequently, we use
# a basis order of 0.
derived_subfields.cauchy_strain.basis_order = 0
derived_subfields.cauchy_stress.basis_order = 0

```

Similarly, we set the fault parameters common to all simulations in the directory. For these simulations, we have three faults that intersect at a common point.

Listing 4.195: General fault parameters common to all simulations in this directory.

```

[pylithapp.problem]
interfaces = [main_fault, west_branch, east_branch]

## main fault
[pylithapp.problem.interfaces.main_fault]
# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
label = fault_main
label_value = 20

# Output `slip` and `traction_changes` on the fault.
observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.west_branch]
# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
label = fault_west
label_value = 21

# Output `slip` and `traction_changes` on the fault.
observers.observer.data_fields = [slip, traction_change]

```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.interfaces.east_branch]
# The `label` and `label_value` correspond to the name and tag of the
## physical group in the Gmsh Python script.
label = fault_east
label_value = 22

# Output `slip` and `traction_changes` on the fault.
observers.observer.data_fields = [slip, traction_change]
```

All simulations in this directory use Dirichlet boundary conditions on the lateral and bottom boundaries of the domain with zero displacements in which we constrain the displacement component perpendicular to the boundary.

Listing 4.196: Dirichlet boundary conditions common to all simulations in this directory.

```
[pylithapp.problem]
bc = [bc_south, bc_east, bc_north, bc_west, bc_bottom]
bc.bc_south = pylith.bc.DirichletTimeDependent
bc.bc_east = pylith.bc.DirichletTimeDependent
bc.bc_north = pylith.bc.DirichletTimeDependent
bc.bc_south = pylith.bc.DirichletTimeDependent
bc.bc_bottom = pylith.bc.DirichletTimeDependent

# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
#
# We constrain the displacement component normal to each boundary.
# We use the specialized `ZeroDB` to specify zero values for the Dirichlet
# BC. We will override this parameter in some of the .cfg files to specify
# nonzero values.
[pylithapp.problem.bc.bc_south]
label = boundary_south
label_value = 10
constrained_dof = [1]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on south boundary

[pylithapp.problem.bc.bc_east]
label = boundary_east
label_value = 11
constrained_dof = [0]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on east boundary

[pylithapp.problem.bc.bc_north]
label = boundary_north
label_value = 12
constrained_dof = [1]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on north boundary
```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.bc.bc_west]
label = boundary_west
label_value = 13
constrained_dof = [0]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on west boundary

[pylithapp.problem.bc.bc_bottom]
label = boundary_bottom
label_value = 14
constrained_dof = [2]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on bottom boundary
```

Step 1: Static Coseismic Slip

Features

- Tetrahedral cells
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- pylith.meshio.MeshIOPetsc

Simulation parameters

This example involves a static simulation that solves for the deformation from prescribed uniform coseismic slip on each of the faults. Fig. 4.127 shows the boundary conditions on the domain. The parameters specific to this example are in `step01_slip.cfg`.

Listing 4.197: Prescribed slip parameters for Step 1.

```
[pylithapp.problem.interfaces.main_fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, -4.0*m, 0.0*m]

[pylithapp.problem.interfaces.west_branch.eq_ruptures.rupture]
```

(continues on next page)

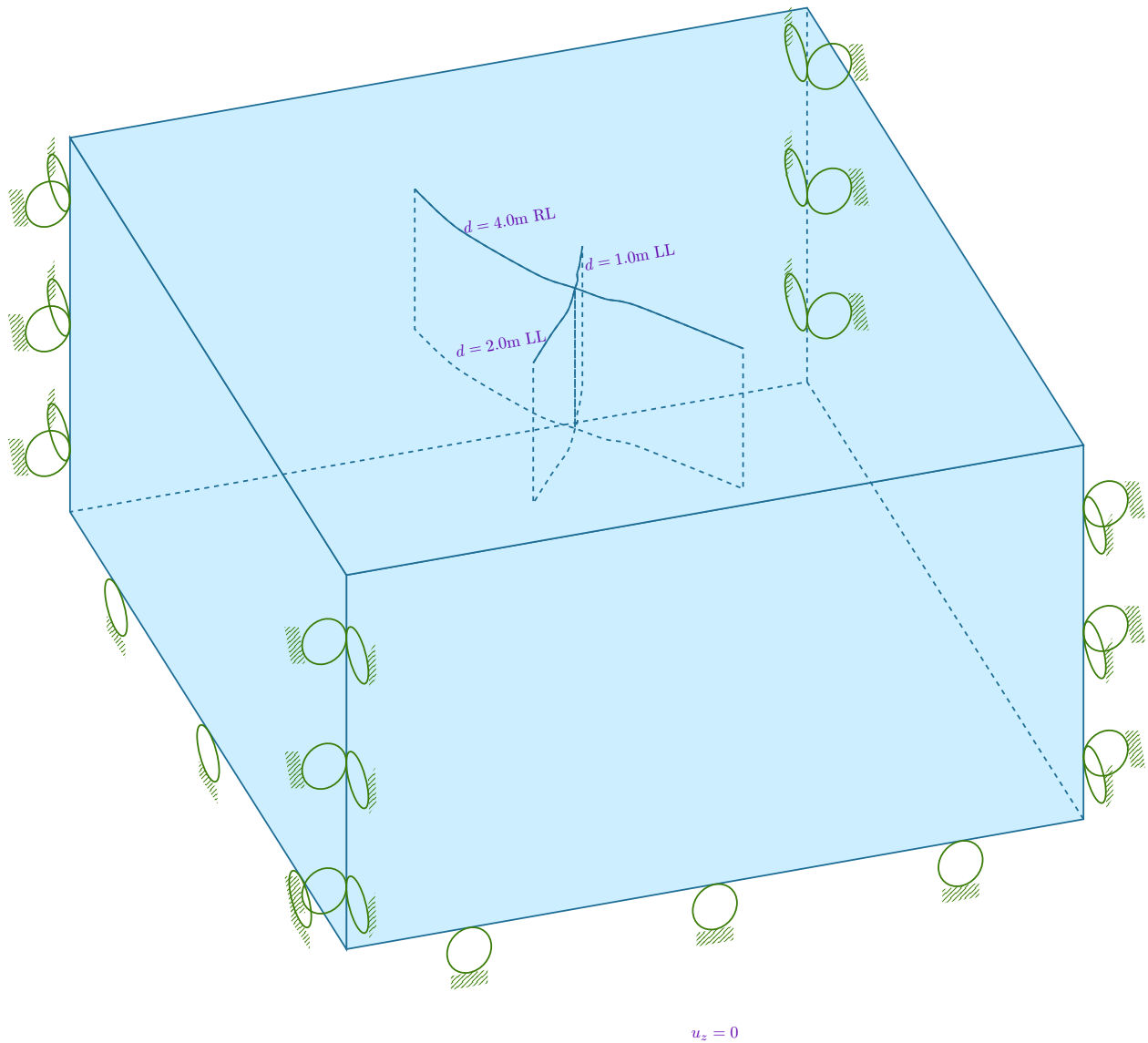


Fig. 4.127: Boundary conditions for static coseismic slip. On the boundary of the domain, we set the displacement component that is perpendicular to the boundary to zero. We prescribe 4.0 m of right-lateral slip on the main fault, 2.0 m of left-lateral slip on the west branch, and 1.0 m of left-lateral slip on the east branch.

(continued from previous page)

```

db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, 3.0*m, 0.0*m]

[pylithapp.problem.interfaces.east_branch.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, final_slip_left_lateral, final_slip_
↪ opening]
db_auxiliary_field.data = [0.0*s, 1.0*m, 0.0*m]

```

Running the simulation

Listing 4.198: Run Step 1 simulation

```

$ pylith step01_slip.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↪ PetscOptions::write(pythia::journal::info_t&, const char*, const_
↪ pylith::utils::PetscOptions&)
-- info (application-flow)
-- Setting PETSc options:
    dm_reorder_section = true
    dm_reorder_section_type = cohesive
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    mg_fine_pc_type = vpbjacobi
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void_

```

(continues on next page)

(continued from previous page)

```

→pylith::meshio::MeshIOPetsc::_read()
-- info (application-flow)
-- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tet.msh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
→pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (413700, 493700)
    (3.917e+06, 3.977e+06)
    (-40000, 0)
>> src/cig/pylith/libsrc/pylith/initializers/MeshDistributor.cc:57:virtual
→pylith::topology::Mesh*
→pylith::initializers::MeshDistributor::run(pylith::topology::Mesh*, const
→pylith::problems::Problem&)
-- info (application-flow)
-- Distributing mesh.
>> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:50:virtual
→pylith::topology::Mesh*
→pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh*, const
→pylith::problems::Problem&)
-- info (application-flow)
-- Inserting cohesive cells.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
→pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
→printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 2500*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
→pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
→pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 3.138739283894e+01
      Linear solve converged due to CONVERGED_ATOL iterations 19
    1 SNES Function norm 1.930576696080e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
→py:222:finalize
-- info (application-flow)

```

(continues on next page)

(continued from previous page)

`-- Finalizing problem.`

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the `MeshIOPetsc` reader and that it found the domain to extend from 410000 to 490000 in the x direction, from $3.91\text{e}+06$ to $3.99\text{e}+06$ in the y direction, and from -40000 to 0 in the z direction. The scales for nondimensionalization remain the default values for a quasistatic problem. PyLith detects the presence of a fault based on the Lagrange multiplier for the fault in the solution field and selects appropriate preconditioning options as discussed in *PETSc Options*.

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 19 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`).

Earthquake rupture parameters

We use the `pylith_eqinfo` utility to compute rupture information, such as earthquake magnitude, seismic moment, seismic potency, and average slip. The file `eqinfoapp.cfg` holds the parameters for `pylith_eqinfo` in this example. By default, `pylith_eqinfo` extracts information for the final time step in the output file. In order to compute the seismic moment and moment magnitude, we need the shear modulus, which in this case is uniform over the domain. Consequently, we specify the density and shear wave speed using a `UniformDB` in `eqinfoapp.cfg`. The average slip, rupture area, seismic moment, and seismic potency are all given in SI units.

Listing 4.199: Run `pylith_eqinfo` for the Step 1 simulation.

```
$ pylith_eqinfo
```

Listing 4.200: Contents of `output/step01_slip-eqinfo.py` generated by `pylith_eqinfo`. The `all` object includes earthquake rupture information combined from the three the individual faults. The average slip matches the uniform slip prescribed on each fault.

```
class RuptureStats(object):
    pass
all = RuptureStats()
all.timestamp = [ 3.155760e+07]
all.ruparea = [ 8.140700e+08]
all.potency = [ 2.623065e+09]
all.moment = [ 5.901897e+19]
all.avgslip = [ 3.222162e+00]
all.mommag = [ 7.147328e+00]
main_fault = RuptureStats()
main_fault.timestamp = [ 3.155760e+07]
main_fault.ruparea = [ 5.424291e+08]
main_fault.potency = [ 2.169716e+09]
main_fault.moment = [ 4.881862e+19]
main_fault.avgslip = [ 4.000000e+00]
main_fault.mommag = [ 7.092390e+00]
east_branch = RuptureStats()
east_branch.timestamp = [ 3.155760e+07]
east_branch.ruparea = [ 8.993288e+07]
east_branch.potency = [ 8.993288e+07]
east_branch.moment = [ 2.023490e+18]
east_branch.avgslip = [ 1.000000e+00]
```

(continues on next page)

(continued from previous page)

```

east_branch.mommag = [ 6.170734e+00]
west_branch = RuptureStats()
west_branch.timestamp = [ 3.155760e+07]
west_branch.ruparea = [ 1.817081e+08]
west_branch.potency = [ 3.634162e+08]
west_branch.moment = [ 8.176864e+18]
west_branch.avgslip = [ 2.000000e+00]
west_branch.mommag = [ 6.575058e+00]

```

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (.h5) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (.xmf) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In Fig. 4.128 we use the `pylith_viz` utility to visualize the *y* displacement field.

Listing 4.201: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step01_slip-domain.h5 warp_grid --component=y
```

Step 1 with Cubit Mesh

Using the Cubit mesh rather than the Gmsh mesh involves two changes:

1. Use the `MeshIOCubit` reader instead of the `MeshIOPetsc` reader and change the filename of the mesh file.
2. Set the `label_value` to 1 for boundary conditions and faults.
We must override the `label_value` settings in `pylithapp.cfg` that were appropriate for our Gmsh reader but are incorrect for the Cubit reader.

The file `step01_slip_cubit.cfg` provides these changes and updates the names for output.

Listing 4.202: Run Step 1 simulation with the Cubit mesh

```

$ pylith step01_slip_cubit.cfg

# The output should look something like the following.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/apps/
↳ PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/meshio/
↳ MeshIOObj.py:41:read
-- meshiocubit(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:185:virtual void_
↳ pylith::meshio::MeshIOCubit::_read()
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Read 5253 vertices.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:189:virtual void_
↳ pylith::meshio::MeshIOCubit::_read()

```

(continues on next page)

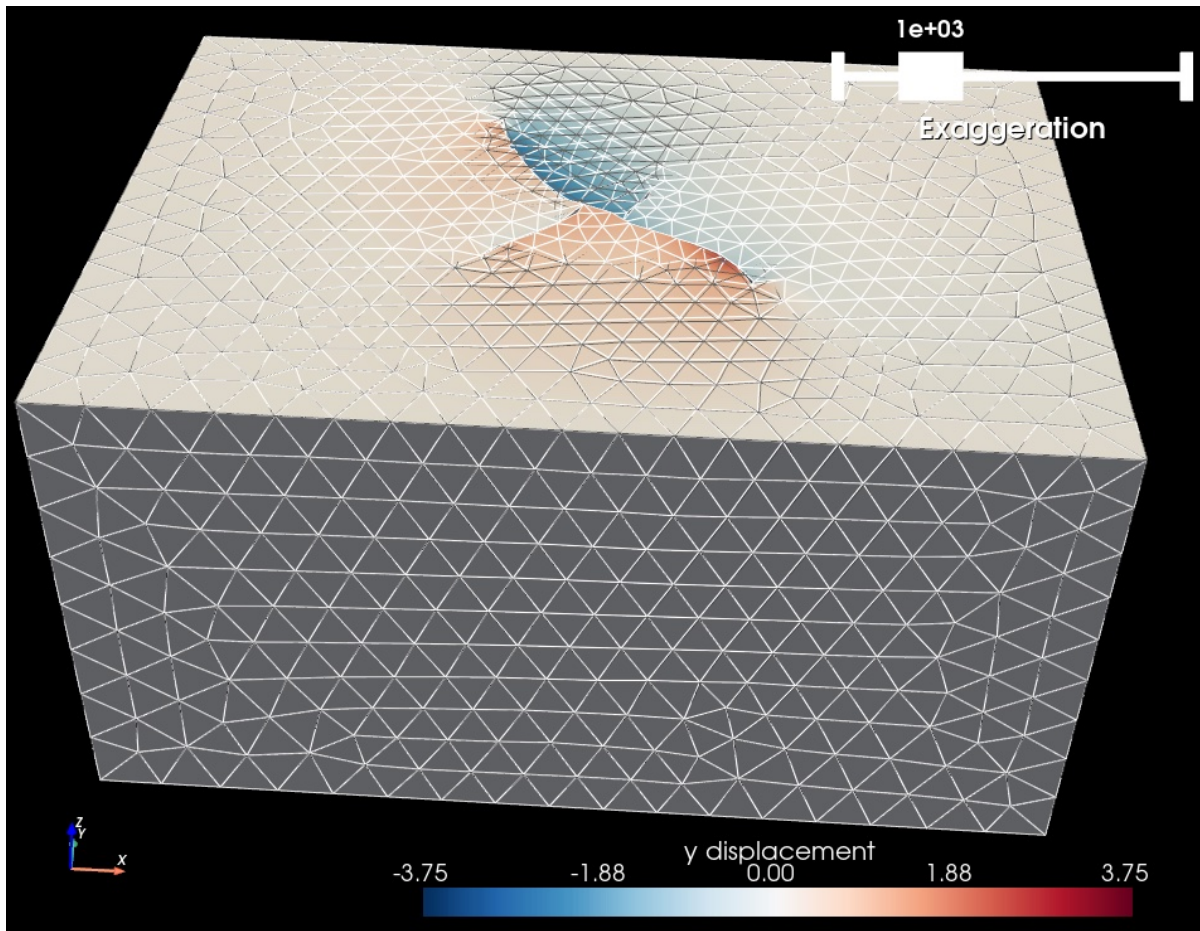


Fig. 4.128: Solution for Step 1. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The contrast in material properties across the faults causes the asymmetry in the y displacement field.

(continued from previous page)

```

-- meshiocubit(info)
-- Component 'meshiocubit.reader': Read 27596 cells.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:356:void_
↳ pylith::meshio::MeshIOCubit::_readNodeSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Found 3 node sets.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:382:void_
↳ pylith::meshio::MeshIOCubit::_readNodeSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading node set 'fault_main_edges' with id 30_
↳ containing 27 nodes.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:382:void_
↳ pylith::meshio::MeshIOCubit::_readNodeSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading node set 'fault_west_edges' with id 31_
↳ containing 18 nodes.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:382:void_
↳ pylith::meshio::MeshIOCubit::_readNodeSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading node set 'fault_east_edges' with id 32_
↳ containing 15 nodes.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:407:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Found 9 side sets.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'boundary_south' with id 10_
↳ containing 286 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'boundary_east' with id 11_
↳ containing 179 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'boundary_north' with id 12_
↳ containing 290 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'boundary_west' with id 13_
↳ containing 172 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'boundary_bottom' with id 14_
↳ containing 583 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void_
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)

```

(continues on next page)

(continued from previous page)

```

-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'boundary_top' with id 15 containing
↳ 930 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'fault_main' with id 20 containing
↳ 197 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'fault_west' with id 21 containing
↳ 72 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIOCubit.cc:439:void
↳ pylith::meshio::MeshIOCubit::_readSideSets(ExodusII &)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Reading side set 'fault_east' with id 22 containing
↳ 38 faces.
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:75:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- meshiocubit(info)
-- Component 'meshiocubit.reader': Domain bounding box:
    (413700, 493700)
    (3.917e+06, 3.977e+06)
    (-40000, 0)
-- many lines omitted --
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ TimeDependent.py:132:run
-- timedependent(info)
-- Solving problem.
0 TS dt 0.001 time 0.
    0 SNES Function norm 2.003292919077e-02
      Linear solve converged due to CONVERGED_ATOL iterations 19
    1 SNES Function norm 1.024898741592e-10
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ Problem.py:199:finalize
-- timedependent(info)
-- Finalizing problem.

```

The MeshIOCubit reader includes diagnostic information in the journal output related to the sizes of the nodesets and material blocks.

Step 2: Static Spatially Variable Coseismic Slip

Features

- Tetrahedral cells
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity

- pylith.faults.FaultCohesiveKin
- pylith.faults.KinSrcStep
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.UniformDB
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- pylith.meshio.MeshIOPetsc
- Gmsh
- spatially variable slip

Simulation parameters

This example involves a static simulation that solves for the deformation from prescribed spatially varying coseismic slip on each of the faults. Fig. 4.129 shows the boundary conditions on the domain. The parameters specific to this example are in `step02_varslip.cfg`.

We prescribe piecewise linear variations in slip along strike on each of the faults using SimpleDB spatial databases. The spatial databases contain coordinates of points in geographic coordinates. PyLith will transform each the coordinates of point on the fault to the geographic coordinates and interpolate the values in the spatial database to the point on the fault.

Tip

In most cases you will not want the interpolation to be done in geographic coordinates, because distances in each of the coordinate directions are different. This is especially true when horizontal coordinates are given in longitude and latitude and vertical distances are given in meters. Transforming the points to a geographic projection is generally preferred.

Listing 4.203: Prescribed slip parameters for Step 2.

```
[pylithapp.problem.interfaces.main_fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Slip parameters for fault 'main'
db_auxiliary_field.iohandler.filename = fault_main_slip.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.interfaces.west_branch.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Slip parameters for fault 'west'
db_auxiliary_field.iohandler.filename = fault_west_slip.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.interfaces.east_branch.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Slip parameters for fault 'east'
db_auxiliary_field.iohandler.filename = fault_east_slip.spatialdb
db_auxiliary_field.query_type = linear
```

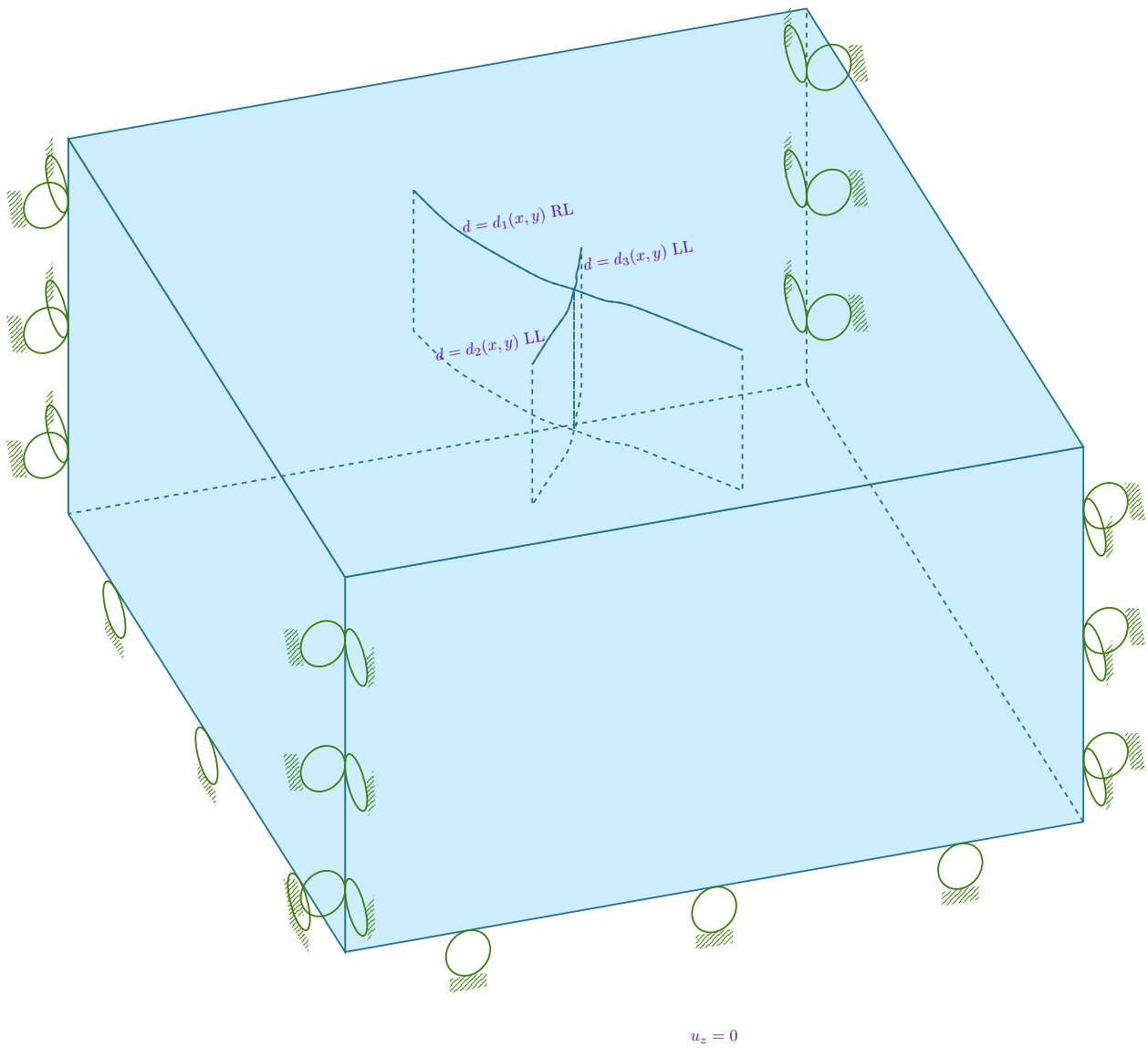


Fig. 4.129: Boundary conditions for static coseismic slip. On the boundary of the domain, we set the displacement component that is perpendicular to the boundary to zero. We prescribe slip that varies linearly over each of the fault surfaces.

Running the simulation

Listing 4.204: Run Step 2 simulation

```

$ pylith step02_varslip.cfg
>> software/pylith-opt/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t&, const char*, const
↳ pylith::utils::PetscOptions&)
-- info (application-flow)
-- Setting PETSc options:
    dm_reorder_section = true
    dm_reorder_section_type = cohesive
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_gmres_restart = 100
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    mg_fine_ksp_max_it = 5
    mg_fine_pc_type = vpbjacobi
    pc_type = gamg
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void
↳ pylith::meshio::MeshIOPetsc::_read()
-- info (application-flow)
-- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tet.msh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (413700, 493700)
    (3.917e+06, 3.977e+06)
    (-40000, 0)
>> src/cig/pylith/libsrc/pylith/initializers/MeshDistributor.cc:57:virtual
↳ pylith::topology::Mesh*
↳ pylith::initializers::MeshDistributor::run(pylith::topology::Mesh*, const
↳ pylith::problems::Problem&)
-- info (application-flow)
-- Distributing mesh.
>> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:50:virtual

```

(continues on next page)

(continued from previous page)

```

↪pylith::topology::Mesh*
↪pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh*, const
↪pylith::problems::Problem&)
-- info (application-flow)
-- Inserting cohesive cells.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
↪pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
↪printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 2500*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↪pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.001 time 0.
    0 SNES Function norm 1.091558353128e+01
      Linear solve converged due to CONVERGED_ATOL iterations 18
    1 SNES Function norm 2.786048497788e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the `MeshIOPetsc` reader and that it found the domain to extend from 413700 to 493700 in the x direction, from $3.92e+06$ to $3.98e+06$ in the y direction, and from -40000 to 0 in the z direction. The scales for nondimensionalization remain the default values for a quasistatic problem. PyLith detects the presence of a fault based on the Lagrange multiplier for the fault in the solution field and selects appropriate preconditioning options as discussed in *PETSc Options*.

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static simulation). The linear solve converged after 18 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snestol`).

Visualizing the results

In Fig. 4.130 we use the `pylith_viz` utility to visualize the y displacement field.

Listing 4.205: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step02_varslip-domain.h5 warp_grid --component=y
```

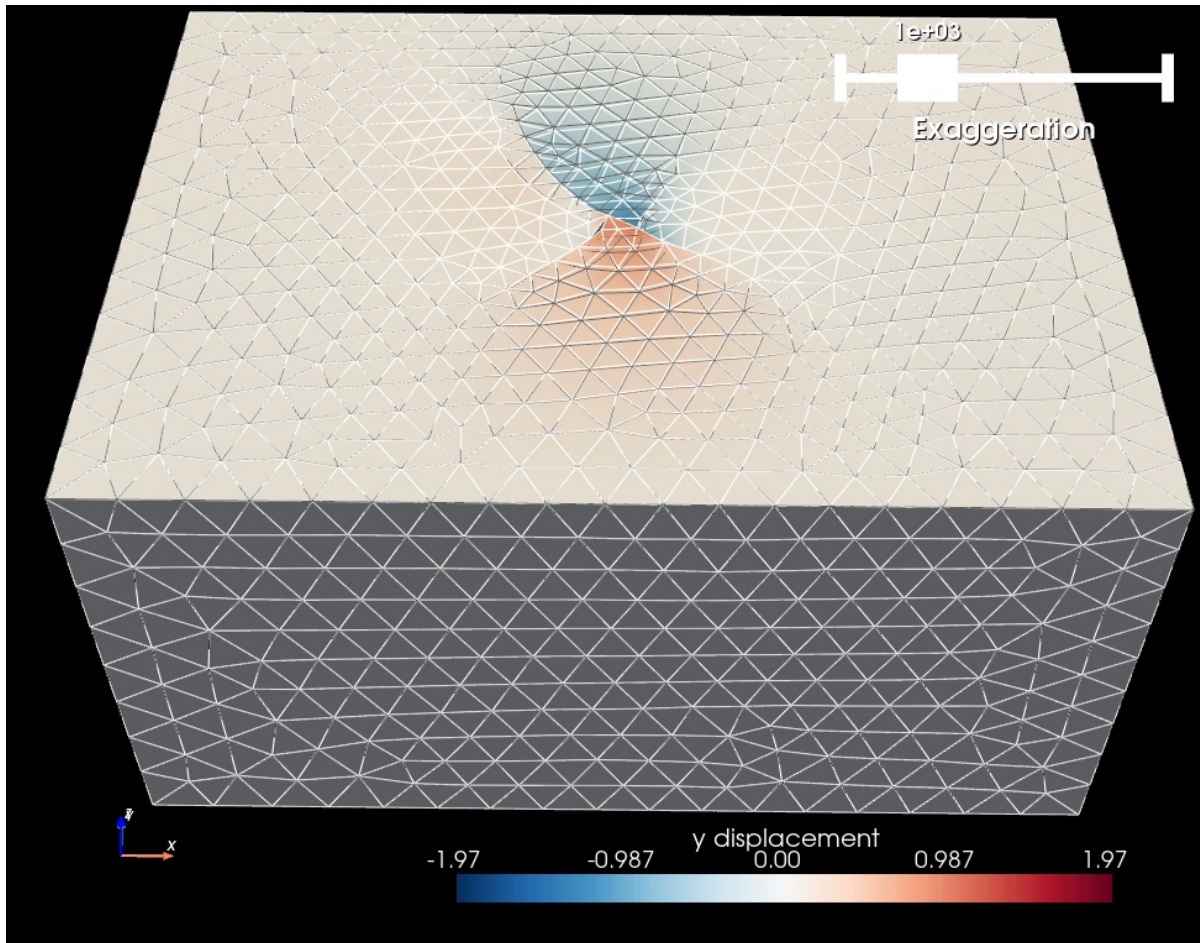


Fig. 4.130: Solution for Step 2. The colors of the shaded surface indicate the y displacement, and the deformation is exaggerated by a factor of 1000. The undeformed configuration is shown by the gray wireframe. The contrast in material properties across the faults causes the asymmetry in the y displacement field.

Suggested Exercises

1. Adjust the discretization size on the fault and the rate at which it increases with distance from the fault.
2. Create a `.cfg` file to run Step 2 with the Cubit mesh.
3. Add additional complexity to the distribution of slip in Step 2.
4. Create a time-dependent problem with earthquake ruptures at different origin times.

Vertical Cross-Section of Subduction Zone (2D)

The files are in the directory `examples/subduction-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

generate_gmsh.py

Python script to generate mesh using Gmsh.

generate_cubit.py

Python script to generate mesh using Cubit.

***.msh**

Gmsh mesh files generated by Gmsh.

***.exo**

Exodus II mesh files generated by Cubit.

***.spatialdb**

Spatial database files associated with the spatial databases.

viz

Directory containing Matplotlib Python scripts for visualizing results.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This example examines quasistatic interseismic and coseismic deformation in 2D for a subduction zone (see Fig. 4.131). It is based on the 2011 M9.0 Tohoku-oki earthquake off the east coast of Japan. We build on the previous examples and add complexity through a series of steps:

Step 1

Static coseismic slip on the subduction interface.

Step 2

Quasistatic interseismic deformation with creep on the top and bottom of the slab, except in the zone of coseismic slip.

Step 3

Quasistatic earthquake cycle with prescribed coseismic slip and creep.

Warning

Steps 4-6 have not yet been updated for PyLith v3.

Important

We describe how to generate the finite-element mesh using both Gmsh and Cubit. The files for both methods are included. We use the Gmsh files in the PyLith parameter files. See [examples/strikeslip-2d/step01_slip_cubit.cfg](#) for a description of how to modify the parameter files to switch from using mesh files from Gmsh to mesh files from Cubit.

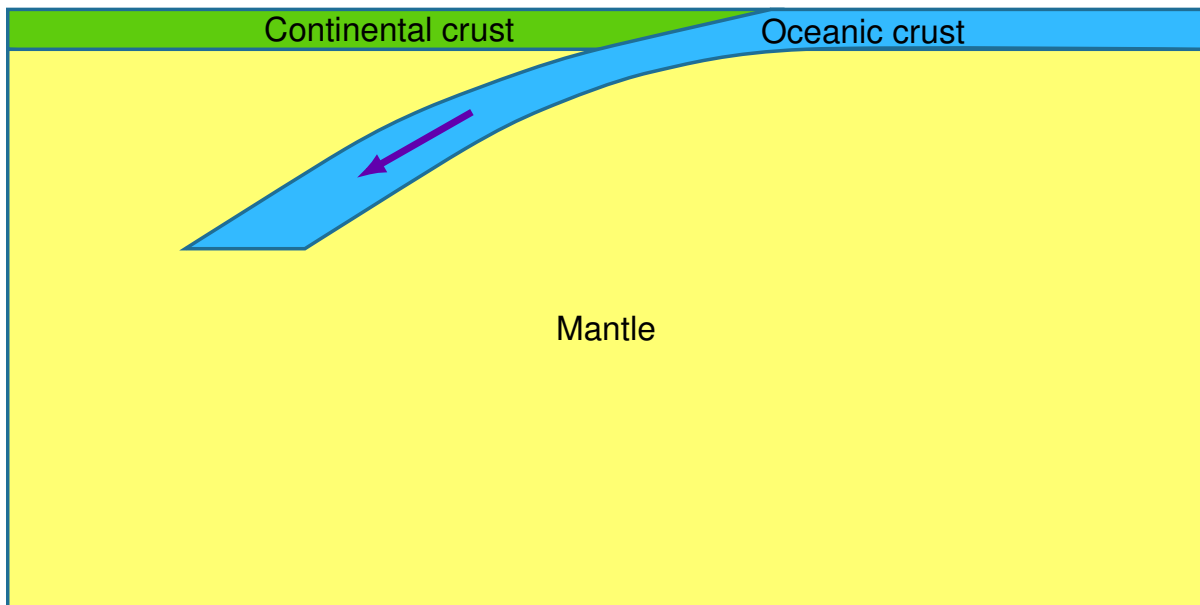


Fig. 4.131: Diagram of 2D subduction zone example.

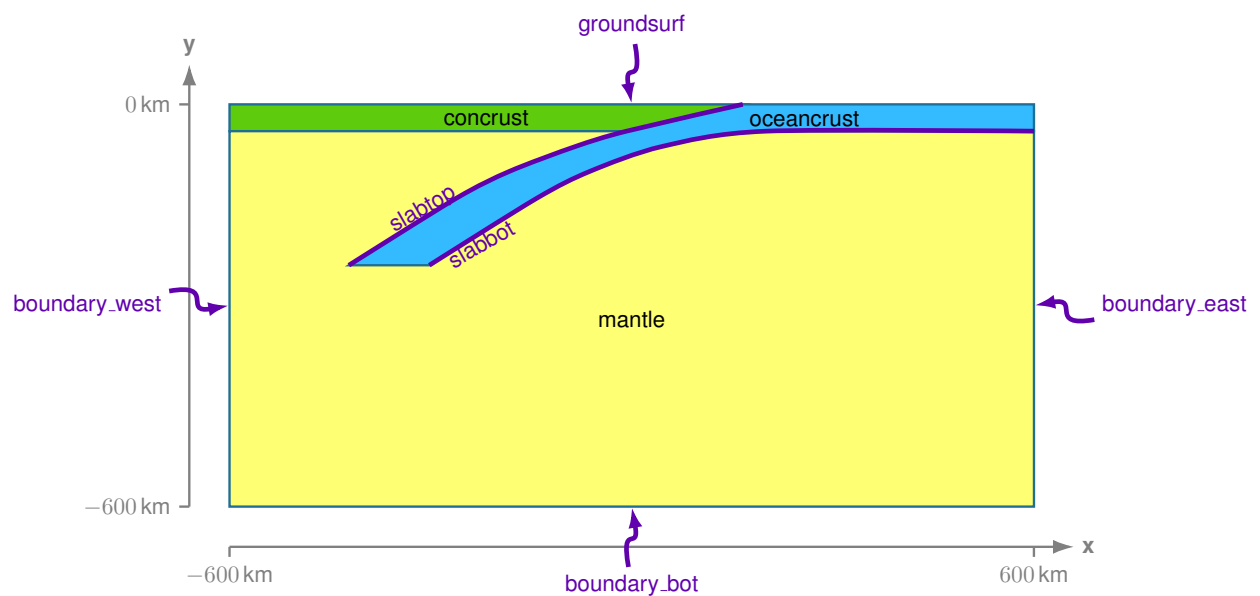


Fig. 4.132: Geometry of the 2D subduction zone example. The domain extends from -600 km to +600 km in the x direction and from -340 km to 0 in the y direction. We refer to the domain boundaries using the names shown in the diagram.

(continued from previous page)

```
# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui
```

By default the Python script will generate a finite-element mesh with triangular cells and save it to the file `mesh_tri.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.207: View the Gmsh mesh file `mesh_tri.msh` using Gmsh.

```
gmsh -open mesh_tri.msh
```

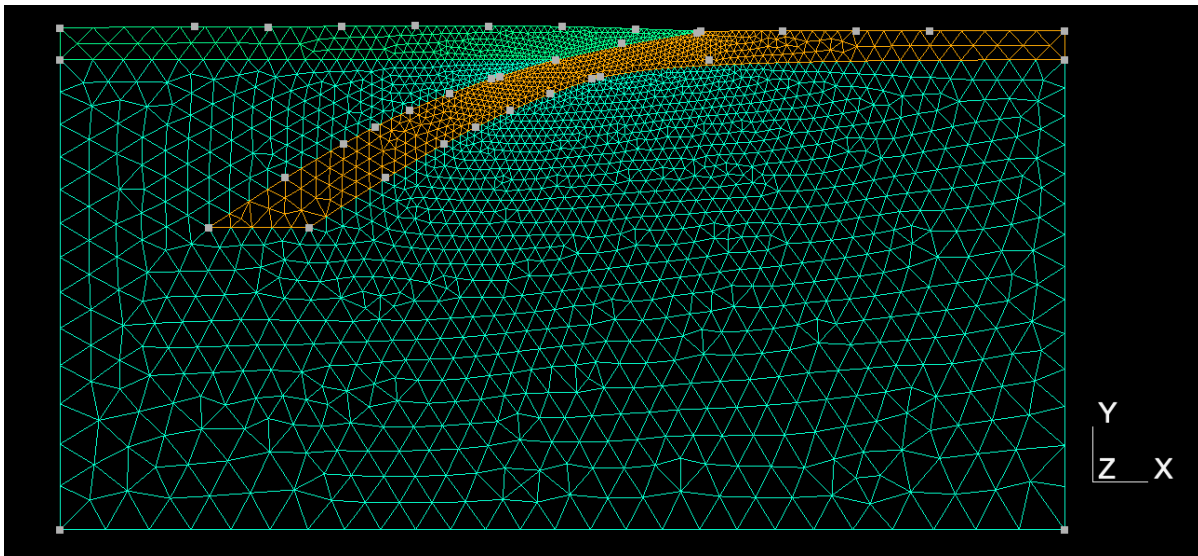


Fig. 4.134: Finite-element mesh with triangular cells generated by Gmsh.

Cubit Mesh

Geometry

We construct the geometry following the same general procedure that we used with Gmsh. We first create points, then connect the points into curves, and finally connect the curves into surfaces. Fig. 4.135 shows the geometry and variables names of the vertices and curves.

Meshing using Python script

Note

New in v4.1.0.

We use a Python script to generate the mesh rather than Journal files. We also use the skeleton sizing function for the surface.

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

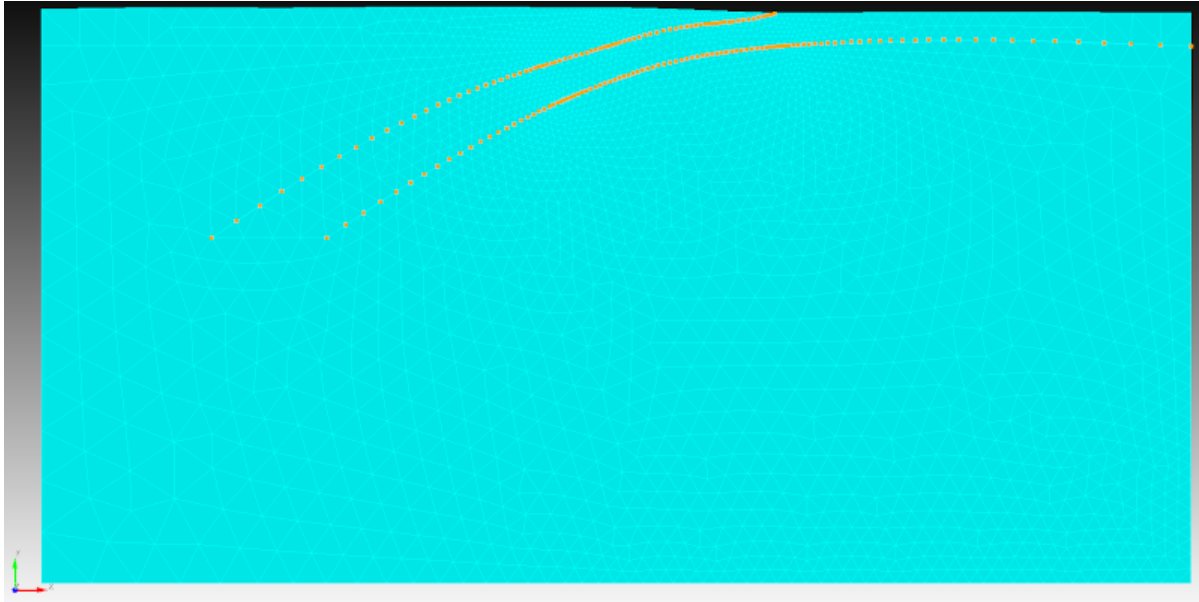


Fig. 4.136: Finite-element mesh with triangular cells generated by Cubit.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh in `pylithapp.mesh_generator`.

Physics

These quasi-static simulations solve the elasticity equation and include a fault, so we have a solution field with both displacement and Lagrange multiplier subfields.

$$\vec{s} = \begin{pmatrix} \vec{u} & \vec{\lambda} \end{pmatrix}^T \quad (4.489)$$

$$\nabla \cdot \sigma(\vec{u}) = \vec{0} \quad (4.490)$$

We use the default `TimeDependent` problem and solution field with a single displacement subfield of basis order 1. In addition to output of the solution over the domain, we output the solution over the ground surface (+y boundary). For the domain we skip one time step between writing the solution to the file.

Listing 4.208: Solution and output parameters for all subduction-2d simulations.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange

[pylithapp.problem]
solution_observers = [domain, groundsurf]
solution_observers.groundsurf = pylith.meshio.OutputSolnBoundary

[pylithapp.problem.solution_observers.domain]
# Skip 1 time step between output for the domain.
trigger.num_skip = 1
```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.solution_observers.groundsurf]
# The `label` and `label_value` correspond to the name and tag of the
# physical group in the Gmsh Python script.
label = groundsurf
label_value = 10
```

The physical properties for each material are specified in spatial database files. For example, the elastic properties for the continental crust are in `mat_concrust.spatialdb`. The provided spatial database files all use just a single point to specify uniform physical properties within each material.

Listing 4.209: Material parameters for the subduction-2d example suite.
We now show the details for the continental crust material.

```
[pylithapp.problem]
materials = [continent_crust, ocean_crust, mantle]

[pylithapp.problem.materials]
[pylithapp.problem.materials.continent_crust]
label_value = 1

db_auxiliary_field.description = Continental crust properties
db_auxiliary_field.iohandler.filename = mat_concrust.spatialdb

observers.observer.trigger.num_skip = 1

auxiliary_subfields.density.basis_order = 0
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0
```

Step 1: Static Coseismic Slip

Features

- Triangular cells
- field split preconditioner
- Schur complement
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- Static simulation
- `pylith.faults.FaultCohesiveKin`
- `pylith.bc.DirichletTimeDependent`
- `spatialdata.spatialdb.SimpleDB`
- `pylith.faults.KinSrcStep`

- pylith.bc.ZeroDB

Simulation parameters

This example involves a static simulation that solves for the deformation from prescribed coseismic slip on the subduction interface. The depth variation in the prescribed slip is based on the 2011 Tohoku-oki earthquake. Fig. 4.137 shows the boundary conditions on the domain. The parameters specific to this example are in `step01_coseismic.cfg`.

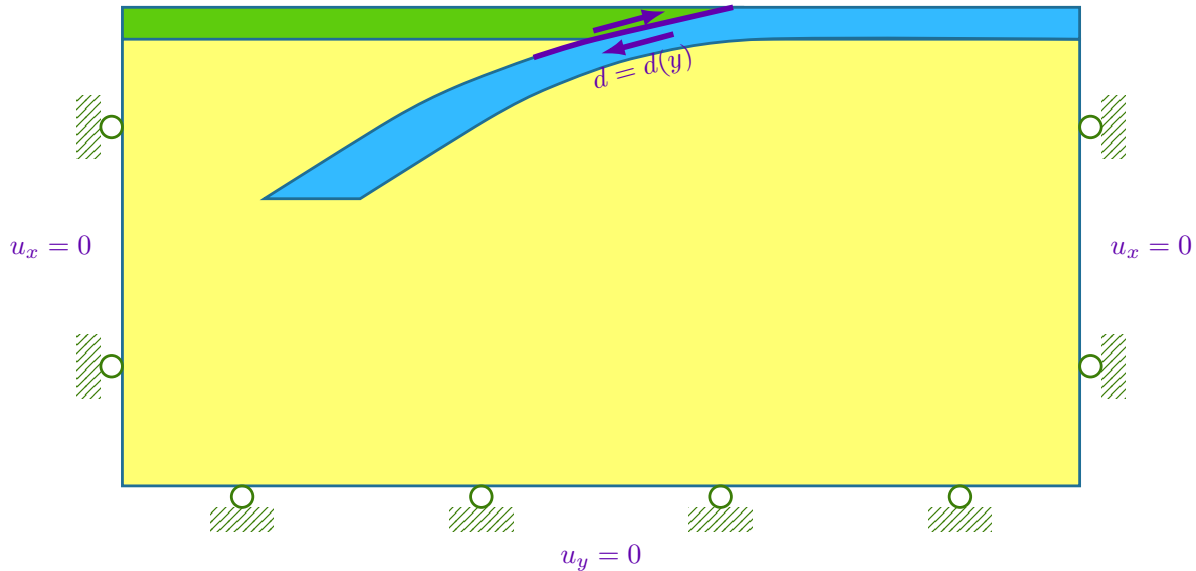


Fig. 4.137: Boundary conditions for static coseismic slip on the subduction interface. We prescribe reverse slip that varies with depth and roller boundary conditions on the lateral sides and bottom of the domain.

We only prescribe slip on the subduction interface, so we create an array with 1 fault. We specify slip as a function of depth, so we use a SimpleDB with linear interpolation.

Listing 4.210: Fault parameters for Step 1.

```
[pylithapp.problem]
interfaces = [fault]

[pylithapp.problem.interfaces.fault]
label = fault_slabtop
label_value = 21

observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.iohandler.filename = fault_coseismic.spatialdb
db_auxiliary_field.query_type = linear
```


Listing 4.211: Dirichlet boundary condition parameters for Step 1. We only show the details for the east boundary of the crust.

```
[pylithapp.problem]
bc = [bc_east_crust, bc_east_mantle, bc_west, bc_bottom]

[pylithapp.problem.bc.bc_east_crust]
label = boundary_east_crust
label_value = 12
constrained_dof = [0]
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on east boundary (crust)
```

Running the simulation

Listing 4.212: Run Step 1 simulation

```
$ pylith step01_coseismic.cfg

# The output should look something like the following.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/apps/
↳ PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/meshio/
↳ MeshIOObj.py:38:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:85:void_
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- meshiopetsc(info)
-- Component 'reader': Domain bounding box:
    (-6000000, 6000000)
    (-6000000, 399.651)
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ Problem.py:116:preinitialize
-- timedependent(info)
-- Performing minimal initialization before verifying configuration.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ Solution.py:39:preinitialize
-- solution(info)
-- Performing minimal initialization of solution.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ Problem.py:174:verifyConfiguration
-- timedependent(info)
-- Verifying compatibility of problem configuration.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ Problem.py:219:_printInfo
-- timedependent(info)
-- Scales for nondimensionalization:
    Length scale: 1000000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
```

(continues on next page)

(continued from previous page)

```

Rigidity scale: 1e+10*m**-1*kg*s**-2
Temperature scale: 1*K
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↪ Problem.py:185:initialize
-- timedependent(info)
-- Initializing timedependent problem with quasistatic formulation.
>> /src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:239:static void pylith::utils::_
↪ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- petscoptions(info)
-- Setting PETSc options:
dm_reorder_section = true
dm_reorder_section_type = cohesive
ksp_atol = 1.0e-7
ksp_converged_reason = true
ksp_error_if_not_converged = true
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-12
mg_fine_ksp_max_it = 5
mg_fine_pc_type = vpbjacobi
mg_levels_pc_type = pbjacobi
pc_gamg_coarse_eq_limit = 200
pc_type = gamg
snes_atol = 4.0e-7
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-12
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_type = beuler
viewer_hdf5_collective = true
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↪ TimeDependent.py:132:run
-- timedependent(info)
-- Solving problem.
0 TS dt 0.001 time 0.
    0 SNES Function norm 5.454651006059e+00
      Linear solve converged due to CONVERGED_ATOL iterations 18
    1 SNES Function norm 2.888986872721e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↪ Problem.py:199:finalize
-- timedependent(info)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the MeshIOPetsc reader and that it found the domain to extend from -600 km to +600 km in the x direction and from -600 km to 0 in the y direction. The output also includes the scales used for nondimensionalization and the default PETSc options.

At the end of the output written to the terminal, we see that the solver advanced the solution one time step (static

simulation). The linear solve converged after 18 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`).

Visualizing the results

The `output` directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (`.h5`) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (`.xmf`) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In Fig. 4.138 we use the `pylith_viz` utility to visualize the `x` displacement field.

Listing 4.213: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step01_coseismic-domain.h5 warp_grid --component=x
```

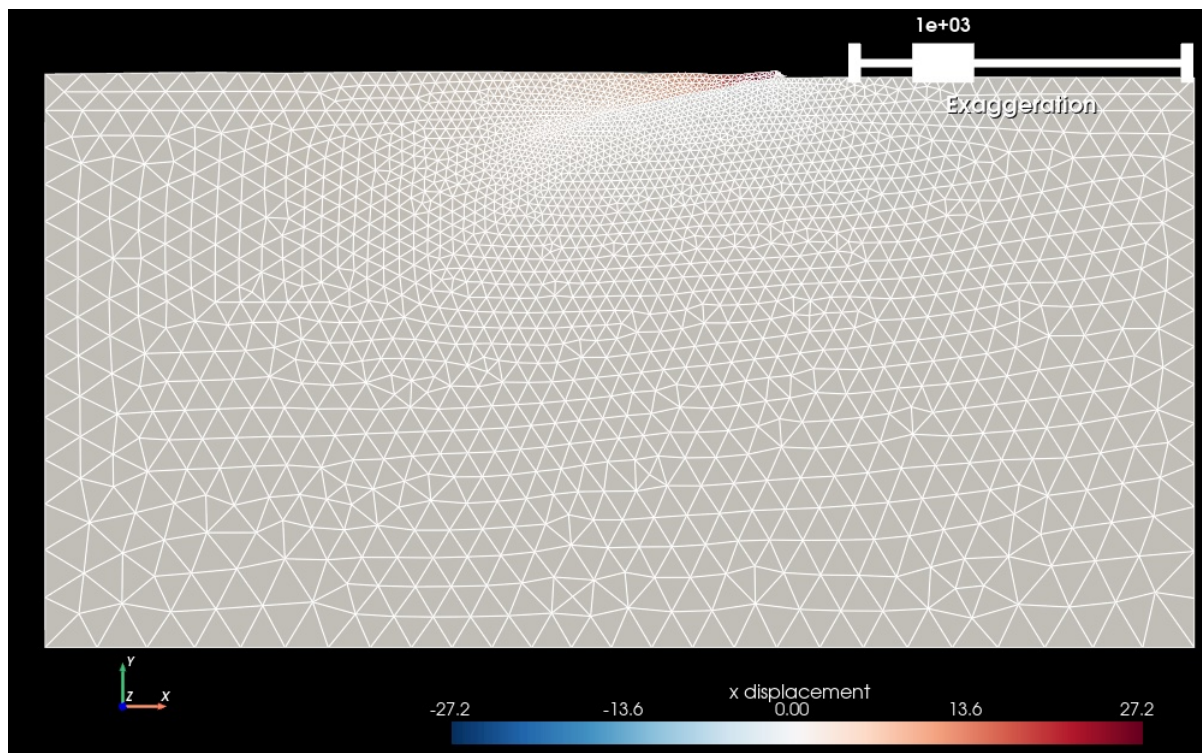


Fig. 4.138: Solution for Step 1. The colors of the shaded surface indicate the `x` displacement, and the deformation is exaggerated by a factor of 1000.

Step 2: Quasistatic Interseismic Deformation

Features

- Triangular cells
- field split preconditioner
- Schur complement
- `pylith.meshio.MeshIOPetsc`

- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Static simulation
- pylith.faults.FaultCohesiveKin
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.UniformDB
- pylith.faults.KinSrcConstRate
- pylith.bc.ZeroDB

Simulation parameters

In this example we simulate the interseismic deformation associated with the oceanic crust subducting beneath the continental crust and into the mantle. We prescribe steady aseismic slip of 8 cm/yr along the interfaces between the oceanic crust and mantle with the interface between the oceanic crust and continental crust locked as shown in Fig. 4.139. The parameters specific to this example are in `step02_interseismic.cfg`.

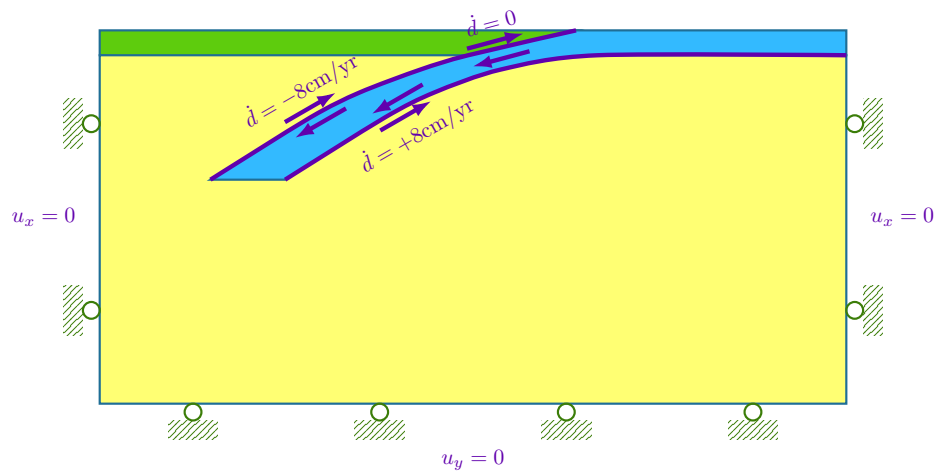


Fig. 4.139: Boundary conditions for quasistatic simulation for interseismic deformation. We prescribe constant creep on the top and bottom of the subduction slab, except for the portion of the subduction interface where we imposed coseismic slip in Step 1. We lock (zero creep) that part of the interface.

The simulation spans 150 years with an initial time step of 5 years.

Listing 4.214: Time stepping parameters for Step 2.

```
[pylithapp.time-dependent]
initial_dt = 5.0*year
start_time = -5.0*year
end_time = 150.0*year
```

We create an array with 2 faults, one for the top of the slab and one for the bottom of the slab. We use the constant slip rate kinematic source model with a uniform slip rate on the bottom of the slab and a slip rate that varies with depth on the top of the slab.

Listing 4.215: Prescribed slip parameters for Step 2.

```
[pylithapp.problem]
interfaces = [fault_slabtop, fault_slabbot]

[pylithapp.problem.interfaces.fault_slabtop]
label = fault_slabtop
label_value = 21

observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.fault_slabtop.eq_ruptures]
rupture = pylith.faults.KinSrcConstRate

[pylithapp.problem.interfaces.fault_slabtop.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.iohandler.filename = fault_slabtop_creep.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.problem.interfaces.fault_slabbot]
label = fault_slabbot
label_value = 22

observers.observer.data_fields = [slip, traction_change]

[pylithapp.problem.interfaces.fault_slabbot.eq_ruptures]
rupture = pylith.faults.KinSrcConstRate

[pylithapp.problem.interfaces.fault_slabbot.eq_ruptures.rupture]
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.values = [initiation_time, slip_rate_left_lateral, slip_rate_opening]
db_auxiliary_field.data = [0.0*year, 8.0*cm/year, 0.0*cm/year]
```

We adjust the Dirichlet (displacement) boundary conditions on the lateral edges and bottom of the domain by pinning only the portions of the boundaries that are mantle and continental crust and not oceanic crust.

Listing 4.216: We use only 3 Dirichlet boundary conditions to allow the slab to move freely on the boundaries.

```
[pylithapp.problem]
bc = [bc_east_mantle, bc_west, bc_bottom]
```

Running the simulation

Listing 4.217: Run Step 2 simulation

```
$ pylith step02_interseismic.cfg

# The output should look something like the following.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/apps/
↳ PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/meshio/
↳ MeshIOObj.py:38:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:85:void_
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- meshiopetsc(info)
-- Component 'reader': Domain bounding box:
    (-6000000, 6000000)
    (-6000000, 399.651)

# -- many lines omitted --

30 TS dt 0.05 time 1.45
    0 SNES Function norm 5.748198602475e-01
      Linear solve converged due to CONVERGED_ATOL iterations 4
      1 SNES Function norm 5.369483396356e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
31 TS dt 0.05 time 1.5
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↳ Problem.py:199:finalize
-- timedependent(info)
-- Finalizing problem.
```

The beginning of the output written to the terminal is identical to that from Step 1. At the end of the output, we see that the simulation advanced the solution 31 time steps. Remember that the PETSc TS monitor shows the nondimensionalized time and time step values.

Visualizing the results

In Fig. 4.140 we use the `pylith_viz` utility to visualize the x displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.218: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step02_interseismic-domain.h5 warp_grid --component=x
```

Step 3: Quasistatic Earthquake Cycle

Features

- Triangular cells
- field split preconditioner
- Schur complement

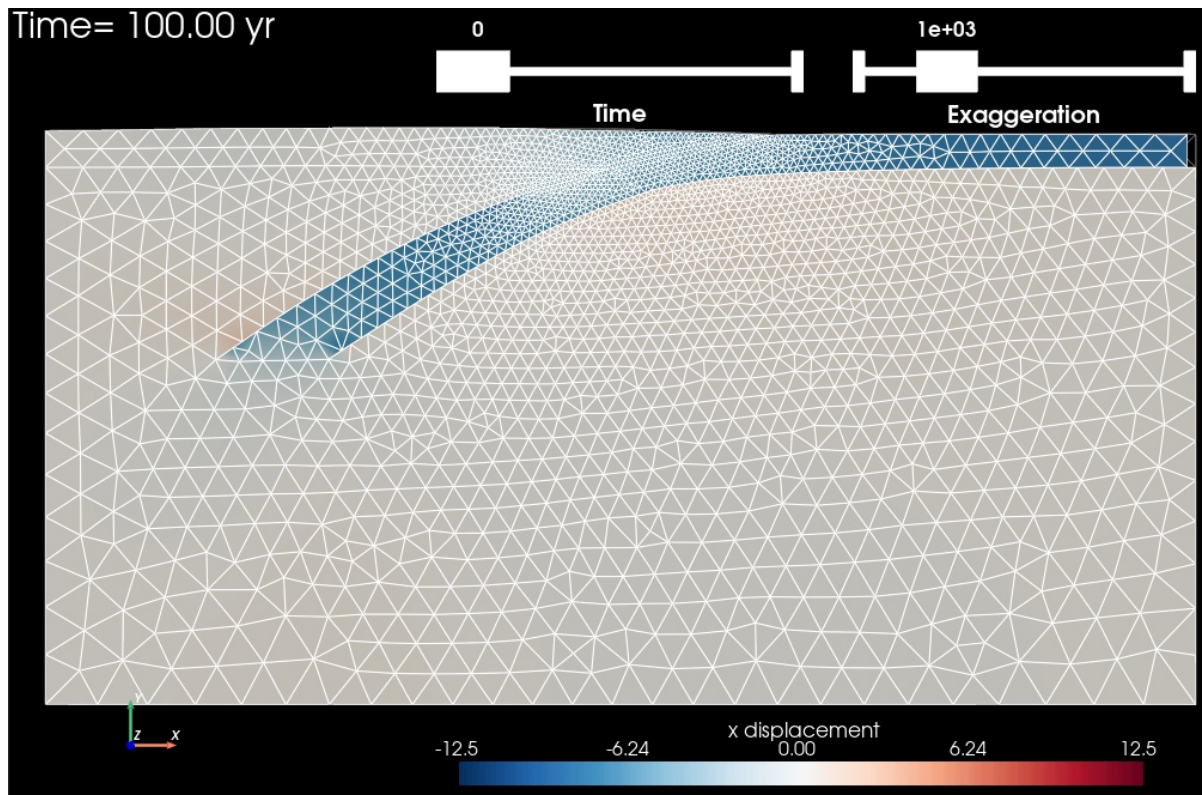


Fig. 4.140: Solution for Step 2 at $t=100$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000.

- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- Quasistatic simulation
- pylith.faults.FaultCohesiveKin
- pylith.bc.DirichletTimeDependent
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.UniformDB
- pylith.faults.KinSrcConstRate
- pylith.bc.ZeroDB

Simulation parameters

This simulation combines 300 years of interseismic deformation from Step 2 with the coseismic deformation from Step 1 applied at 150 years to create a simple model of an earthquake cycle. Fig. 4.141 shows the schematic of the boundary conditions. The parameters specific to this example are in `step03_eqcycle.cfg`.

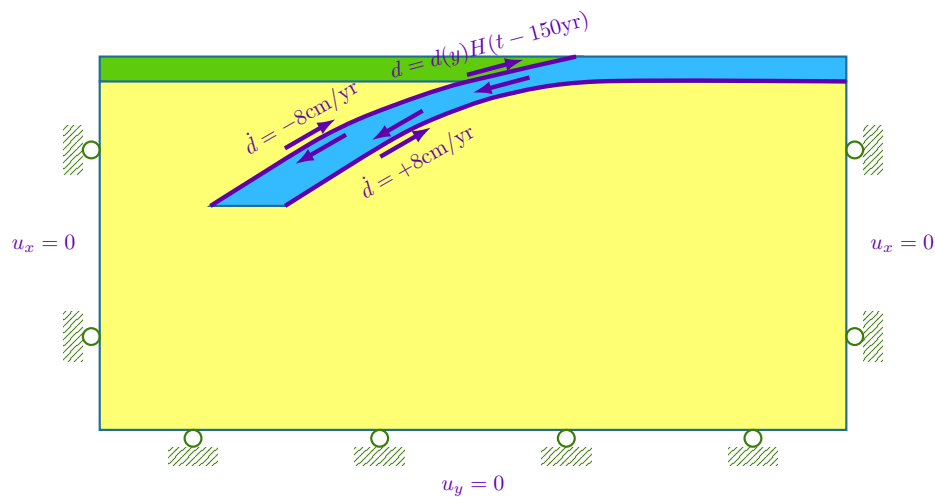


Fig. 4.141: Boundary conditions for a simple earthquake cycle with prescribed coseismic slip and creep. We combine the coseismic slip from Step 1 with the interseismic slip from Step 2.

On the interface along the top of the subducting oceanic crust and the continental crust and mantle we create two earthquake ruptures. The first rupture applies the coseismic slip from Step 1 at 150 years, while the second rupture prescribes the same steady, aseismic slip as in Step 2. On the interface between the bottom of the subducting oceanic crust and the mantle, we prescribe the same steady, aseismic slip as that in Step 2.

Listing 4.219: Prescribed slip parameters for Step 3. We only show the details for the top of the slab.

```
[pylithapp.problem]
interfaces = [fault_slabtop, fault_slabbot]

[pylithapp.problem.interfaces.fault_slabtop]
label = fault_slabtop
label_value = 21

observers.observer.data_fields = [slip, traction_change]

eq_ruptures = [creep, earthquake]

# Creep
eq_ruptures.creep = pylith.faults.KinSrcConstRate
eq_ruptures.creep.origin_time = 0.0*year

# Earthquake
eq_ruptures.earthquake = pylith.faults.KinSrcStep
eq_ruptures.earthquake.origin_time = 150.0*year

[pylithapp.timedependent.interfaces.fault_slabtop.eq_ruptures.earthquake]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.iohandler.filename = fault_coseismic.spatialdb
db_auxiliary_field.query_type = linear

[pylithapp.timedependent.interfaces.fault_slabtop.eq_ruptures.creep]
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Fault rupture auxiliary field spatial database
db_auxiliary_field.iohandler.filename = fault_slabtop_creep.spatialdb
db_auxiliary_field.query_type = linear
```

Running the simulation

Listing 4.220: Run Step 3 simulation

```
$ pylith step03_eqcycle.cfg

# The output should look something like the following.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/apps/
↳ PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/meshio/
↳ MeshIOObj.py:38:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:85:void_
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
```

(continues on next page)

(continued from previous page)

```
-- meshiopetsc(info)
-- Component 'reader': Domain bounding box:
  (-6000000, 6000000)
  (-6000000, 399.651)

# -- many lines omitted --

60 TS dt 0.05 time 2.95
  0 SNES Function norm 5.748198611842e-01
    Linear solve converged due to CONVERGED_ATOL iterations 1
  1 SNES Function norm 2.523053498061e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
61 TS dt 0.05 time 3.
>> /software/unix/py3.12-venv/pylith-debug/lib/python3.12/site-packages/pylith/problems/
↪ Problem.py:199:finalize
-- timedependent(info)
-- Finalizing problem.
```

The beginning of the output written to the terminal is identical to that from Steps 1 and 2. At the end of the output, we see that the simulation advanced the solution 61 time steps. Remember that the PETSc TS monitor shows the nondimensionalized time and time step values.

Visualizing the results

In Fig. 4.142 we use the `pylith_viz` utility to visualize the `x` displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.221: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step03_eqcycle-domain.h5 warp_grid --component=x
```

Suggested Exercises

1. Change the resolution of the mesh by editing the `generate_gmsh.py` Gmsh Python script or the `mesh_tri.jou` Cubit journal files. Change the resolution and bias factor.
2. Add depth dependent viscosity to the mantle and crust. This requires using the linear Maxwell plane strain bulk constitutive model in the crust as well and creating spatial databases that include viscosity for the crust. Specifying a depth dependent variation in the parameters will require adding points, updating `num-locs` accordingly, and changing `data-dim` to 1.
3. Modify the spatial database files for the material properties to use depth-dependent elastic properties based on PREM (Dziewonski and Anderson, 1981, 10.1016/0031-9201(81)90046-7). See <http://ds.iris.edu/ds/products/emc-prem/> for a simple table of values. Add points, update `num-locs` accordingly, and change `data-dim` to 1.
4. Create a Cubit journal file `mesh_quad.jou` for generating a mesh with quadrilateral cells instead of triangular cells. This requires using the pave mesh scheme.

Subduction Zone (3D)

The files are in the directory `examples/subduction-3d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

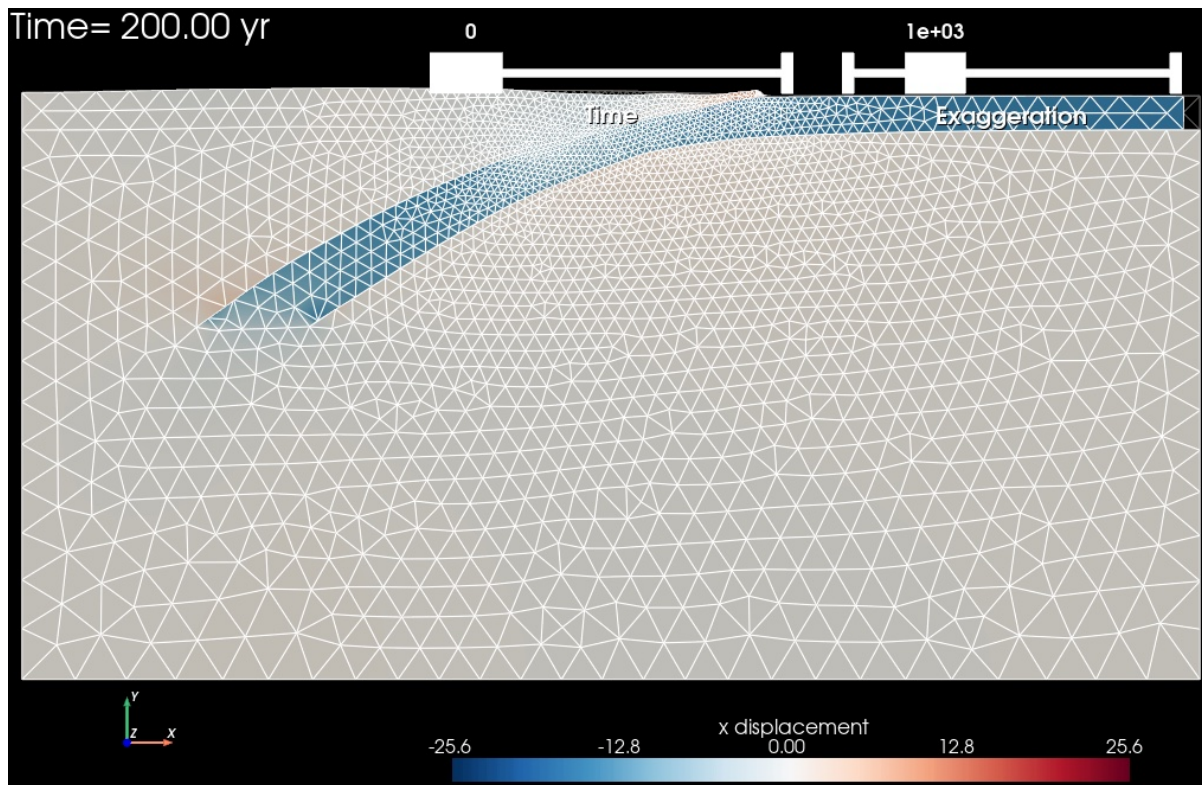


Fig. 4.142: Solution for Step 3 at $t=200$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 1000.

step*.cfg

PyLith parameter files.

mat_*.cfg

Additional PyLith parameter files that are specific to materials used.

***.cfg**

Parameter files in addition to those above that are used with example Python scripts.

***.spatialdb**

Spatial database files associated with the spatial databases.

***.py**

Python scripts used in the examples.

viz

Directory containing Matplotlib Python scripts for visualizing results.

input

Directory for simulation input data that must be downloaded.

output

Directory containing simulation output. It is created automatically when running the simulations.

Details for specific files

cas_contours_dep.in.txt.gz

Contours of the top of the slab geometry of the Cascadia subduction zone from Slab1.

cgns_stations_flat.txt

Coordinates of fake continuous GNSS stations for the CUBIT mesh, which has a flat mesh (top surface $z=0$).

cgns_stations_latlon.txt

Latitude and longitude coordinates of fake continuous GNSS stations.

cgns_stations.txt

Coordinates of fake continuous GNSS stations for the Gmsh mesh, which includes topology and bathymetry; produced by `generate_cgns.py` using the output from Step 1.

coordsys.py

Utility functions to get coordinate systems used in the examples. `:etopo2020_bedrock_local.nc` NetCDF file with local topography and bathymetric from ETopo 2020; used in creating the Gmsh mesh.

fault_slabtop_coseismic.spatialdb

Coseismic slip distribution used in Step 4.

fault_slabtop_creep.spatialdb

Aseismic creep distribution used in Steps 3 and 4.

fault_slabtop_slipweakening.spatialdb

Slip-weakening friction parameters used in Step 5.

fault_slabtop_tractions.spatialdb

Initial fault tractions used in Step 5.

Generated files

These files are not included in the source and are generated by Python scripts as part of the process of running the examples.

cgns_synthetic_displacement.txt

Synthetic data for Step 7 inversion generated from the output of Step 6 with added noise; generated by `generate_synthetic_gnssdisp.py` using the output from Step 6.

fault_slabtop_slowslip.spatialdb

Slip distribution used in Step 6; generated by `generate_slowslip.py`.

fault_slabtop_slowslip.timedb

Time history of slip used in Step 6; generated by `generate_slowslip.py`.

mat_refstate_grav-*.spatialdb

Reference stress state for Step 8c; generated by `generate_refstate_grav.py`.

Important

The input mesh files for this example are not included in the source distribution. Instructions for downloading the mesh files are available in `input/README.md`.

Overview

This suite of examples demonstrates use of a wide variety of features and the general workflow often used in research simulations. We base the model on the Cascadia subduction zone (Fig. 4.143). These examples focus on modeling the deformation associated with the the subducting slab, including interseismic deformation with aseismic slip (creep) and viscoelastic relaxation, coseismic slip on the slab interface and a splay fault, and slow slip events on the subduction interface. We account for the 3D material properties associated with different elastic properties for the subducting slab, mantle, continental crust, and accretionary wedge. To keep the computation time in these examples short, we limit our model to an $800 \text{ km} \times 800 \text{ km} \times 400 \text{ km}$ domain and we use a relatively coarse discretization.

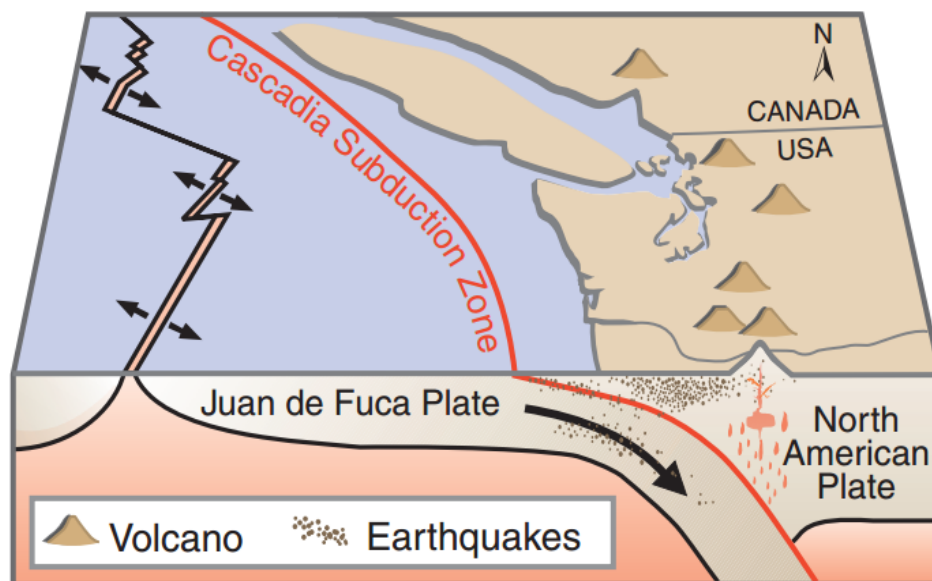


Fig. 4.143: Cartoon of the Cascadia Subduction Zone showing the subduction of the Juan de Fuca Plate under the North American Plate. Source - U.S. Geological Survey Fact Sheet 060-00

Fig. 4.144 shows our conceptual model with a slab, mantle, continental crust, and accretionary wedge. We cut off the slab at a depth of 100 km. We use a transverse geographic projection coordinate system with Portland, Oregon, as the origin to georeference our model. To model the motion of the slab, we include a fault for the subduction interface (the interface between the top of the slab and the mantle, crust, and wedge), as well as a fault between the bottom of the slab and the mantle.

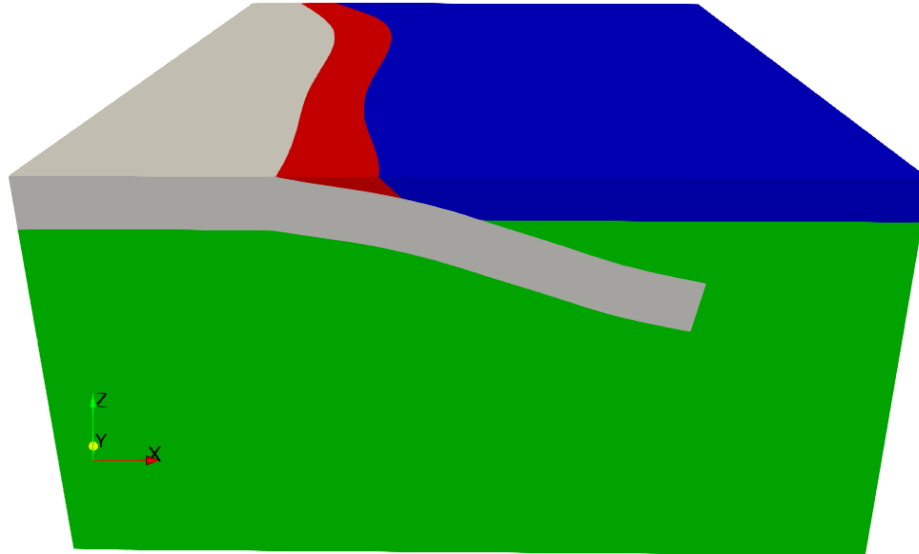


Fig. 4.144: Conceptual model based on the Cascadia Subduction Zone. The model includes the subduction slab (white), the mantle (green), continental crust (blue), and an accretionary wedge (red).

Step 1

Static axial compression.

Step 2

Quasi-static coseismic and postseismic deformation from an earthquake rupture in the center of the subduction zone interface.

Step 3

Quasi-static interseismic deformation with creep on the top and bottom of the slab.

Step 4

Quasi-static earthquake cycle with prescribed earthquake rupture and creep.

Step 5

Spontaneous rupture driven by subducting slab.

Step 6

Prescribed slow-slip event.

Step 7a,b

Generation of 3D Green's functions and inversion of displacements for slow slip event from Step 6.

Step 8a,b,c

Stress field due to gravitational body forces.

We provide Python scripts for generating finite-element meshes using Gmsh and Cubit and PyLith parameter files for using each of the mesh files.

Warning

Step 5 has not yet been updated for PyLith v3+.

Danger

These simulations are significantly larger than those for the other suites of examples. Some of the time-dependent simulations may take 30 minutes or more to run and use up to 4 GB of memory.

Example Workflow

Gmsh Mesh

We use Gmsh to generate the finite-element mesh. Due to its size, we do not include the finite-element mesh in the PyLith source or binary distributions. See the instructions in the `input/README.md` file for how to download the mesh produced by Gmsh.

Overview

We use contours of the Cascadia Subduction Zone from Slab v1.0 [Hayes *et al.*, 2012] for the geometry of the subduction interface. We define the top surface using topography and bathymetry from Etopo 2022 [NOAA National Centers for Environmental Information, 2022]. The data files `cas_contours_dep.in.txt.gz` and `etopo2020_bedrock_local.nc` provide the subduction interface contours and the topography and bathymetry, respectively. The Python script `generate_gmsh.py` reads in the subduction interface contour files and the topography and bathymetry grid file, constructs the geometry, marks the boundaries, fault surfaces, and materials, and generates the finite-element mesh.

Tip

We define the coordinate systems we use in the simulations in the Python script `coordsys.py` to make it easier to convert to and from various georeference coordinate systems in the pre- and post-processing. PyLith will automatically convert among compatible coordinate systems during the simulation.

Listing 4.222: [Optional] Generate the finite-element mesh using `generate_gmsh.py`. If you do not want to generate the finite-element mesh, refer to the instructions in `input/README.md` on how to download the Gmsh file.

```
$ ./generate_gmsh.py --write --filename=mesh_tet.msh

# When creating the geometry you should see the following output. If your output does not
# match, you likely need to update the id values in the generate_gmsh.py script.
Fragment domain with topography, tags=[(3, 2), (3, 3), (2, 27)]. Setting domain=(3, 2),
↳ deleting=[(3, 3), (2, 27)]
Fragment domain with slab.
Fragment domain with slab, tags=[(3, 1), (3, 2), (3, 3)]. Setting domain=(3, 1), slab=(3,
↳ 2), deleting=[(3, 3)]
Fragment domain with splay fault, tags=[(3, 3), (3, 4), (2, 41)]. Setting domain=(3, 3),
↳ slab=(3, 2) wedge=(3, 4), deleting=[(2, 41)]
Fragment domain with moho, tags=[(3, 5), (3, 6), (2, 53)]. Setting domain=(3, 5),
↳ deleting=[(3, 6), (2, 53)]
```

(continues on next page)

(continued from previous page)

```
↪crust=(3, 6), slab=(3, 2) wedge=(3, 4), deleting=[(2, 53)]  
Fragment slab with patch, tags=[(3, 7), (3, 8), (3, 9), (3, 10)]. Setting domain=(3, 5), ↪  
↪crust=(3, 6), slab=[(3, 7), (3, 8)] wedge=(3, 4), deleting=[(3, 9), (3, 10)]
```

Geometry

We construct the geometry with the following steps:

1. Construct basic building blocks.
 1. Load the topography and bathymetry values and construct a surface.
 2. Load the subduction interface contours and construct a surface.
 3. Calculate the average surface normal for the top of the slab.
 4. Construct the bottom of the slab and build a volume for the slab.
 5. Construct a splay fault using the slab interface contours.
 6. Construct a box for the domain.
 7. Construct a surface corresponding to the continental Moho.
2. Combine the basic building blocks.
 1. Chop off the top of the domain using the topography and bathymetry surface.
 2. Fragment the domain with the slab volume.
 3. Fragment the domain with the splay surface to create a wedge.
 4. Fragment the domain with the continental Moho surface.
 5. Create a patch for the subduction interface and fragment the domain with it.
3. Get tags of the various surfaces for use in marking boundaries, faults, and materials.

Mark boundaries, faults, and materials

1. Create material groups for each volume to make it easy to assign different material properties to each one.
2. Create boundary groups corresponding to boundary conditions and faults.

Generate the mesh

We create a cell size field to increase the discretization size with distance from the top of the subduction interface. We generate the mesh and then use Gmsh mesh quality tools to improve the shape of the cells.

Cubit Mesh

We use Cubit to generate the finite-element mesh. Due to its size, we do not include the finite-element mesh in the PyLith source or binary distributions. See the instructions in the `input/README.md` file for how to download the mesh.

Overview

We use contours of the Cascadia Subduction Zone from Slab v1.0 [Hayes *et al.*, 2012] for the geometry of the subduction interface. We ignore topography and bathymetry and use a flat surface at an elevation of mean sea level. The data file

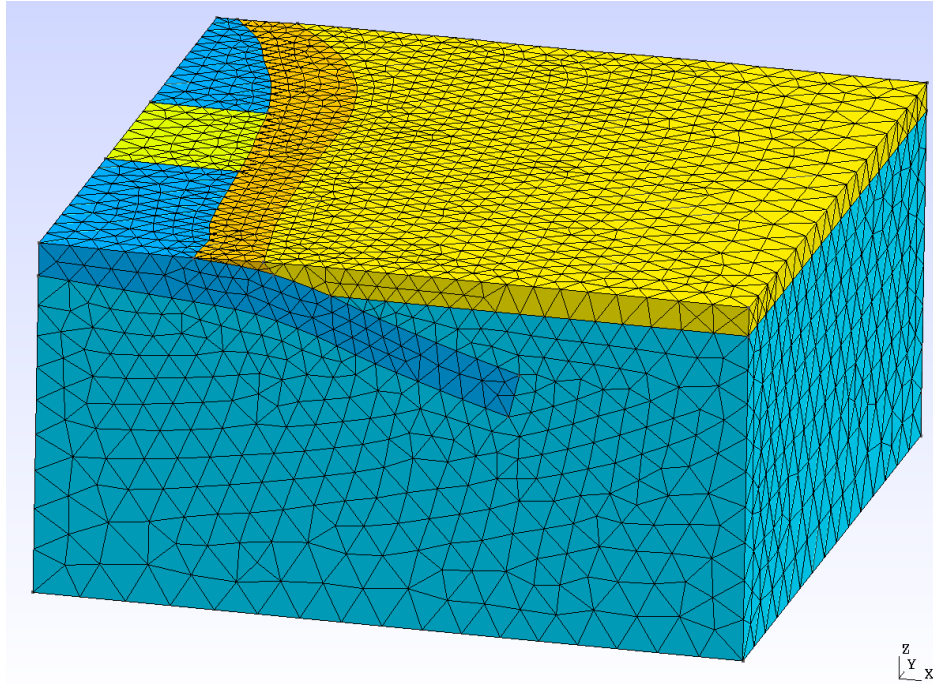


Fig. 4.145: Finite-element mesh with tetrahedral cells generated by Gmsh.

`cas_contours_dep.in.txt.gz` provides the subduction interface contours. The Python script `generate_cubit.py` reads in the subduction interface contour files, constructs the geometry, marks the boundaries, fault surfaces, and materials, and generates the finite-element mesh.

Tip

We define the coordinate systems we use in the simulations in the Python script `coordsys.py` to make it easier to convert to and from various georeference coordinate systems in the pre- and post-processing. PyLith will automatically convert among compatible coordinate systems during the simulation.

Important

We recommend running the `generate_cubit.py` Python script from within the Cubit application. You can copy and paste the lines into the command console or load the Python script into the Cubit script editor.

Geometry

We construct the geometry with the following steps:

1. Construct basic building blocks.
 1. Construct a box for the domain.
 2. Load the subduction interface contours and construct a surface.
 3. Calculate the average surface normal for the top of the slab.
 4. Construct the bottom of the slab and build a volume for the slab.

5. Construct a splay fault using the slab interface contours.
 6. Create a box to subdivide the surface for the top of the slab.
 7. Construct a surface corresponding to the continental Moho.
2. Combine the basic building blocks.
 1. Fragment the domain with the slab volume.
 2. Fragment the domain with the continental Moho surface.
 3. Fragment the domain with the splay surface to create a wedge.
 4. Inscribe the patch onto the top of the slab.
 3. Get ids of the various surfaces for use in marking boundaries, faults, and materials.

Generate the mesh

We increase the discretization size with distance from the top of the subduction interface. We generate the mesh and then use Gmsh mesh quality tools to improve the shape of the cells.

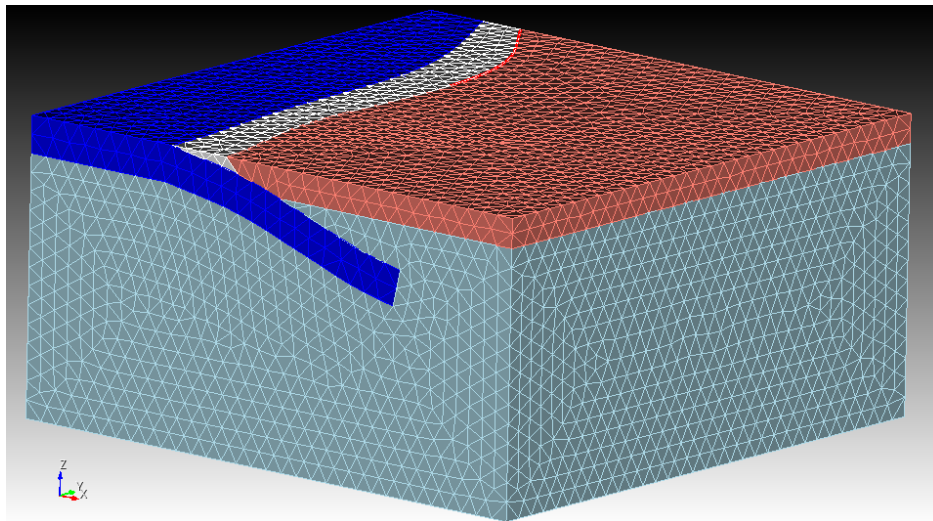


Fig. 4.146: Finite-element mesh with tetrahedral cells generated by Cubit.

Mark boundaries, faults, and materials

1. Create material blocks for each volume to make it easy to assign different material properties to each one.
2. Create sidesets corresponding to boundary conditions and faults.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`, which contains numerous comments, so we only summarize the parameters here.

The settings contained in `pylithapp.cfg` for this problem consist of:

`pylithapp.metadata`

Metadata common to all of the simulations in the directory.

pylithapp.mesh_generator

Parameters for importing the finite-element mesh.

pylithapp.problem

Parameters that define the boundary value problem and its solution, such as the type of solver, solution fields, and output over the domain.

pylithapp.problem.materials

Parameters that specify the governing equation and bulk rheologies.

pylithapp.problem.bc

Parameters that specify the boundary conditions.

To make it easier to switch between different bulk rheologies for different simulations, we separate the material parameters relevant to the bulk rheologies into three parameter files: `mat_elastic.cfg`, `mat_viscoelastic.cfg`, and `mat_elastic_incompressible.cfg`. We use `mat_elastic.cfg` when we want all materials to use elastic bulk rheologies. We use `mat_viscoelastic.cfg` when we want the mantle and bottom of the slab to use viscoelastic bulk rheologies. We use `mat_elastic_incompressible.cfg` only for Step 8b in which we use incompressible elasticity. The physical properties for each material are specified in spatial database files. For example, the elastic properties for the crust are in `mat_crust_elastic.spatialdb`. The spatial database files for elastic properties all use just a single point to specify uniform physical properties within each material.

The distorted cells at the trench result in a small degradation in the preconditioning, so the default KSP absolute tolerance is not quite small enough for the nonlinear solver in a single iteration. Consequently, we reduce the KSP absolute tolerance to $2.0\text{e-}8$, which allow the nonlinear solver to converge in a single iteration.

Step 1: Axial Compression

This example involves a static simulation that solves for the deformation from axial compression. Fig. 4.147 shows the boundary conditions on the domain.

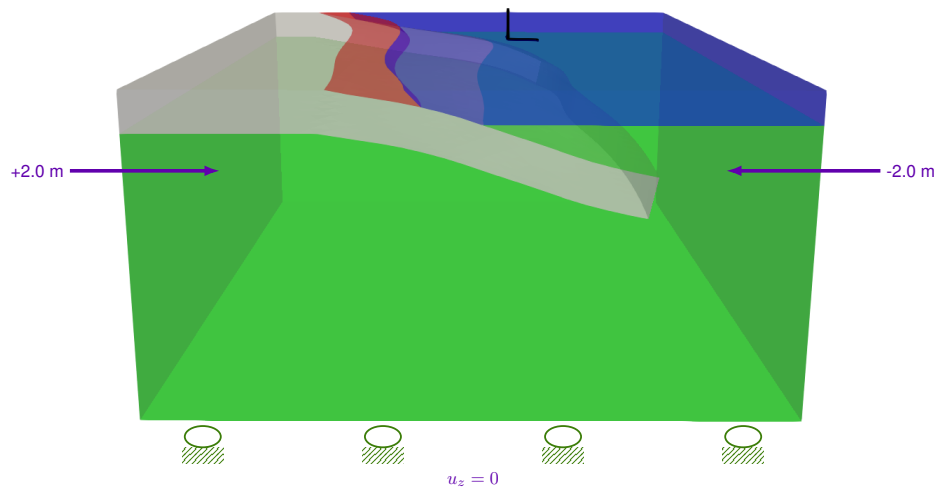


Fig. 4.147: Boundary conditions for axial compression in the x direction with roller boundary conditions on the -y, +y, and -z boundaries.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOcubit`
- `pylith.problems.TimeDependent`

- pylith.meshio.OutputSolnBoundary
- pylith.meshio.DataWriterHDF5
- pylith.bc.DirichletTimeDependent
- pylith.bc.ZeroDB
- spatialdata.geocoords.CSGeo
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.SimpleDB
- Static simulation
- pylith.materials.IsotropicLinearElasticity
- spatialdata.spatialdb.SimpleDB

Simulation parameters

The parameters specific to this example are in `step01_axialdisp.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output.
- `pylithapp.problem.bc` Parameters for describing the boundary conditions that override the defaults.

We override the parameters for the Dirichlet displacement boundary conditions on the `-x` and `+x` boundaries. We replace the `ZeroDB` spatial database for zero displacement values with a `UniformDB` to impose axial compression with 2.0 m of displacement on the two boundaries. For a more efficient solve we use the PETSc default solver options for elasticity in parallel; for larger simulations these are sometimes more efficient than the defaults for running in serial.

Listing 4.223: Run Step 1 simulation using the Gmsh mesh.

```
$ pylith step01_axialdisp.cfg mat_elastic.cfg

# The output should look something like the following.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/meshio/MeshIOObj.py:41:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:74:void_
↪pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- meshiopetsc(info)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-400000, 400000)
    (-400000, 400000)
    (-400000, 2017.5)

# -- many lines omitted --
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.py:250:_
↪printInfo
```

(continues on next page)

(continued from previous page)

```

-- timedependent(info)
-- Scales for nondimensionalization:
    Length scale: 20000*m
    Displacement scale: 1*m
    Time scale: 3.15576e+09*s
    Rigidity scale: 1e+10*m**-1*kg*s**-2
    Temperature scale: 1*K
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:217:initialize
-- timedependent(info)
-- Initializing timedependent problem with quasistatic formulation.
>> /src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:262:static void pylith::utils::_
↳PetscOptions::write(pythia::journal::info_t&, const char*, const_
↳pylith::utils::PetscOptions&)
-- petscoptions(info)
-- Setting PETSc options:
ksp_converged_reason = true
ksp_error_if_not_converged = true
ksp_gmres_restart = 100
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-14
mg_fine_ksp_max_it = 5
pc_type = gamg
snes_atol = 5.0e-7
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-14
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_type = beuler
viewer_hdf5_collective = true
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/TimeDependent.
↳py:145:run
-- timedependent(info)
-- Solving problem.
0 TS dt 0.001 time 0.
    0 SNES Function norm 1.673924902061e+03
      Linear solve converged due to CONVERGED_ATOL iterations 14
    1 SNES Function norm 1.647222821241e-07
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 0.001 time 0.001
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:232:finalize
-- timedependent(info)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the MeshIOPetsc reader and that it found the domain to extend from -400 km to +400 km in the x direction, from -400 km to +400 km in the y direction, and from -400 km to +2 km in the z direction. The output also includes the scales used for nondimensionalization and the default PETSc options.

At the end of the output written to the terminal, we see that the solver advanced the solution 1 time step (static simulation). The linear solve converged after 14 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`).

Listing 4.224: Alternatively, run Step 1 simulation using the Cubit mesh.

```
$ pylith step01_axialdisp.cfg step01_axialdisp_cubit.cfg mat_elastic.cfg
```

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (`.h5`) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (`.xmf`) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In [Fig. 4.148](#) we use the `pylith_viz` utility to visualize the x displacement field.

Listing 4.225: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step01_axialdisp-domain.h5 warp_grid --component=x --  
↪exaggeration=10000
```

Step 2: Earthquake Rupture and Postseismic Relaxation

This example involves a quasi-static simulation for coseismic earthquake rupture and postseismic relaxation. We use linear Maxwell viscoelastic bulk rheologies in the mantle and deeper part of the slab. [Fig. 4.149](#) shows the boundary conditions on the domain.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOcubit`
- `pylith.problems.TimeDependent`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.ZeroDB`
- `spatialdata.geocoords.CSGeo`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.materials.IsotropicLinearMaxwell`
- `spatialdata.spatialdb.CompositeDB`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.SimpleGridDB`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`

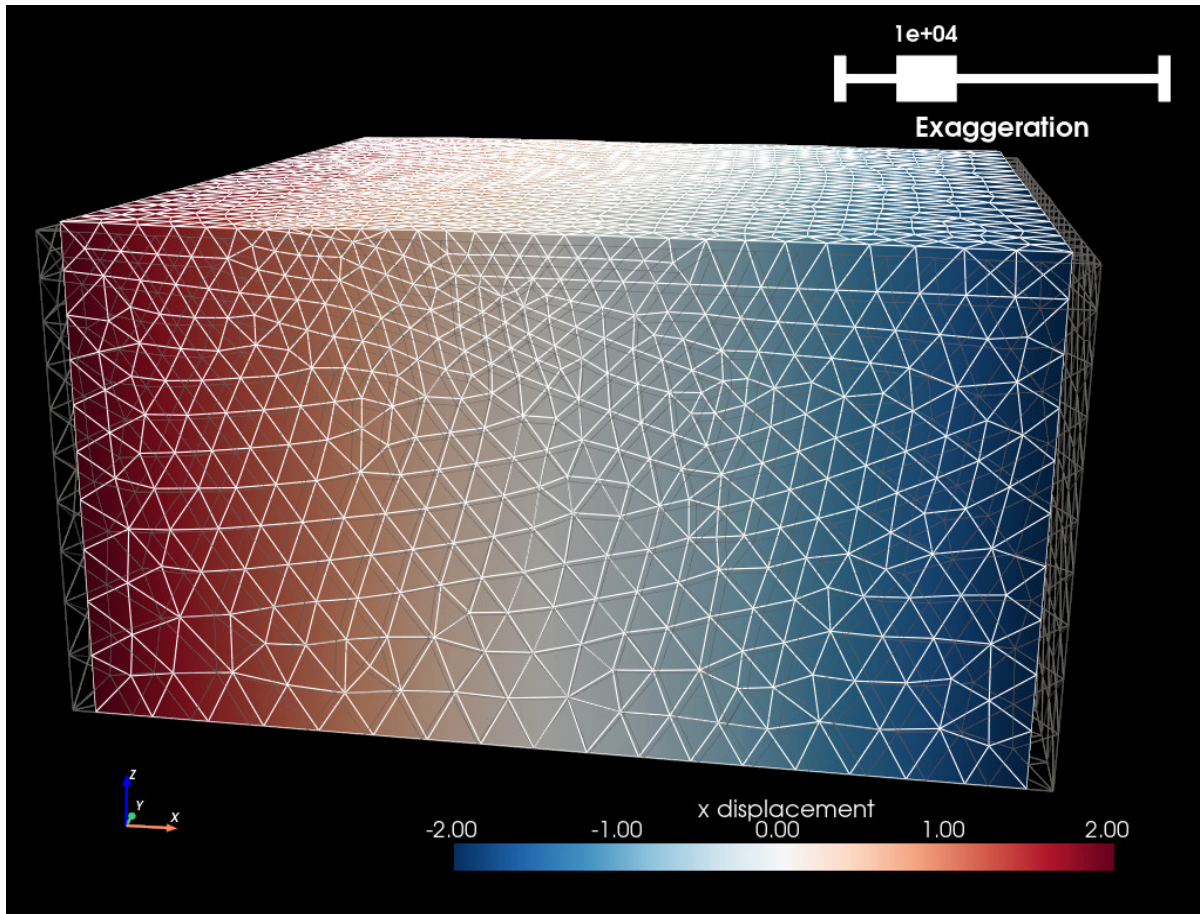


Fig. 4.148: Solution for Step 1. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 10,000.

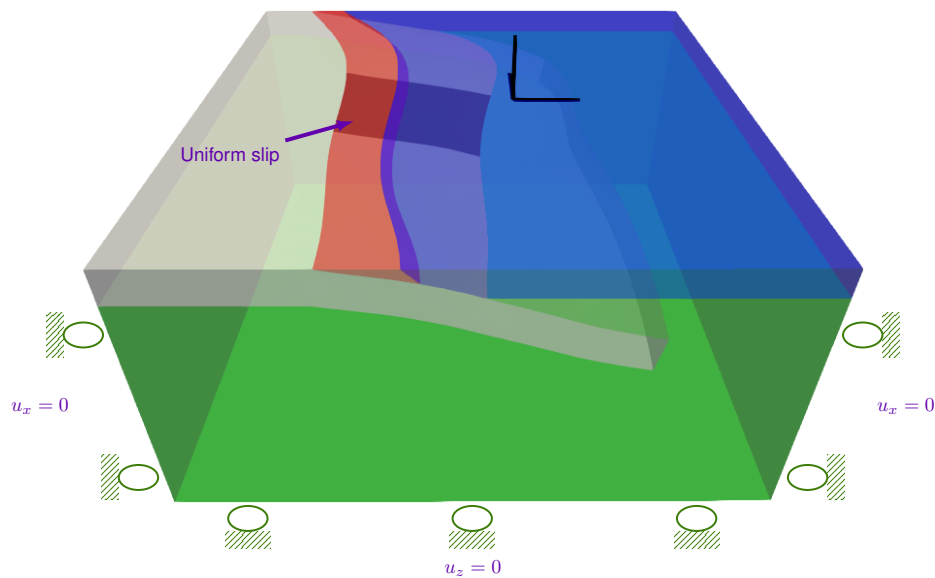


Fig. 4.149: Boundary conditions for quasi-static coseismic slip on the subduction interface and postseismic relaxation. We prescribe uniform oblique slip in the center of the subduction interfaces with roller boundary conditions on the lateral sides and bottom of the domain.

- `pylith.materials.IsotropicLinearMaxwell`
- `spatialdata.spatialdb.CompositeDB`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.SimpleGridDB`
- Quasi-static simulation
- `pylith.faults.KinSrcStep`

Simulation parameters

The parameters specific to this example are in `step02_coseismic.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output.
- `pylithapp.problem` Parameters for the time step information as well as solution field with displacement and Lagrange multiplier subfields.
- `pylithapp.interfaces` Parameters for the earthquake rupture.

We define the duration of the simulation to be 200 years with an initial time step of 0.1 years. We enable adaptive time stepping and start with a small time step to resolve the viscoelastic deformation.

For the fault slip, we use the boundary groups for the fault and its buried edges corresponding to the central patch of the top of the slab. We set the slip initiation time to 10 years with the default step time function for the earthquake rupture. Alternatively, we could have set the `origin_time` of the earthquake rupture to 10 years and used an `initiation_time` of 0 years. We impose oblique slip with 1.0 m of right-lateral slip and 4.0 m of reverse slip.

Listing 4.226: Run Step 2 simulation using the Gmsh mesh.

```
$ pylith step02_coseismic.cfg mat_viscoelastic.cfg

# The output should look something like the following.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/meshio/MeshIOObj.py:41:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:74:void
↳pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- meshiopetsc(info)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-4000000, 4000000)
    (-4000000, 4000000)
    (-4000000, 2017.5)

# -- many lines omitted --
>> /src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:262:static void pylith::utils::_
↳PetscOptions::write(pythia::journal::info_t&, const char*, const
↳pylith::utils::PetscOptions&)
-- petscoptions(info)
```

(continues on next page)

(continued from previous page)

```

-- Setting PETSc options:
dm_reorder_section = true
dm_reorder_section_type = cohesive
ksp_converged_reason = true
ksp_error_if_not_converged = true
ksp_gmres_restart = 100
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-14
mg_fine_ksp_max_it = 5
mg_fine_pc_type = vpbjacobi
pc_type = gamg
snes_atol = 5.0e-7
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-14
ts_adapt_monitor = true
ts_adapt_reject_safety = 0.1
ts_adapt_safety = 0.2
ts_adapt_type = basic
ts_atol = 0.05
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_rtol = 0.05
ts_type = beuler
viewer_hdf5_collective = true
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/TimeDependent.
↳py:145:run
-- timedependent(info)
-- Solving problem.
0 TS dt 0.001 time -0.001
    0 SNES Function norm 2.954073482783e+01
      Linear solve converged due to CONVERGED_ATOL iterations 16
    1 SNES Function norm 1.956521733099e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
      TSAdapt basic beuler 0: step 0 accepted t=-0.001 + 1.000e-03 dt=1.000e-03

# -- many lines omitted --

9 TS dt 0.496292 time 1.50371
    0 SNES Function norm 2.765617769230e+00
      Linear solve converged due to CONVERGED_ATOL iterations 12
    1 SNES Function norm 1.288354163470e-07
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
      TSAdapt basic beuler 0: step 9 accepted t=1.50371 + 4.963e-01 dt=1.373e+00
↳wlte=0.00523 wltea= -1 wlter= -1
9 TS dt 0. time 1.50371
    0 SNES Function norm 2.765639325307e+00
      Linear solve converged due to CONVERGED_ATOL iterations 14
    1 SNES Function norm 9.313028842369e-09

```

(continues on next page)

(continued from previous page)

```

    Nonlinear496294ar solve converged due to CONVERGED_FNORM_ABS iterations 1
      TSAdapt basic beuler 0: step 9 accepted t=1.50371 + 4.963e-01 dt=1.373e+00
↪ wlte=0.00523 wltea= -1 wlter= -1
10 TS dt 1.37262 time 2.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
↪ py:232:finalize
-- timedependent(info)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the MeshIOPetsc reader. We also see the PETSc solver options, which show use of the variable point-block Jacobi and GAMG (algebraic multigrid) preconditioner settings.

At the end of the output written to the terminal, we see that the solver advanced the solution 10 time steps. The linear solve converged after 14 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snes_atol`).

Listing 4.227: Alternatively, run Step 2 simulation using the Cubit mesh.

```
$ pylith step02_coseismic.cfg step02_coseismic_cubit.cfg mat_viscoelastic.cfg
```

Visualizing the results

In Fig. 4.150 we use the `pylith_viz` utility to visualize the x displacement field. You can move the slider or use the p and n keys to change the increment or decrement time.

Listing 4.228: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filename=output/step02_coseismic-domain.h5 warp_grid --component=x --
↪ exaggeration=10000

```

Step 3: Interseismic Deformation

This example involves a quasi-static simulation for interseismic deformation. We prescribe aseismic slip (creep) on the bottom of the slab and the deeper portion of the top of the slab; the shallow portion of the top of the slab remains locked. We use linear Maxwell viscoelastic bulk rheologies in the mantle and deeper part of the slab. Fig. 4.151 shows the boundary conditions on the domain.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOCubit`
- `pylith.problems.TimeDependent`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.ZeroDB`
- `spatialdata.geocoords.CSGeo`
- `pylith.materials.Elasticity`

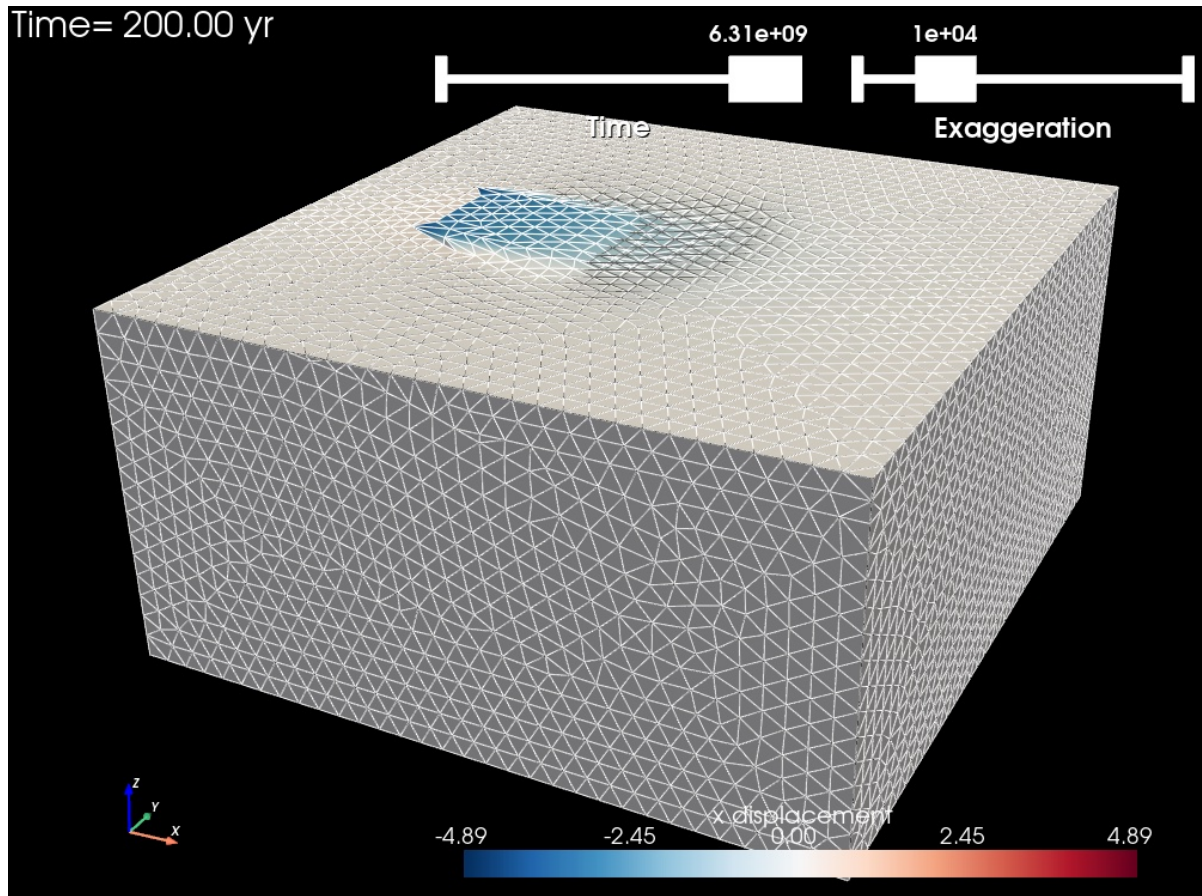


Fig. 4.150: Solution for Step 2 at $t=200$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 10,000.

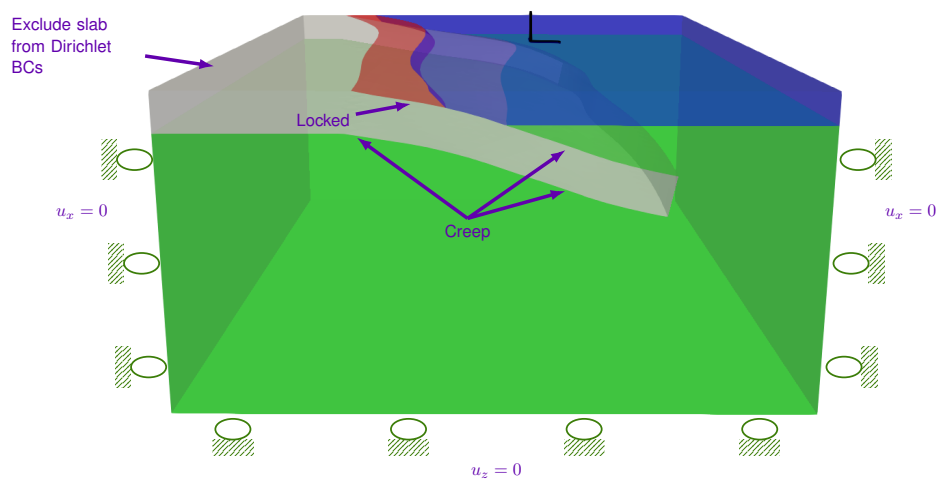


Fig. 4.151: Boundary conditions for quasi-static interseismic deformation. We prescribe aseismic slip (creep) on the bottom of the slab and the deeper portion of the top of the slab; the shallow portion of the top of the slab remains locked.

- pylith.materials.IsotropicLinearElasticity
- pylith.materials.IsotropicLinearMaxwell
- spatialdata.spatialdb.CompositeDB
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.SimpleGridDB
- pylith.materials.Elasticity
- pylith.materials.IsotropicLinearElasticity
- pylith.materials.IsotropicLinearMaxwell
- spatialdata.spatialdb.CompositeDB
- spatialdata.spatialdb.SimpleDB
- spatialdata.spatialdb.SimpleGridDB
- Quasi-static simulation
- pylith.faults.KinSrcConstRate

Simulation parameters

The parameters specific to this example are in `step03_interseismic.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output.
- `pylithapp.problem` Parameters for the time step information as well as solution field with displacement and Lagrange multiplier subfields.
- `pylithapp.interfaces` Parameters for the aseismic slip (creep) on the top and bottom of the slab.
- `pylithapp.problem.bc` Parameters for describing the boundary conditions that override the defaults.

For aseismic slip we use the `KinSrcConstRate` kinematic source to prescribe a constant slip rate. We also adjust the groups used for the boundary conditions to remove overlap with the slab to allow the slab to move independently. We adjust the displacement scale to match the slow deformation and tighten the KSP absolute tolerance to facilitate convergence in a single SNES iteration.

Listing 4.229: Run Step 3 simulation using the Gmsh mesh.

```
$ pylith step03_interseismic.cfg mat_viscoelastic.cfg

# The output should look something like the following.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/meshio/MeshIOObj.py:41:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:74:void_
↪pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- meshiopetsc(info)
-- Component 'meshiopetsc.reader': Domain bounding box:
```

(continues on next page)

(continued from previous page)

```

(-400000, 400000)
(-400000, 400000)
(-400000, 2017.5)

# -- many lines omitted --
>> /src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:262:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t&, const char*, const
↳ pylith::utils::PetscOptions&)
-- petscoptions(info)
-- Setting PETSc options:
dm_reorder_section = true
dm_reorder_section_type = cohesive
ksp_converged_reason = true
ksp_error_if_not_converged = true
ksp_gmres_restart = 100
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-14
mg_fine_ksp_max_it = 5
mg_fine_pc_type = vpbjacobi
pc_type = gamg
snes_atol = 5.0e-7
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-14
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_type = beuler
viewer_hdf5_collective = true

# -- many lines omitted --

20 TS dt 0.1 time 1.9
   0 SNES Function norm 3.124244823506e+03
     Linear solve converged due to CONVERGED_ATOL iterations 22
   1 SNES Function norm 1.157676866316e-07
     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
21 TS dt 0.1 time 2.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:232:finalize
-- timedependent(info)
-- Finalizing problem.

```

The beginning of the output is nearly the same as in Step 2. The simulation advances 21 time steps. The linear solve converged after 22 iterations and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). In this simulation the fault interfaces on the top and bottom of the slab occupy a significant fraction of the domain. As a result, the linear solver requires more iterations to converge compared to the limited fault interface in Step 2.

Listing 4.230: Alternatively, run Step 3 simulation using the Cubit mesh.

```
$ pylith step03_interseismic.cfg step03_interseismic_cubit.cfg mat_viscoelastic.cfg
```

Visualizing the results

In Fig. 4.152 we use the `pylith_viz` utility to visualize the x displacement field. You can move the slider or use the p and n keys to change the increment or decrement time.

Listing 4.231: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step03_interseismic-domain.h5 warp_grid --component=x --  
↩exaggeration=5000
```

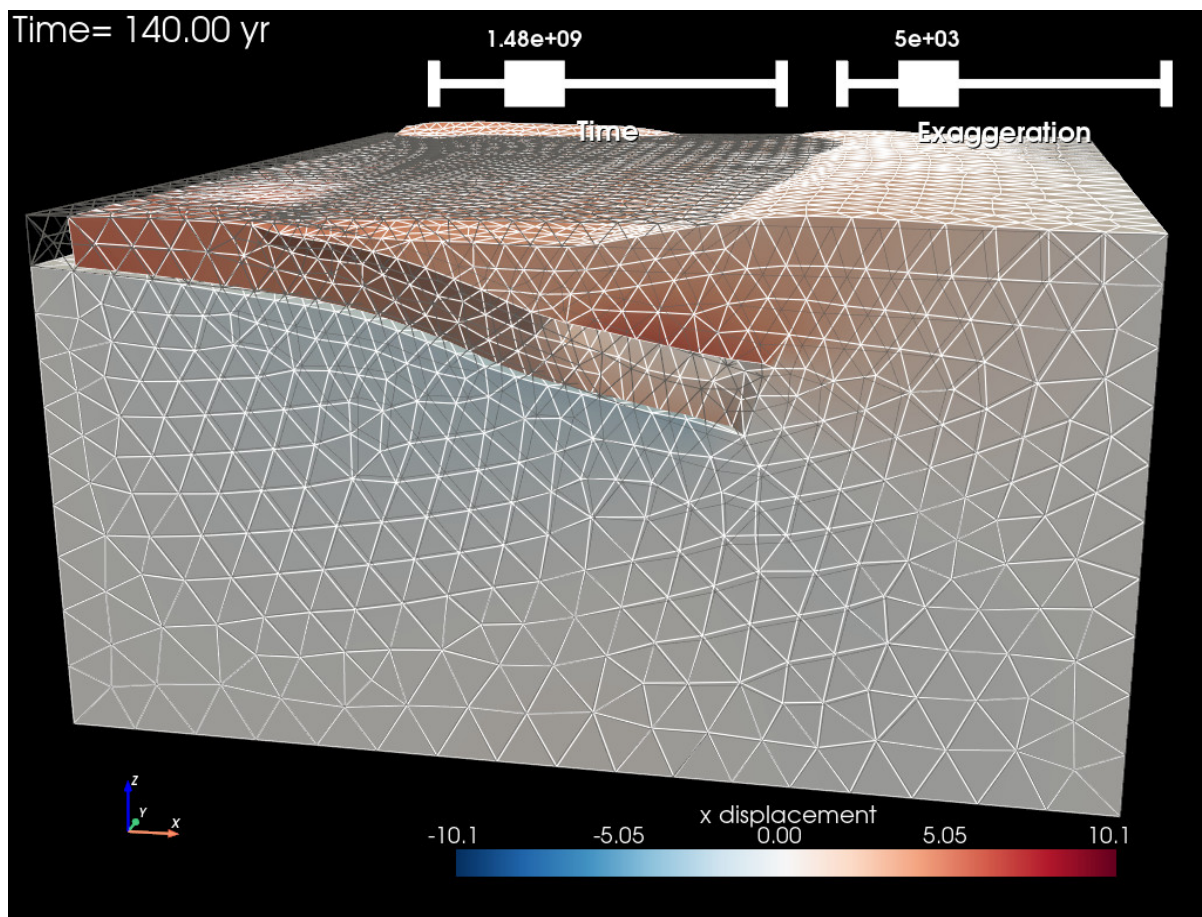


Fig. 4.152: Solution for Step 3 at $t=140$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 5000.

Step 4: Earthquake Cycle with Prescribed Slip

This example combines the interseismic deformation from Step 3 and expands on the earthquake ruptures from Step 2. We expand the earthquake ruptures to span the entire along strike length of the top of the slab and also consider earthquake rupture on the splay fault. We use linear Maxwell viscoelastic bulk rheologies in the mantle and deeper part of the slab. Fig. 4.153 shows the boundary conditions on the domain.

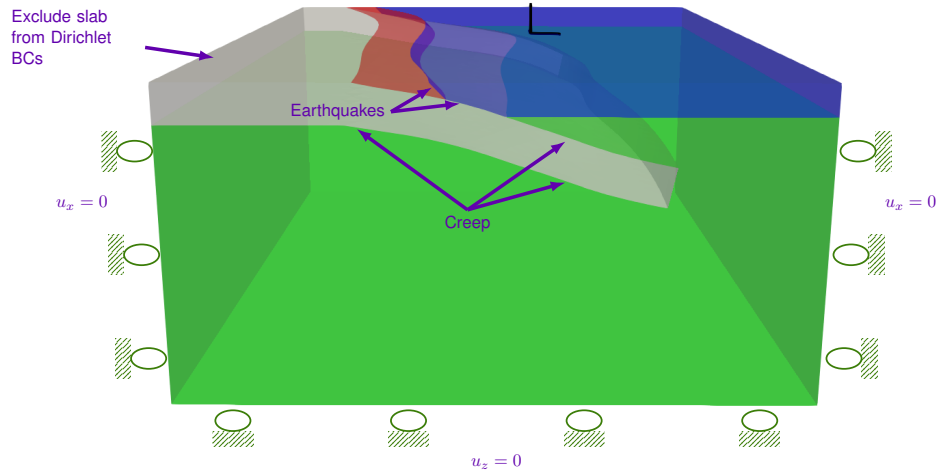


Fig. 4.153: Boundary conditions for quasi-static interseismic deformation. We prescribe aseismic slip (creep) on the bottom of the slab and the deeper portion of the top of the slab; the shallow portion of the top of the slab remains locked.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOCubit`
- `pylith.problems.TimeDependent`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.ZeroDB`
- `spatialdata.geocoords.CSGeo`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.materials.IsotropicLinearMaxwell`
- `spatialdata.spatialdb.CompositeDB`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.SimpleGridDB`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `pylith.materials.IsotropicLinearMaxwell`
- `spatialdata.spatialdb.CompositeDB`
- `spatialdata.spatialdb.SimpleDB`
- `spatialdata.spatialdb.SimpleGridDB`
- Quasi-static simulation

- pylith.faults.KinSrcConstRate
- pylith.faults.KinSrcStep

Simulation parameters

The parameters specific to this example are in `step04_eqcycle.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output.
- `pylithapp.problem` Parameters for the time step information as well as solution field with displacement and Lagrange multiplier subfields.
- `pylithapp.interfaces` Parameters for the earthquake ruptures and aseismic slip (creep) on the top and bottom of the slab.
- `pylithapp.problem.bc` Parameters for describing the boundary conditions that override the defaults.

We extend the duration of the simulation to 300 years. We impose two earthquake ruptures on slab interface at $t=100$ and $t=200$ years and one earthquake rupture on the splay fault at $t=250$ years. Now that we have both earthquake rupture and aseismic creep on the top of the slab, we use `SimpleDB` spatial databases to give depth-dependent, complementary slip. We impose uniform slip on the splay fault using a `UniformDB`. As in Step 3 we also adjust the nodesets used for the boundary conditions to remove overlap with the slab to allow the slab to move independently. Note that the aseismic slip uses `KinSrcConstRate`, while the two ruptures use the default `KinSrcStep`.

Listing 4.232: Run Step 4 simulation using the Gmsh mesh.

```
$ pylith step04_eqcycle.cfg mat_viscoelastic.cfg

# The output should look something like the following.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/meshio/MeshIOObj.py:41:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:74:void_
↳pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- meshiopetsc(info)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-4000000, 4000000)
    (-4000000, 4000000)
    (-4000000, 2017.5)

# -- many lines omitted --
>> /src/cig/pylith/libsrc/pylith/utils/PetscOptions.cc:262:static void pylith::utils::_
↳PetscOptions::write(pythia::journal::info_t&, const char*, const_
↳pylith::utils::PetscOptions&)
-- petscoptions(info)
-- Setting PETSc options:
dm_reorder_section = true
dm_reorder_section_type = cohesive
ksp_converged_reason = true
```

(continues on next page)

(continued from previous page)

```

ksp_error_if_not_converged = true
ksp_gmres_restart = 100
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-14
mg_fine_ksp_max_it = 5
mg_fine_pc_type = vpbjacobi
pc_type = gamg
snes_atol = 5.0e-7
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-14
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_type = beuler
viewer_hdf5_collective = true

# -- many lines omitted --

30 TS dt 0.1 time 2.9
    0 SNES Function norm 3.515714246561e+03
      Linear solve converged due to CONVERGED_ATOL iterations 22
    1 SNES Function norm 8.157356157583e-07
      Linear solve converged due to CONVERGED_ATOL iterations 1
    2 SNES Function norm 6.740861225135e-07
      Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE iterations 2
31 TS dt 0.1 time 3.
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
→py:232:finalize
-- timedependent(info)
-- Finalizing problem.

```

The beginning of the output is near the same as in Steps 2 and 3. The simulation advances 31 time steps. As in Step 3 each linear solve requires about 20 iterations to converge. Some time steps require 2 SNES iterations (likley due to poor mesh quality along the trench) although the linear solve in the second SNES iteration requires no more than 1 iterations in most cases.

Listing 4.233: Alternatively, run Step 4 simulation using the Cubit mesh.

```
$ pylith step04_eqcycle.cfg step04_eqcycle_cubit.cfg mat_viscoelastic.cfg
```

Visualizing the results

In Fig. 4.154 we use the `pylith_viz` utility to visualize the `x` displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

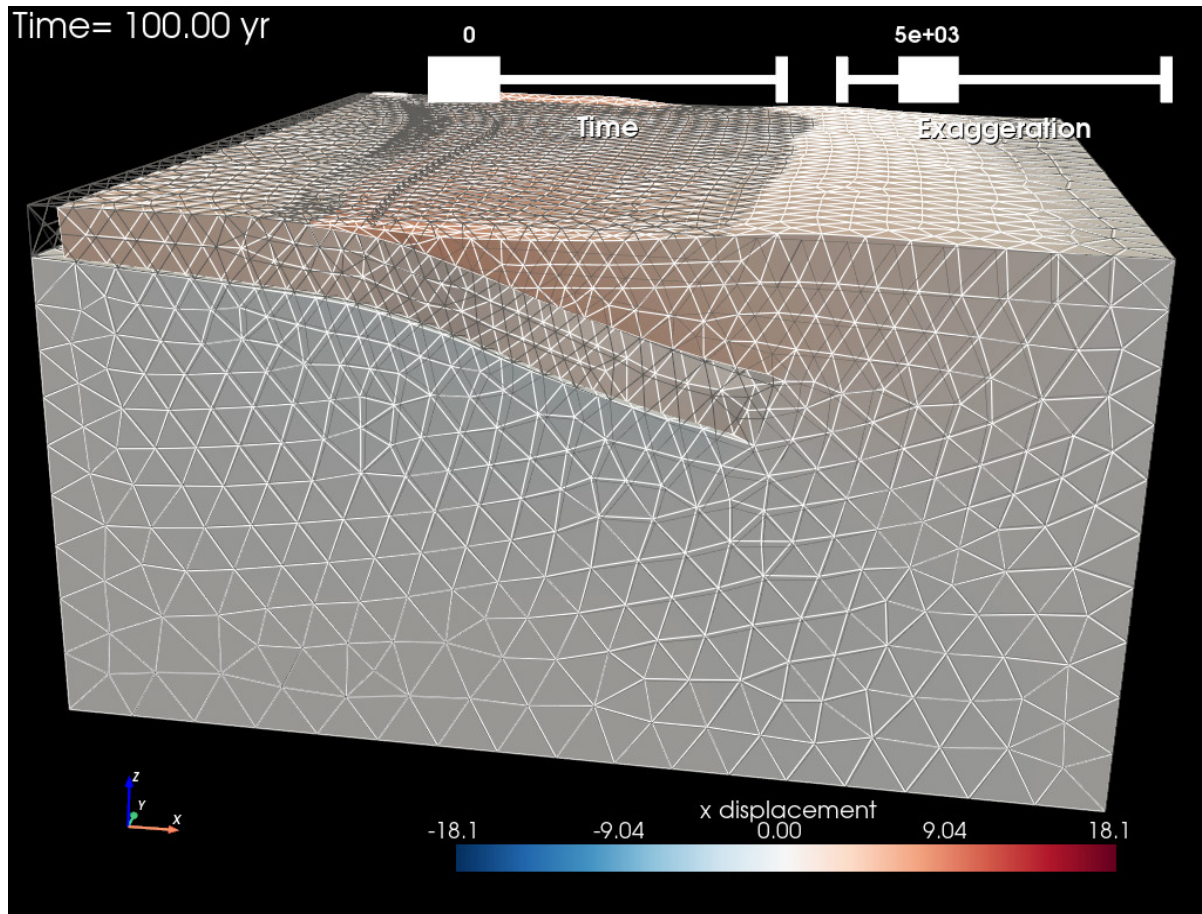


Fig. 4.154: Solution for Step 4 at $t=100$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 5000.

Listing 4.234: Visualize PyLith output using pylith_viz.

```
pylith_viz --filename=output/step04_eqcycle-domain.h5 warp_grid --component=x --
↳exaggeration=5000
```

Step 5: Spontaneous Rupture Driven by Subducting Slab

Warning

This example has not been updated for use with PyLith v3 and later.

Step 6: Prescribed Slow Slip Events

This example simulates several slow slip events on the slab interface. We prescribe a time history of slip, varying the slip amplitude as a function of time. We also output the displacement field at fake GNSS stations using `OutputSolnPoints`. For this simulation, we use linearly elastic materials. Fig. 4.155 shows the boundary conditions on the domain.

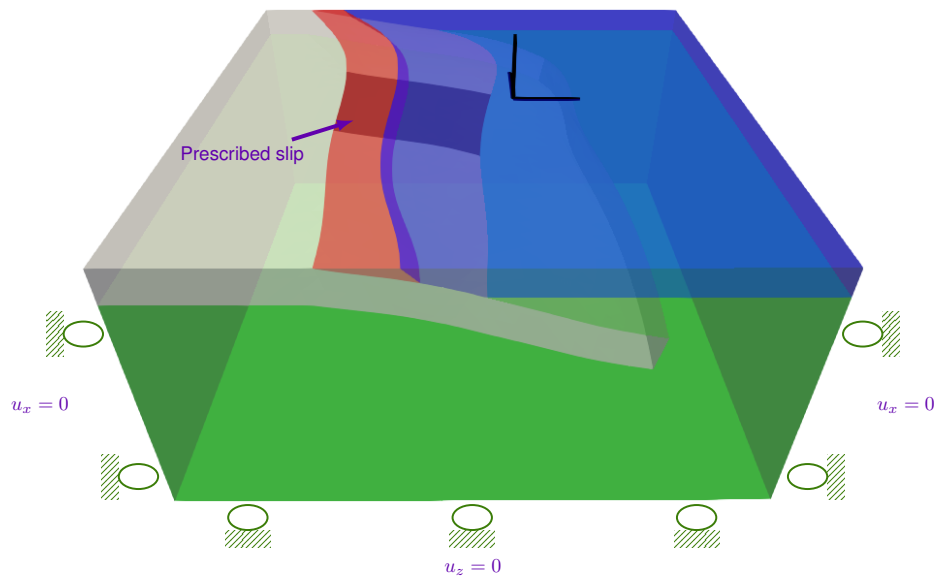


Fig. 4.155: Boundary conditions for prescribed slow slip. We prescribe time-varying slip on a patch within the subduction interface with roller boundary conditions on the lateral sides and bottom of the domain.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOCubit`
- `pylith.problems.TimeDependent`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.ZeroDB`

- `spatialdata.geocoords.CSGeo`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.SimpleDB`
- Quasi-static simulation
- Temporal database
- `pylith.meshio.OutputSolnPoints`
- `pylith.faults.KinSrcTimeHistory`
- `spatialdata.geocoords.CSGeo`
- `spatialdata.spatialdb.UniformDB`
- `spatialdata.spatialdb.SimpleGridDB`
- `spatialdata.spatialdb.SimpleGridAscii`

Simulation parameters

The parameters specific to this example are in `step06_slowslip.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output.
- `pylithapp.problem` Parameters for the time step information as well as solution field with displacement and Lagrange multiplier subfields.
- `pylithapp.interfaces` Parameters for the time-varying slip on the top of the slab.

For slow slip we use the `KinSrcTimeHistory` kinematic source to prescribe time-varying slip. We also use `OutputSolnPoints` to simulate output at continuous GNSS stations.

Prior to running the simulation, we use the `generate_slowslip.py` Python script to generate spatial database input files for the prescribed slip.

Listing 4.235: Generate database files needed for Step 6.

```
# Generate fault_slabtop_slowslip.spatialdb and fault_slabtop_slowslip.timedb.  
$ ./generate_slowslip.py
```

Note

The GNSS station file for use with the mesh from Gmsh (`cgns_stations.txt`) requires elevation values consistent with the topography in the model. The `generate_cgns.py` Python script uses the ground surface output from Step 1 to set the elevation of stations just below the ground surface consistent with the topography.

Listing 4.236: Run Step 6 simulation using the Gmsh mesh.

```
$ pylith step06_slowslip.cfg mat_elastic.cfg  
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:77:main
```

(continues on next page)

(continued from previous page)

```

-- pylithapp(info)
-- Running on 1 process(es).
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/meshio/MeshIOObj.py:41:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:74:void
↳pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- meshiopetsc(info)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-4000000, 4000000)
    (-4000000, 4000000)
    (-4000000, 2017.5)

# -- many lines omitted --
>> /src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:262:static void pylith::utils::
↳PetscOptions::write(pythia::journal::info_t&, const char*, const
↳pylith::utils::PetscOptions&)
-- petscoptions(info)
-- Setting PETSc options:
dm_reorder_section = true
dm_reorder_section_type = cohesive
ksp_converged_reason = true
ksp_error_if_not_converged = true
ksp_gmres_restart = 100
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-14
mg_fine_ksp_max_it = 5
mg_fine_pc_type = vpbjacobi
pc_type = gamg
snes_atol = 5.0e-7
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-14
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_type = beuler
viewer_hdf5_collective = true

# -- many lines omitted --

15 TS dt 5.4757e-05 time 0.000766598
    0 SNES Function norm 1.140833293430e-01
      Linear solve converged due to CONVERGED_ATOL iterations 0
    1 SNES Function norm 1.386302589692e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
16 TS dt 5.4757e-05 time 0.000821355
>> /software/pylith-opt/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:232:finalize
-- timedependent(info)

```

(continues on next page)

(continued from previous page)

```
-- Finalizing problem.
```

The beginning of the output is nearly the same as in previous examples. The simulation advances 16 time steps.

Listing 4.237: Alternatively, run Step 6 simulation using the Cubit mesh.

```
$ pylith step06_slowslip.cfg step06_slowslip_subit.cfg mat_elastic.cfg
```

Visualizing the results

In Fig. 4.156 we use the `pylith_viz` utility to visualize the x displacement field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time.

Listing 4.238: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step06_slowslip-domain.h5 warp_grid --component=x --  
↪exaggeration=10000
```

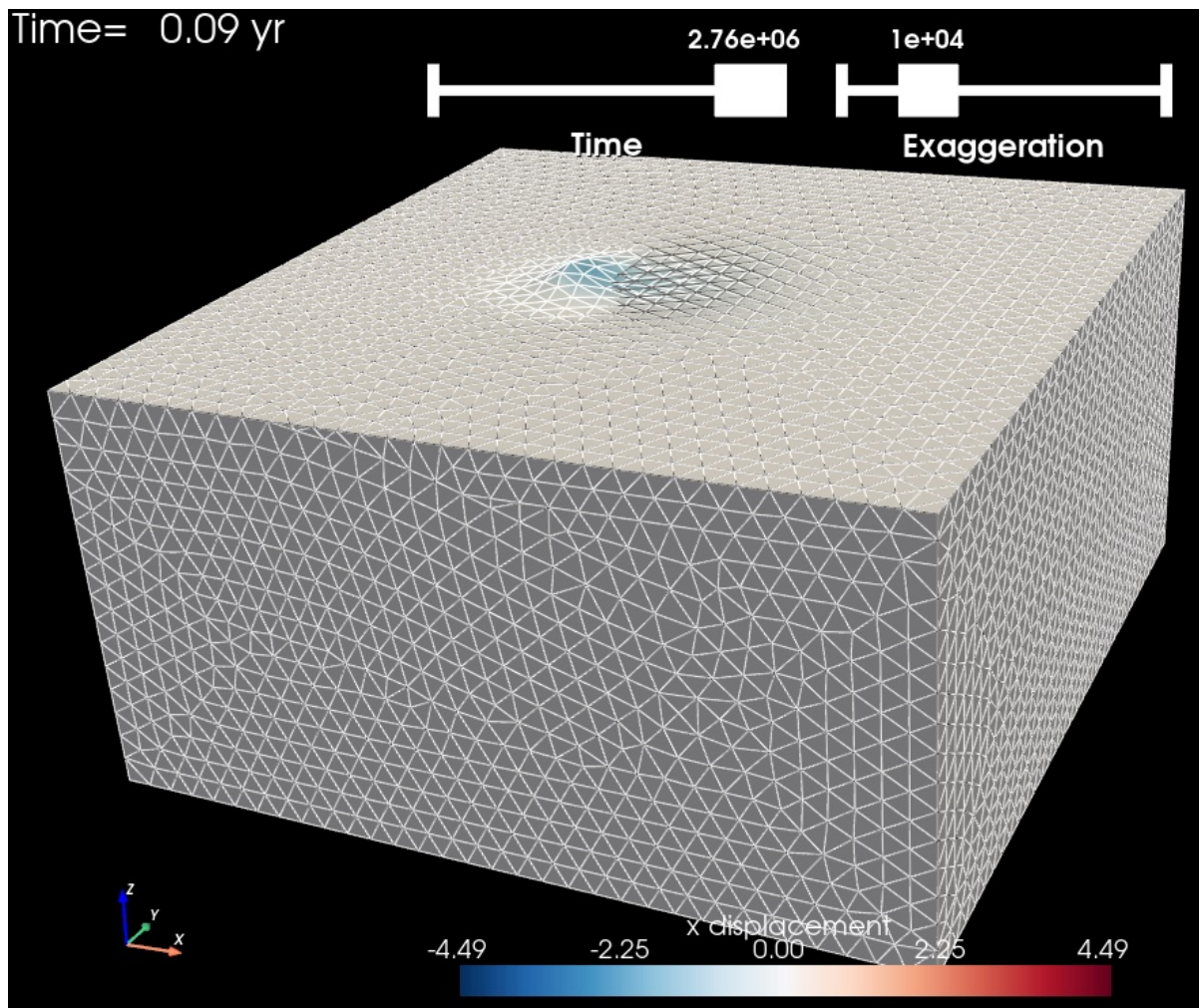


Fig. 4.156: Solution for Step 6 at $t=0.08$ yr. The colors of the shaded surface indicate the x displacement, and the deformation is exaggerated by a factor of 10,000.

Step 7: Generation of Green's Functions and Slow Slip Inversion

In this example we compute the static Green's function for prescribed slip impulses on the central portion of the subduction interface. We then use the responses to perform a simple geodetic inversion of the slip from Step 6. We divide generating Green's functions into two sub-problems:

- **Step 7a:** Left-lateral slip component
- **Step 7b:** Reverse slip component

Although PyLith can generate the two components in one simulation, we often prefer to speed up the process by running simulations for each of the components simultaneously using multiple processes on a cluster.

Fig. 4.157 shows the boundary conditions on the domain.

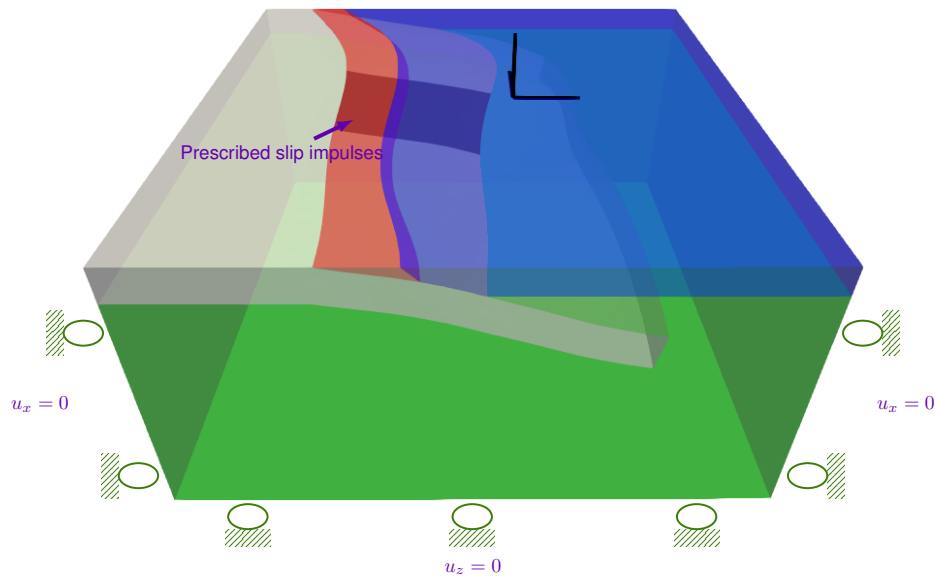


Fig. 4.157: Boundary conditions for Green's function generation. We apply left-lateral (Step 7a) and reverse (Step 7b) slip impulses on a patch within the subduction interface with roller boundary conditions on the lateral sides and bottom of the domain.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOcubit`
- `pylith.problems.TimeDependent`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.ZeroDB`
- `spatialdata.geocoords.CSGeo`
- `pylith.materials.Elasticity`
- `pylith.materials.IsotropicLinearElasticity`
- `spatialdata.spatialdb.SimpleDB`

- “Green’s functions”
- `pylith.problems.GreensFns`
- `pylith.meshio.OutputSolnPoints`
- `pylith.faults.FaultCohesiveImpulses`
- `spatialdata.spatialdb.UniformDB`
- `spatialdata.geocoords.CSGeo`
-

Simulation parameters

The parameters specific to this example are in `step07a_leftlateral.cfg` and `step07b_reverse.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output. We also change the problem type to `pylith.problems.GreensFns`.
- `pylithapp.problem` Parameters for the time step information as well as solution field with displacement and Lagrange multiplier subfields.
- `pylithapp.interfaces` Parameters for the slip impulses applied on the top of the slab.

For Green’s functions we use the `FaultCohesiveImpulses` kinematic source to prescribe slip impulses for left-lateral (Step 7a – `impulse_dof = [1]`) and reverse (Step 7b – `impulse_dof = [2]`) slip directions. As for Step 6, we use `OutputSolnPoints` to output the solution at fake CGNSS stations.

Listing 4.239: Run Step 7a and 7b simulations using the Gmsh mesh.

```
$ pylith step07a_leftlateral.cfg mat_elastic.cfg
$ pylith step07b_reverse.cfg mat_elastic.cfg

# The output should look something like the following.
>> /home/baagaard/software/micromamba/envs/pylith-opt/lib/python3.12/site-packages/
↳ pylith/apps/PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /home/baagaard/software/micromamba/envs/pylith-opt/lib/python3.12/site-packages/
↳ pylith/meshio/MeshIOobj.py:41:read
-- meshiopetsc(info)
-- Reading finite-element mesh
>> /home/baagaard/src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:74:void_
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
-- meshiopetsc(info)
-- Component 'meshiopetsc.reader': Domain bounding box:
    (-4000000, 4000000)
    (-4000000, 4000000)
    (-4000000, 2017.5)

# -- many lines omitted --
>> /home/baagaard/software/micromamba/envs/pylith-opt/lib/python3.12/site-packages/
↳ pylith/problems/GreensFns.py:105:run
```

(continues on next page)

(continued from previous page)

```

-- greensfns(info)
-- Solving problem.
>> /home/baagaard/src/cig/pylith/libsrc/pylith/problems/GreensFns.cc:321:void_
↳ pylith::problems::GreensFns::solve()
-- greensfns(info)
-- Component 'greensfns.problem': Computing Green's function 1 of 96.
  0 SNES Function norm 5.684788384324e-02
    Linear solve converged due to CONVERGED_ATOL iterations 11
  1 SNES Function norm 1.157087800256e-08
    Nonlinear solve converged due to CONVERGED_ITS iterations 1

# -- many lines omitted --
>> /home/baagaard/src/cig/pylith/libsrc/pylith/problems/GreensFns.cc:321:void_
↳ pylith::problems::GreensFns::solve()
-- greensfns(info)
-- Component 'greensfns.problem': Computing Green's function 96 of 96.
  0 SNES Function norm 6.580456779117e-02
    Linear solve converged due to CONVERGED_ATOL iterations 10
  1 SNES Function norm 6.832632664165e-09
    Nonlinear solve converged due to CONVERGED_ITS iterations 1
>> /home/baagaard/software/micromamba/envs/pylith-opt/lib/python3.12/site-packages/
↳ pylith/problems/Problem.py:232:finalize
-- greensfns(info)
-- Finalizing problem.
WARNING! There are options you set that were not used!
WARNING! could be spelling mistake, etc!
There are 4 unused database options. They are:
Option left: name:-ts_error_if_step_fails (no value) source: code
Option left: name:-ts_exact_final_time value: matchstep source: code
Option left: name:-ts_monitor (no value) source: code
Option left: name:-ts_type value: beuler source: code

```

The beginning of the output is nearly the same as in several previous examples. In this case, however, rather than stepping through time, we are stepping through impulse numbers. In each simulation, there are 96 impulses applied on the fault. Because we are not time stepping, we do not use all of the default PETSc options.

Listing 4.240: Alternatively, run Step 7a and 7b simulations using the Cubit mesh.

```

$ pylith step07a_leftlateral.cfg step07a_leftlateral_cubit.cfg mat_elastic.cfg
$ pylith step07b_reverse.cfg step07b_reverse_subit.cfg mat_elastic.cfg

```

Visualizing the results

In Fig. 4.158 we use the `pylith_viz` utility to visualize the slip for the impulses on the fault surface. You can move the slider or use the `p` and `n` keys to increment or decrement the slip impulse (shown as time).

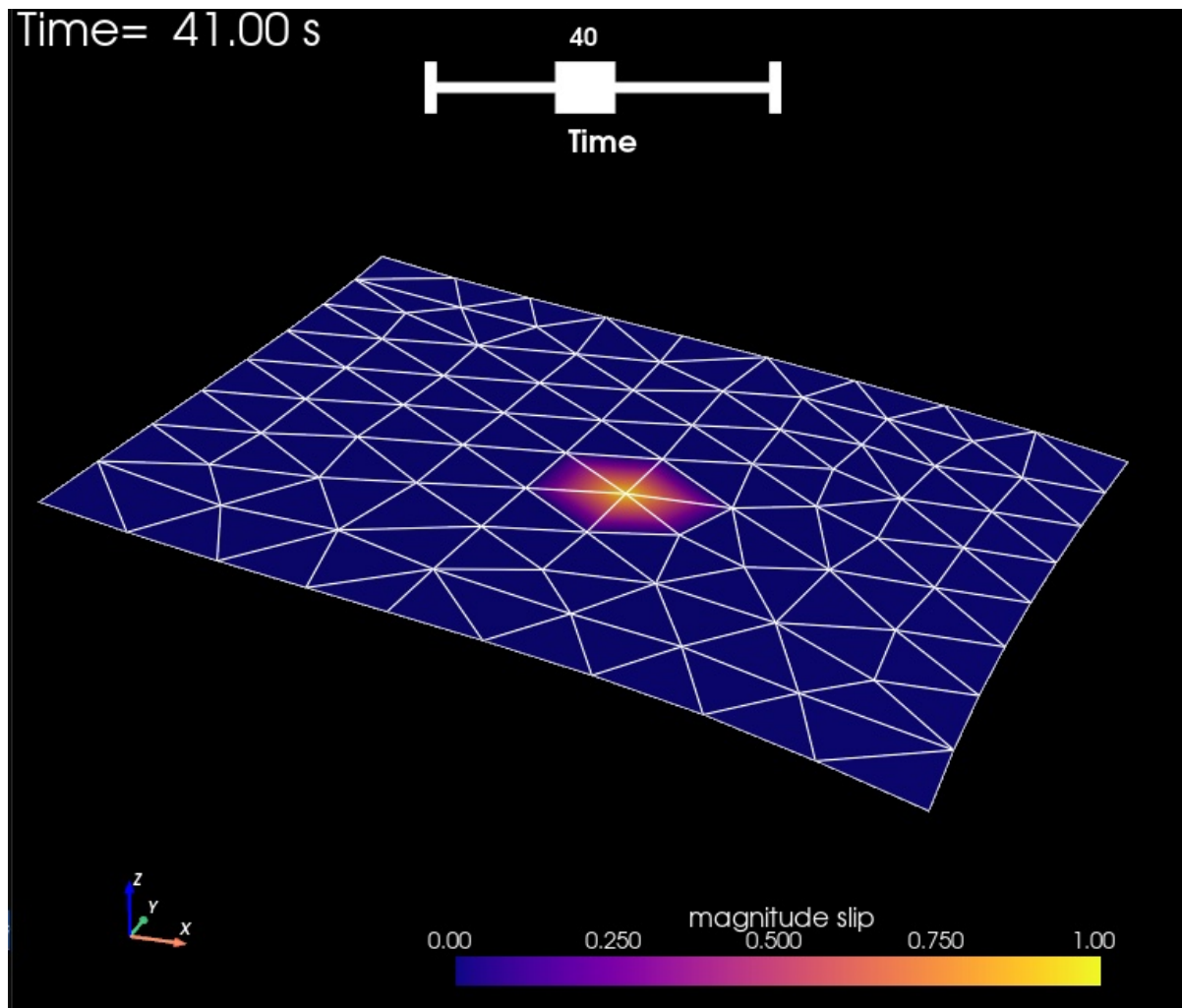


Fig. 4.158: Distribution of left-lateral fault slip for slip impulse 41 (time corresponds to the zero-based index) for Step 7a. The colors of the shaded surface indicate the magnitude of the slip.

Listing 4.241: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filename=output/step07a_leftlateral-fault_slabtop.h5 plot_field --field=slip
```

Performing a simple inversion

To perform our simulated inversion, we first postprocess the displacement results from Step 6. We do this using the `generate_synthetic_gnssdisp.py` Python script, which in turn reads the parameters from the `generate_synthetic_gnssdisp.cfg` file:

Listing 4.242: Generate synthetic data for inversion

```
$ ./generate_synthetic_gnssdisp.py
```

This script will produce the files `cgns_synthetic_displacement.txt`, which is used in the inversion. Once these have been created (and both sets of Green's functions have been generated), we can perform a simple inversion using the `invert_slip.py` script, which in turn reads the inversion parameters from `invert_slip.cfg`.

Listing 4.243: Perform inversion of synthetic data

```
$ ./invert_slip.py
```

The Python script performs a simple weighted generalized least-squares inversion, with penalty values applied to the deviation of the solution from the a priori value (zero, in this case). The slip inversion will produce several files in the output directory:

- `step07-inversion-summary.txt` A summary of the model residuals for each penalty parameter.
- `step07-inversion-displacement.h5` The predicted site displacements for different penalty parameters.
- `step07-inversion-slip.h5` The predicted fault slip for different penalty parameters.

Note that the inversion produces a `.xmf` file for each of the HDF5 files. We provide a Matplotlib Python script to visualize the data misfit for various penalty parameters:

Listing 4.244: Visualize data misfit

```
$ viz/plot_inversion_misfit.py
```

Fig. 4.159 shows the data residual versus the penalty residual, showing very clearly the 'corner' of the L-curve.

Step 8: Use of Gravitational Body Forces

This example demonstrates the use of gravitational body forces as well as the use of initial stresses to balance the body forces. This involves enabling gravity within our domain with Dirichlet roller boundary conditions on the lateral and bottom boundaries; we do not include faults in this example. We also demonstrate what happens when the initial stresses are not in balance with the gravitational stresses, and show how viscoelastic problems with gravitational stresses will in general not reach a steady-state solution. This example consists of three sub-problems:

- **Step 8a:** Gravitational body forces with 3-D density variations in elastic materials and initial stresses for a uniform density.
- **Step 8b:** Gravitational body forces with 3-D density variations in incompressible elastic materials.
- **Step 8c:** Gravitational body forces with 3-D density variations in elastic and viscoelastic materials and initial stresses from Step 8a plus finite strain formulation (does not reach a steady-state solution). This example is not yet implemented for PyLith v3+.

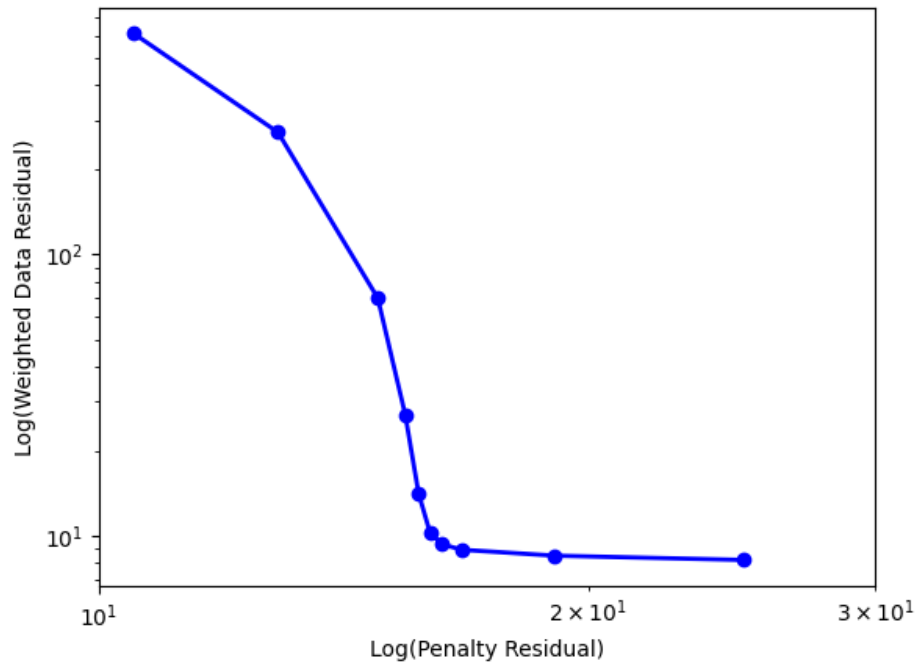


Fig. 4.159: Data residual versus penalty residual for inversion from Step 7.

Danger

Step 8b takes an extremely long time to run and does not presently run in parallel. We are working to implement better preconditioners to improve the solvers and reduce the runtime.

All these simulations use the default ZeroDB displacement boundary conditions on the lateral and bottom boundaries, and they all use `spatialdata.spatialdb.GravityField` to apply gravitational body forces. Step 8a uses reference stresses (provided in the associated spatial databases for each material) to balance the deformation induced by turning on body forces. The reference stresses assume a constant density, while there is actually a density contrast between materials, meaning that the stresses don't balance and there is some resulting deformation. Step 8b uses incompressible elasticity (using `pylith.problems.SolnDispPres`) to prevent the vertical deformation that would otherwise occur. In this case it is necessary to provide an additional Dirichlet zero pressure boundary condition on the ground surface.

Fig. 4.160 shows the boundary conditions on the domain.

Features

- Tetrahedral cells
- `pylith.meshio.MeshIOcubit`
- `pylith.problems.TimeDependent`
- `pylith.meshio.OutputSolnBoundary`
- `pylith.meshio.DataWriterHDF5`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.ZeroDB`

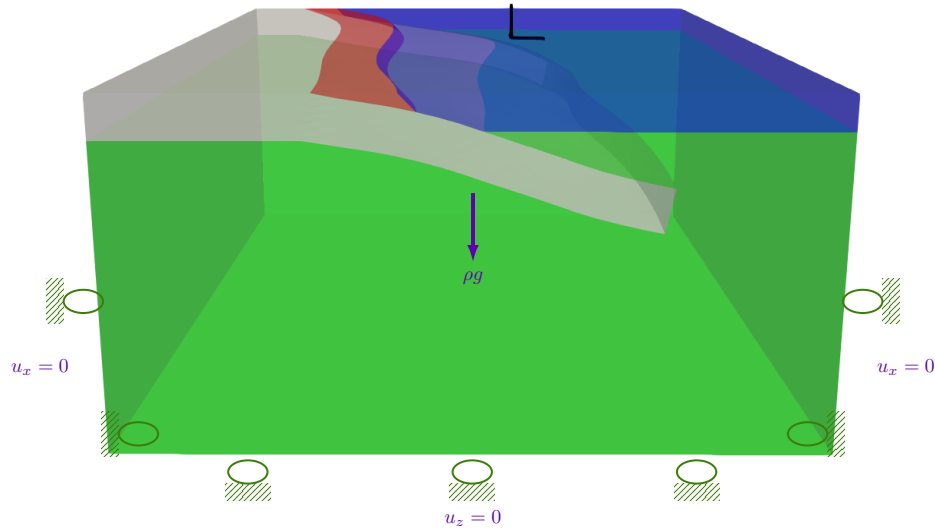


Fig. 4.160: Boundary conditions for gravitational body forces examples. Body forces are applied along with roller boundary conditions on the lateral sides and bottom of the domain. For Step 8b, we also apply zero pressure BC along the upper surface of the mesh.

- `spatialdata.geocoords.CSGeo`
- Static simulation
- `spatialdata.spatialdb.GravityField`
- Reference state

Simulation parameters

The parameters specific to this example are in `step08a_gravity_refstate.cfg` and `step08b_gravity_incompressible.cfg` and include:

- `pylithapp.metadata` Metadata for this simulation. Even when the author and version are the same for all simulations in a directory, we prefer to keep that metadata in each simulation file as a reminder to keep it up-to-date for each simulation.
- `pylithapp` Parameters defining where to write the output.
- `pylithapp.problem` Parameters to set quadrature and basis order, as well as provide a `gravity_field`. For Step 8b, we also set the solution to `pylith.problems.SolnDispPres`.
- `pylithapp.problem.bc` For Step 8b, we add a zero pressure BC on the ground surface.
- `pylithapp.problem.materials` For Step 8a, we set `use_reference_state` and provide spatial databases that include reference stresses.

For all of the problems involving gravitational body forces we use `spatialdata.spatialdb.GravityField` to define the `gravity_field`. For Step 8a, we use higher order `quadrature_order` and `basis_order` to accurately capture the displacement field. This is not needed for Step 8b, which has much smaller displacements. Note that Step 8b also requires `mat_elastic_incompressible.cfg`, which provides all of the necessary properties using a `UniformDB`.

Listing 4.245: Run Step 8a and 8b simulations using the Gmsh mesh.

```

$ pylith step08a_gravity_refstate.cfg
$ pylith step08b_gravity_incompressible.cfg mat_elastic_incompressible.cfg

# The output should look something like the following. This is for Step 8b.
>> /work/charlesw/virtualenv/python3.12/lib/python3.12/site-packages/pylith/apps/
↳ PyLithApp.py:77:main
-- pylithapp(info)
-- Running on 1 process(es).
>> /work/charlesw/virtualenv/python3.12/lib/python3.12/site-packages/pylith/meshio/
↳ MeshIOObj.py:38:read
-- meshiocubit(info)
-- Reading finite-element mesh
>> /home/charlesw/cig/pylith3/source/pylith-fork/libsrc/pylith/meshio/MeshIOCubit.
↳ cc:148:void pylith::meshio::MeshIOCubit::_readVertices(pylith::meshio::ExodusII&,
↳ pylith::scalar_array*, int*, int*) const
-- meshiocubit(info)
-- Component 'reader': Reading 24824 vertices.
>> /home/charlesw/cig/pylith3/source/pylith-fork/libsrc/pylith/meshio/MeshIOCubit.
↳ cc:208:void pylith::meshio::MeshIOCubit::_readCells(pylith::meshio::ExodusII&,
↳ pylith::int_array*, pylith::int_array*, int*, int*) const
-- meshiocubit(info)
-- Component 'reader': Reading 134381 cells in 4 blocks.

# -- many lines omitted --
>> /home/charlesw/cig/pylith3/source/pylith-fork/libsrc/pylith/Utils/PetscOptions.
↳ cc:239:static void pylith::utils::PetscOptions::write(pythia::journal::info_t&, const
↳ char*, const pylith::utils::PetscOptions&)
-- petscoptions(info)
-- Setting PETSc options:
fieldsplit_displacement_pc_type = lu
fieldsplit_pressure_pc_type = lu
ksp_atol = 1.0e-12
ksp_converged_reason = true
ksp_error_if_not_converged = true
ksp_guess_pod_size = 8
ksp_guess_type = pod
ksp_rtol = 1.0e-12
pc_fieldsplit_schur_factorization_type = full
pc_fieldsplit_schur_precondition = full
pc_fieldsplit_type = schur
pc_type = fieldsplit
snes_atol = 1.0e-9
snes_converged_reason = true
snes_error_if_not_converged = true
snes_monitor = true
snes_rtol = 1.0e-12
ts_error_if_step_fails = true
ts_monitor = true
ts_type = beuler
>> /work/charlesw/virtualenv/python3.12/lib/python3.12/site-packages/pylith/problems/

```

(continues on next page)

(continued from previous page)

```

↪TimeDependent.py:132:run
-- timedependent(info)
-- Solving problem.
0 TS dt 3.15576e+07 time 0.
  0 SNES Function norm 1.847563517818e+03
    Linear solve converged due to CONVERGED_ATOL iterations 2
  1 SNES Function norm 1.534546536971e-09
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 1
1 TS dt 3.15576e+07 time 3.15576e+07
>> /work/charlesw/virtualenv/python3.12/lib/python3.12/site-packages/pylith/problems/
↪Problem.py:199:finalize
-- timedependent(info)
-- Finalizing problem.

```

The beginning of the output is nearly the same as in several previous examples. Both of these problems are static, however, so there is only a single time step.

Listing 4.246: Alternatively, run Step 8a and 8b simulations using the Cubit mesh.

```

$ pylith step08a_gravity_refstate.cfg step08a_gravity_refstate_cubit.cfg
$ pylith step08b_gravity_incompressible.cfg step08b_gravity_incompressible_cubit.cfg mat_
↪elastic_incompressible.cfg

```

Visualizing the results

In Fig. 4.161 we use the `pylith_viz` utility to visualize the *z* displacement field.

Listing 4.247: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filename=output/step08a_gravity_refstate-domain.h5 warp_grid --component=z -
↪-exaggeration=50

```

Note that the vertical displacements for Step 8b are very close to zero, because the material is nearly incompressible.

Suggested Exercises

1. Change the elastic bulk rheology properties to be depth dependent.
2. Add spatial variation to the slip distribution in Step 2.
3. Change the distribution of aseismic slip (creep) in Steps 3 and 4.
4. Change the distribution of slow slip in Step 6 and the GNSS stations used in Steps 6 and 7.
5. Optimize the depth variation of the initial stresses in Step 8a to minimize the deformation.

2D Magma Reservoir Using Poroelasticity

The files are in the directory `examples/magma-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

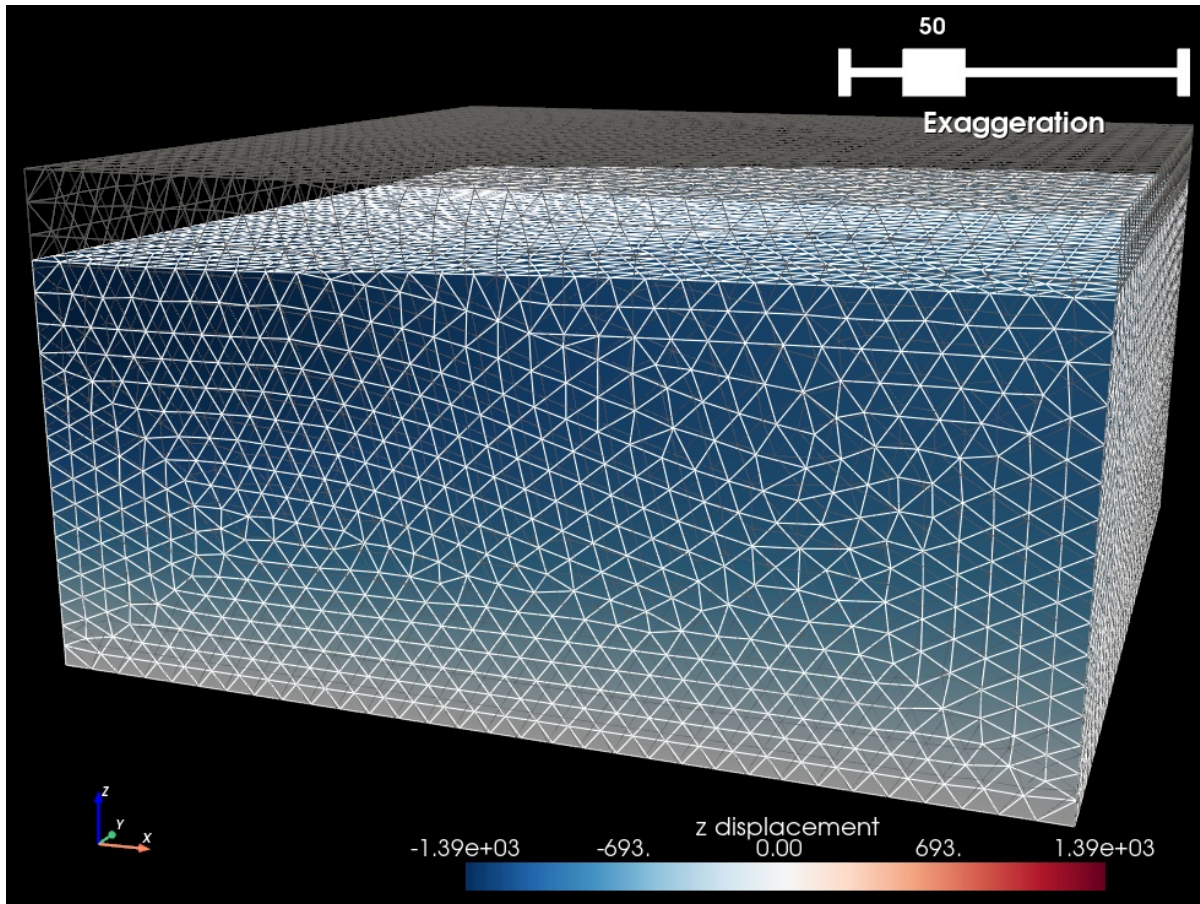


Fig. 4.161: Solution for Step 8a. The colors of the shaded surface indicate the z displacement, and the deformation is exaggerated by a factor of 50.

generate_gmsh.py

Python script to generate mesh using Gmsh.

generate_cubit.py

Python script to generate mesh using Cubit.

***.msh**

Gmsh mesh files generated by Gmsh.

***.exo**

Exodus II finite-element mesh files generated by Cubit.

***.spatialdb**

Spatial database files associated with the spatial databases.

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This example demonstrates the use of poroelasticity to model magma flow in a conduit and reservoir in the crust.

Step 1

Magma influx with displacement and pressure boundary conditions.

Step 2

Same as Step 1 with evolution of porosity state variable.

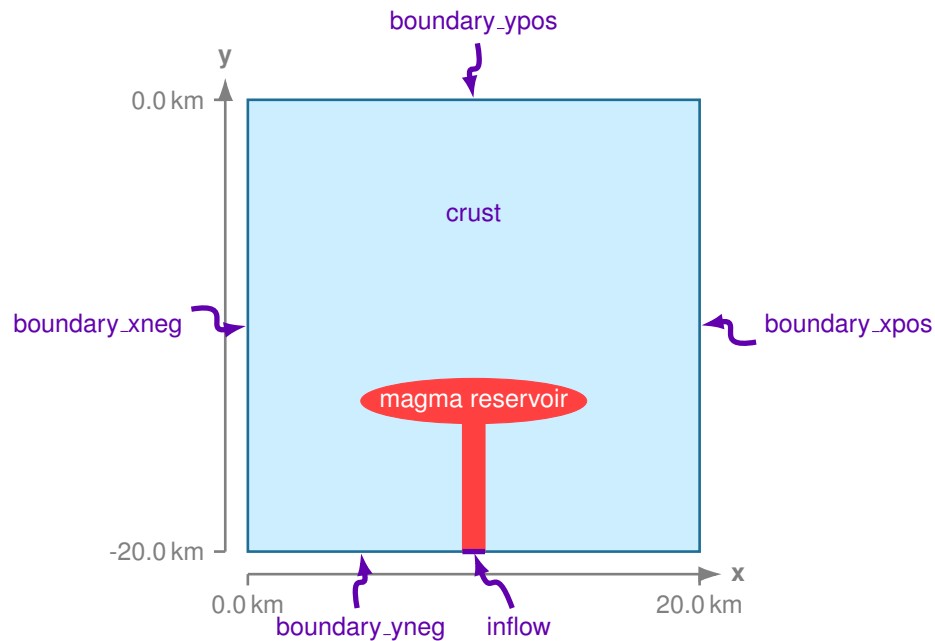


Fig. 4.162: Diagram of geometry for magma reservoir. The domain extends from 0 km to +20 km in the x direction and from -20 km to 0 km in the y direction. We refer to the domain boundaries using the names shown in the diagram.

Example Workflow

Gmsh Mesh

Geometry

We construct the geometry for the vertical cross section using 2D primitives available in the Gmsh OpenCASCADE geometry engine. Fig. 4.163 shows the geometry and variables names of the curves and surfaces.

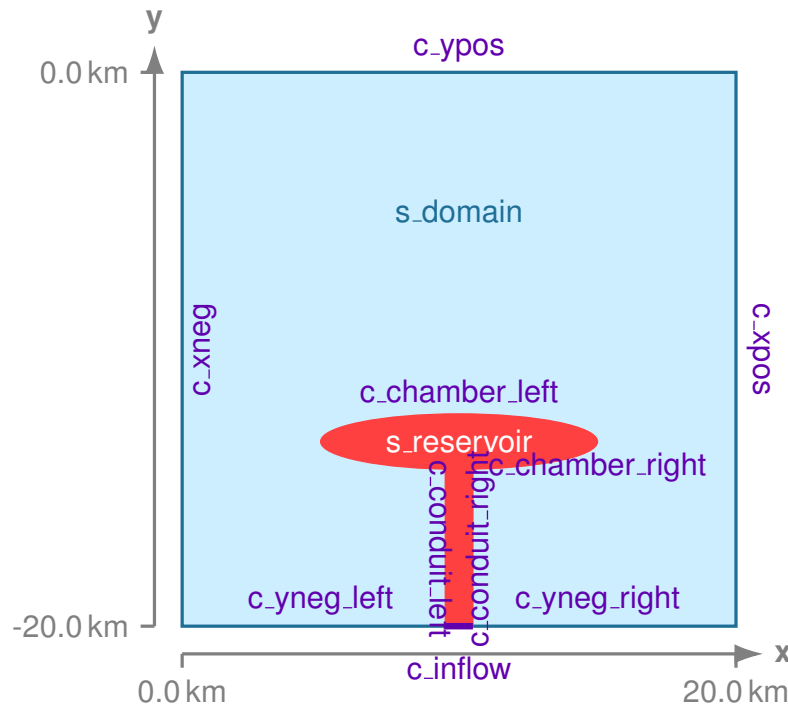


Fig. 4.163: Geometry created in Gmsh for generating the finite-element mesh. We create a rectangle for the domain, an ellipse for the magma chamber, and a second rectangle for the conduit. We join the ellipse and second rectangle to form the shape of the magma chamber, and then embed it in the domain.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. We create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class. The main difference is that the geometry is slightly more complex, and we calculate the location of the points on the fault using trigonometry.

We use the Gmsh `MeshSize` options to define a discretization size that grows slowly at a geometric rate with distance from the main fault.

Listing 4.248: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with triangular cells and save it to `mesh_tri.msh` (default filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui
```

(continues on next page)

(continued from previous page)

```
# Create only the geometry and start the Gmsh graphical interface.  
$ ./generate_gmsh.py --geometry --gui  
  
# Show available command line arguments.  
$ ./generate_gmsh.py --help
```

By default the Python script will generate a finite-element mesh with triangular cells and save it to the file `mesh_tri.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.249: View the Gmsh mesh file `mesh_tri.msh` using Gmsh.

```
gmsh -open mesh_tri.msh
```

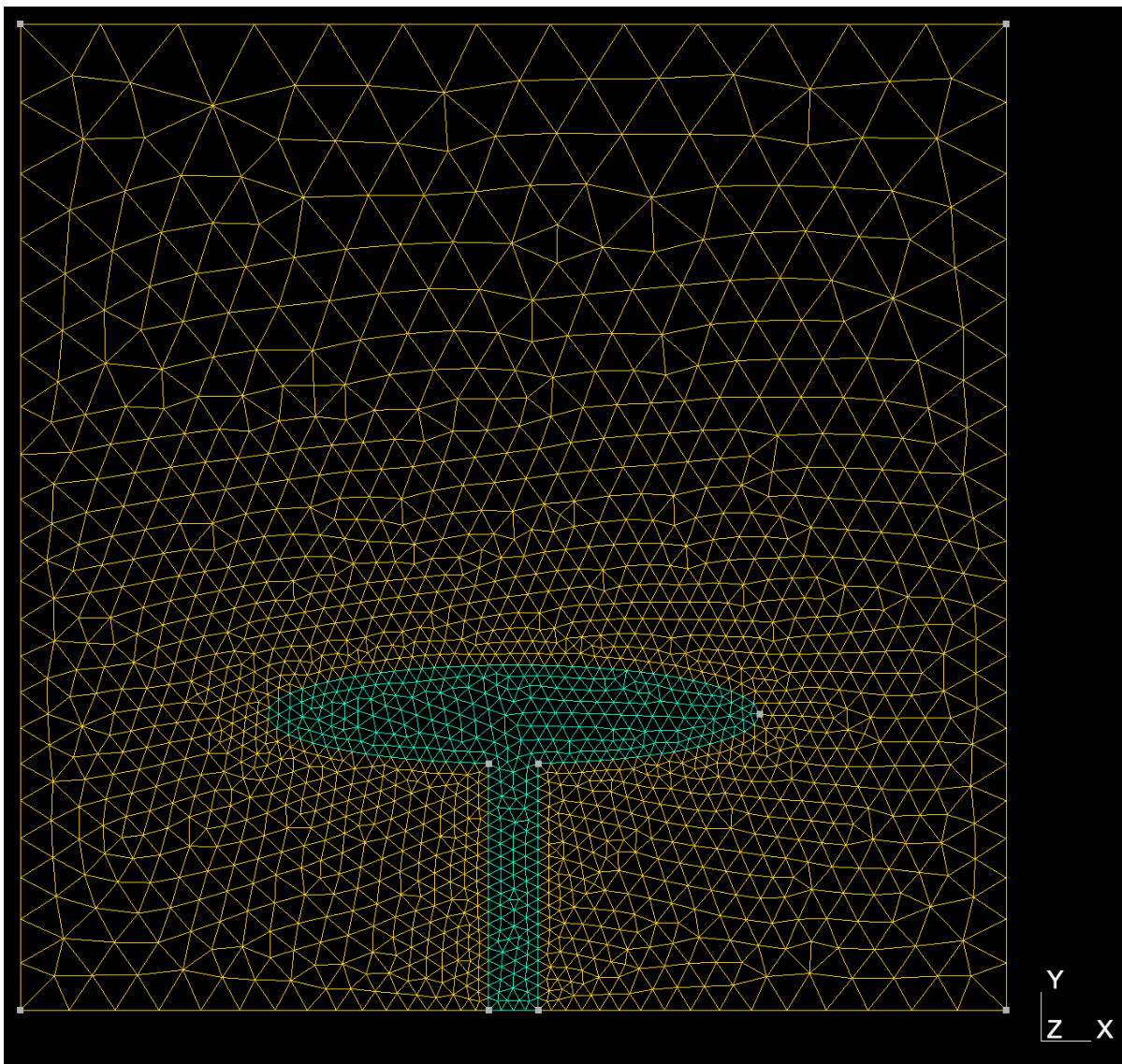


Fig. 4.164: Finite-element mesh with triangular cells generated by Gmsh.

Cubit Mesh

Geometry

We construct the geometry for the vertical cross section using 2D primitives available in Cubit.

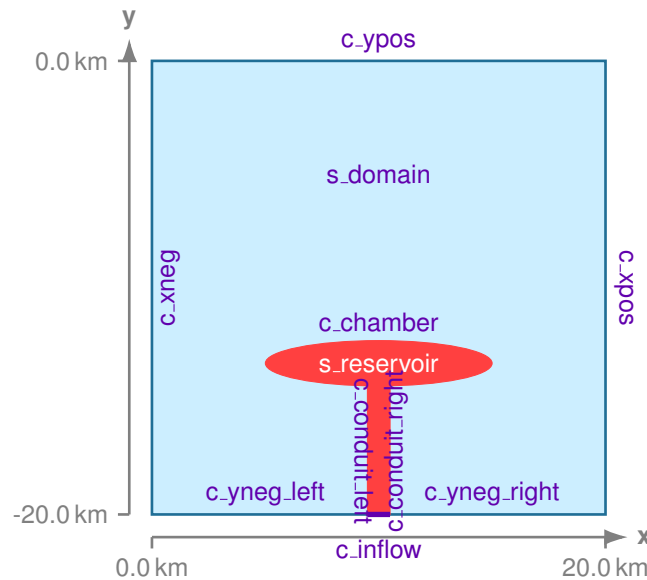


Fig. 4.165: Geometry created in Cubit for generating the finite-element mesh. The names of the vertices and curves match the ones we use in the Cubit Python script.

Meshing using Python script

Note

New in v4.1.3.

We use a Python script to generate the mesh rather than Journal files. We also use the skeleton sizing function for the surface.

We use the Python script `generate_cubit.py` to generate the mesh. The Python script is setup so that it can be run from within Cubit or as a standalone Python script without the Cubit GUI interface. In this example, we will run the script from within Cubit using the Journal editor.

Open the Python script `generate_cubit.py` in the Cubit journal editor. Play the selected script or play the lines, making sure you play the first line so that Cubit uses the Python interpreter when running the script. We specify the parameters controlling the geometry, mesh size, and cell shape near the top of the script.

Important

We reference objects by their centroid so the resulting script should be independent of the version of Cubit that is being used.

Note

We use the skeleton sizing function to gradually increase the cell size with distance from the magma reservoir. This is much simpler than applying the bias sizing function to the curves and surfaces. We set the minimum size equal to the cell size on the reservoir boundary and the maximum gradient equal to the “bias”.

After running the Python script in Cubit, you will have a corresponding Exodus-II file (`mesh_tri.exo` or `mesh_quad.exo`).

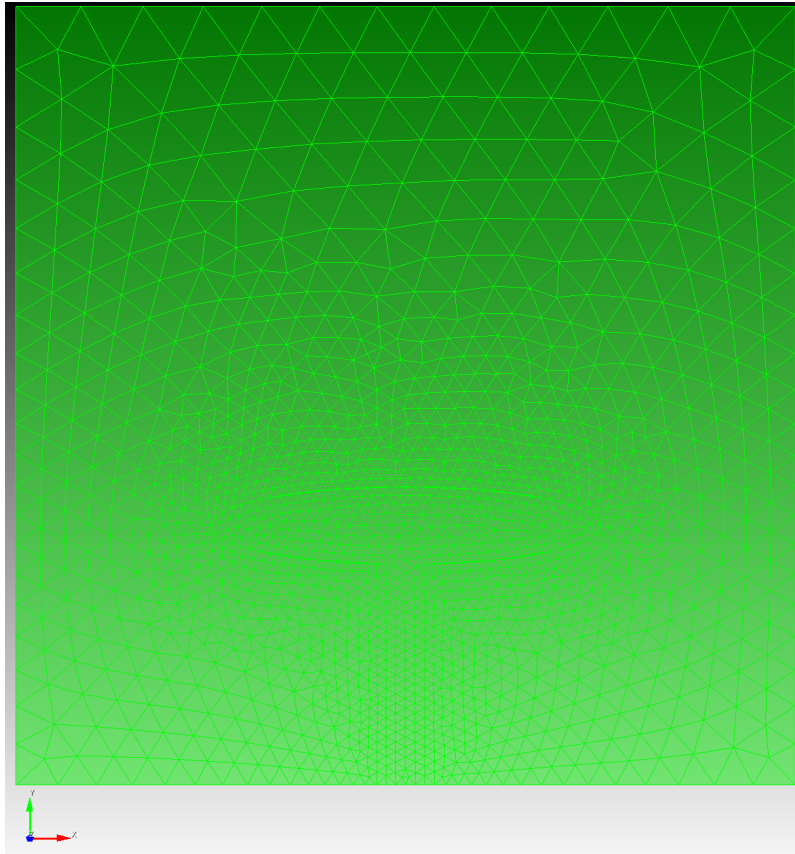


Fig. 4.166: Finite-element mesh with triangular cells generated by Cubit.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`. The `pylithapp.cfg` file contains numerous comments, so we only summarize the parameters here.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh from Gmsh in `pylithapp.mesh_generator`.

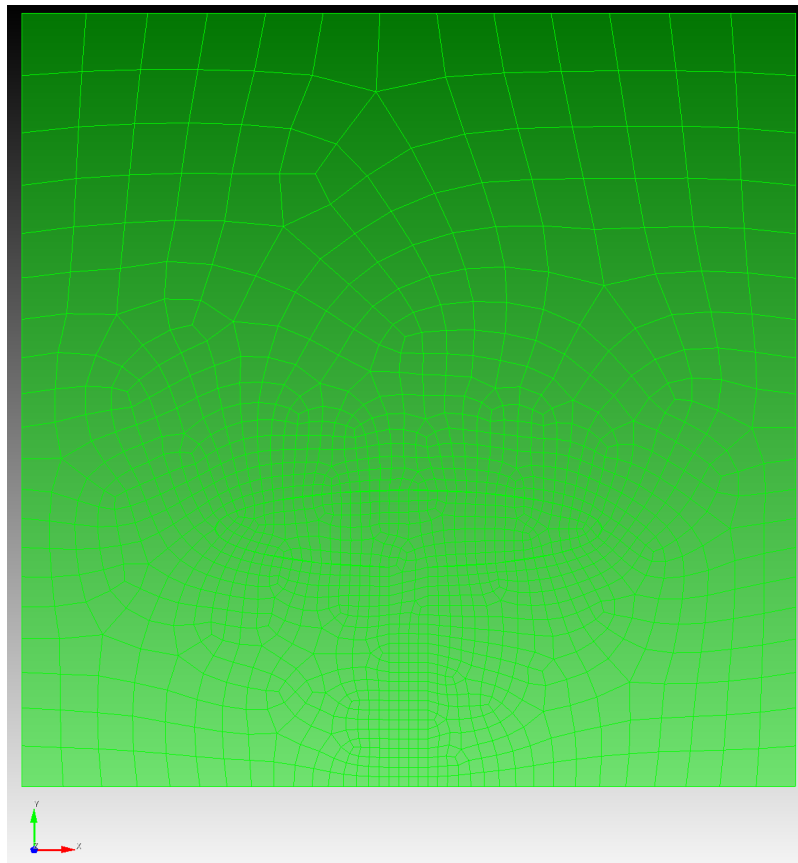


Fig. 4.167: Finite-element mesh with quadrilateral cells generated by Cubit.

Physics

These quasi-static simulations solve the poroelasticity equation, so we have a solution field with displacement, pressure, and volumetric strain subfields.

$$\vec{s} = (\vec{u} \quad p \quad \epsilon_v)^T, \quad (4.491)$$

$$\nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) = \vec{0}, \quad (4.492)$$

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} + \nabla \cdot \vec{q}(p) = 0, \quad (4.493)$$

$$\nabla \cdot \vec{u} - \epsilon_v = 0. \quad (4.494)$$

We specify a basis order of 2 for the displacement subfield, and 1 for each of the remaining subfields. We also adjust the scales for nondimensionalization.

Listing 4.250: Solution and nondimensionalization parameters for magma-2d examples.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresTracStrain
defaults.quadrature_order = 2

[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
pressure.basis_order = 1
trace_strain.basis_order = 1

[pylithapp.problem]
scales = pylith.scales.NondimElasticQuasistatic
scales.length_scale = 100.0*m
scales.relaxation_time = 0.2*year
scales.shear_modulus = 10.0*GPa
```

We use the material properties in all of the simulations in this directory, so we specify them in `pylithapp.cfg` to avoid repeating the information in the file with parameters for each simulation. We create an array of 2 materials; with uniform material properties we use `UniformDB` spatial databases and set the basis order of the auxiliary subfields to 0.

Listing 4.251: Material parameters for the magma-2d examples.

```
[pylithapp.problem]
# We have two different poroelastic materials each with a linear bulk rheology.
materials = [crust, intrusion]
materials.crust = pylith.materials.Poroelasticity
materials.intrusion = pylith.materials.Poroelasticity

[pylithapp.problem.materials]
crust.bulk_rheology = pylith.materials.IsotropicLinearPoroelasticity
intrusion.bulk_rheology = pylith.materials.IsotropicLinearPoroelasticity

[pylithapp.problem.materials.crust]

# `label_value` must match the blocks in `bc.jou` Cubit Journal file.
label_value = 1

# We will use uniform material properties, so we use the UniformDB
```

(continues on next page)

(continued from previous page)

```

# spatial database.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Poroelastic properties for the crust
db_auxiliary_field.values = [solid_density, fluid_density, fluid_viscosity, porosity,
↪shear_modulus, drained_bulk_modulus, biot_coefficient, fluid_bulk_modulus, solid_bulk_
↪modulus, isotropic_permeability]
db_auxiliary_field.data = [ 2500*kg/m**3, 1000*kg/m**3, 0.001*Pa*s, 0.01, 6.0*GPa, 10.
↪0*GPa, 1.0, 2.0*GPa, 20.0*GPa, 1e-15*m**2]

# Set basis order to 0 for uniform properties and a basis order of 1 for Cauchy stress_
↪and strain.
auxiliary_subfields.body_force.basis_order = 0
auxiliary_subfields.solid_density.basis_order = 0
auxiliary_subfields.fluid_density.basis_order = 0
auxiliary_subfields.fluid_viscosity.basis_order = 0
auxiliary_subfields.gravitational_acceleration.basis_order = 0
auxiliary_subfields.porosity.basis_order = 0
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.crust.bulk_rheology]
auxiliary_subfields.drained_bulk_modulus.basis_order = 0
auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.biot_coefficient.basis_order = 0
auxiliary_subfields.biot_modulus.basis_order = 0
auxiliary_subfields.isotropic_permeability.basis_order = 0

[pylithapp.problem.materials.intrusion]
# `label_value` must match the blocks in `bc.jou` Cubit Journal file.
label_value = 2

# We will use uniform material properties, so we use the UniformDB
# spatial database.
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Poroelastic properties
db_auxiliary_field.values = [solid_density, fluid_density, fluid_viscosity, porosity,
↪shear_modulus, drained_bulk_modulus, biot_coefficient, fluid_bulk_modulus, solid_bulk_
↪modulus, isotropic_permeability]
db_auxiliary_field.data = [ 2500*kg/m**3, 1000*kg/m**3, 0.001*Pa*s, 0.1, 6.0*GPa, 10.
↪0*GPa, 0.8, 2.0*GPa, 20.0*GPa, 1e-13*m**2]

auxiliary_subfields.body_force.basis_order = 0
auxiliary_subfields.solid_density.basis_order = 0
auxiliary_subfields.fluid_density.basis_order = 0
auxiliary_subfields.fluid_viscosity.basis_order = 0
auxiliary_subfields.gravitational_acceleration.basis_order = 0
auxiliary_subfields.porosity.basis_order = 0
derived_subfields.cauchy_strain.basis_order = 1
derived_subfields.cauchy_stress.basis_order = 1

[pylithapp.problem.materials.intrusion.bulk_rheology]

```

(continues on next page)

(continued from previous page)

```
# Set basis order to 0 for uniform properties
auxiliary_subfields.drained_bulk_modulus.basis_order = 0
auxiliary_subfields.shear_modulus.basis_order = 0
auxiliary_subfields.biot_coefficient.basis_order = 0
auxiliary_subfields.biot_modulus.basis_order = 0
auxiliary_subfields.isotropic_permeability.basis_order = 0
```

Step 1: Magma inflation

Features

- Quasi-static problem
- LU preconditioner
- pylith.materials.Poroelasticity
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.problems.SolnDispPresTracStrain
- pylith.problems.InitialConditionDomain
- pylith.bc.DirichletTimeDependent
- pylith.bc.NeumannTimeDependent
- pylith.meshio.DataWriterHDF5
- spatialdata.spatialdb.SimpleGridDB
- spatialdata.spatialdb.UniformDB

Simulation parameters

This example uses poroelasticity to model flow of magma up through a conduit and into a magma reservoir. The magma reservoir and conduit have a higher permeability than the surrounding crust. We generate flow by imposing a pressure on the external boundary of the conduit that is higher than the uniform initial pressure in the domain. Fig. 4.168 shows the boundary conditions on the domain. The parameters specific to this example are in `step01_inflation.cfg`.

Listing 4.252: Time stepping parameters for Step 1.

```
[pylithapp.timedependent]
start_time = -0.2*year
initial_dt = 0.2*year
end_time = 10.0*year
```

Listing 4.253: Initial condition parameters for Step 1. We impose an initial fluid pressure over the entire domain that increases from 0 at the top boundary to 5 MPa at the bottom boundary.

```
[pylithapp.problem]
ic = [domain]
ic.domain = pylith.problems.InitialConditionDomain

[pylithapp.problem.ic.domain]
```

(continues on next page)

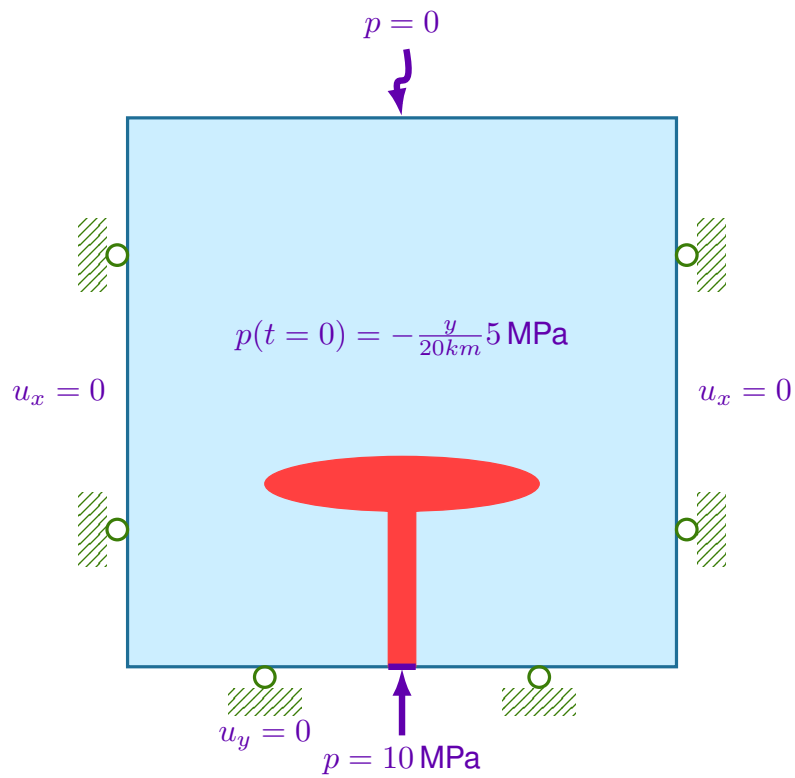


Fig. 4.168: Boundary and initial conditions for magma inflation. We apply roller boundary conditions on the +x, -x, and -y boundaries. We impose zero pressure (undrained conditions) on the +y boundary and a pressure on the external boundary of the conduit to generate fluid flow.

(continued from previous page)

```

subfields = [pressure]
db = spatialdata.spatialdb.SimpleDB
db.description = Initial conditions for domain
db.iohandler.filename = initial_pressure.spatialdb

```

We create an array of 5 Dirichlet boundary conditions: 3 for displacement and 2 for fluid pressure. We have zero displacement perpendicular to the -x, +x, and -y boundaries, zero pressure on the +y boundary, and 10 MPa of fluid pressure on the external boundary of the conduit.

Listing 4.254: Dirichlet boundary conditions for Step 1.

```

[pylithapp.problem]
bc = [bc_xneg, bc_xpos, bc_yneg, bc_ypos, bc_flow]

bc.bc_xneg = pylith.bc.DirichletTimeDependent
bc.bc_xpos = pylith.bc.DirichletTimeDependent
bc.bc_yneg = pylith.bc.DirichletTimeDependent
bc.bc_ypos = pylith.bc.DirichletTimeDependent
bc.bc_flow = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xneg]
constrained_dof = [0]
label = boundary_xneg
field = displacement
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC -x

[pylithapp.problem.bc.bc_xpos]
constrained_dof = [0]
label = boundary_xpos
field = displacement
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC +x

[pylithapp.problem.bc.bc_yneg]
constrained_dof = [1]
label = boundary_yneg
field = displacement
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC -y

[pylithapp.problem.bc.bc_ypos]
constrained_dof = [0]
label = boundary_ypos
field = pressure
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC +z

[pylithapp.problem.bc.bc_flow]
constrained_dof = [0]
label = boundary_flow
field = pressure

```

(continues on next page)

(continued from previous page)

```
db_auxiliary_field = spatialdata.spatialdb.UniformDB
db_auxiliary_field.description = Flow into external boundary of conduit
db_auxiliary_field.values = [initial_amplitude]
db_auxiliary_field.data = [10.0*MPa]
```

Running the simulation

Listing 4.255: Run Step 1 simulation

```
$ pylith step01_inflation.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:80:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳ PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    fieldsplit_displacement_pc_type = lu
    fieldsplit_pressure_pc_type = bjacobi
    fieldsplit_trace_strain_pc_type = bjacobi
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    pc_fieldsplit_0_fields = 2
    pc_fieldsplit_1_fields = 1
    pc_fieldsplit_2_fields = 0
    pc_fieldsplit_type = multiplicative
    pc_type = fieldsplit
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
    snes_monitor = true
    snes_rtol = 1.0e-14
    ts_error_if_step_fails = true
    ts_exact_final_time = matchstep
    ts_monitor = true
    ts_type = beuler
    viewer_hdf5_collective = true
>> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void
↳ pylith::meshio::MeshIOPetsc::_read()
-- info (application-flow)
-- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
>> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
```

(continues on next page)

(continued from previous page)

```

(0, 20000)
(-20000, 0)
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
↳pylith::problems::TimeDependent::verifyConfiguration() const
-- info (application-flow)
-- Component 'timedependent.problem': Verifying problem configuration.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
↳printInfo
-- info (application-flow)
-- Scales for nondimensionalization:
    Length scale: 5000*m
    Displacement scale: 10*m
    Time scale: 4.16667e+07*s
    Rigidity scale: 6e+09*m**-1*kg*s**-2
    Temperature scale: 1*K
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
↳pylith::problems::TimeDependent::initialize()
-- info (application-flow)
-- Component 'timedependent.problem': Initializing problem.
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 0.151476 time -0.151476
    0 SNES Function norm 9.951368195027e-01
      Linear solve converged due to CONVERGED_ATOL iterations 160
    1 SNES Function norm 3.315089885638e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --

50 TS dt 0.151476 time 7.42235
    0 SNES Function norm 1.755481622154e-03
      Linear solve converged due to CONVERGED_ATOL iterations 44
    1 SNES Function norm 3.243946411840e-08
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
51 TS dt 0.151476 time 7.57382
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

At the beginning of the output written to the terminal, we see that PyLith is reading the mesh using the `MeshIO Petsc` reader and that it found the domain to extend from 0 to 20 km in the x direction and from -20 km to 0 in the y direction. PyLith detects the use of poroelasticity without a fault and selects appropriate preconditioning options as discussed in *PETSc Options*.

At the end of the output written to the terminal, we see that the solver advanced the solution 51 time steps. The linear solver converges in about 50 or less iteration at most time steps and the norm of the residual met the absolute convergence tolerance (`ksp_atol`). The nonlinear solve converged in 1 iteration, which we expect because this is a linear problem, and the residual met the absolute convergence tolerance (`snestol`).

Visualizing the results

In Fig. 4.169 we use the `pylith_viz` utility to visualize the pressure field. You can move the slider or use the `p` and `n` keys to change the increment or decrement time. The initial linear variation in fluid pressure adjusts to match the boundary conditions and spatial variations in constitutive behavior.

Listing 4.256: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step01_inflation-domain.h5 warp_grid --field=pressure
```

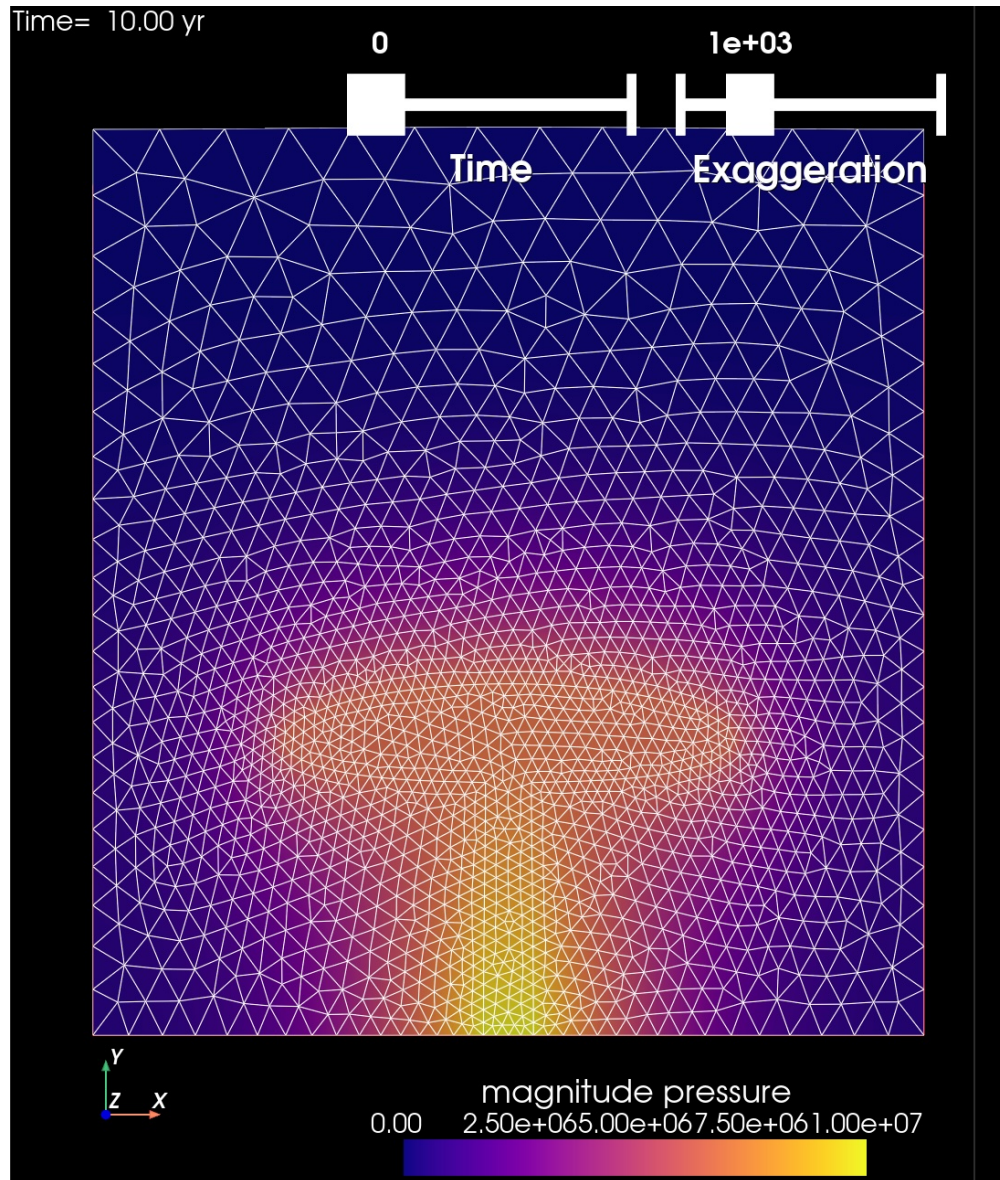


Fig. 4.169: Solution for Step 1 at $t=100$ yr. The colors of the shaded surface indicate the fluid pressure, and the deformation is exaggerated by a factor of 1000.

Step 1b: Adaptive Time Stepping

In Step 1b we demonstrate the use adaptive time stepping. We start with an initial time step of 0.2 years and let the adaptive time stepping algorithm increase the time step as the rate of deformation decreases. We decrease the tolerances slightly (the default tolerances are 0.05) for better agreement with the solution from uniform time stepping.

Listing 4.257: Adaptive time stepping parameters for Step 1b.

```
[pylithapp.timedependent]
start_time = -0.2*year
initial_dt = 0.2*year
end_time = 10.0*year

petsc_defaults.adaptive_time_stepping = True

[pylithapp.petsc]
ts_atol = 0.02
ts_rtol = 0.02
```

Listing 4.258: Run Step 1b simulation

```
$ pylith step01_inflation.cfg step01b_inflation.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/Utils/PetscManager.
↳py:55:initialize
-- info (application-flow)
-- Initialized PETSc with user options
    ts_atol = 0.02
    ts_rtol = 0.02
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:80:main
-- info (application-flow)
-- Running on 1 process(es).
>> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
↳PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
-- info (application-flow)
-- Setting PETSc options:
    fieldsplit_displacement_pc_type = lu
    fieldsplit_pressure_pc_type = bjacobi
    fieldsplit_trace_strain_pc_type = bjacobi
    ksp_atol = 1.0e-7
    ksp_converged_reason = true
    ksp_error_if_not_converged = true
    ksp_guess_pod_size = 8
    ksp_guess_type = pod
    ksp_rtol = 1.0e-14
    pc_fieldsplit_0_fields = 2
    pc_fieldsplit_1_fields = 1
    pc_fieldsplit_2_fields = 0
    pc_fieldsplit_type = multiplicative
    pc_type = fieldsplit
    snes_atol = 5.0e-7
    snes_converged_reason = true
    snes_error_if_not_converged = true
```

(continues on next page)

(continued from previous page)

```

snes_monitor = true
snes_rtol = 1.0e-14
ts_adapt_monitor = true
ts_adapt_reject_safety = 0.1
ts_adapt_safety = 0.2
ts_adapt_type = basic
ts_error_if_step_fails = true
ts_exact_final_time = matchstep
ts_monitor = true
ts_type = beuler
viewer_hdf5_collective = true

# -- many lines omitted --

19 TS dt 1.2231 time 6.35073
   0 SNES Function norm 1.893597646523e-03
     Linear solve converged due to CONVERGED_ATOL iterations 77
   1 SNES Function norm 2.064447881404e-08
     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
   TSAdapt basic beuler 0: step 19 accepted t=6.35073 + 1.223e+00 dt=1.412e+00 wlte=
→ 0.03 wltea= -1 wlter= -1
20 TS dt 1.41194 time 7.57382
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
→ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The adaptive time stepping algorithm shortens a few of the early time steps but then lengthens the time step as the rate of deformation decreases. We end up with 20 time steps with adaptive time stepping compared to 51 with uniform time steps.

Listing 4.259: Compare the results from Step 1 ad 1b using the `plot_compare.py` Python script.

```
./plot_compare.py
```

Step 2: Magma inflation with evolution of porosity

New in v4.0.0.

Features

- Quasi-static problem
- LU preconditioner
- `pylith.materials.Poroelasticity`
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.problems.SolnDispPresTracStrain`
- `pylith.problems.InitialConditionDomain`
- `pylith.bc.DirichletTimeDependent`

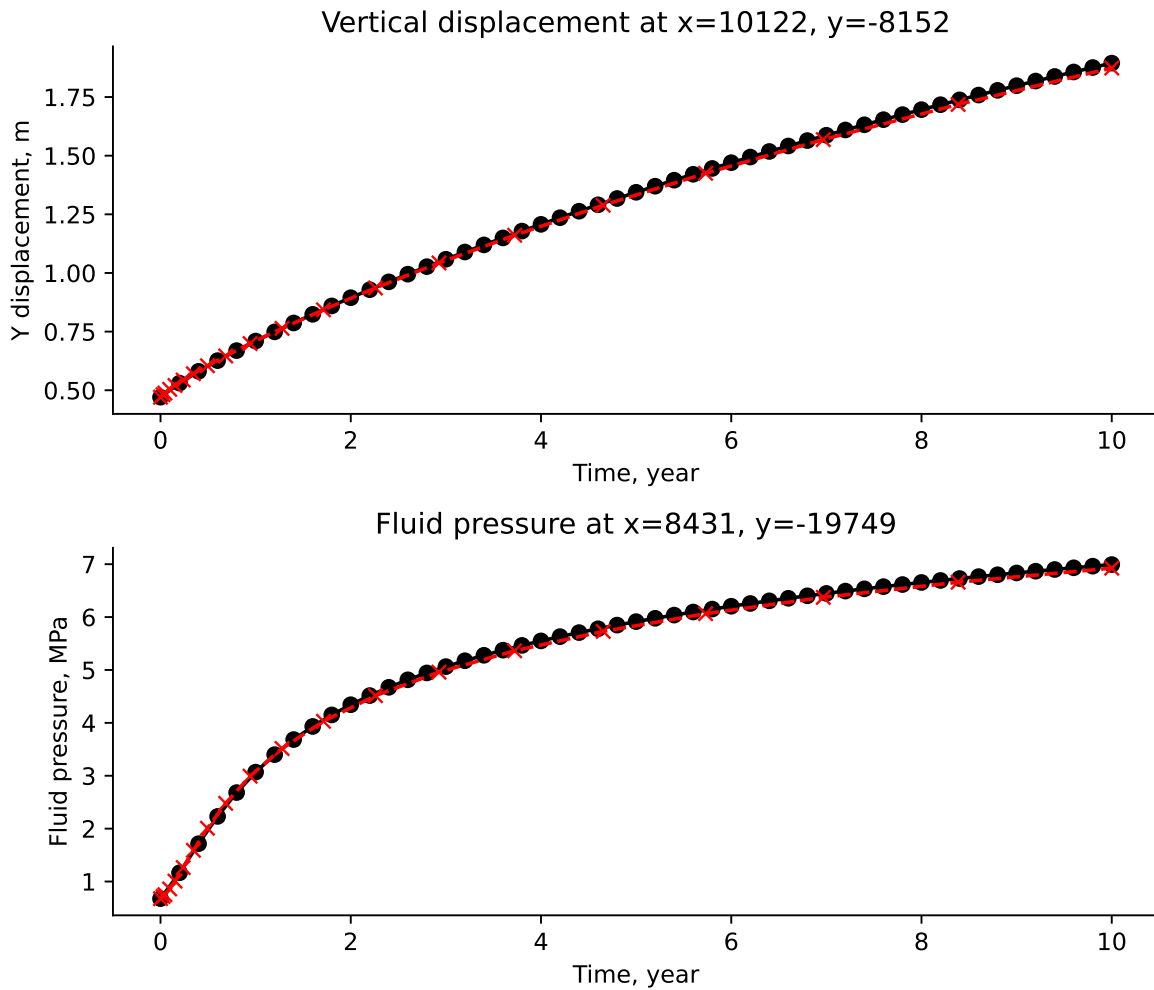


Fig. 4.170: Vertical displacement and fluid pressure time histories for Step 1 and 1b at locations with the largest changes. We find very little difference between the solutions from uniform time stepping (51 time steps) and adaptive time stepping (20 time steps).

- pylith.bc.NeumannTimeDependent
- pylith.meshio.DataWriterHDF5
- spatialdata.spatialdb.SimpleGridDB
- spatialdata.spatialdb.UniformDB
- Poroelasticity with porosity state variable
- Isotropic linear poroelasticity with reference state

Simulation parameters

We extend the simulation in Step 1 by including evolution of the porosity, which depends on the time derivative of the pressure and trace strain. We also compute the deformation relative to a uniform reference compressive pressure of 5 MPa to illustrate how to use a reference state with poroelasticity. We use the same initial conditions and boundary conditions as in Step 1.

Because the evolution of porosity depends on the time derivative of the solution subfields, we need to include the time derivatives in the solution field. As a result, we have 6 subfields in our solution field.

Listing 4.260: Solution subfields for Step 2.

```
# Poroelasticity with porosity state variable requires solution with time derivatives
solution = pylith.problems.SolnDispPresTracStrainVelPdotTdot

# Set basis order for all solution subfields
[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
pressure.basis_order = 1
trace_strain.basis_order = 1
velocity.basis_order = 2
pressure_t.basis_order = 1
trace_strain_t.basis_order = 1
```

Listing 4.261: Material parameters for poroelasticity with state variables and reference state for Step 2.

```
[pylithapp.problem.materials.crust]
use_state_variables = True

db_auxiliary_field.values = [
    solid_density, fluid_density, fluid_viscosity, porosity, shear_modulus, drained_bulk_
    ↪ modulus, biot_coefficient, fluid_bulk_modulus, solid_bulk_modulus, isotropic_
    ↪ permeability,
    reference_stress_xx, reference_stress_yy, reference_stress_zz, reference_stress_xy,
    reference_strain_xx, reference_strain_yy, reference_strain_zz, reference_strain_xy
]
db_auxiliary_field.data = [
    2500*kg/m**3, 1000*kg/m**3, 0.001*Pa*s, 0.01, 6.0*GPa, 10.0*GPa, 1.0, 2.0*GPa, 20.
    ↪ 0*GPa, 1e-15*m**2,
    -5.0*MPa, -5.0*MPa, -5.0*MPa, 0.0*MPa,
    0.0, 0.0, 0.0, 0.0
]
```

(continues on next page)

(continued from previous page)

```

auxiliary_subfields.porosity.basis_order = 1

[pylithapp.problem.materials.crust.bulk_rheology]
use_reference_state = True

[pylithapp.problem.materials.intrusion]
use_state_variables = True

db_auxiliary_field.values = [
    solid_density, fluid_density, fluid_viscosity, porosity, shear_modulus, drained_bulk_
    ↪modulus, biot_coefficient, fluid_bulk_modulus, solid_bulk_modulus, isotropic_
    ↪permeability,
    reference_stress_xx, reference_stress_yy, reference_stress_zz, reference_stress_xy,
    reference_strain_xx, reference_strain_yy, reference_strain_zz, reference_strain_xy
]
db_auxiliary_field.data = [
    2500*kg/m**3, 1000*kg/m**3, 0.001*Pa*s, 0.1, 6.0*GPa, 10.0*GPa, 0.8, 2.0*GPa, 20.
    ↪0*GPa, 1e-13*m**2,
    -5.0*MPa, -5.0*MPa, -5.0*MPa, 0.0*Pa,
    0.0, 0.0, 0.0, 0.0
]

auxiliary_subfields.porosity.basis_order = 1

[pylithapp.problem.materials.intrusion.bulk_rheology]
use_reference_state = True

```

The changes in the physics as well as the default solver settings that impact the initial guess leads to a large initial residual at the second time step. Consequently, we increase the divergence tolerance for the linear solver.

Listing 4.262: Adjustment of the divergence tolerance for the linear solver for Step 2.

```

[pylithapp.petsc]
# Increase divergence tolerance. Initial guess at second time step is not accurate.
ksp_divtol = 1.0e+5

```

Running the simulation

Listing 4.263: Run Step 2 simulation

```

$ pylith step02_inflation_statevars.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
-- info (application-flow)
-- Running on 1 process(es).
>> software/pylith-debug/lib/python3.12/site-packages/pylith/Utils/PetscManager.
    ↪py:64:showOptions
-- info (application-flow)
-- PETSc user options:

```

(continues on next page)

(continued from previous page)

```

ksp_divtol = 1.0e+6

# -- many lines omitted --

50 TS dt 0.151476 time 7.42235
    0 SNES Function norm 1.207876038262e-02
      Linear solve converged due to CONVERGED_ATOL iterations 79
    1 SNES Function norm 6.174330339004e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
51 TS dt 0.151476 time 7.57382
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

The linear solver exhibits similar performance with less than 50 iterations at most time steps. Furthermore, the problem is still linear, so the nonlinear solver converges in one iteration.

Visualizing the results

In Fig. 4.171 we use the `pylith_viz` utility to visualize the pressure field. You can move the slider or use the p and n keys to change the increment or decrement time.

Listing 4.264: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filenames=output/step02_inflation_statevars-domain.h5 warp_grid --
↪field=pressure

```

Suggested Exercises

1. Change the material properties. How do the changes affect the fluid flow?
2. Add time-dependence to the pressure (Neumann) boundary condition.
3. Change the geometry of the magma reservoir. How does this affect the results?

2D Outer-Rise Hydration Using Poroelasticity

The files are in the directory `examples/poroelastic-outerise-2d`. The files and directories for this set of examples includes:

README.md

README file containing a brief description of the various examples.

***.cfg**

PyLith parameter files.

generate_gmsh.py

Python script to generate mesh using Gmsh.

***.msh**

Gmsh mesh files generated by Gmsh.

***.spatialdb**

Spatial database files associated with the spatial databases.

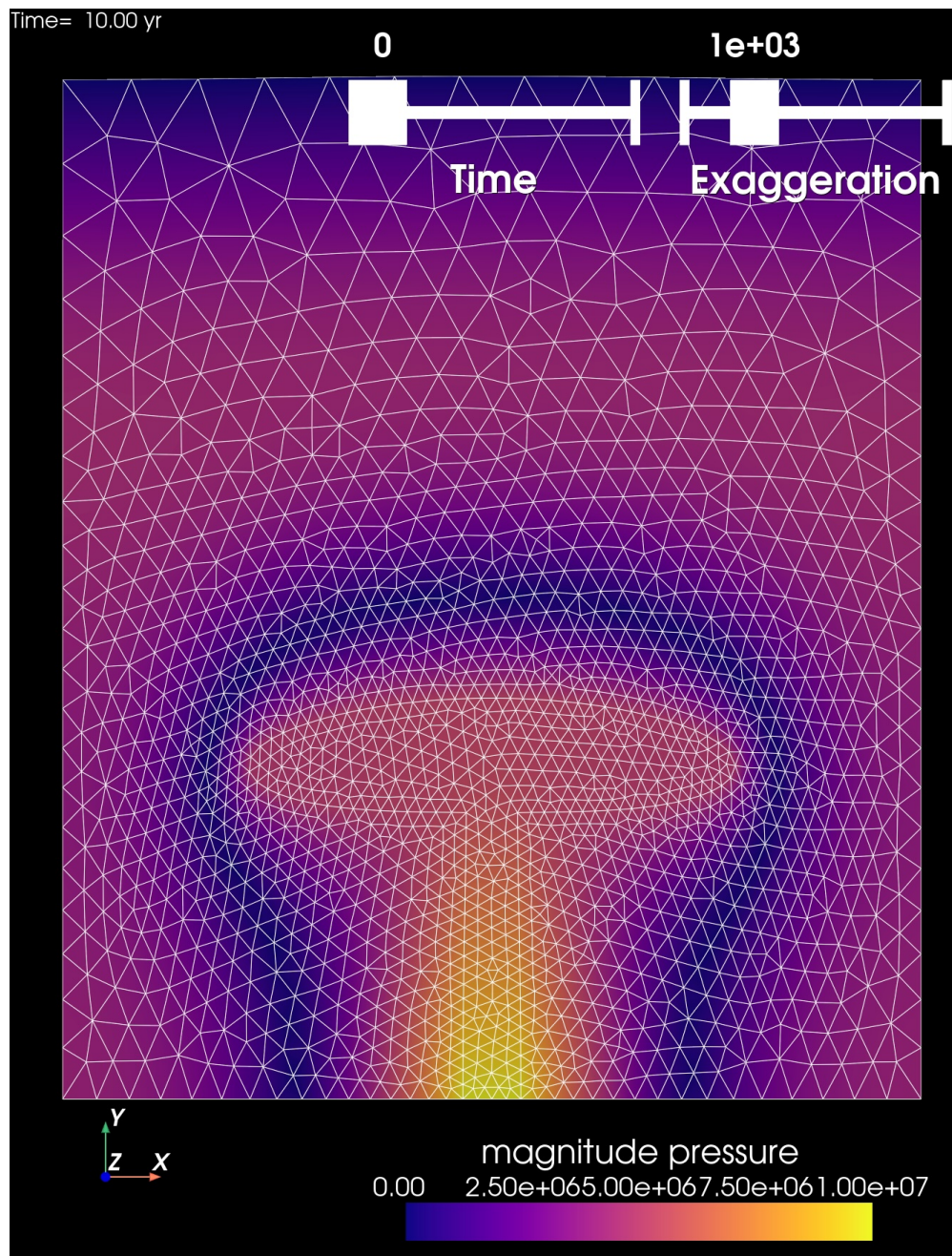


Fig. 4.171: Solution for Step 2 at $t=100$ yr. The colors of the shaded surface indicate the fluid pressure, and the deformation is exaggerated by a factor of 1000. The reference state gives rise to greater vertical deformation compared to Step 1. The choice of material properties does not lead to significant changes in porosity in either material during the simulation (not shown in the figure).

output

Directory containing simulation output. It is created automatically when running the simulations.

Overview

This example examines how bending stresses and permeability structure influence the hydration state of a slab of oceanic lithosphere in the outer-rise of a subduction zone. The poroelastic slab is deformed based on bathymetry along a trench-orthogonal transect of the Central-American subduction system. Enhanced permeability is introduced in the outer-rise to simulate the effect of normal faulting.

Step 1

Underformed lithosphere with a uniform fluid-pressure applied to the top-boundary

Step 2

Lithosphere is deformed to the current bathymetry over 0.3 Myr.

Step 3

Lithosphere is deformed to the current bathymetry over 0.3 Myr with enhanced permeability from outer-rise faults.

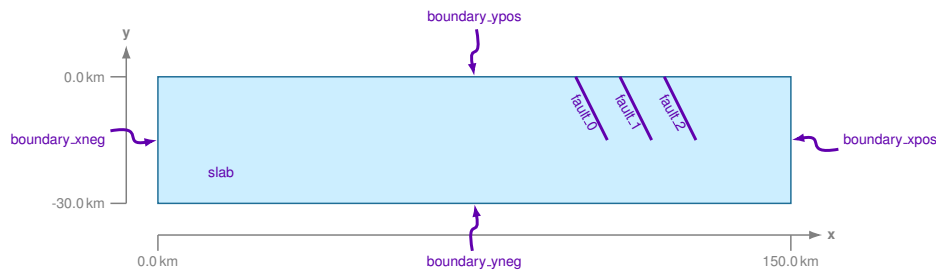


Fig. 4.172: Diagram of geometry used for poroelastic-outrise-2d. The domain extends from 0 km to +150 km in the x direction and from -30 km to 0 km in the y direction. We refer to the domain boundaries using the names shown in the diagram.

Example Workflow**Gmsh Mesh****Geometry**

We construct the geometry by first creating points, then connecting the points into curves, and finally the curves into surfaces. Fig. 4.173 shows the geometry and variables names of the vertices and curves.

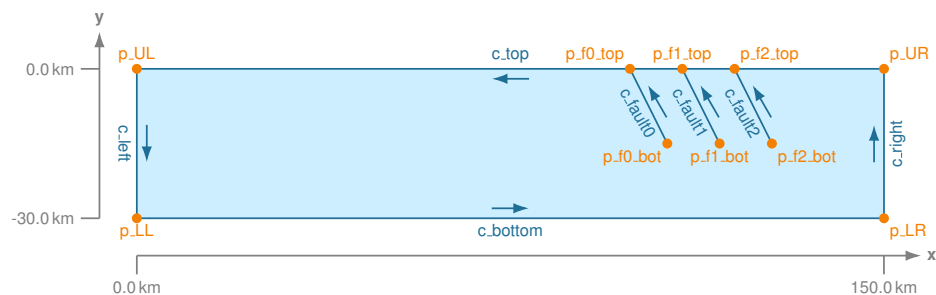


Fig. 4.173: Geometry created in Gmsh for generating the finite-element mesh. We construct curves from points (p_*) and surfaces from the curves (c_*). The arrows indicate the direction (orientation) of the curves.

Meshing using Python Script

We use the Python script `generate_gmsh.py` to create the geometry and generate the mesh. The script is structured identically to the one used in `examples/strikeslip-2d` and `examples/reverse-2d`. We create a class `App` that implements the functionality missing in `gmsh_utils.GenerateMesh`. We must implement the `create_geometry()`, `mark()`, and `generate_mesh()` methods that are abstract in the `GenerateMesh` base class. In this case the geometry is significantly more complex with arrays of points defining the top surface (topography and bathymetry) and the geometry of the slab.

We use the Gmsh `MeshSize` options to define a discretization size that grows slowly at a geometric rate with distance from the main fault.

Listing 4.265: Run the `generate_gmsh.py` Python script to generate the mesh.

```
# Generate a mesh with triangular cells and save it to `mesh_tri.msh` (default filename).
$ ./generate_gmsh.py --write

# Save as above but start the Gmsh graphical interface after saving the mesh.
$ ./generate_gmsh.py --write --gui

# Create only the geometry and start the Gmsh graphical interface.
$ ./generate_gmsh.py --geometry --gui
```

By default the Python script will generate a finite-element mesh with triangular cells and save it to the file `mesh_tri.msh`. You can view the mesh using Gmsh either by using the `--gui` command line argument when you generate the mesh or running Gmsh from the command line and opening the file.

Listing 4.266: View the Gmsh mesh file `mesh_tri.msh` using Gmsh.

```
gmsh -open mesh_tri.msh
```

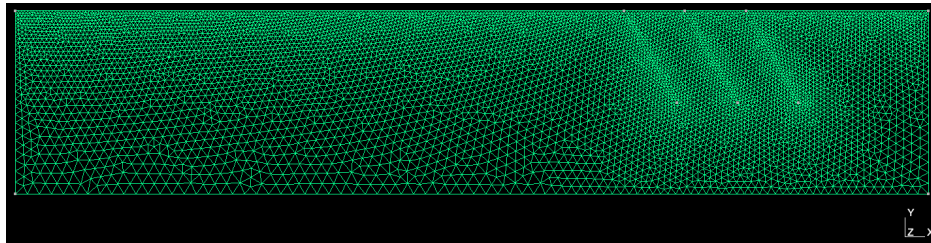


Fig. 4.174: Finite-element mesh with triangular cells generated by Gmsh.

Common Information

In addition to the finite-element mesh, PyLith requires files to specify the simulation parameters. We specify parameters common to all simulations in a directory in `pylithapp.cfg`. The `pylithapp.cfg` file contains numerous comments, so we only summarize the parameters here.

Metadata, Mesh, and Output

The `pylithapp.metadata` section specifies metadata common to all simulations in the directory. We set the parameters for importing the finite-element mesh in `pylithapp.mesh_generator`.

Physics

These quasi-static simulations solve the poroelasticity equation with state-variables, so we have a solution field with displacement, pressure, volumetric strain, velocity, the time derivative of pressure, and the volumetric strain rate subfields.

$$\vec{s} = (\vec{u} \quad p \quad \epsilon_v \quad \vec{v} \quad \dot{p} \quad \dot{\epsilon}_v)^T, \quad (4.495)$$

$$\nabla \cdot \boldsymbol{\sigma}(\vec{u}, p) = \vec{0}, \quad (4.496)$$

$$\frac{\partial \zeta(\vec{u}, p)}{\partial t} + \nabla \cdot \vec{q}(p) = 0, \quad (4.497)$$

$$\nabla \cdot \vec{u} - \epsilon_v = 0, \frac{\partial \phi(\vec{x}, t)}{\partial t} = (\alpha(\vec{x}) - \phi(\vec{x}, t)) \left(\dot{\epsilon}_v + \frac{1 - \alpha(\vec{x})}{K_d(\vec{x})} \dot{p}(\vec{x}, t) \right) \quad (4.498)$$

We specify a basis order of 2 for the displacement, the velocity, and the isotropic permeability subfields, and a basis order of 1 for each of the remaining subfields.

Listing 4.267: Discretization parameters for the 2D outer-rise examples with poroelasticity.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispPresTracStrainVelPdotTdot
defaults.quadrature_order = 2

[pylithapp.problem.solution.subfields]
displacement.basis_order = 2
pressure.basis_order = 1
trace_strain.basis_order = 1

velocity.basis_order = 2
pressure_t.basis_order = 1
trace_strain_t.basis_order = 1

[pylithapp.problem.materials.slabs.bulk_rheology]
auxiliary_subfields.isotropic_permeability.basis_order = 2
```

Listing 4.268: Nondimensionalization parameters for the 2D outer-rise examples with poroelasticity.

```
[pylithapp.problem]
scales = pylith.scales.NondimElasticQuasistatic
scales.length_scale = 100.0*m
scales.relaxation_time = 1*year
scales.shear_modulus = 10.0*GPa
```

Listing 4.269: Time stepping parameters for the 2D outer-rise examples with poroelasticity.

```
[pylithapp.time-dependent]
start_time = -6e3*year
initial_dt = 6e3*year
end_time = 300e3*year
```

We set the material parameters that are common across the three steps in the `pylithapp.cfg` file to avoid repeating parameters. We only have a single material with spatially varying material properties so we use a `SimpleGridDB`

spatial database. We use differ SimpleGridDB files to initialize the material properties for the three steps, so we specify the filename in the parameter file for each step.

Listing 4.270: Material parameters common to all steps in the 2D outer-rise examples with poroelasticity.

```
[pylithapp.problem]
materials = [slab]
materials.slab = pylith.materials.Poroelasticity

[pylithapp.problem.materials]
slab.bulk_rheology = pylith.materials.IsotropicLinearPoroelasticity

[pylithapp.problem.materials.slab]
label_value = 1
use_state_variables = True
db_auxiliary_field = spatialdata.spatialdb.SimpleGridDB
db_auxiliary_field.description = Spatial database for material properties and state_
↳variables
db_auxiliary_field.query_type = linear

observers.observer.data_fields = [displacement, pressure, cauchy_stress, velocity,
↳porosity, isotropic_permeability, water_content]
```

For all steps, the left boundary is held fixed, which allows the rest of the boundary to pivot about this anchor point. The top boundary always has a fluid pressure boundary condition, while the displacement boundary condition varies between steps for the top boundary. The bottom and right boundaries remain unconstrained.

Listing 4.271: Boundary conditions for the 2D outer-rise examples with poroelasticity.

```
[pylithapp.problem]
bc = [bc_xneg, bc_ypos, bc_ypos_fluid]

bc.bc_west = pylith.bc.DirichletTimeDependent
bc.bc_ypos = pylith.bc.DirichletTimeDependent
bc.bc_ypos_fluid = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.bc_xneg]
constrained_dof = [0, 1]
label = boundary_xneg
label_value = 11
field = displacement
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on -x for displacement

[pylithapp.problem.bc.bc_ypos]
constrained_dof = [0, 1]
label = boundary_ypos
label_value = 10
field = displacement
db_auxiliary_field = pylith.bc.ZeroDB
db_auxiliary_field.description = Dirichlet BC on +y for displacement
```

(continues on next page)

(continued from previous page)

```
[pylithapp.problem.bc.bc_ypos_fluid]
constrained_dof = [0]
label = boundary_ypos
label_value = 10
field = pressure
use_initial = True
db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC on +y for fluid pressure
db_auxiliary_field.iohandler.filename = simpleDB_files/surface_fluid_pressure.txt
```

Generate spatial databases

Important

The spatial database files are not provided because some of them are large. You generate them using the provided Python scripts *before* running the simulations.

Listing 4.272: Generate the spatial database files.

```
# Generate spatial databases for the Dirichlet BC on the top surface.
./generate_spatialdb_ypos.py

# Generate spatial databases for the material properties.
./generate_spatialdb_matprops.py
```

Step 1: No faults, no flexure

Features

- Quasi-static problem
- triangular cells
- LU preconditioner
- pylith.materials.Poroelasticity
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.problems.SolnDispPresTracStrainVelPdotTdot
- pylith.problems.InitialConditionDomain
- pylith.bc.DirichletTimeDependent
- pylith.bc.NeumannTimeDependent
- pylith.meshio.DataWriterHDF5
- spatialdata.spatialdb.SimpleGridDB
-

Simulation parameters

This example uses poroelasticity to model the infiltration of seawater through a slab of oceanic lithosphere. The permeability field is depth dependent, decreasing with depth but does not vary laterally. The lithosphere is not subject to any deformation, but a fluid pressure is applied to the top boundary that is equivalent to the pressure exerted on the seafloor by the water column. This simulates what the hydration state of the oceanic lithosphere is far from any spreading centers, or convergent margins. It also serves as a comparison to see how the hydration state varies for later steps where more complexities are introduced.

Fig. 4.175 shows the boundary conditions on the domain. The parameters specific to this example are in `step01_no_faults_no_flexure.cfg`.

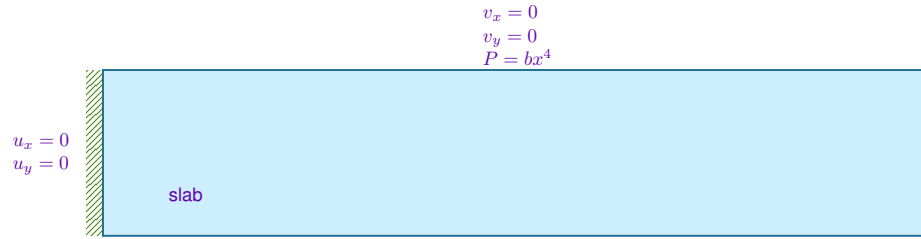


Fig. 4.175: Boundary and initial conditions for Step 1. We fix the left and the top boundaries with a zero displacement boundary condition, while leaving the right and bottom boundaries unconstrained. We impose a fluid pressure on the +y boundary equal to the weight of the water column to generate fluid flow.

Listing 4.273: Initial condition parameters for Step 1. We use a SimpleGridDB file that does not contain enhanced permeability due to outer rise faults.

```
[pylithapp.problem]

[pylithapp.problem.materials.slabs]
db_auxiliary_field.filename = no_faultzone_permeability.spatialdb
```

Running the simulation

Listing 4.274: Run Step 1 simulation

```
$ pylith step01_no_faults_no_flexure.cfg

# The output should look something like the following.
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 8.41536 time -8.41536
  0 SNES Function norm 1.780933174156e+03
    Linear solve converged due to CONVERGED_ATOL iterations 23
  1 SNES Function norm 9.489171115501e-10
```

(continues on next page)

(continued from previous page)

```

Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 8.41536 time 0.
  0 SNES Function norm 2.440019354463e-03
    Linear solve converged due to CONVERGED_ATOL iterations 22
  1 SNES Function norm 3.000361522615e-13
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --

50 TS dt 8.41536 time 412.353
  0 SNES Function norm 2.201389334854e-05
    Linear solve converged due to CONVERGED_ATOL iterations 18
  1 SNES Function norm 3.145787316430e-13
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
51 TS dt 8.41536 time 420.768
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

Visualizing the results

The output directory contains the simulation output. Each “observer” writes its own set of files, so the solution over the domain is in one set of files, the boundary condition information is in another set of files, and the material information is in yet another set of files. The HDF5 (.h5) files contain the mesh geometry and topology information along with the solution fields. The Xdmf (.xmf) files contain metadata that allow visualization tools like ParaView to know where to find the information in the HDF5 files. To visualize the data using ParaView or Visit, load the Xdmf files.

In Fig. 4.176 we use the `pylith_viz` utility to visualize the porosity field.

Listing 4.275: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filenames=output/step01_no_faults_no_flexure-slab.h5 warp_grid --
↪field=porosity --exaggeration=1 --hide-edges

```

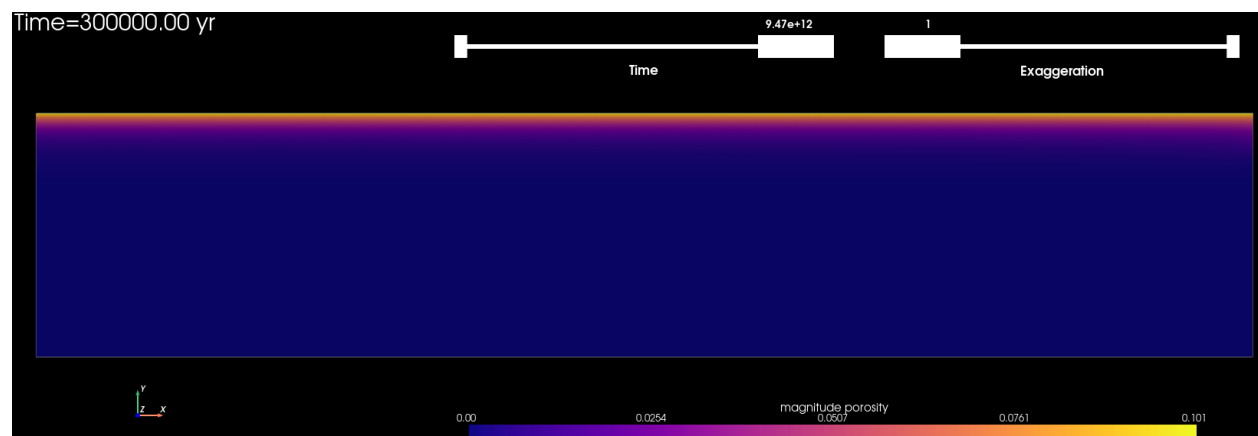


Fig. 4.176: Porosity field at the end of model run time for Step 1.

Step 2: No faults with flexure

Features

- Quasi-static problem
- triangular cells
- LU preconditioner
- pylith.materials.Poroelasticity
- pylith.meshio.MeshIOPetsc
- pylith.problems.TimeDependent
- pylith.problems.SolnDispPresTracStrainVelPdotTdot
- pylith.problems.InitialConditionDomain
- pylith.bc.DirichletTimeDependent
- pylith.bc.NeumannTimeDependent
- pylith.meshio.DataWriterHDF5
- spatialdata.spatialdb.SimpleGridDB
-

Simulation parameters

This example uses poroelasticity to model the infiltration of seawater through a slab of oceanic lithosphere. The permeability field is depth dependent, decreasing with depth but does not vary laterally. The lithosphere is now subject to deformation, over 300 kyr the slab bends to simulate extensional stresses in the outer-rise of a subduction zone. A fluid pressure is applied to the top boundary that is equivalent to the pressure exerted on the seafloor by the water column. This simulates what the hydration state of the oceanic lithosphere as it is about to enter a convergent margin.

Fig. 4.177 shows the boundary conditions on the domain. The parameters specific to this example are in `step02_no_faults_flexure.cfg`.

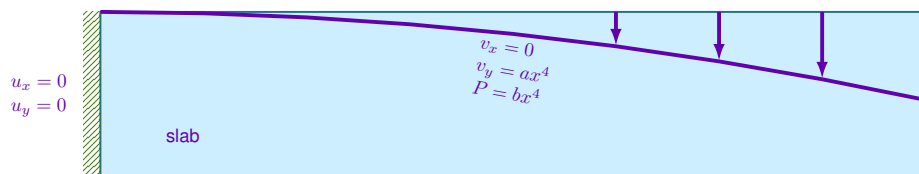


Fig. 4.177: Boundary and initial conditions for Step 2. We fix the left boundary, but we now apply a spatially varying velocity condition on the top boundary using a SimpleDB file, while leaving the right and bottom boundaries unconstrained. We impose a fluid pressure on the +y boundary equal to the weight of the water column to generate fluid flow.

Listing 4.276: Modified top boundary condition for Step 2.

```
[pylithapp.problem.bc.boundary_top]
use_initial = False
use_rate = True

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC +y boundary
db_auxiliary_field.iohandler.filename = top_velocity_boundary.spatialdb
```

Listing 4.277: Initial condition parameters for Step 2. We use a SimpleGridDB file that does not contain enhanced permeability due to outer rise faults.

```
[pylithapp.problem]

[pylithapp.problem.materials.slab]
db_auxiliary_field.filename = no_faultzone_permeability.spatialdb
```

Running the simulation

Listing 4.278: Run Step 2 simulation

```
$ pylith step02_no_faults_flexure.cfg
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
↪pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 8.41536 time -8.41536
  0 SNES Function norm 1.780933174156e+03
    Linear solve converged due to CONVERGED_ATOL iterations 23
  1 SNES Function norm 9.489171115501e-10
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 8.41536 time 0.
  0 SNES Function norm 1.624022924774e+03
    Linear solve converged due to CONVERGED_ATOL iterations 34
  1 SNES Function norm 1.245577106021e-08
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --
50 TS dt 8.41536 time 412.353
  0 SNES Function norm 1.624022924815e+03
    Linear solve converged due to CONVERGED_ATOL iterations 36
  1 SNES Function norm 8.879603719365e-09
    Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
51 TS dt 8.41536 time 420.768
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↪py:222:finalize
-- info (application-flow)
-- Finalizing problem.
```

Visualizing the results

In Fig. 4.178 we use the `pylith_viz` utility to visualize the porosity field.

Listing 4.279: Visualize PyLith output using `pylith_viz`.

```
pylith_viz --filenames=output/step02_no_faults_flexure-slab.h5 warp_grid --
↔field=porosity --exaggeration=1 --hide-edges
```

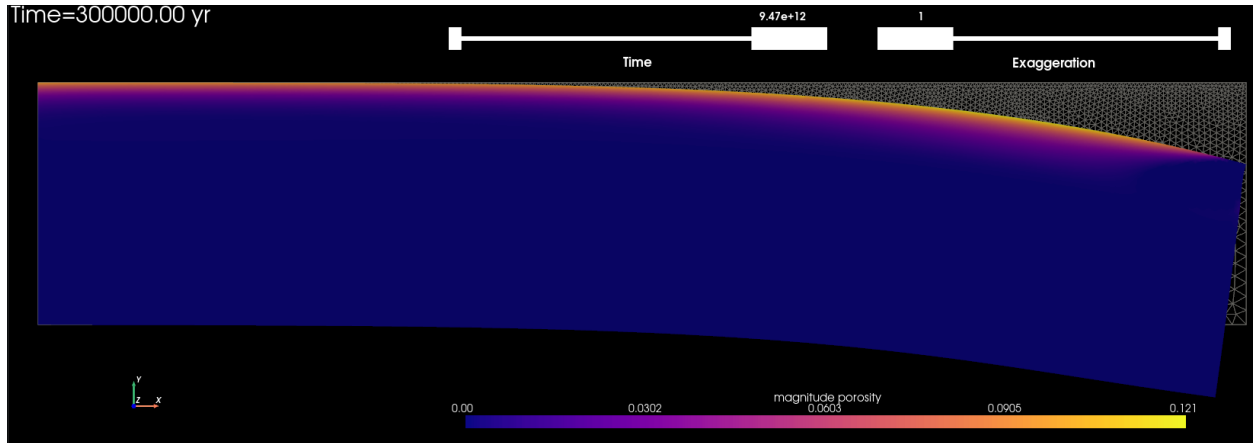


Fig. 4.178: Porosity field at the end of the simulation for Step 2.

Step 3: Faults with flexure

Features

- Quasi-static problem
- triangular cells
- LU preconditioner
- `pylith.materials.Poroelasticity`
- `pylith.meshio.MeshIOPetsc`
- `pylith.problems.TimeDependent`
- `pylith.problems.SolnDispPresTracStrainVelPdotTdot`
- `pylith.problems.InitialConditionDomain`
- `pylith.bc.DirichletTimeDependent`
- `pylith.bc.NeumannTimeDependent`
- `pylith.meshio.DataWriterHDF5`
- `spatialdata.spatialdb.SimpleGridDB`
-

Simulation parameters

This example uses poroelasticity to model the infiltration of seawater through a slab of oceanic lithosphere. The permeability field is depth dependent, decreasing with depth and also varies laterally, simulating the enhanced permeability within normal faults in the outer-rise. The lithosphere is subject to deformation, over 300 kyr the slab bends to simulate extensional stresses in the outer-rise of a subduction zone. A fluid pressure is applied to the top boundary that is equivalent to the pressure exerted on the seafloor by the water column. This simulates what the hydration state of the oceanic lithosphere as it is about to enter a convergent margin with the addition of enhanced permeability within faults.

Fig. 4.179 shows the boundary conditions on the domain. The parameters specific to this example are in `step03_faults_flexure.cfg`.

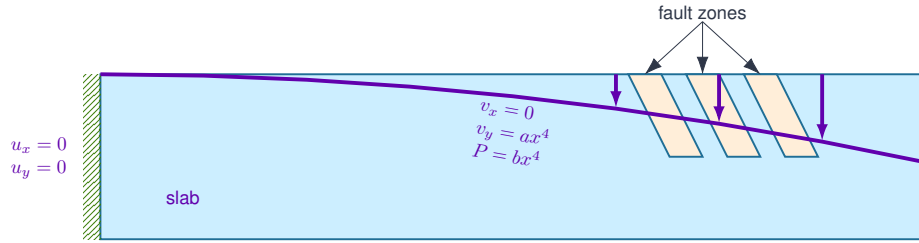


Fig. 4.179: Boundary and initial conditions for Step 3. We fix the left boundary, and we apply a spatially varying velocity condition on the top boundary using a SimpleDB file, while leaving the right and bottom boundaries unconstrained. We add laterally varying permeability by increasing the permeability within the vicinity of faults in the outer-rise. We impose a fluid pressure on the +y boundary equal to the weight of the water column to generate fluid flow.

Listing 4.280: Modified top boundary condition for Step 3.

```
[pylithapp.problem.bc.boundary_top]
use_initial = False
use_rate = True

db_auxiliary_field = spatialdata.spatialdb.SimpleDB
db_auxiliary_field.description = Dirichlet BC +y boundary
db_auxiliary_field.iohandler.filename = top_velocity_boundary.spatialdb
```

Listing 4.281: Initial condition parameters for Step 3. We use a different SimpleGridDB file that contains enhanced permeability around outer rise faults.

```
[pylithapp.problem]

[pylithapp.problem.materials.slabs]
db_auxiliary_field.filename = enhanced_faultzone_permeability.spatialdb
```

Running the simulation

Listing 4.282: Run Step 3 simulation

```
$ pylith step03_faults_flexure.cfg
>> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:79:main
-- info (application-flow)
-- Running on 1 process(es).
```

(continues on next page)

(continued from previous page)

```

# -- many lines omitted --
>> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
↳ pylith::problems::TimeDependent::solve()
-- info (application-flow)
-- Component 'timedependent.problem': Solving equations.
0 TS dt 8.41536 time -8.41536
    0 SNES Function norm 2.230305189106e+03
      Linear solve converged due to CONVERGED_ATOL iterations 46
    1 SNES Function norm 2.535484482781e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
1 TS dt 8.41536 time 0.
    0 SNES Function norm 1.624022924774e+03
      Linear solve converged due to CONVERGED_ATOL iterations 41
    1 SNES Function norm 1.238578421139e-10
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1

# -- many lines omitted --

50 TS dt 8.41536 time 412.353
    0 SNES Function norm 1.624022924802e+03
      Linear solve converged due to CONVERGED_ATOL iterations 40
    1 SNES Function norm 3.703317710018e-09
      Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
51 TS dt 8.41536 time 420.768
>> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
↳ py:222:finalize
-- info (application-flow)
-- Finalizing problem.

```

Visualizing the results

In Fig. 4.180 we use the `pylith_viz` utility to visualize the porosity field.

Listing 4.283: Visualize PyLith output using `pylith_viz`.

```

pylith_viz --filenames=output/step03_faults_flexure-slab.h5 warp_grid --field=porosity --
↳ exaggeration=1 --hide-edges

```

Suggested Exercises

1. Change the amount of bending by increasing the model runtime, or increasing the velocity imposed on the top boundary. How does this impact the amount of hydration?
2. Change the depth of the faults in step03, and the permeability within the fault-zone. How does this impact the hydration?

Troubleshooting (2D)

This set of examples demonstrates how to troubleshoot PyLith simulations. We consider Steps 1a and 6 from `examples/reverse-2d` and introduce errors into the parameter files and spatial databases. We illustrate how to dissect a variety of error messages, strategies for diagnosing errors, and the resolution.

The files are in the directory `examples/troubleshooting-2d`. The files and directories for this set of examples

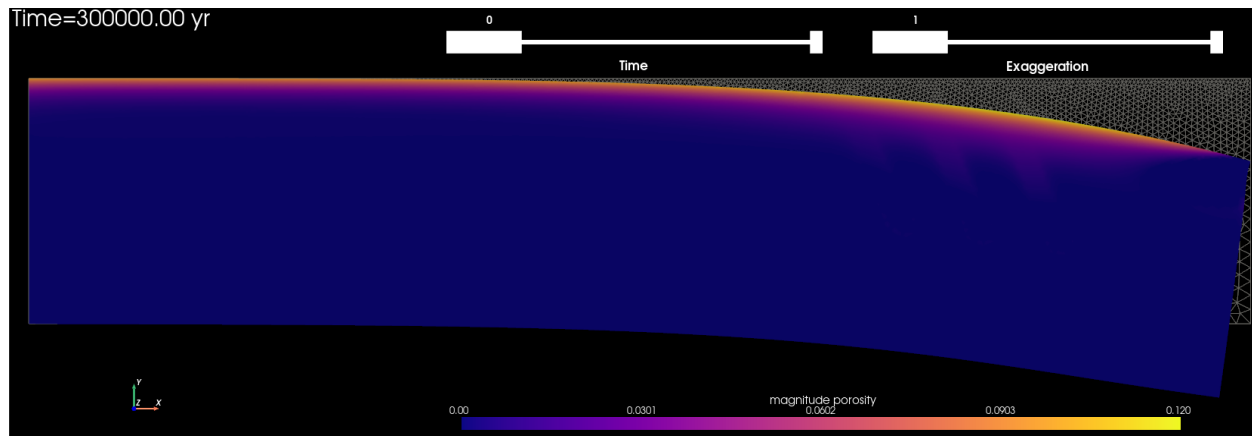


Fig. 4.180: Porosity field at the end of the simulation for Step 3.

includes:

:README.md README file containing a brief description of the various examples. :*.cfg PyLith parameter files. :*.msh Gmsh finite-element mesh files generated by Gmsh. :*.spatialdb Spatial database files associated with the spatial databases. :output Directory containing simulation output. It is created automatically when running the simulations.

Example Workflow

Error Messages

Summary of error messages addressed in this set of examples.

Configuration Errors

Listing 4.284: Refer to *Step 1, Error 1*.

```
>> {default}::
-- error (pyre.inventory)
-- timedependent.problem_defaults.name <- ''
-- Missing required property 'name' in default options for problem.
```

Listing 4.285: Refer to *Step 1, Error 2*.

```
>> {default}::
-- error (pyre.inventory)
-- timedependent.bc.dirichlettimedependent.label <- ''
-- Label for boundary condition group/nodeset/pset in mesh not specified.
```

Listing 4.286: Refer to *Step 1, Error 3*.

```
>> ./pylithapp.cfg:112:
-- error (pyre.inventory)
-- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.bulk_modulus.basis_
  ↳ order <- '0'
-- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
  ↳ bulk_modulus'
```

Listing 4.287: Refer to *Step 6, Error 1*.

```
>> {default}::
-- error (pyre.inventory)
-- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.
↪description <- ''
-- Description for spatial database not specified.
```

Listing 4.288: Refer to *Step 6, Error 2*.

```
>> {default}::
-- error (pyre.inventory)
-- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.
↪simpleioascii.filename <- ''
-- Filename for spatial database not specified.
```

Runtime Errors

Listing 4.289: Refer to *Step 1, Error 4*.

```
>> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:802:static void
↪pylith::topology::MeshOps::checkMaterialLabels(const pylith::topology::Mesh&,
↪pylith::int_array&)
-- error (user-input)
-- Material label_value '3' for cell '609' does not match the label_value of any
↪materials or interfaces.
```

Listing 4.290: Refer to *Step 1, Error 5*.

```
[0]PETSC ERROR: Error in external library
[0]PETSC ERROR: Error converting spatial database values for gravitational_acceleration
↪at ( -92285.3 -72772.7) in spatial database 'Gravity field'. Found near zero
↪magnitude (0) for gravity field vector ( 0 0).
```

Listing 4.291: Refer to *Step 6, Error 3*.

```
>> src/cig/pylith/libsrc/pylith/topology/FieldOps.cc:224:static void
↪pylith::topology::FieldOps::checkSubfieldsExist(const pylith::string_vector &, const
↪std::string &, const pylith::topology::Field &)
-- error (user-input)
-- Could not find 'lagrange_multiplier_fault' in domain solution field. Field contains:
↪'displacement'; the missing fields are required for interface 'FaultCohesiveKin'.
```

Listing 4.292: Refer to *Step 6, Error 4*.

```
RuntimeError: Error occurred while reading spatial database file 'fault_slip.spatialdb'.
Read data for 3 out of 4 points.
```

Listing 4.293: Refer to *Step 6, Error 5*.

```
RuntimeError: Could not find value 'final_slip_opening' in spatial database 'Fault
```

(continues on next page)

(continued from previous page)

```

↪rupture for main fault'. Available values are:
    final-slip-left-lateral
    final-slip-opening
    initiation-time

```

Listing 4.294: Refer to *Step 6, Error 6*.

```

>> src/cig/pylith/libsrc/pylith/topology/FieldQuery.cc:473:static void_
↪pylith::topology::_FieldQuery::findQueryIndices(FieldQuery::DBQueryContext *, const_
↪pylith::string_vector &)
-- error (user-input)
-- Could not find value 'final_slip_opening' in spatial database 'Fault rupture for main_
↪fault'. Available values are:
    final-slip-left-lateral
    final-slip-opening
    initiation-time

```

Listing 4.295: Refer to *Step 6, Error 7*.

```

[0]PETSC ERROR: Error in external library
[0]PETSC ERROR: Could not find values for initiation_time at ( -21860.3 -27621) in_
↪spatial database 'Fault rupture for main fault'.

```

Step 1: Gravitational Body Forces

This example involves a static simulation that solves for the deformation from loading by gravitational body forces. Fig. 4.181 shows the boundary conditions on the domain. The parameters specific to this example are in `step01_gravity.cfg`.

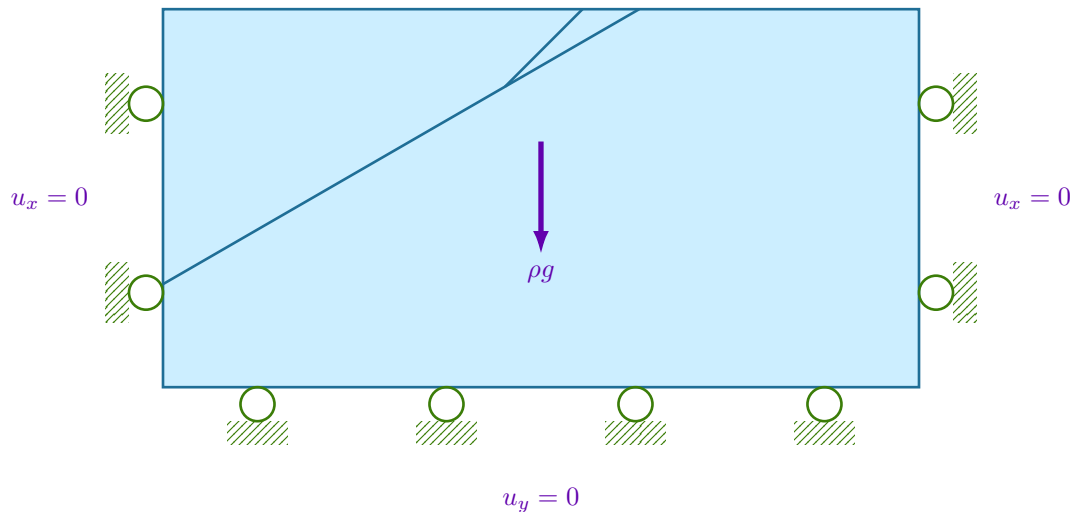


Fig. 4.181: We apply roller boundary conditions on the lateral sides and bottom of the domain.

See *examples/reverse-2d Step 1* for more information.

Step 1: Error 1**Error Message**

Listing 4.296: Error message 1 when running Step 1.

```

1 $ pylith step01a_gravity.cfg
2
3 # Output
4 >> {default}::
5 -- error (pyre.inventory)
6 -- timedependent.problem_defaults.name <- ''
7 -- Missing required property 'name' in default options for problem.
8 >> {default}::
9 -- error (pyre.inventory)
10 -- timedependent.bc.dirichlettimedependent.label <- ''
11 -- Label for boundary condition group/nodeset/pset in mesh not specified.
12 >> {default}::
13 -- error (pyre.inventory)
14 -- timedependent.bc.dirichlettimedependent.simplifiedb.description <- ''
15 -- Description for spatial database not specified.
16 >> {default}::
17 -- error (pyre.inventory)
18 -- timedependent.bc.dirichlettimedependent.simplifiedb.simpleioascii.filename <- ''
19 -- Filename for spatial database not specified.
20 >> {default}::
21 -- error (pyre.inventory)
22 -- timedependent.bc.dirichlettimedependent.constrained_dof <- '[]'
23 -- No constrained degrees of freedom found for time-dependent Dirichlet boundary_
   ↳ condition 'bc_xpos'. 'constrained_dof' must be a zero-based integer array (0=x, 1=y,
   ↳ 2=z).
24 >> ./pylithapp.cfg:112:
25 -- error (pyre.inventory)
26 -- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.bulk_modulus.basis_
   ↳ order <- '0'
27 -- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
   ↳ bulk_modulus'
28 >> ./pylithapp.cfg:113:
29 -- error (pyre.inventory)
30 -- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.shear_modulus.basis_
   ↳ order <- '0'
31 -- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
   ↳ shear_modulus'
32 >> ./pylithapp.cfg:135:
33 -- error (pyre.inventory)
34 -- pylithapp.timedependent.bc.xpos.label <- 'boundary_xpos'
35 -- unknown component 'pylithapp.timedependent.bc.xpos'
36 >> ./pylithapp.cfg:136:
37 -- error (pyre.inventory)
38 -- pylithapp.timedependent.bc.xpos.label_value <- '11'
39 -- unknown component 'pylithapp.timedependent.bc.xpos'
40 >> ./pylithapp.cfg:137:
41 -- error (pyre.inventory)

```

(continues on next page)

(continued from previous page)

```

42 -- pylithapp.timedependent.bc.xpos.constrained_dof <- '[0]'
43 -- unknown component 'pylithapp.timedependent.bc.xpos'
44 >> ./pylithapp.cfg:138:
45 -- error (pyre.inventory)
46 -- pylithapp.timedependent.bc.xpos.db_auxiliary_field <- 'pylith.bc.ZeroDB'
47 -- unknown component 'pylithapp.timedependent.bc.xpos'
48 >> ./pylithapp.cfg:139:
49 -- error (pyre.inventory)
50 -- pylithapp.timedependent.bc.xpos.db_auxiliary_field.description <- 'Dirichlet BC +x_
    ↪ edge'
51 -- unknown component 'pylithapp.timedependent.bc.xpos.db_auxiliary_field'
52 >> ./pylithapp.cfg:141:
53 -- error (pyre.inventory)
54 -- pylithapp.timedependent.bc.xpos.auxiliary_subfields.initial_amplitude.basis_order <-
    ↪ '0'
55 -- unknown component 'pylithapp.timedependent.bc.xpos.auxiliary_subfields.initial_
    ↪ amplitude'
56 usage: pylith [--<property>=<value>] [--<facility>.<property>=<value>] [FILE.cfg] ...
57 component 'pylithapp'
58   properties: dump_parameters, help, help-components, help-persistence, help-
    ↪ properties, include-citations, initialize_only, job, launcher, metadata, nodes, petsc,
    ↪ problem, scheduler, show_application_flow, start_python_debugger, typos, weaver
59   facilities: dump_parameters, job, launcher, metadata, petsc, problem, scheduler, weaver
60 For more information:
61   --help-properties: prints details about user settable properties
62   --help-components: prints details about user settable facilities and components
63 pylithapp: configuration error(s)

```

Troubleshooting Strategy

The full error message shows several errors associated with our parameter files. Because errors often cascade, the best place to start is to address the first error (lines 3-6). We see that the default name for `problem_defaults` is empty and PyLith requires one. This name will be used as the first part of the filename for output. We resolve this error by examining the parameters for `pylithapp.timedependent.problem_defaults` in `step01_gravity.cfg`. We see that we do not set any of the problem defaults.

Resolution

We must set the simulation name in the problem defaults.

Listing 4.297: Correct error in step01a_gravity.cfg.

```
[pylithapp]
...
# Set the name of the problem that will be used to construct the
# output filenames. The default directory for output is 'output'.
problem.defaults.name = step01a_gravity
```

Step 1: Error 2**Error Message**

Listing 4.298: Error message 2 when running Step 1.

```
1 $ pylith step01a_gravity.cfg
2
3 # Output
4 >> {default}::
5 -- error (pyre.inventory)
6 -- timedependent.bc.dirichlettimedependent.label <- ''
7 -- Label for boundary condition group/nodeset/pset in mesh not specified.
8 >> {default}::
9 -- error (pyre.inventory)
10 -- timedependent.bc.dirichlettimedependent.simplifiedb.description <- ''
11 -- Description for spatial database not specified.
12 >> {default}::
13 -- error (pyre.inventory)
14 -- timedependent.bc.dirichlettimedependent.simplifiedb.simpleioascii.filename <- ''
15 -- Filename for spatial database not specified.
16 >> {default}::
17 -- error (pyre.inventory)
18 -- timedependent.bc.dirichlettimedependent.constrained_dof <- '[]'
19 -- No constrained degrees of freedom found for time-dependent Dirichlet boundary_
   ↳ condition 'bc_xpos'. 'constrained_dof' must be a zero-based integer array (0=x, 1=y,
   ↳ 2=z).
20 >> ./pylithapp.cfg:112:
21 -- error (pyre.inventory)
22 -- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.bulk_modulus.basis_
   ↳ order <- '0'
23 -- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
   ↳ bulk_modulus'
24 >> ./pylithapp.cfg:113:
25 -- error (pyre.inventory)
26 -- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.shear_modulus.basis_
   ↳ order <- '0'
27 -- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
   ↳ shear_modulus'
28 >> ./pylithapp.cfg:135:
29 -- error (pyre.inventory)
30 -- pylithapp.timedependent.bc.xpos.label <- 'boundary_xpos'
31 -- unknown component 'pylithapp.timedependent.bc.xpos'
32 >> ./pylithapp.cfg:136:
```

(continues on next page)

(continued from previous page)

```

33 -- error (pyre.inventory)
34 -- pylithapp.timedependent.bc.xpos.label_value <- '11'
35 -- unknown component 'pylithapp.timedependent.bc.xpos'
36 >> ./pylithapp.cfg:137:
37 -- error (pyre.inventory)
38 -- pylithapp.timedependent.bc.xpos.constrained_dof <- '[0]'
39 -- unknown component 'pylithapp.timedependent.bc.xpos'
40 >> ./pylithapp.cfg:138:
41 -- error (pyre.inventory)
42 -- pylithapp.timedependent.bc.xpos.db_auxiliary_field <- 'pylith.bc.ZeroDB'
43 -- unknown component 'pylithapp.timedependent.bc.xpos'
44 >> ./pylithapp.cfg:139:
45 -- error (pyre.inventory)
46 -- pylithapp.timedependent.bc.xpos.db_auxiliary_field.description <- 'Dirichlet BC +x_
    ↪edge'
47 -- unknown component 'pylithapp.timedependent.bc.xpos.db_auxiliary_field'
48 >> ./pylithapp.cfg:141:
49 -- error (pyre.inventory)
50 -- pylithapp.timedependent.bc.xpos.auxiliary_subfields.initial_amplitude.basis_order <-
    ↪'0'
51 -- unknown component 'pylithapp.timedependent.bc.xpos.auxiliary_subfields.initial_
    ↪amplitude'
52 usage: pylith [--<property>=<value>] [--<facility>.<property>=<value>] [FILE.cfg] ...
53 component 'pylithapp'
54   properties: dump_parameters, help, help-components, help-persistence, help-
    ↪properties, include-citations, initialize_only, job, launcher, metadata, nodes, petsc,
    ↪problem, scheduler, show_application_flow, start_python_debugger, typos, weaver
55   facilities: dump_parameters,job,launcher,metadata,petsc,problem,scheduler,weaver
56 For more information:
57   --help-properties: prints details about user settable properties
58   --help-components: prints details about user settable facilities and components
59 pylithapp: configuration error(s)

```

Troubleshooting Strategy

The full error message shows several errors associated with our parameter files. Because errors often cascade, the best place to start is to address the first error (lines 3-6). This does not give us anything specific, so we look further down the output until we find an error for a specific component. At lines 15-18, we find that an error is associated with the `bc_xpos` boundary condition. We look for this section in the parameter files and find it in `pylithapp.cfg` and notice the name of the boundary condition is missing `bc_`.

Resolution

Listing 4.299: Correct error in pylithapp.cfg.

```
# Error
[pylithapp.timedependent.bc.xpos]

# Correct
[pylithapp.timedependent.bc.bc_xpos]
```

Step 1: Error 3

Error Message

Listing 4.300: Error message 3 when running Step 1.

```
1 $ pylith step01a_gravity.cfg
2
3 # Output
4 >> ./pylithapp.cfg:112:
5 -- error (pyre.inventory)
6 -- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.bulk_modulus.basis_
   ↳ order <- '0'
7 -- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
   ↳ bulk_modulus'
8 >> ./pylithapp.cfg:113:
9 -- error (pyre.inventory)
10 -- pylithapp.timedependent.materials.elasticity.auxiliary_subfields.shear_modulus.basis_
   ↳ order <- '0'
11 -- unknown component 'pylithapp.timedependent.materials.elasticity.auxiliary_subfields.
   ↳ shear_modulus'
12 usage: pylith [--<property>=<value>] [--<facility>.<property>=<value>] [FILE.cfg] ...
13 component 'pylithapp'
14   properties: dump_parameters, help, help-components, help-persistence, help-
   ↳ properties, include-citations, initialize_only, job, launcher, metadata, nodes, petsc,
   ↳ problem, scheduler, show_application_flow, start_python_debugger, typos, weaver
15   facilities: dump_parameters, job, launcher, metadata, petsc, problem, scheduler, weaver
16 For more information:
17   --help-properties: prints details about user settable properties
18   --help-components: prints details about user settable facilities and components
19 pylithapp: configuration error(s)
```

Troubleshooting Strategy

Lines 3-6 indicate that we are trying to set the basis order for component `pylithapp.timedependent.materials.elasticity.auxiliary_subfields` that does not exist. If we look up the documentation for components [Elasticity](#) and [IsotropicLinearElasticity](#), then we see that the `bulk_modulus` auxiliary subfield is a subfield of the bulk rheology, not of the material.

Resolution

Listing 4.301: Correct error in pylithapp.cfg.

```
[pylithapp.problem.materials.wedge]
...
bulk_rheology.auxiliary_subfields.bulk_modulus.basis_order = 0
bulk_rheology.auxiliary_subfields.shear_modulus.basis_order = 0
```

Step 1: Error 4**Error Message**

Listing 4.302: Error message 4 when running Step 1.

```
1 $ pylith step01a_gravity.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
  ↳ PetscOptions::write(pythia::journal::info_t&, const char*, const
  ↳ pylith::utils::PetscOptions&)
8 -- info (application-flow)
9 -- Setting PETSc options:
10     ksp_atol = 1.0e-7
11     ksp_converged_reason = true
12     ksp_error_if_not_converged = true
13     ksp_gmres_restart = 100
14     ksp_guess_pod_size = 8
15     ksp_guess_type = pod
16     ksp_rtol = 1.0e-14
17     mg_fine_ksp_max_it = 5
18     pc_type = gamg
19     snes_atol = 5.0e-7
20     snes_converged_reason = true
21     snes_error_if_not_converged = true
22     snes_monitor = true
23     snes_rtol = 1.0e-14
24     ts_error_if_step_fails = true
25     ts_exact_final_time = matchstep
26     ts_monitor = true
27     ts_type = beuler
28     viewer_hdf5_collective = true
29 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void
  ↳ pylith::meshio::MeshIOPetsc::_read()
30 -- info (application-flow)
31 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
  ↳ pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Domain bounding box:
35     (-1000000, 1000000)
```

(continues on next page)

(continued from previous page)

```

36     (-1000000, 0)
37 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
    ↪ pylith::problems::TimeDependent::verifyConfiguration() const
38 -- info (application-flow)
39 -- Component 'timedependent.problem': Verifying problem configuration.
40 >> src/cig/pylith/libsrc/pylith/topology/MeshOps.cc:802:static void
    ↪ pylith::topology::MeshOps::checkMaterialLabels(const pylith::topology::Mesh&,
    ↪ pylith::int_array&)
41 -- error (user-input)
42 -- Material label_value '3' for cell '609' does not match the label_value of any
    ↪ materials or interfaces.
43 Fatal error. Calling MPI_Abort() to abort PyLith application.
44 Traceback (most recent call last):
45   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication.
    ↪ py", line 55, in onComputeNodes
46     self.main(*args, **kws)
47   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py",
    ↪ line 84, in main
48     self.problem.verifyConfiguration()
49   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py",
    ↪ line 206, in verifyConfiguration
50     ModuleProblem.verifyConfiguration(self)
51   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/problems.py",
    ↪ line 162, in verifyConfiguration
52     return _problems.Problem_verifyConfiguration(self)
53     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
54 RuntimeError: Material label_value '3' for cell '609' does not match the label_value of
    ↪ any materials or interfaces.
55 C++ traceback (9 frames):
56   [0] software/pylith-debug/lib/libpylith.so.0(_
    ↪ ZN6pylith8topology7MeshOps19checkMaterialLabelsERKNS0_4MeshERSt8valarrayIiE+0x7b9)
    ↪ [0x74544c5780c7]
57   [1] software/pylith-debug/lib/libpylith.so.0(_ZNK6pylith8problems7Problem20_
    ↪ checkMaterialLabelsEv+0x4f6) [0x74544c51e69e]
58   [2] software/pylith-debug/lib/libpylith.so.0(_
    ↪ ZNK6pylith8problems7Problem19verifyConfigurationEv+0x552) [0x74544c51d046]
59   [3] software/pylith-debug/lib/libpylith.so.0(_
    ↪ ZNK6pylith8problems13TimeDependent19verifyConfigurationEv+0x253) [0x74544c52c8e5]
60   [4] software/pylith-debug/lib/python3.12/site-packages/pylith/problems/_problems.
    ↪ so(+0x15af4) [0x745445fd8af4]
61   [5] software/pylith-debug/bin/mpinemesi(+0x15a51) [0x612c79077a51]
62   [6] /lib/x86_64-linux-gnu/libc.so.6(+0x29d90) [0x745468c29d90]
63   [7] /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0x80) [0x745468c29e40]
64   [8] software/pylith-debug/bin/mpinemesi(+0x5805) [0x612c79067805]
65
66 -----
67 MPI_ABORT was invoked on rank 0 in communicator MPI_COMM_WORLD
68   Proc: [[1354,1],0]
69   Errorcode: -1
70
71 NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
72 You may or may not see output from other processes, depending on

```

(continues on next page)

(continued from previous page)

```

73 exactly when Open MPI kills them.
74 -----
75 -----
76 prterun has exited due to process rank 0 with PID 0 on node igskci164warlng calling
77 "abort". This may have caused other processes in the application to be
78 terminated by signals sent by prterun (as reported here).
79 -----
80 software/pylith-debug/bin/nemesis: mpiexec: exit 255
81 software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesis: exit 1

```

Troubleshooting Strategy

The output shows a Python Traceback and then the error message on Line 52. The error indicates the finite-element mesh file contains a cell with a label value of 3, but the parameter files do not have a material with a label value of 3. We examine the `pylithapp.problem.materials` sections of `pylithapp.cfg` and see that the label values are 0, 1, and 2 in the parameter file rather than 1, 2, and 3 that we set in the Gmsh Python script.

Resolution

Listing 4.303: Correct error in `pylithapp.cfg`.

```

[pylithapp.problem.materials.slabs]
label_value = 1
...

[pylithapp.problem.materials.crust]
label_value = 2
...

[pylithapp.problem.materials.wedge]
label_value = 3
...

```

Step 1: Error 5

Error Message

Listing 4.304: Error message 5 when running Step 1.

```

1 $ pylith step01a_gravity.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
  ↪ PetscOptions::write(pythia::journal::info_t&, const char*, const_
  ↪ pylith::utils::PetscOptions&)
8 -- info (application-flow)
9 -- Setting PETSc options:
10    ksp_atol = 1.0e-7

```

(continues on next page)

(continued from previous page)

```

11     ksp_converged_reason = true
12     ksp_error_if_not_converged = true
13     ksp_gmres_restart = 100
14     ksp_guess_pod_size = 8
15     ksp_guess_type = pod
16     ksp_rtol = 1.0e-14
17     mg_fine_ksp_max_it = 5
18     pc_type = gamg
19     snes_atol = 5.0e-7
20     snes_converged_reason = true
21     snes_error_if_not_converged = true
22     snes_monitor = true
23     snes_rtol = 1.0e-14
24     ts_error_if_step_fails = true
25     ts_exact_final_time = matchstep
26     ts_monitor = true
27     ts_type = beuler
28     viewer_hdf5_collective = true
29 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void
    ↪ pylith::meshio::MeshIOPetsc::_read()
30 -- info (application-flow)
31 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
    ↪ pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Domain bounding box:
35     (-100000, 100000)
36     (-100000, 0)
37 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
    ↪ pylith::problems::TimeDependent::verifyConfiguration() const
38 -- info (application-flow)
39 -- Component 'timedependent.problem': Verifying problem configuration.
40 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
    ↪ printInfo
41 -- info (application-flow)
42 -- Scales for nondimensionalization:
43     Length scale: 5000*m
44     Displacement scale: 1*m
45     Time scale: 3.15576e+09*s
46     Rigidity scale: 1e+10*m**-1*kg*s**-2
47     Temperature scale: 1*K
48 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
    ↪ pylith::problems::TimeDependent::initialize()
49 -- info (application-flow)
50 -- Component 'timedependent.problem': Initializing problem.
51 [0]PETSC ERROR: ----- Error Message -----
    ↪ -----
52 [0]PETSC ERROR: Error in external library
53 [0]PETSC ERROR: Error converting spatial database values for gravitational_acceleration
    ↪ at ( -92285.3 -72772.7) in spatial database 'Gravity field'. Found near zero
    ↪ magnitude (0) for gravity field vector ( 0 0).
54 [0]PETSC ERROR: WARNING! There are unused option(s) set! Could be the program crashed

```

(continues on next page)

(continued from previous page)

```

→before usage or a spelling mistake, etc!
55 [0]PETSC ERROR: Option left: name:-ksp_atol value: 1.0e-7 source: code
56 [0]PETSC ERROR: Option left: name:-ksp_converged_reason (no value) source: code
57 [0]PETSC ERROR: Option left: name:-ksp_error_if_not_converged (no value) source: code
58 [0]PETSC ERROR: Option left: name:-ksp_gmres_restart value: 100 source: code
59 [0]PETSC ERROR: Option left: name:-ksp_guess_pod_size value: 8 source: code
60 [0]PETSC ERROR: Option left: name:-ksp_guess_type value: pod source: code
61 [0]PETSC ERROR: Option left: name:-ksp_rtol value: 1.0e-14 source: code
62 [0]PETSC ERROR: Option left: name:-mg_fine_ksp_max_it value: 5 source: code
63 [0]PETSC ERROR: Option left: name:-pc_type value: gamg source: code
64 [0]PETSC ERROR: Option left: name:-snes_atol value: 5.0e-7 source: code
65 [0]PETSC ERROR: Option left: name:-snes_converged_reason (no value) source: code
66 [0]PETSC ERROR: Option left: name:-snes_error_if_not_converged (no value) source: code
67 [0]PETSC ERROR: Option left: name:-snes_monitor (no value) source: code
68 [0]PETSC ERROR: Option left: name:-snes_rtol value: 1.0e-14 source: code
69 [0]PETSC ERROR: Option left: name:-ts_error_if_step_fails (no value) source: code
70 [0]PETSC ERROR: Option left: name:-ts_exact_final_time value: matchstep source: code
71 [0]PETSC ERROR: Option left: name:-ts_monitor (no value) source: code
72 [0]PETSC ERROR: Option left: name:-ts_type value: beuler source: code
73 [0]PETSC ERROR: Option left: name:-viewer_hdf5_collective (no value) source: code
74 [0]PETSC ERROR: See https://petsc.org/release/faq/ for trouble shooting.
75 [0]PETSC ERROR: PETSc Development Git Revision: v3.25.1-168-g70613f6caab Git Date: 2026-
→05-21 15:15:28 +0000
76 [0]PETSC ERROR: software/pylith-debug/bin/mpinemesis with 1 MPI process(es) and PETSC_
→ARCH arch-pylith-debug on igskci164warlng.gs.doi.net by baagaard Tue May 26 11:31:47
→2026
77 [0]PETSC ERROR: Configure options: --PETSC_ARCH=arch-pylith-debug --with-debugging=1 --
→with-clanguage=c --with-mpi-compilers=1 --with-shared-libraries=1 --with-64-bit-
→points=1 --with-large-file-io=1 --with-lgrind=0 --download-parmetis=1 --download-
→metis=1 --download-triangle --download-ml=1 --download-superlu=1 --with-fc=0 --
→download-f2cblaslapack --with-hdf5=1 --with-hdf5-dir=software/pylith-debug --with-
→zlib=1 CFLAGS+=-g
78 [0]PETSC ERROR: #1 static PetscErrorCode
→pylith::topology::FieldQuery::queryDBPointFn(PylithInt, PylithReal, const PylithReal*,
→PylithInt, PylithScalar*, void*)() at src/cig/pylith/libsrc/pylith/topology/FieldQuery.
→cc:334
79 [0]PETSC ERROR: #2 DMProjectPoint_Func_Private() at /software/baagaard/petsc-dev/src/dm/
→impls/plex/plexproject.c:143
80 [0]PETSC ERROR: #3 DMProjectPoint_Private() at /software/baagaard/petsc-dev/src/dm/impls/
→plex/plexproject.c:545
81 [0]PETSC ERROR: #4 DMProjectLocal_Generic_Plex() at /software/baagaard/petsc-dev/src/dm/
→impls/plex/plexproject.c:1083
82 [0]PETSC ERROR: #5 DMProjectFunctionLocal_Plex() at /software/baagaard/petsc-dev/src/dm/
→impls/plex/plexproject.c:1114
83 [0]PETSC ERROR: #6 DMProjectFunctionLocal() at /software/baagaard/petsc-dev/src/dm/
→interface/dm.c:8361
84 [0]PETSC ERROR: #7 void pylith::topology::FieldQuery::queryDB()() at src/cig/pylith/
→libsrc/pylith/topology/FieldQuery.cc:227
85 Fatal error. Calling MPI_Abort() to abort PyLith application.
86 Traceback (most recent call last):
87 File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication.
→py", line 55, in onComputeNodes

```

(continues on next page)

(continues on next page)

(continued from previous page)

```
126 software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesis: exit 1
```

Troubleshooting Strategy

During initialization of the time-dependent problem, PyLith encountered an error while calling PETSc functions. When an error occurs within a PETSc routine, PETSc dumps a lot of information to the screen. First, it displays the error message; second, it displays configuration information; third, it displays the stacktrace. PyLith traps the PETSc error and displays the Python Traceback. In this case, the error message on lines 8-9 indicate that the gravity field vector has a magnitude of zero. We notice that it shows only the first two components because this is a 2D problem, whereas the default is (0, 0, -1) and intended for 3D problems.

Resolution

Listing 4.305: Correct error in step01_gravity.cfg.

```
[pylithapp.problem]
gravity_field = spatialdata.spatialdb.GravityField
gravity_field.gravity_dir = [0.0, -1.0, 0.0]
```

Our simulation now runs without errors and the output looks correct.

Step 1: Successful run

Listing 4.306: Output when running Step 1.

```
1 $ pylith step01a_gravity.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/utils/PetscOptions.cc:251:static void pylith::utils::_
   ↪ PetscOptions::write(pythia::journal::info_t&, const char*, const_
   ↪ pylith::utils::PetscOptions&)
8 -- info (application-flow)
9 -- Setting PETSc options:
10     ksp_atol = 1.0e-7
11     ksp_converged_reason = true
12     ksp_error_if_not_converged = true
13     ksp_gmres_restart = 100
14     ksp_guess_pod_size = 8
15     ksp_guess_type = pod
16     ksp_rtol = 1.0e-14
17     mg_fine_ksp_max_it = 5
18     pc_type = gamg
19     snes_atol = 5.0e-7
20     snes_converged_reason = true
21     snes_error_if_not_converged = true
22     snes_monitor = true
23     snes_rtol = 1.0e-14
24     ts_error_if_step_fails = true
```

(continues on next page)

(continued from previous page)

```

25     ts_exact_final_time = matchstep
26     ts_monitor = true
27     ts_type = beuler
28     viewer_hdf5_collective = true
29 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void_
    ↪ pylith::meshio::MeshIOPetsc::_read()
30 -- info (application-flow)
31 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
    ↪ pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Domain bounding box:
35     (-1000000, 1000000)
36     (-1000000, 0)
37 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void_
    ↪ pylith::problems::TimeDependent::verifyConfiguration() const
38 -- info (application-flow)
39 -- Component 'timedependent.problem': Verifying problem configuration.
40 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
    ↪ printInfo
41 -- info (application-flow)
42 -- Scales for nondimensionalization:
43     Length scale: 5000*m
44     Displacement scale: 1*m
45     Time scale: 3.15576e+09*s
46     Rigidity scale: 1e+10*m**-1*kg*s**-2
47     Temperature scale: 1*K
48 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void_
    ↪ pylith::problems::TimeDependent::initialize()
49 -- info (application-flow)
50 -- Component 'timedependent.problem': Initializing problem.
51 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
    ↪ pylith::problems::TimeDependent::solve()
52 -- info (application-flow)
53 -- Component 'timedependent.problem': Solving equations.
54 0 TS dt 0.001 time 0.
55     0 SNES Function norm 3.007829881319e+03
56     Linear solve converged due to CONVERGED_ATOL iterations 11
57     1 SNES Function norm 3.524261285349e-07
58     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
59 1 TS dt 0.001 time 0.001
60 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
    ↪ py:222:finalize
61 -- info (application-flow)
62 -- Finalizing problem.

```

Our simulation now runs without errors and the output looks correct.

Step 6: Slip on Two Faults

In this example we include coseismic prescribed slip on both the main and splay faults. We specify 2 meters of reverse slip on the main fault and 1 meter of reverse slip on the splay fault. Fig. 4.182 shows the boundary conditions on the domain. The parameters specific to this example are in `step06_twofaults.cfg`.

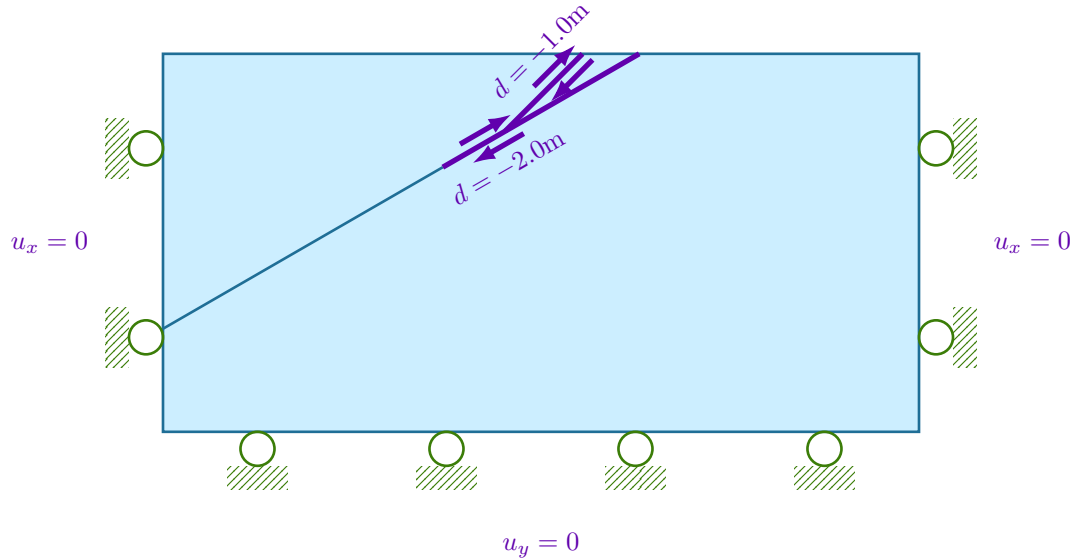


Fig. 4.182: Boundary conditions for static coseismic slip on both the main and splay faults. We prescribe 2 meters of reverse slip on the main fault with 1 meter of reverse slip on the splay fault. We use roller boundary conditions on the lateral sides and bottom of the domain.

See *examples/reverse-2d Step 6* for more information.

Step 6: Error 1

In troubleshooting Step 1 we resolved all of the errors associated with `pylithapp.cfg`. In this example, we expect all errors to be associated with inputs files specific to Step 6, such as `step06_twofaults.cfg`.

Error Message

Listing 4.307: Error message 1 when running Step 6.

```

1  $ pylith step06_twofaults.cfg
2
3  # Output
4  >> {default}::
5  -- error (pyre.inventory)
6  -- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.
   ↳ description <- ''
7  -- Description for spatial database not specified.
8  >> {default}::
9  -- error (pyre.inventory)
10 -- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.
   ↳ simpleioascii.filename <- ''
11 -- Filename for spatial database not specified.
```

(continues on next page)

(continued from previous page)

```

12 >> step06_twofaults.cfg:81:
13 -- error (pyre.inventory)
14 -- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.filename
15 ↪<- 'fault_slip.spatialdb'
16 -- unrecognized property 'timedependent.interfaces.faultcohesivekin.singlerupture.
17 ↪kinsrcstep.simplifiedb.filename'
18 usage: pylith [--<property>=<value>] [--<facility>.<property>=<value>] [FILE.cfg] ...
19 component 'pylithapp'
20   properties: dump_parameters, help, help-components, help-persistence, help-
21 ↪properties, include-citations, initialize_only, job, launcher, metadata, nodes, petsc,
22 ↪problem, scheduler, show_application_flow, start_python_debugger, typos, weaver
23   facilities: dump_parameters, job, launcher, metadata, petsc, problem, scheduler, weaver
For more information:
  --help-properties: prints details about user settable properties
  --help-components: prints details about user settable facilities and components
pylithapp: configuration error(s)

```

Troubleshooting Strategy

We have several configuration errors, so we start with the first one at lines 3-6. Spatial databases require a description, and PyLith cannot find one for `timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb`. We examine the fault parameters for `step06_twofaults.cfg` and see that `db_auxiliary_field` for the earthquake rupture is missing the description.

Resolution

Listing 4.308: Correct error in `step06_twofaults.cfg`.

```

[pylithapp.problem.interfaces.fault.eq_ruptures.rupture]
...
db_auxiliary_field.description = Fault rupture for main fault

```

Step 6: Error 2

Error Message

Listing 4.309: Error message 2 when running Step 6.

```

1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> {default}::
5 -- error (pyre.inventory)
6 -- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.
7 ↪simpleioascii.filename <- ''
8 -- Filename for spatial database not specified.
9 >> step06_twofaults.cfg:84:
10 -- error (pyre.inventory)
11 -- timedependent.interfaces.faultcohesivekin.singlerupture.kinsrcstep.simplifiedb.filename
12 ↪<- 'fault_slip.spatialdb'
13 -- unrecognized property 'timedependent.interfaces.faultcohesivekin.singlerupture.

```

(continues on next page)

(continued from previous page)

```

12 ↪knsrsrcstep.simplesdb.filename'
usage: pylith [--<property>=<value>] [--<facility>.<property>=<value>] [FILE.cfg] ...
13 component 'pylithapp'
14   properties: dump_parameters, help, help-components, help-persistence, help-
15 ↪properties, include-citations, initialize_only, job, launcher, metadata, nodes, petsc,
16 ↪problem, scheduler, show_application_flow, start_python_debugger, typos, weaver
17   facilities: dump_parameters, job, launcher, metadata, petsc, problem, scheduler, weaver
18 For more information:
19   --help-properties: prints details about user settable properties
   --help-components: prints details about user settable facilities and components
pylithapp: configuration error(s)

```

Troubleshooting Strategy

We still have problem configuration errors. The filename for the SimpleIOAscii reader for a SimpleDB is missing. We examine the earthquake rupture parameters in `step06_twofaults.cfg` again and see that the filename is associated with the SimpleDB; however, if we look at the [SimpleDB documentation](#) we notice that it has a reader SimpleIOAscii, which has a filename.

Resolution

Listing 4.310: Correct error in `step06_twofaults.cfg`.

```

# Error
db_auxiliary_field.filename = fault_slip.spatialdb

# Correct
db_auxiliary_field.iohandler.filename = fault_slip.spatialdb

```

Step 6: Error 3

Error Message

Listing 4.311: Error message 3 when running Step 6.

```

1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
8 ↪PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
9 -- info (application-flow)
10 -- Setting PETSc options:
11   dm_reorder_section = true
12   dm_reorder_section_type = cohesive
13   ksp_atol = 1.0e-7
14   ksp_converged_reason = true
15   ksp_error_if_not_converged = true
   ksp_gmres_restart = 100

```

(continues on next page)

(continued from previous page)

```

16     ksp_guess_pod_size = 8
17     ksp_guess_type = pod
18     ksp_rtol = 1.0e-14
19     mg_fine_ksp_max_it = 5
20     mg_fine_pc_type = vpbjacobi
21     pc_type = gamg
22     snes_atol = 5.0e-7
23     snes_converged_reason = true
24     snes_error_if_not_converged = true
25     snes_monitor = true
26     snes_rtol = 1.0e-14
27     ts_error_if_step_fails = true
28     ts_exact_final_time = matchstep
29     ts_monitor = true
30     ts_type = beuler
31     viewer_hdf5_collective = true
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void
    ↪ pylith::meshio::MeshIOPetsc::_read()
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
35 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
    ↪ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
36 -- info (application-flow)
37 -- Component 'meshiopetsc.reader': Domain bounding box:
38     (-1000000, 1000000)
39     (-1000000, 0)
40 >> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:51:virtual
    ↪ pylith::topology::Mesh
    ↪ *pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh *, const
    ↪ pylith::problems::Problem &)
41 -- info (application-flow)
42 -- Inserting cohesive cells.
43 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
    ↪ pylith::problems::TimeDependent::verifyConfiguration() const
44 -- info (application-flow)
45 -- Component 'timedependent.problem': Verifying problem configuration.
46 >> src/cig/pylith/libsrc/pylith/topology/FieldOps.cc:224:static void
    ↪ pylith::topology::FieldOps::checkSubfieldsExist(const pylith::string_vector &, const
    ↪ std::string &, const pylith::topology::Field &)
47 -- error (user-input)
48 -- Could not find 'lagrange_multiplier_fault' in domain solution field. Field contains:
    ↪ 'displacement'; the missing fields are required for interface 'FaultCohesiveKin'.
49 Fatal error. Calling MPI_Abort() to abort PyLith application.
50 Traceback (most recent call last):
51   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication.
    ↪ py", line 55, in onComputeNodes
52     self.main(*args, **kws)
53   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py",
    ↪ line 84, in main
54     self.problem.verifyConfiguration()
55   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py",
    ↪ line 206, in verifyConfiguration

```

(continues on next page)

(continued from previous page)

```

56     ModuleProblem.verifyConfiguration(self)
57     File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/problems.py",
↳ line 162, in verifyConfiguration
58         return _problems.Problem.verifyConfiguration(self)
59         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
60 RuntimeError: Could not find 'lagrange_multiplier_fault' in domain solution field. Field
↳ contains: 'displacement'; the missing fields are required for interface
↳ 'FaultCohesiveKin'.
61 C++ traceback (8 frames):
62 [0] 3  libpylith.0.dylib                0x00000000108002e40 _
↳ ZN6pylith10ValueErrorCI1NS_5ErrorMessageE + 36
63 [1] 4  libpylith.0.dylib                0x0000000010824f548 _
↳ ZN6pylith8topology8FieldOps19checkSubfieldsExistERKNSt3__16vectorINS2_12basic_
↳ stringIcNS2_11char_traitsIcEENS2_9allocatorIcEEEEENS7_IS9_EEEERKS9_RKNS0_5FieldE + 1212
64 [2] 5  libpylith.0.dylib                0x0000000010803dc14 _
↳ ZNK6pylith6faults16FaultCohesiveKin19verifyConfigurationERKNS_8topology5FieldE + 648
65 [3] 6  libpylith.0.dylib                0x000000001081d8068 _
↳ ZNK6pylith8problems7Problem19verifyConfigurationEv + 1024
66 [4] 7  libpylith.0.dylib                0x000000001081e8f4c _
↳ ZNK6pylith8problems13TimeDependent19verifyConfigurationEv + 548
67 [5] 8  _problems.so                    0x0000000010895e218 _ZL33_wrap_Problem_
↳ verifyConfigurationP7_objectS0_ + 216
68 [6] 42 mpinemesis                     0x00000000100b91d70 main + 616
69 [7] 43 dyld                          0x00000000197e0eb98 start + 6076
70
71 Abort(-1) on node 0 (rank 0 in comm 0): application called MPI_Abort(MPI_COMM_WORLD, -1)
↳ - process 0
72 software/pylith-debug/bin/nemesis: mpiexec: exit 255
73 software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesis: exit 1

```

Troubleshooting Strategy

After the Python Traceback, we see the error message on lines 64-65. The faults check to make sure the solution field contains the necessary subfields, and the fault cannot find the `lagrange_multiplier_fault` subfield. The easiest way to diagnose an error like this is to view the JSON file automatically generated by PyLith; it contains all of the parameters, including any defaults used. We point our web browser to https://geodynamics.github.io/pylith_parameters/ and load the parameter file `output/step06_twofaults-parameters.json`. In the left panel we navigate to the solution field and see that the subfields are set to `pylith.problems.SolnDisp`, so that the solution field only contains a single subfield, `displacement`. We want the solution field to contain both `displacement` and `lagrange_multiplier_fault`.

Resolution

Listing 4.312: Correct error in step06_twofaults.cfg.

```
[pylithapp.problem]
solution = pylith.problems.SolnDispLagrange
```

Step 6: Error 4

Error Message

Listing 4.313: Error message 4 when running Step 6.

```
1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/utils/PetscOptions.cc:251:static void pylith::utils::_
8 ↪PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
9 -- info (application-flow)
10 -- Setting PETSc options:
11     dm_reorder_section = true
12     dm_reorder_section_type = cohesive
13     ksp_atol = 1.0e-7
14     ksp_converged_reason = true
15     ksp_error_if_not_converged = true
16     ksp_gmres_restart = 100
17     ksp_guess_pod_size = 8
18     ksp_guess_type = pod
19     ksp_rtol = 1.0e-14
20     mg_fine_ksp_max_it = 5
21     mg_fine_pc_type = vpbjacobi
22     pc_type = gamg
23     snes_atol = 5.0e-7
24     snes_converged_reason = true
25     snes_error_if_not_converged = true
26     snes_monitor = true
27     snes_rtol = 1.0e-14
28     ts_error_if_step_fails = true
29     ts_exact_final_time = matchstep
30     ts_monitor = true
31     ts_type = beuler
32     viewer_hdf5_collective = true
33 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void_
34 ↪pylith::meshio::MeshIOPetsc::_read()
35 -- info (application-flow)
36 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
37 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
38 ↪pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
-- Component 'meshiopetsc.reader': Domain bounding box:
(-100000, 100000)
```

(continues on next page)

(continued from previous page)

```

39     (-1000000, 0)
40 >> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:51:virtual
    ↪ pylith::topology::Mesh
    ↪ *pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh *, const
    ↪ pylith::problems::Problem &)
41 -- info (application-flow)
42 -- Inserting cohesive cells.
43 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
    ↪ pylith::problems::TimeDependent::verifyConfiguration() const
44 -- info (application-flow)
45 -- Component 'timedependent.problem': Verifying problem configuration.
46 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
    ↪ printInfo
47 -- info (application-flow)
48 -- Scales for nondimensionalization:
49     Length scale: 2500*m
50     Displacement scale: 1*m
51     Time scale: 3.15576e+09*s
52     Rigidity scale: 1e+10*m**-1*kg*s**-2
53     Temperature scale: 1*K
54 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
    ↪ pylith::problems::TimeDependent::initialize()
55 -- info (application-flow)
56 -- Component 'timedependent.problem': Initializing problem.
57 Fatal error. Calling MPI_Abort() to abort PyLith application.
58 Traceback (most recent call last):
59   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication.
    ↪ py", line 55, in onComputeNodes
60     self.main(*args, **kwds)
61   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py",
    ↪ line 85, in main
62     self.problem.initialize()
63   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py",
    ↪ line 212, in initialize
64     ModuleProblem.initialize(self)
65   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/problems.py",
    ↪ line 165, in initialize
66     return _problems.Problem_initialize(self)
67     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
68 RuntimeError: Error occurred while reading spatial database file 'fault_slip.spatialdb'.
69 Read data for 3 out of 4 points.
70 Error reading coordinates from buffer ''.
71 Abort(-1) on node 0 (rank 0 in comm 0): application called MPI_Abort(MPI_COMM_WORLD, -1)
    ↪ - process 0
72 software/pylith-debug/bin/nemesis: mpiexec: exit 255
73 software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesis: exit 1

```

Troubleshooting Strategy

The error message on lines 97-98 indicates there is an error reading the `fault_slip.spatialdb` spatial database for the fault slip. PyLith was able to read data for 3 of 4 points. The file `fault_slip.spatialdb` contains only 3 points but `num-locs` is 4.

Resolution

Listing 4.314: Correct error in `fault_slip.spatialdb`.

```
# Error
num-locs = 4

# Correct
num-locs = 3
```

Step 6: Error 5

Error Message

Listing 4.315: Error message 5 when running Step 6.

```
1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
8 ↪PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
9 -- info (application-flow)
10 -- Setting PETSc options:
11     dm_reorder_section = true
12     dm_reorder_section_type = cohesive
13     ksp_atol = 1.0e-7
14     ksp_converged_reason = true
15     ksp_error_if_not_converged = true
16     ksp_gmres_restart = 100
17     ksp_guess_pod_size = 8
18     ksp_guess_type = pod
19     ksp_rtol = 1.0e-14
20     mg_fine_ksp_max_it = 5
21     mg_fine_pc_type = vpbjacobi
22     pc_type = gamg
23     snes_atol = 5.0e-7
24     snes_converged_reason = true
25     snes_error_if_not_converged = true
26     snes_monitor = true
27     snes_rtol = 1.0e-14
28     ts_error_if_step_fails = true
29     ts_exact_final_time = matchstep
30     ts_monitor = true
31     ts_type = beuler
32     viewer_hdf5_collective = true
33 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void_
34 ↪pylith::meshio::MeshIOPetsc::_read()
35 -- info (application-flow)
36 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
37 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
```

(continues on next page)

(continued from previous page)

```

36 ↪ pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
37 -- info (application-flow)
38 -- Component 'meshiopetsc.reader': Domain bounding box:
39   (-100000, 100000)
40   (-100000, 0)
41 >> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:51:virtual ↪
42 ↪ pylith::topology::Mesh ↪
43 ↪ *pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh *, const ↪
44 ↪ pylith::problems::Problem &)
45 -- info (application-flow)
46 -- Inserting cohesive cells.
47 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void ↪
48 ↪ pylith::problems::TimeDependent::verifyConfiguration() const
49 -- info (application-flow)
50 -- Component 'timedependent.problem': Verifying problem configuration.
51 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238: ↪
52 ↪ printInfo
53 -- info (application-flow)
54 -- Scales for nondimensionalization:
55   Length scale: 2500*m
56   Displacement scale: 1*m
57   Time scale: 3.15576e+09*s
58   Rigidity scale: 1e+10*m**-1*kg*s**-2
59   Temperature scale: 1*K
60 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void ↪
61 ↪ pylith::problems::TimeDependent::initialize()
62 -- info (application-flow)
63 -- Component 'timedependent.problem': Initializing problem.
64 >> src/cig/pylith/libsrc/pylith/topology/FieldQuery.cc:473:static void ↪
65 ↪ pylith::topology::_FieldQuery::findQueryIndices(FieldQuery::DBQueryContext *, const ↪
66 ↪ pylith::string_vector &)
67 -- error (user-input)
68 -- Could not find value 'final_slip_opening' in spatial database 'Fault rupture for main ↪
69 ↪ fault'. Available values are:
70   final-slip-left-lateral
71   final-slip-opening
72   initiation-time
73
74 Fatal error. Calling MPI_Abort() to abort PyLith application.
75 Traceback (most recent call last):
76   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication. ↪
77   ↪ py", line 55, in onComputeNodes
78     self.main(*args, **kwds)
79   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py", ↪
80   ↪ line 85, in main
81     self.problem.initialize()
82   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py", ↪
83   ↪ line 212, in initialize
84     ModuleProblem.initialize(self)
85   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/problems.py", ↪
86   ↪ line 165, in initialize
87     return _problems.Problem_initialize(self)

```

(continues on next page)

```

RuntimeError: Could not find value 'final_slip_opening' in spatial database 'Fault_
↳rupture for main fault'. Available values are:
    final-slip-left-lateral
    final-slip-opening
    initiation-time

C++ traceback (13 frames):
[0] 3  libpylith.0.dylib                                0x000000001041c2e40 _
↳ZN6pylith10ValueErrorCI1NS_5ErrorEERKNS_12ErrorMessageE + 36
[1] 4  libpylith.0.dylib                                0x0000000010441a864 _ZN6pylith8topology11_
↳FieldQuery16findQueryIndicesEPNS0_10FieldQuery14DBQueryContextERKNS_t3__16vectorINS5_
↳12basic_stringIcNS5_11char_traitsIcEENS5_9allocatorIcEEEEISA_ISC_EEEE + 1624
[2] 5  libpylith.0.dylib                                0x00000000104418ae4 _
↳ZN6pylith8topology10FieldQuery6openDBEPN11spatialdata9spatialdb9SpatialDBEd + 1720
[3] 6  libpylith.0.dylib                                0x0000000010428b090 _
↳ZN6pylith10feassemble16AuxiliaryFactory15setValuesFromDBEv + 596
[4] 7  libpylith.0.dylib                                0x00000000104216d48 _
↳ZN6pylith6faults6KinSrc10initializeERKNS_8topology5FieldERKNS_
↳6scales6ScalesEPKN11spatialdata9geocoords8CoordSysE + 1180
[5] 8  libpylith.0.dylib                                0x000000001041fef64 _
↳ZN6pylith6faults16FaultCohesiveKin20createAuxiliaryFieldERKNS_8topology5FieldERKNS2_
↳4MeshE + 2764
[6] 9  libpylith.0.dylib                                0x000000001042349bc _
↳ZN6pylith10feassemble10Integrator10initializeERKNS_8topology5FieldE + 560
[7] 10 libpylith.0.dylib                                0x00000000104255944 _
↳ZN6pylith10feassemble19IntegratorInterface10initializeERKNS_8topology5FieldE + 924
[8] 11 libpylith.0.dylib                                0x00000000104399014 _
↳ZN6pylith8problems7Problem10initializeEv + 936
[9] 12 libpylith.0.dylib                                0x000000001043a9610 _
↳ZN6pylith8problems13TimeDependent10initializeEv + 652
[10] 13 _problems.so                                    0x000000001079ce3f0 _ZL24_wrap_Problem_
↳initializeP7_objectS0_ + 216
[11] 47 mpinemesis                                       0x0000000010006e9d70 main + 616
[12] 48 dyld                                             0x00000000197e0eb98 start + 6076

Abort(-1) on node 0 (rank 0 in comm 0): application called MPI_Abort(MPI_COMM_WORLD, -1)
↳- process 0
software/pylith-debug/bin/nemesiis: mpiexec: exit 255
software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesiis: exit 1

```

Resolution

Listing 4.316: Correct error in `fault_slip.spatialdb`.

```
# Error
value-names = final-slip-left-lateral final-slip-opening initiation-time

# Correct
value-names = final_slip_left_lateral final_slip_opening initiation_time
```

Step 6: Error 6

Error Message

Listing 4.317: Error message 6 when running Step 6.

```
1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
  ↳ PetscOptions::write(pythia::journal::info_t&, const char*, const_
  ↳ pylith::utils::PetscOptions&)
8 -- info (application-flow)
9 -- Setting PETSc options:
10     dm_reorder_section = true
11     dm_reorder_section_type = cohesive
12     ksp_atol = 1.0e-7
13     ksp_converged_reason = true
14     ksp_error_if_not_converged = true
15     ksp_gmres_restart = 100
16     ksp_guess_pod_size = 8
17     ksp_guess_type = pod
18     ksp_rtol = 1.0e-14
19     mg_fine_ksp_max_it = 5
20     mg_fine_pc_type = vpbjacobi
21     pc_type = gamg
22     snes_atol = 5.0e-7
23     snes_converged_reason = true
24     snes_error_if_not_converged = true
25     snes_monitor = true
26     snes_rtol = 1.0e-14
27     ts_error_if_step_fails = true
28     ts_exact_final_time = matchstep
29     ts_monitor = true
30     ts_type = beuler
31     viewer_hdf5_collective = true
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void_
  ↳ pylith::meshio::MeshIOPetsc::_read()
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
```

(continues on next page)

(continued from previous page)

```

35 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
    ↪pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
36 -- info (application-flow)
37 -- Component 'meshiopetsc.reader': Domain bounding box:
38     (-100000, 100000)
39     (-100000, 0)
40 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void_
    ↪pylith::problems::TimeDependent::verifyConfiguration() const
41 -- info (application-flow)
42 -- Component 'timedependent.problem': Verifying problem configuration.
43 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
    ↪printInfo
44 -- info (application-flow)
45 -- Scales for nondimensionalization:
46     Length scale: 2500*m
47     Displacement scale: 1*m
48     Time scale: 3.15576e+09*s
49     Rigidity scale: 1e+10*m**-1*kg*s**-2
50     Temperature scale: 1*K
51 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void_
    ↪pylith::problems::TimeDependent::initialize()
52 -- info (application-flow)
53 -- Component 'timedependent.problem': Initializing problem.
54 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void_
    ↪pylith::problems::TimeDependent::solve()
55 -- info (application-flow)
56 -- Component 'timedependent.problem': Solving equations.
57 0 TS dt 0.2 time -0.2
58     0 SNES Function norm 7.734839381701e+00
59     Linear solve converged due to CONVERGED_ATOL iterations 14
60     1 SNES Function norm 5.384136432949e-08
61     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
62 1 TS dt 0.2 time 0.
63     0 SNES Function norm 5.384136432949e-08
64     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
65 2 TS dt 0.2 time 0.2
66     0 SNES Function norm 3.245094761947e+00
67     Linear solve converged due to CONVERGED_ATOL iterations 13
68     1 SNES Function norm 4.882143081495e-08
69     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
70 3 TS dt 0.2 time 0.4
71 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
    ↪py:222:finalize
72 -- info (application-flow)
73 -- Finalizing problem.

```

Troubleshooting Strategy

The simulation ran without errors. In visualizing the output we notice the slip distribution contains a sharp transition from 0 m to 2.0 m; we intended to prescribe slip that is uniform above $y = -20$ km and tapers linearly to 0 at $y = -30$ km. We load the JSON parameter file into the PyLith Parameter Viewer and find that we are using the default `query_type` of nearest for the earthquake rupture parameters. For our intended piecewise linear variation in slip, we need to use

linear for the query_type.

Resolution

Listing 4.318: Correct error in step06_twofaults.cfg.

```
[pylithapp.problem.interfaces.fault]
...
db_auxiliary_field.query_type = linear
```

Step 6: Error 7

Error Message

Listing 4.319: Output when running Step 6.

```
1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)
6 -- Running on 1 process(es).
7 >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
8 ↪PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
9 -- info (application-flow)
10 -- Setting PETSc options:
11     dm_reorder_section = true
12     dm_reorder_section_type = cohesive
13     ksp_atol = 1.0e-7
14     ksp_converged_reason = true
15     ksp_error_if_not_converged = true
16     ksp_gmres_restart = 100
17     ksp_guess_pod_size = 8
18     ksp_guess_type = pod
19     ksp_rtol = 1.0e-14
20     mg_fine_ksp_max_it = 5
21     mg_fine_pc_type = vpbjacobi
22     pc_type = gamg
23     snes_atol = 5.0e-7
24     snes_converged_reason = true
25     snes_error_if_not_converged = true
26     snes_monitor = true
27     snes_rtol = 1.0e-14
28     ts_error_if_step_fails = true
29     ts_exact_final_time = matchstep
30     ts_monitor = true
31     ts_type = beuler
32     viewer_hdf5_collective = true
33 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void_
34 ↪pylith::meshio::MeshIOPetsc::_read()
35 -- info (application-flow)
36 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
37 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
```

(continues on next page)

(continued from previous page)

```

36  ↪pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
-- info (application-flow)
37 -- Component 'meshiopetsc.reader': Domain bounding box:
38     (-100000, 100000)
39     (-100000, 0)
40 >> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:51:virtual ↪
    ↪pylith::topology::Mesh ↪
    ↪*pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh *, const ↪
    ↪pylith::problems::Problem &)
41 -- info (application-flow)
42 -- Inserting cohesive cells.
43 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void ↪
    ↪pylith::problems::TimeDependent::verifyConfiguration() const
44 -- info (application-flow)
45 -- Component 'timedependent.problem': Verifying problem configuration.
46 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238: ↪
    ↪printInfo
47 -- info (application-flow)
48 -- Scales for nondimensionalization:
49     Length scale: 2500*m
50     Displacement scale: 1*m
51     Time scale: 3.15576e+09*s
52     Rigidity scale: 1e+10*m**-1*kg*s**-2
53     Temperature scale: 1*K
54 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void ↪
    ↪pylith::problems::TimeDependent::initialize()
55 -- info (application-flow)
56 -- Component 'timedependent.problem': Initializing problem.
57 [0]PETSC ERROR: ----- Error Message -----
    ↪-----
58 [0]PETSC ERROR: Error in external library
59 [0]PETSC ERROR: Could not find values for initiation_time at ( -21860.3 -27621) in ↪
    ↪spatial database 'Fault rupture for main fault'.
60 [0]PETSC ERROR: WARNING! There are unused option(s) set! Could be the program crashed ↪
    ↪before usage or a spelling mistake, etc!
61 [0]PETSC ERROR: Option left: name:-ksp_atol value: 1.0e-7 source: code
62 [0]PETSC ERROR: Option left: name:-ksp_converged_reason (no value) source: code
63 [0]PETSC ERROR: Option left: name:-ksp_error_if_not_converged (no value) source: code
64 [0]PETSC ERROR: Option left: name:-ksp_gmres_restart value: 100 source: code
65 [0]PETSC ERROR: Option left: name:-ksp_guess_pod_size value: 8 source: code
66 [0]PETSC ERROR: Option left: name:-ksp_guess_type value: pod source: code
67 [0]PETSC ERROR: Option left: name:-ksp_rtol value: 1.0e-14 source: code
68 [0]PETSC ERROR: Option left: name:-mg_fine_ksp_max_it value: 5 source: code
69 [0]PETSC ERROR: Option left: name:-mg_fine_pc_type value: vpbjacobi source: code
70 [0]PETSC ERROR: Option left: name:-pc_type value: gamg source: code
71 [0]PETSC ERROR: Option left: name:-snes_atol value: 5.0e-7 source: code
72 [0]PETSC ERROR: Option left: name:-snes_converged_reason (no value) source: code
73 [0]PETSC ERROR: Option left: name:-snes_error_if_not_converged (no value) source: code
74 [0]PETSC ERROR: Option left: name:-snes_monitor (no value) source: code
75 [0]PETSC ERROR: Option left: name:-snes_rtol value: 1.0e-14 source: code
76 [0]PETSC ERROR: Option left: name:-ts_error_if_step_fails (no value) source: code
77 [0]PETSC ERROR: Option left: name:-ts_exact_final_time value: matchstep source: code

```

(continues on next page)

(continued from previous page)

```

78 [0]PETSC ERROR: Option left: name:-ts_monitor (no value) source: code
79 [0]PETSC ERROR: Option left: name:-ts_type value: beuler source: code
80 [0]PETSC ERROR: See https://petsc.org/release/faq/ for trouble shooting.
81 [0]PETSC ERROR: PETSc Development Git Revision: v3.25.1-142-g990e4f00326 Git Date: 2026-
    ↪ 05-15 01:03:36 -0400
82 [0]PETSC ERROR: software/pylith-debug/bin/mpinemesi with 1 MPI process(es) and PETSC_
    ↪ ARCH arch-pylith-debug on IGSKCI164LM006 by baagaard Thu Jun 4 12:12:58 2026
83 [0]PETSC ERROR: Configure options: --PETSC_ARCH=arch-pylith-debug --with-debugging=1 --
    ↪ with-clanguage=c --with-mpi-compilers=1 --with-shared-libraries=1 --with-64-bit-
    ↪ points=1 --with-large-file-io=1 --with-lgrind=0 --download-parmetis=1 --download-
    ↪ metis=1 --download-triangle --download-ml=1 --download-superlu=1 --with-fc=0 --
    ↪ download-f2cblaslapack --with-hdf5=1 --with-hdf5-include=software/pylith-debug/include_
    ↪ --with-hdf5-lib=software/pylith-debug/lib/libhdf5.dylib --with-zlib=1 CFLAGS+=-g
84 [0]PETSC ERROR: #1 static PetscErrorCode_
    ↪ pylith::topology::FieldQuery::queryDBPointFn(PylithInt, PylithReal, const PylithReal *,
    ↪ PylithInt, PylithScalar *, void *)() at src/cig/pylith/libsrc/pylith/topology/
    ↪ FieldQuery.cc:320
85 [0]PETSC ERROR: #2 DMProjectPoint_Func_Private() at software/unix/petsc-dev/src/dm/impls/
    ↪ plex/plexproject.c:143
86 [0]PETSC ERROR: #3 DMProjectPoint_Private() at software/unix/petsc-dev/src/dm/impls/plex/
    ↪ plexproject.c:545
87 [0]PETSC ERROR: #4 DMProjectLocal_Generic_Plex() at software/unix/petsc-dev/src/dm/impls/
    ↪ plex/plexproject.c:1083
88 [0]PETSC ERROR: #5 DMProjectFunctionLocal_Plex() at software/unix/petsc-dev/src/dm/impls/
    ↪ plex/plexproject.c:1114
89 [0]PETSC ERROR: #6 DMProjectFunctionLocal() at software/unix/petsc-dev/src/dm/interface/
    ↪ dm.c:8361
90 [0]PETSC ERROR: #7 void pylith::topology::FieldQuery::queryDB()() at src/cig/pylith/
    ↪ libsrc/pylith/topology/FieldQuery.cc:228
91 Fatal error. Calling MPI_Abort() to abort PyLith application.
92 Traceback (most recent call last):
93   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication.
    ↪ py", line 55, in onComputeNodes
94     self.main(*args, **kwds)
95   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py", ↪
    ↪ line 85, in main
96     self.problem.initialize()
97   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py", ↪
    ↪ line 212, in initialize
98     ModuleProblem.initialize(self)
99   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/problems.py", ↪
    ↪ line 165, in initialize
100     return _problems.Problem_initialize(self)
101     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
102 RuntimeError: Error detected while in PETSc function.
103 C++ traceback (12 frames):
104   [0] 3 libpylith.0.dylib 0x000000001081209fc _
    ↪ ZN6pylith13ExternalErrorCI1NS_5ErrorEERKNS_12ErrorMessageE + 36
105   [1] 4 libpylith.0.dylib 0x00000000108376c28 _
    ↪ ZN6pylith8topology10FieldQuery7queryDBEv + 592
106   [2] 5 libpylith.0.dylib 0x000000001081e709c _
    ↪ ZN6pylith10feassemble16AuxiliaryFactory15setValuesFromDBEv + 608

```

(continues on next page)

(continued from previous page)

```

107 [3] 6 libpylith.0.dylib 0x00000000108172d48 _
    ↳ZN6pylith6faults6KinSrc10initializeERKNS_8topology5FieldERKNS_
    ↳6scales6ScalesEPKN11spatialdata9geocoords8CoordSysE + 1180
108 [4] 7 libpylith.0.dylib 0x0000000010815af64 _
    ↳ZN6pylith6faults16FaultCohesiveKin20createAuxiliaryFieldERKNS_8topology5FieldERKNS2_
    ↳4MeshE + 2764
109 [5] 8 libpylith.0.dylib 0x000000001081909bc _
    ↳ZN6pylith10feassemble10Integrator10initializeERKNS_8topology5FieldE + 560
110 [6] 9 libpylith.0.dylib 0x000000001081b1944 _
    ↳ZN6pylith10feassemble19IntegratorInterface10initializeERKNS_8topology5FieldE + 924
111 [7] 10 libpylith.0.dylib 0x000000001082f5014 _
    ↳ZN6pylith8problems7Problem10initializeEv + 936
112 [8] 11 libpylith.0.dylib 0x00000000108305610 _
    ↳ZN6pylith8problems13TimeDependent10initializeEv + 652
113 [9] 12 _problems.so 0x00000000109a363f0 _ZL24_wrap_Problem_
    ↳initializeP7_objectS0_ + 216
114 [10] 46 mpinemesiis 0x00000000100a65d70 main + 616
115 [11] 47 dyld 0x00000000197e0eb98 start + 6076
116
117 Abort(-1) on node 0 (rank 0 in comm 0): application called MPI_Abort(MPI_COMM_WORLD, -1)
    ↳- process 0
118 software/pylith-debug/bin/nemesis: mpiexec: exit 255
119 software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesis: exit 1

```

Troubleshooting Strategy

With the linear interpolation we get an error about not being able to find an initiation time for a point. This suggests there are errors in our spatial database file related to interpolation. We examine the header and data points for errors. We notice that our points lie along a line (data dimension is 1), but our header has data-dim=2.

Resolution

Listing 4.320: Correct error in fault_slip.spatialdb.

```

# Error
data-dim = 2

# Correct
data-dim = 1

```

Step 6: Error 8

Error Message

Listing 4.321: Error message 8 when running Step 6.

```

1 $ pylith step06_twofaults.cfg
2
3 # Output
4 >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5 -- info (application-flow)

```

(continues on next page)

(continued from previous page)

```

6  -- Running on 1 process(es).
7  >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
    ↪PetscOptions::write(pythia::journal::info_t &, const char *, const PetscOptions &)
8  -- info (application-flow)
9  -- Setting PETSc options:
10     dm_reorder_section = true
11     dm_reorder_section_type = cohesive
12     ksp_atol = 1.0e-7
13     ksp_converged_reason = true
14     ksp_error_if_not_converged = true
15     ksp_gmres_restart = 100
16     ksp_guess_pod_size = 8
17     ksp_guess_type = pod
18     ksp_rtol = 1.0e-14
19     mg_fine_ksp_max_it = 5
20     mg_fine_pc_type = vpbjacobi
21     pc_type = gamg
22     snes_atol = 5.0e-7
23     snes_converged_reason = true
24     snes_error_if_not_converged = true
25     snes_monitor = true
26     snes_rtol = 1.0e-14
27     ts_error_if_step_fails = true
28     ts_exact_final_time = matchstep
29     ts_monitor = true
30     ts_type = beuler
31     viewer_hdf5_collective = true
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:205:virtual void_
    ↪pylith::meshio::MeshIOPetsc::_read()
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
35 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void_
    ↪pylith::meshio::MeshIO::read(pylith::topology::Mesh *, const bool)
36 -- info (application-flow)
37 -- Component 'meshiopetsc.reader': Domain bounding box:
38     (-100000, 100000)
39     (-100000, 0)
40 >> src/cig/pylith/libsrc/pylith/initializers/MeshInsertInterfaces.cc:51:virtual_
    ↪pylith::topology::Mesh_
    ↪*pylith::initializers::MeshInsertInterfaces::run(pylith::topology::Mesh *, const_
    ↪pylith::problems::Problem &)
41 -- info (application-flow)
42 -- Inserting cohesive cells.
43 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void_
    ↪pylith::problems::TimeDependent::verifyConfiguration() const
44 -- info (application-flow)
45 -- Component 'timedependent.problem': Verifying problem configuration.
46 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
    ↪printInfo
47 -- info (application-flow)
48 -- Scales for nondimensionalization:
49     Length scale: 2500*m

```

(continues on next page)

(continued from previous page)

```

50     Displacement scale: 1*m
51     Time scale: 3.15576e+09*s
52     Rigidity scale: 1e+10*m**-1*kg*s**-2
53     Temperature scale: 1*K
54 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
    ↪ pylith::problems::TimeDependent::initialize()
55 -- info (application-flow)
56 -- Component 'timedependent.problem': Initializing problem.
57 [0]PETSC ERROR: ----- Error Message -----
    ↪ -----
58 [0]PETSC ERROR: Error in external library
59 [0]PETSC ERROR: Could not find values for initiation_time at ( -21860.3 -27621) in
    ↪ spatial database 'Fault rupture for main fault'.
60 [0]PETSC ERROR: WARNING! There are unused option(s) set! Could be the program crashed
    ↪ before usage or a spelling mistake, etc!
61 [0]PETSC ERROR: Option left: name:-ksp_atol value: 1.0e-7 source: code
62 [0]PETSC ERROR: Option left: name:-ksp_converged_reason (no value) source: code
63 [0]PETSC ERROR: Option left: name:-ksp_error_if_not_converged (no value) source: code
64 [0]PETSC ERROR: Option left: name:-ksp_gmres_restart value: 100 source: code
65 [0]PETSC ERROR: Option left: name:-ksp_guess_pod_size value: 8 source: code
66 [0]PETSC ERROR: Option left: name:-ksp_guess_type value: pod source: code
67 [0]PETSC ERROR: Option left: name:-ksp_rtol value: 1.0e-14 source: code
68 [0]PETSC ERROR: Option left: name:-mg_fine_ksp_max_it value: 5 source: code
69 [0]PETSC ERROR: Option left: name:-mg_fine_pc_type value: vpbjacobi source: code
70 [0]PETSC ERROR: Option left: name:-pc_type value: gamg source: code
71 [0]PETSC ERROR: Option left: name:-snes_atol value: 5.0e-7 source: code
72 [0]PETSC ERROR: Option left: name:-snes_converged_reason (no value) source: code
73 [0]PETSC ERROR: Option left: name:-snes_error_if_not_converged (no value) source: code
74 [0]PETSC ERROR: Option left: name:-snes_monitor (no value) source: code
75 [0]PETSC ERROR: Option left: name:-snes_rtol value: 1.0e-14 source: code
76 [0]PETSC ERROR: Option left: name:-ts_error_if_step_fails (no value) source: code
77 [0]PETSC ERROR: Option left: name:-ts_exact_final_time value: matchstep source: code
78 [0]PETSC ERROR: Option left: name:-ts_monitor (no value) source: code
79 [0]PETSC ERROR: Option left: name:-ts_type value: beuler source: code
80 [0]PETSC ERROR: See https://petsc.org/release/faq/ for trouble shooting.
81 [0]PETSC ERROR: PETSc Development Git Revision: v3.25.1-142-g990e4f00326 Git Date: 2026-
    ↪ 05-15 01:03:36 -0400
82 [0]PETSC ERROR: software/pylith-debug/bin/mpinemesi with 1 MPI process(es) and PETSC_
    ↪ ARCH arch-pylith-debug on IGSKCI164LM006 by baagaard Thu Jun 4 12:15:03 2026
83 [0]PETSC ERROR: Configure options: --PETSC_ARCH=arch-pylith-debug --with-debugging=1 --
    ↪ with-clanguage=c --with-mpi-compilers=1 --with-shared-libraries=1 --with-64-bit-
    ↪ points=1 --with-large-file-io=1 --with-lgrind=0 --download-parmetis=1 --download-
    ↪ metis=1 --download-triangle --download-ml=1 --download-superlu=1 --with-fc=0 --
    ↪ download-f2cblaslapack --with-hdf5=1 --with-hdf5-include=software/pylith-debug/include
    ↪ --with-hdf5-lib=software/pylith-debug/lib/libhdf5.dylib --with-zlib=1 CFLAGS+=-g
84 [0]PETSC ERROR: #1 static PetscErrorCode
    ↪ pylith::topology::FieldQuery::queryDBPointFn(PylithInt, PylithReal, const PylithReal *,
    ↪ PylithInt, PylithScalar *, void *)() at src/cig/pylith/libsrc/pylith/topology/
    ↪ FieldQuery.cc:320
85 [0]PETSC ERROR: #2 DMProjectPoint_Func_Private() at software/unix/petsc-dev/src/dm/impls/
    ↪ plex/plexproject.c:143
86 [0]PETSC ERROR: #3 DMProjectPoint_Private() at software/unix/petsc-dev/src/dm/impls/plex/

```

(continues on next page)

(continued from previous page)

```

87 ↪plexproject.c:545
[0]PETSC ERROR: #4 DMProjectLocal_Generic_Plex() at software/unix/petsc-dev/src/dm/impls/
88 ↪plex/plexproject.c:1083
[0]PETSC ERROR: #5 DMProjectFunctionLocal_Plex() at software/unix/petsc-dev/src/dm/impls/
89 ↪plex/plexproject.c:1114
[0]PETSC ERROR: #6 DMProjectFunctionLocal() at software/unix/petsc-dev/src/dm/interface/
90 ↪dm.c:8361
[0]PETSC ERROR: #7 void pylith::topology::FieldQuery::queryDB()() at src/cig/pylith/
91 ↪libsrc/pylith/topology/FieldQuery.cc:228
Fatal error. Calling MPI_Abort() to abort PyLith application.
92 Traceback (most recent call last):
93   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PetscApplication.
94 ↪py", line 55, in onComputeNodes
95     self.main(*args, **kwds)
96   File "software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py", ↪
97 ↪line 85, in main
98     self.problem.initialize()
99   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py", ↪
100 ↪line 212, in initialize
101     ModuleProblem.initialize(self)
102   File "software/pylith-debug/lib/python3.12/site-packages/pylith/problems/problems.py", ↪
103 ↪line 165, in initialize
104     return _problems.Problem_initialize(self)
105     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
106 RuntimeError: Error detected while in PETSc function.
107 C++ traceback (12 frames):
108 [0] 3  libpylith.0.dylib                0x0000000010653c9fc _
109 ↪ZN6pylith13ExternalErrorCI1NS_5ErrorEERKNS_12ErrorMessageE + 36
110 [1] 4  libpylith.0.dylib                0x00000000106792c28 _
111 ↪ZN6pylith8topology10FieldQuery7queryDBEv + 592
112 [2] 5  libpylith.0.dylib                0x0000000010660309c _
113 ↪ZN6pylith10feassemble16AuxiliaryFactory15setValuesFromDBEv + 608
114 [3] 6  libpylith.0.dylib                0x0000000010658ed48 _
115 ↪ZN6pylith6faults6KinSrc10initializeERKNS_8topology5FieldERKNS_
116 ↪6scales6ScalesEPKN11spatialdata9geocoords8CoordSysE + 1180
117 [4] 7  libpylith.0.dylib                0x00000000106576f64 _
118 ↪ZN6pylith6faults16FaultCohesiveKin20createAuxiliaryFieldERKNS_8topology5FieldERKNS2_
119 ↪4MeshE + 2764
120 [5] 8  libpylith.0.dylib                0x000000001065ac9bc _
121 ↪ZN6pylith10feassemble10Integrator10initializeERKNS_8topology5FieldE + 560
122 [6] 9  libpylith.0.dylib                0x000000001065cd944 _
123 ↪ZN6pylith10feassemble19IntegratorInterface10initializeERKNS_8topology5FieldE + 924
124 [7] 10 libpylith.0.dylib                0x00000000106711014 _
125 ↪ZN6pylith8problems7Problem10initializeEv + 936
126 [8] 11 libpylith.0.dylib                0x00000000106721610 _
127 ↪ZN6pylith8problems13TimeDependent10initializeEv + 652
128 [9] 12 _problems.so                    0x000000001096e23f0 _ZL24_wrap_Problem_
129 ↪initializeP7_objectS0_ + 216
130 [10] 46 mpinemesis                     0x00000000102addd70 main + 616
131 [11] 47 dyld                           0x00000000197e0eb98 start + 6076
132
133 Abort(-1) on node 0 (rank 0 in comm 0): application called MPI_Abort(MPI_COMM_WORLD, -1)↪

```

(continues on next page)

(continued from previous page)

```

118 ↪- process 0
software/pylith-debug/bin/nemesis: mpiexec: exit 255
119 software/pylith-debug/bin/pylith: software/pylith-debug/bin/nemesis: exit 1

```

Troubleshooting Strategy

We still get an error about not being able to find an initiation time for a point. This suggests there are still one or more errors in our spatial database file related to interpolation. We examine the header and data points for errors. We notice that our deepest point has a y coordinate of -25 km, but PyLith is looking for values at a point with a y coordinate of -27.621 km. We need to add an additional point to our spatial database. This explains why we had `num-locs=4` when we started!

Resolution

Listing 4.322: Correct error in `fault_slip.spatialdb`.

```

num-locs = 4
...
0.0  99.0   -2.0   0.0  0.0
0.0 -20.0   -2.0   0.0  0.0
0.0 -25.0    0.0   0.0  0.0
0.0 -99.0    0.0   0.0  0.0

```

Step 6: Successful run

Listing 4.323: Output when running Step 6.

```

1  $ pylith step06_twofaults.cfg
2
3  # Output
4  >> software/pylith-debug/lib/python3.12/site-packages/pylith/apps/PyLithApp.py:76:main
5  -- info (application-flow)
6  -- Running on 1 process(es).
7  >> src/cig/pylith/libsrc/pylith/Utils/PetscOptions.cc:251:static void pylith::utils::_
   ↪PetscOptions::write(pythia::journal::info_t&, const char*, const_
   ↪pylith::utils::PetscOptions&)
8  -- info (application-flow)
9  -- Setting PETSc options:
10     dm_reorder_section = true
11     dm_reorder_section_type = cohesive
12     ksp_atol = 1.0e-7
13     ksp_converged_reason = true
14     ksp_error_if_not_converged = true
15     ksp_gmres_restart = 100
16     ksp_guess_pod_size = 8
17     ksp_guess_type = pod
18     ksp_rtol = 1.0e-14
19     mg_fine_ksp_max_it = 5
20     mg_fine_pc_type = vpbjacobi
21     pc_type = gamg
22     snes_atol = 5.0e-7

```

(continues on next page)

(continued from previous page)

```

23     snes_converged_reason = true
24     snes_error_if_not_converged = true
25     snes_monitor = true
26     snes_rtol = 1.0e-14
27     ts_error_if_step_fails = true
28     ts_exact_final_time = matchstep
29     ts_monitor = true
30     ts_type = beuler
31     viewer_hdf5_collective = true
32 >> src/cig/pylith/libsrc/pylith/meshio/MeshIOPetsc.cc:204:virtual void
    ↪ pylith::meshio::MeshIOPetsc::_read()
33 -- info (application-flow)
34 -- Component 'meshiopetsc.reader': Reading finite-element mesh from 'mesh_tri.msh'.
35 >> src/cig/pylith/libsrc/pylith/meshio/MeshIO.cc:76:void
    ↪ pylith::meshio::MeshIO::read(pylith::topology::Mesh*, bool)
36 -- info (application-flow)
37 -- Component 'meshiopetsc.reader': Domain bounding box:
38     (-100000, 100000)
39     (-100000, 0)
40 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:316:virtual void
    ↪ pylith::problems::TimeDependent::verifyConfiguration() const
41 -- info (application-flow)
42 -- Component 'timedependent.problem': Verifying problem configuration.
43 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.py:238:_
    ↪ printInfo
44 -- info (application-flow)
45 -- Scales for nondimensionalization:
46     Length scale: 2500*m
47     Displacement scale: 1*m
48     Time scale: 3.15576e+09*s
49     Rigidity scale: 1e+10*m**-1*kg*s**-2
50     Temperature scale: 1*K
51 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:342:virtual void
    ↪ pylith::problems::TimeDependent::initialize()
52 -- info (application-flow)
53 -- Component 'timedependent.problem': Initializing problem.
54 >> src/cig/pylith/libsrc/pylith/problems/TimeDependent.cc:473:void
    ↪ pylith::problems::TimeDependent::solve()
55 -- info (application-flow)
56 -- Component 'timedependent.problem': Solving equations.
57 0 TS dt 0.2 time -0.2
58     0 SNES Function norm 7.713717485060e+00
59     Linear solve converged due to CONVERGED_ATOL iterations 14
60     1 SNES Function norm 5.241642052618e-08
61     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
62 1 TS dt 0.2 time 0.
63     0 SNES Function norm 5.241642052618e-08
64     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 0
65 2 TS dt 0.2 time 0.2
66     0 SNES Function norm 3.245094761956e+00
67     Linear solve converged due to CONVERGED_ATOL iterations 13
68     1 SNES Function norm 7.651348491638e-08

```

(continues on next page)

(continued from previous page)

```

69     Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 1
70 3 TS dt 0.2 time 0.4
71 >> software/pylith-debug/lib/python3.12/site-packages/pylith/problems/Problem.
    ↪ py:222:finalize
72 -- info (application-flow)
73 -- Finalizing problem.

```

Our simulation now runs without errors and the output looks correct.

Additional Examples

Cubit Meshing Examples

The directory `examples/meshing-cubit` contains several examples of using Cubit to construct finite-element meshes for complex geometry. This includes features such as constructing nonplanar fault geometry from contours, constructing topography from a DEM, and merging sheet bodies (surfaces). A separate examples discusses defining the discretization size using a vertex field in an Exodus-II file. See the `README` files in the subdirectories for more detailed descriptions of these examples.

Troubleshooting Examples

The directory `examples/troubleshooting-2d` contains a few examples to practice troubleshooting a variety of user errors in parameters files and problem setup. The files with the errors corrected are in `examples/troubleshooting-2d/correct`.

Code Verification Benchmarks

The CIG GitHub software repository https://github.com/geodynamics/pylith_benchmarks contains input files for a number of community benchmarks. The benchmarks do not include the mesh files because they are so large; instead they include the Cubit Python scripts that can be used to generate the meshes. Most, but not all, of the input files in the repository are updated for PyLith v2.0.0, so you will need to modify them if you use another version of PyLith.

4.2.10 Benchmarks

Overview

We have started developing a suite of performance benchmarks to document current PyLith performance and improvements with adjustments to solver settings. The current benchmarks focus on solver performance for linear elasticity with prescribed fault slip.

Performance Benchmarks

Linear Elasticity Performance Benchmarks

Solver Parameters

We consider the solver parameters used in PyLith versions 3.x (Schur complement) and 4.x (variable point block Jacobi), including adjustments that improve performance.

Schur Complement

In PyLith versions 3.0–4.0, the default solver parameters for linear elasticity with prescribed slip use a Schur complement approach (Listing 4.324). We use lower factorization (`pc_fieldsplit_schur_factorization_type = lower`) with the preconditioning for the Schur complement generated from an explicitly-assembled approximation (`pc_fieldsplit_schur_precondition = selfp`), $S_p = A_{11} - A_{10}(\text{diag}(A_{00}))^{-1}A_{01}$ [cite{petsc}]. Within the blocks, we use algebraic multigrid preconditioning with only one application of the preconditioner (`preonly`).

Listing 4.324: Schur complement solver parameters (`fieldsplit` (`selfp`)).

```
[pylithapp.problem.scales]
length_scale = 1.0*km
displacement_scale = 1.0*km

[pylithapp.problem.petsc_defaults]
solver = False

[pylithapp.petsc]
pc_type = fieldsplit
pc_use_amat = true
pc_fieldsplit_type = schur

pc_fieldsplit_schur_factorization_type = lower
pc_fieldsplit_schur_precondition = selfp
pc_fieldsplit_schur_scale = 1.0

fieldsplit_displacement_ksp_type = preonly
fieldsplit_displacement_pc_type = ml

fieldsplit_lagrange_multiplier_fault_ksp_type = preonly
fieldsplit_lagrange_multiplier_fault_pc_type = ml
```

Variable Point Block Jacobi

In PyLith versions 4.1–4.2, the default solver parameters for linear elasticity with prescribed slip use a variable point block Jacobian preconditioner (Listing 4.325). We reorder the degrees of freedom in the solution vector (`dm_reorder_section_type = cohesive`) to form small blocks with the degrees of freedom associated with the Lagrange multipliers at a point and the corresponding displacements. We use the PETSc geometric algebraic multigrid preconditioner with variable point block Jacobi preconditioning at the fine scale (`mg_fine_pc_type = vpbjacob`). We use the default parameters for the geometric algebraic multigrid preconditioner.

Listing 4.325: Variable point block Jacobi solver parameters (`vpbjacob` (`defaults`)). The highlighted lines indicate the differences relative to the field split solver parameters.

```
[pylithapp.problem.scales]
length_scale = 1.0*km
displacement_scale = 1.0*km

[pylithapp.problem.petsc_defaults]
solver = False

[pylithapp.petsc]
```

(continues on next page)

(continued from previous page)

```

pc_type = gamg

dm_reorder_section = True
dm_reorder_section_type = cohesive

mg_fine_pc_type = vpbjacobi
ksp_gmres_restart = 100

```

Tuning Fine-Scale Parameters

In PyLith version 5.0, we adjust the fine-scale preconditioner settings to improve solver performance (`mg_fine_ksp_max_it = 5`).

Listing 4.326: Variable point block Jacobi solver parameters with adjustments at the fine scale of the multigrid preconditioner (vpbjacobi (tunefine)).

```

[pylithapp.problem.scales]
length_scale = 1.0*km
displacement_scale = 1.0*km

[pylithapp.problem.petsc_defaults]
solver = False

[pylithapp.petsc]
pc_type = gamg

dm_reorder_section = True
dm_reorder_section_type = cohesive
mg_fine_pc_type = vpbjacobi
ksp_gmres_restart = 100

mg_fine_ksp_max_it = 5

```

Tuning Fine- and Coarse-Scale Parameters

We also considered tuning at the parameters at the coarse levels of the multigrid preconditioner (Listing 4.327). These changes resulted in slight increases in the number of iterations required for the linear solver and a small increase in the runtime.

Listing 4.327: Variable point block Jacobi solver parameters with adjustments at the fine and coarse scales of the multigrid preconditioner (vpbjacobi (tunecoarse)). Adjusting the parameters and the coarse scale did not improve performance.

```

[pylithapp.problem.scales]
length_scale = 1.0*km
displacement_scale = 1.0*km

[pylithapp.problem.petsc_defaults]
solver = False

```

(continues on next page)

(continued from previous page)

```
[pylithapp.petsc]
pc_type = gamg

dm_reorder_section = True
dm_reorder_section_type = cohesive
mg_fine_pc_type = vpbjacobi
ksp_gmres_restart = 100

mg_fine_ksp_max_it = 5

pc_gamg_coarse_eq_limit = 200
mg_levels_pc_type = pbjacobi
pc_gamg_agg_nsmoothing = 3
```

Length and Displacement Scales

In PyLith version 5.0, we also added independent specification displacement scale, whereas in previous versions the displacement scale matched the length scale. Setting the displacement scale to the displacement corresponding to the average slip (Listing 4.328) normalizes the displacement and Lagrange multiplier values in the solution vector independent of the length scale. Consequently, the solver tolerances become independent of the discretization size and we can specify meaningful defaults ($ksp_atol=1.0e-7$, $snes_atol=4.0e-7$).

Listing 4.328: Variable point block Jacobi solver parameters with adjustments at the fine scale of the multigrid preconditioner and separation of length and displacement scales (vpbjacobi (dispscale)).

```
[pylithapp.problem.scales]
length_scale = 1.0*km
displacement_scale = 1.0*m

[pylithapp.problem.petsc_defaults]
solver = False

[pylithapp.petsc]
pc_type = gamg

dm_reorder_section = True
dm_reorder_section_type = cohesive
mg_fine_pc_type = vpbjacobi
ksp_gmres_restart = 100

mg_fine_ksp_max_it = 5
```

Solver Tolerances

With the variable point block Jacobi preconditioner at the fine scale, for the scales to match in the blocks that include displacement and Lagrange multiplier terms, the length scale (x_0) must match the nominal discretization size (h_0). The displacement and Lagrange multiplier values in the solution vector are normalized by the length scale, so the normalization depends on the discretization size. To achieve a given level of accuracy in the displacement field relative to the scale of deformation requires setting the KSP and SNES absolute tolerances relative to the discretization size.

For example, to achieve an accuracy in the displacement field of $1.0e-7$ relative to the nominal displacement, $u_{nominal}$,

we use

$$\text{ksp_atol} = 1.0 \times 10^{-7} u_{\text{nominal}}/h_o, \quad (4.499)$$

$$\text{snes_atol} = 4.0 \times 10^{-7} u_{\text{nominal}}/h_o. \quad (4.500)$$

2-D Linear Elasticity with Prescribed Slip

We evaluate solver performance for 2-D plane strain linear elasticity with prescribed slip using Step 2 (variable slip) from `examples/crustal-strikeslip-2d`. We create a suite of simulations in which we use uniform global refinement to reduce the discretization size by a factor of 2 with up to 5 levels of refinement. We assess the performance based on the number of iterations in the linear solver, the time required to solve, and the total runtime.

Note

We generate the meshes with a gradient in the mesh size in which the mesh size increases at a geometric rate with distance from the faults. Uniform global refinement of the meshes does not preserve this geometric rate, because the refinement within a cell is uniform. Generating new meshes for each desired discretization size on the fault allows preserving the geometric rate and will result in a smaller number of unknowns relative to the discretization size on the fault compared with uniform global refinement.

Running the Benchmark

```
# Generate the finite-element mesh using Gmsh
./generate_gmsh.py --write --filename=mesh_tri.msh

# Run the benchmark simulations
pylith step02_fieldsplit_selfp.cfg >& logs/step02a_fieldsplit_selfp.stdout
pylith step02_fieldsplit_selfp.cfg step02b_fieldsplit_selfp.cfg >& logs/step02b_
↳ fieldsplit_selfp.stdout

pylith step03_vpbjacobi.cfg >& step03a_vpbjacobi.stdout
pylith step03_vpbjacobi.cfg step03b_vpbjacobi.cfg >& step03b_vpbjacobi.stdout
pylith step03_vpbjacobi.cfg step03c_vpbjacobi.cfg >& step03c_vpbjacobi.stdout
pylith step03_vpbjacobi.cfg step03d_vpbjacobi.cfg >& step03d_vpbjacobi.stdout
pylith step03_vpbjacobi.cfg step03e_vpbjacobi.cfg >& step03e_vpbjacobi.stdout
pylith step03_vpbjacobi.cfg step03f_vpbjacobi.cfg >& step03f_vpbjacobi.stdout

pylith step04_vpbjacobi_tunefine.cfg >& step04a_vpbjacobi_tunefine.stdout
pylith step04_vpbjacobi_tunefine.cfg step04b_vpbjacobi_tunefine.cfg >& step04b_vpbjacobi_
↳ tunefine.stdout
pylith step04_vpbjacobi_tunefine.cfg step04c_vpbjacobi_tunefine.cfg >& step04c_vpbjacobi_
↳ tunefine.stdout
pylith step04_vpbjacobi_tunefine.cfg step04d_vpbjacobi_tunefine.cfg >& step04d_vpbjacobi_
↳ tunefine.stdout
pylith step04_vpbjacobi_tunefine.cfg step04e_vpbjacobi_tunefine.cfg >& step04e_vpbjacobi_
↳ tunefine.stdout
pylith step04_vpbjacobi_tunefine.cfg step04f_vpbjacobi_tunefine.cfg >& step04f_vpbjacobi_
↳ tunefine.stdout

pylith step06_vpbjacobi_dispscale.cfg >& step06a_vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06b_vpbjacobi_dispscale.cfg >& step06b_
↳ vpbjacobi_dispscale.stdout
```

(continues on next page)

(continued from previous page)

```

pylith step06_vpbjacobi_dispscale.cfg step06c_vpbjacobi_dispscale.cfg >& step06c_
↪vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06d_vpbjacobi_dispscale.cfg >& step06d_
↪vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06e_vpbjacobi_dispscale.cfg >& step06e_
↪vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06f_vpbjacobi_dispscale.cfg >& step06f_
↪vpbjacobi_dispscale.stdout

```

Results

For small problems (about 10,000) unknowns we find little difference in the runtimes across the different solver settings. As the number of unknowns increases towards 100,000, the number of iterations required by the Schur complement field split preconditioner increases dramatically, resulting in dramatic increases in runtime. The geometric algebraic multigrid preconditioner with variable point block Jacobi preconditioner at the fine scale with default parameters works well up to about 1 million unknowns. Fine-tuning the fine-scale multigrid preconditioner parameters gives good performance across the entire range of problem sizes considered.

Note

Using independent displacement and length scales does not improve solver performance, but it allows use of solver tolerances that are relative to the displacement scale and independent of discretization size.

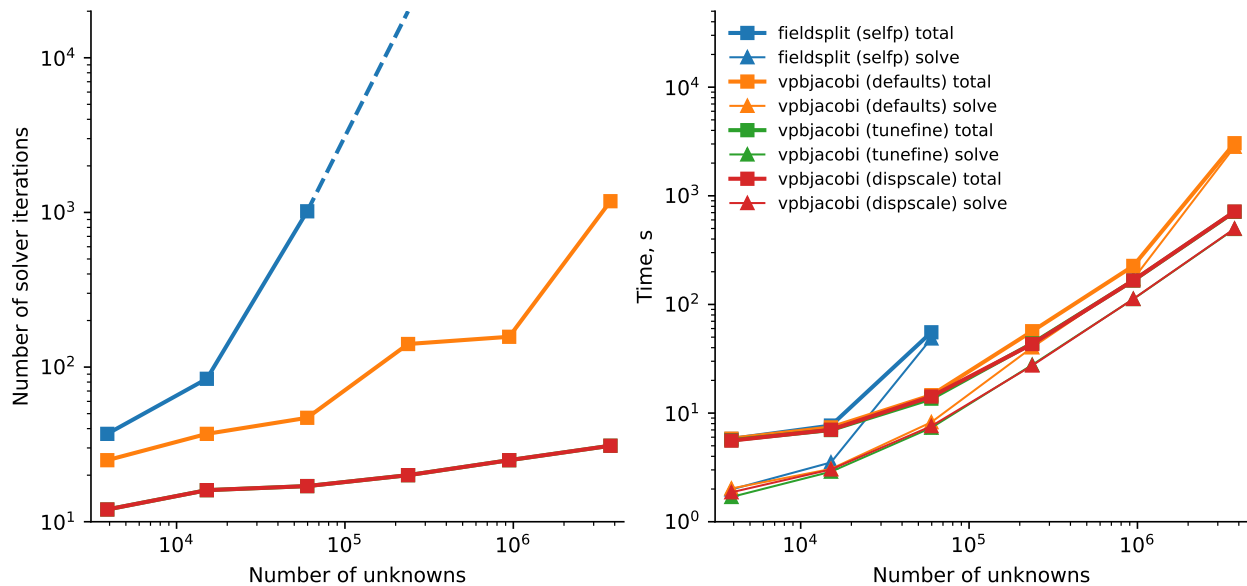


Fig. 4.183: Solver performance for 2-D plane strain elasticity with prescribed slip. The left panel shows the number of iterations used by the linear solver as a function of the problem size (number of unknowns). The right panels shows the total (squares) and solve (triangles) runtime as a function of the problem size (number of unknowns). The dashed lines indicate the solves required more than 10,000 iterations at the next level of refinement.

3-D Linear Elasticity with Prescribed Slip

We evaluate solver performance for 3-D plane strain linear elasticity with prescribed slip using Step 2 (variable slip) from `examples/crustal-strikeslip-3d`. We create a suite of simulations in which we generate meshes with factors of 2 reduction in cell size on the fault. We assess the performance based on the number of iterations in the linear solver, the time required to solve, and the total runtime.

Important

Generating high quality meshes with tetrahedral cells is much more difficult than it is in 2-D for triangular cells. In this benchmark, we found degradation of mesh quality with uniform global refinement, which resulted in a substantial increase in the number of iterations required by the linear solver. Generating meshes for each discretization size on the fault provided an alternative approach. Furthermore, generating meshes for each discretization size maintains the geometric rate of increase in cell size with distance from the faults. In this case the number of unknowns increases by about a factor of 4 each each factor of 2 reduction in the cell size on the fault; this is a factor of 2 smaller than that for uniform global refinement.

Running the Benchmark

```
# Generate the finite-element meshes using Gmsh
./generate_gmsh.py --write --level=0 --filename=mesh_tet2500.msh
./generate_gmsh.py --write --level=1 --filename=mesh_tet1250.msh
./generate_gmsh.py --write --level=2 --filename=mesh_tet0625.msh
./generate_gmsh.py --write --level=3 --filename=mesh_tet0312.msh
./generate_gmsh.py --write --level=4 --filename=mesh_tet0156.msh
./generate_gmsh.py --write --level=5 --filename=mesh_tet0078.msh

# Run the benchmark simulations
pyolith step02_fieldsplit_selfp.cfg >& logs/step02a_fieldsplit_selfp.stdout
pyolith step02_fieldsplit_selfp.cfg step02b_fieldsplit_selfp.cfg >& logs/step02b_
↳fieldsplit_selfp.stdout

pyolith step03_vpbjacobi.cfg >& step03a_vpbjacobi.stdout
pyolith step03_vpbjacobi.cfg step03b_vpbjacobi.cfg >& step03b_vpbjacobi.stdout
pyolith step03_vpbjacobi.cfg step03c_vpbjacobi.cfg >& step03c_vpbjacobi.stdout
pyolith step03_vpbjacobi.cfg step03d_vpbjacobi.cfg >& step03d_vpbjacobi.stdout

pyolith step04_vpbjacobi_tunefine.cfg >& step04a_vpbjacobi_tunefine.stdout
pyolith step04_vpbjacobi_tunefine.cfg step04b_vpbjacobi_tunefine.cfg >& step04b_vpbjacobi_
↳tunefine.stdout
pyolith step04_vpbjacobi_tunefine.cfg step04c_vpbjacobi_tunefine.cfg >& step04c_vpbjacobi_
↳tunefine.stdout
pyolith step04_vpbjacobi_tunefine.cfg step04d_vpbjacobi_tunefine.cfg >& step04d_vpbjacobi_
↳tunefine.stdout
pyolith step04_vpbjacobi_tunefine.cfg step04e_vpbjacobi_tunefine.cfg >& step04e_vpbjacobi_
↳tunefine.stdout
pyolith step04_vpbjacobi_tunefine.cfg step04f_vpbjacobi_tunefine.cfg >& step04f_vpbjacobi_
↳tunefine.stdout

pyolith step06_vpbjacobi_dispscale.cfg >& step06a_vpbjacobi_dispscale.stdout
pyolith step06_vpbjacobi_dispscale.cfg step06b_vpbjacobi_dispscale.cfg >& step06b_
↳vpbjacobi_dispscale.stdout
```

(continues on next page)

(continued from previous page)

```

pylith step06_vpbjacobi_dispscale.cfg step06c_vpbjacobi_dispscale.cfg >& step06c_
↪vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06d_vpbjacobi_dispscale.cfg >& step06d_
↪vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06e_vpbjacobi_dispscale.cfg >& step06e_
↪vpbjacobi_dispscale.stdout
pylith step06_vpbjacobi_dispscale.cfg step06f_vpbjacobi_dispscale.cfg >& step06f_
↪vpbjacobi_dispscale.stdout

```

Results

For small problems (about 10,000) unknowns we find little difference in the runtimes across the different solver settings. As the number of unknowns increases towards 100,000, the number of iterations required by the Schur complement field split preconditioner increases dramatically, resulting in dramatic increases in runtime. The geometric algebraic multigrid preconditioner with variable point block Jacobi preconditioner at the fine scale with default parameters works well up to about 100,000 unknowns. Fine-tuning the fine-scale multigrid preconditioner parameters gives good performance across the entire range of problem sizes considered.

Note

Using independent displacement and length scales does not improve solver performance, but it allows use of solver tolerances that are relative to the displacement scale and independent of discretization size.

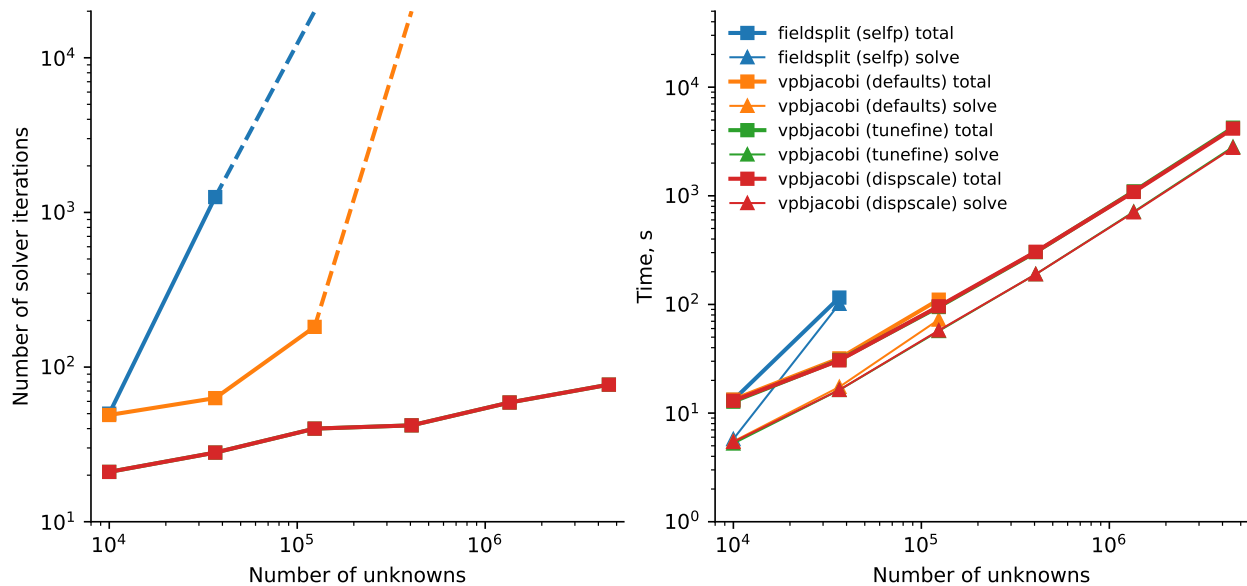


Fig. 4.184: Solver performance for 3-D elasticity with prescribed slip. The left panel shows the number of iterations used by the linear solver as a function of the problem size (number of unknowns). The right panels shows the total (squares) and solve (triangles) runtime as a function of the problem size (number of unknowns). The dashed lines indicate the solves required more than 10,000 iterations at the next level of refinement.

Legacy Benchmarks

The Crustal Deformation Modeling and Earthquake Source Physics Focus Groups within the Southern California Earthquake Center and the Short-Term Tectonics Working Group within CIG developed a suite of benchmarks to test the accuracy and performance of 3D numerical codes for quasistatic crustal deformation and earthquake rupture dynamics. The benchmark definitions for the quasistatic crustal deformation benchmarks are posted on the CIG website at Short-Term Tectonics Benchmarks <https://geodynamics.org/groups/shortterm/wiki/MainPage/Benchmarks> and the definitions for the earthquake rupture benchmarks are posted on the SCEC website <https://strike.scec.org/cvws/cgi-bin/cvws.cgi>. This suite of benchmarks permits evaluating the relative performance of different types of basis functions, quadrature schemes, and discretizations for geophysical applications. The files needed to run the 3D benchmarks are in the CIG GitHub Repository https://github.com/geodynamics/pylith_benchmarks. In addition to evaluating the efficiency and accuracy of numerical codes, the benchmarks also make good test problems, where users can perform simulations based on actual geophysical problems. The benchmarks are performed at various resolutions and using different element types. By comparing the runtime and accuracy for different resolutions and element types, users can evaluate which combination will be best for their problems of interest.

Warning

None of the benchmark input files in the PyLith benchmarks repository on GitHub have been updated for v3.0.

Strike-Slip Benchmark

This benchmark problem computes the viscoelastic (Maxwell) relaxation of stresses from a single, finite, strike-slip earthquake in 3D without gravity. Dirichlet boundary conditions equal to the analytical elastic solution are imposed on the sides of a cube with sides of length 24 km. Anti-plane strain boundary conditions are imposed at $y = 0$, so the solution is equivalent to that for a domain with a 48 km length in the y direction. We can use the analytical solution of Okada [1992] both to apply the boundary conditions and to compare against the numerically-computed elastic solution.

Problem Description

Fig. 4.185 shows the geometry of the strike-slip fault (red surface) embedded in the cube consisting of an elastic material (yellow block) over a Maxwell viscoelastic material (blue block).

Domain The domain spans the region

$$0 \leq x \leq 24 \text{ km}, \quad (4.501)$$

$$0 \leq y \leq 24 \text{ km}, \quad (4.502)$$

$$-24 \text{ km} \leq z \leq 0. \quad (4.503)$$

The top (elastic) layer occupies the region $-12 \text{ km} \leq z \leq 0$ and the bottom (viscoelastic) layer occupies the region $-24 \text{ km} \leq z \leq -12 \text{ km}$.

Material properties The material is a Poisson solid with a shear modulus of 30 GPa. The domain is modeled using an elastic isotropic material for the top layer and a Maxwell viscoelastic material for the bottom layer. The bottom layer has a viscosity of $1.0\text{e}+18$ Pa-s.

Fault The fault is a vertical, right-lateral strike-slip fault. The strike is parallel to the y -direction at the center of the model:

$$x = 12 \text{ km}, \quad (4.504)$$

$$0 \leq y \leq 16 \text{ km}, \quad (4.505)$$

$$-16 \text{ km} \leq z \leq 0. \quad (4.506)$$

Uniform slip of 1 m is applied over the region $0 \leq y \leq 12 \text{ km}$ and $-12 \text{ km} \leq z \leq 0$ with a linear taper to 0 at $y = 16 \text{ km}$ and $z = -16 \text{ km}$. The tapered region is the light red portion of the fault surface in Fig. 4.185. In the region where the two tapers overlap, each slip value is the minimum of the two tapers (so that the taper remains linear).

Boundary conditions Bottom and side displacements are set to the elastic analytical solution, and the top of the model is a free surface. There are two exceptions to these applied boundary conditions. The first is on the $y=0$ plane, where y -displacements are left free to preserve symmetry, and the x - and z -displacements are set to zero. The second is along the line segment between $(12, 0, -24)$ and $(12, 24, -24)$, where the analytical solution blows up in some cases. Along this line segment, all three displacement components are left free.

Discretization The model is discretized with nominal spatial resolutions of 1000 m, 500 m, and 250 m.

Basis functions We use trilinear hexahedral cells and linear tetrahedral cells.

Solution We compute the error in the elastic solution and compare the solution over the domain after 0, 1, 5, and 10 years.

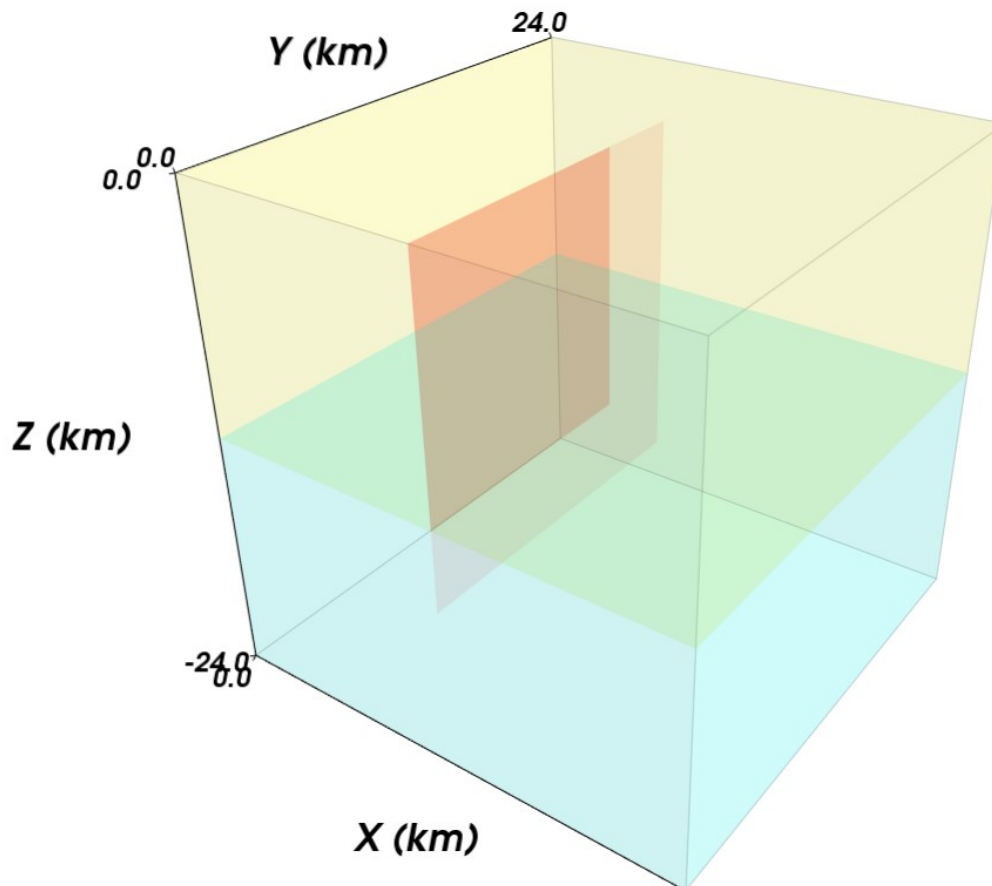


Fig. 4.185: Geometry of strike-slip benchmark problem.

Running the Benchmark

Each benchmark uses three `.cfg` files in the parameters directory: `pylithapp.cfg`, a mesher related file (`strikeslip_cubit.cfg`), and a resolution and cell related file (e.g., `strikeslip_hex8_1000m.cfg`).

```
# Checkout the benchmark files from the CIG Git repository.
$ git clone https://github.com/geodynamics/pylith_benchmarks.git
# Change to the quasistatic/sceccrustdeform/strikeslip directory.
```

(continues on next page)

(continued from previous page)

```

$ cd quasistatic/sceccrustdeform/strikeslip
# Decompress the gzipped files in the meshes and parameters directories.
$ gunzip meshes/*.gz parameters/*.gz
# Change to the parameters directory.
$ cd parameters
# Examples of running static (elastic solution only) cases.
$ pylith strikeslip_cubit.cfg strikeslip_hex8_1000m.cfg
$ pylith strikeslip_cubit.cfg strikeslip_hex8_0500m.cfg
$ pylith strikeslip_cubit.cfg strikeslip_tet4_1000m.cfg
# Append timedep.cfg to run the time-dependent (viscoelastic cases).
$ pylith strikeslip_cubit.cfg strikeslip_hex8_1000m.cfg timedep.cfg
$ pylith strikeslip_cubit.cfg strikeslip_hex8_0500m.cfg timedep.cfg
$ pylith strikeslip_cubit.cfg strikeslip_tet4_1000m.cfg timedep.cfg

```

This will run the problem for 10 years, using a time-step size of 0.1 years, and results will be output for each year. The benchmarks at resolutions of 1000 m, 500 m, and 250 m require approximately 150 MB, 960 MB, and 8 GB, respectively.

Benchmark Results

Fig. 4.186 shows the displacement field from the simulation with hexahedral cells using trilinear basis functions at a resolution of 1000 m. For each resolution and set of basis functions, we measure the accuracy by comparing the numerical solution against the semi-analytical Okada solution [Okada, 1992]. We also compare the accuracy and runtime across resolutions and different cell types. This provides practical information about what cell types and resolutions are required to achieve a given level of accuracy with the shortest runtime.

Solution Accuracy

We quantify the error in the finite-element solution by integrating the L2 norm of the difference between the finite-element solution and the semi-analytical solution evaluated at the quadrature points. We define the local error (error for each finite-element cell) to be

$$\epsilon_{local} = \frac{1}{V_{cell}} \sqrt{\int_{cell} (u_i^t - u_i^{fem})^2 dV}, \quad (4.507)$$

where u_i^t is the i th component of the displacement field for the semi-analytical solution, and u_i^{fem} is the i th component of the displacement field for the finite-element solution. Taking the square root of the L2 norm and normalizing by the volume of the cell results in an error metric with dimensions of length. This roughly corresponds to the error in the magnitude of the displacement field in the finite element solution. We define the global error in a similar fashion,

$$\epsilon_{global} = \frac{1}{V_{domain}} \sqrt{\int_{domain} (u_i^t - u_i^{fem})^2 dV}, \quad (4.508)$$

where we sum the L2 norm computed for the local error over all of the cells before taking the square root and dividing by the volume of the domain.

Figures Fig. 4.187 through Fig. 4.192 show the local error for each of the three resolutions and two cell types. The error decreases with decreasing cell size as expected for a converging solution. The largest errors, which approach 1 mm for 1 m of slip for a discretization size of 250 m, occur where the gradient in slip is discontinuous at the boundary between the region of uniform slip and linear taper in slip. The linear basis functions cannot match this higher order variation. The trilinear basis functions in the hexahedral element provide more terms in the polynomial defining the variation in the displacement field within each cell compared to the linear basis functions for the tetrahedral cell. Consequently, for this problem the error for the hexahedral cells at a given resolution is smaller than that for the tetrahedral cells. Both sets of cell types and basis functions provide the same rate of convergence as shown in Fig. 4.193.

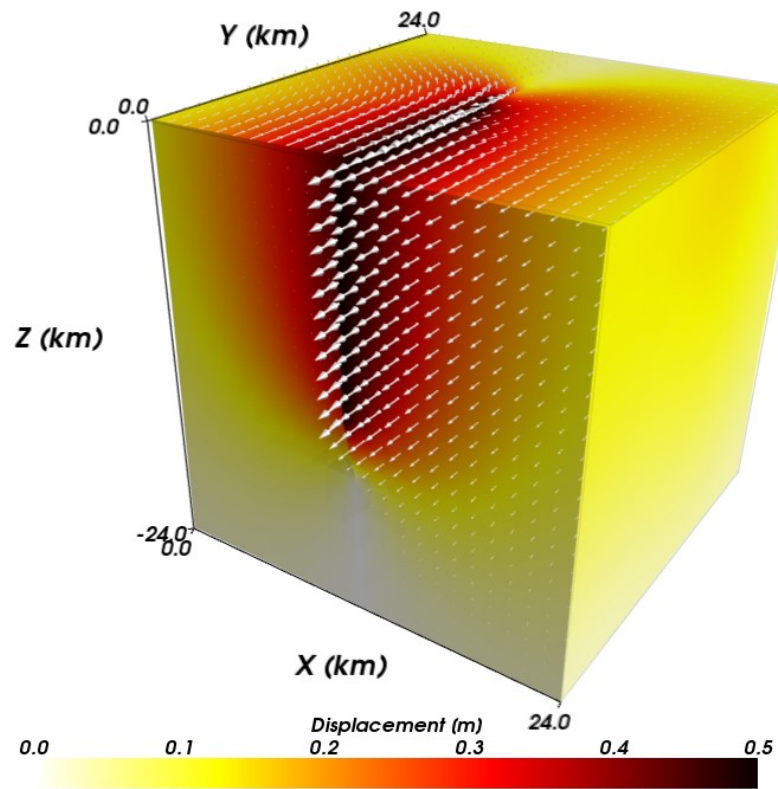


Fig. 4.186: Displacement field for strike-slip benchmark problem.

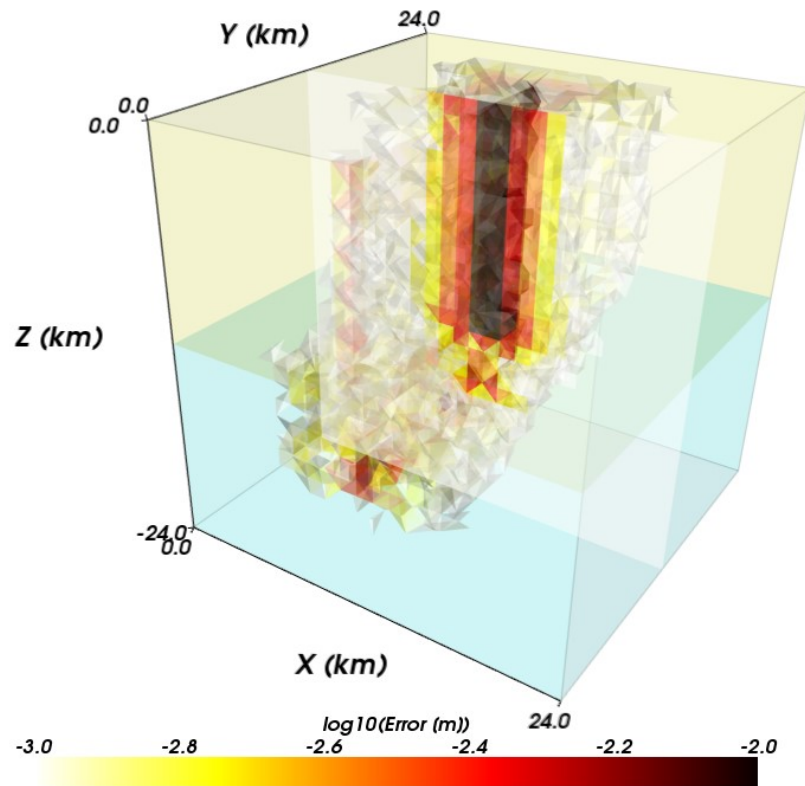


Fig. 4.187: Local error for strike-slip benchmark problem with tetrahedral cells and linear basis functions with a uniform discretization size of 1000 m.

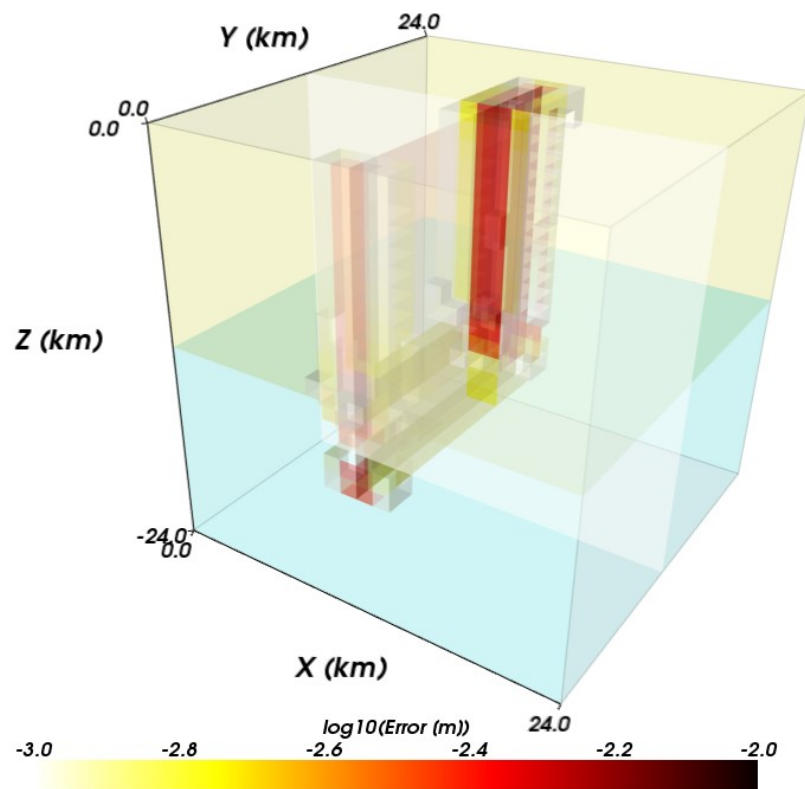


Fig. 4.188: Local error for strike-slip benchmark problem with hexahedral cells and trilinear basis functions with a uniform discretization size of 1000 m.

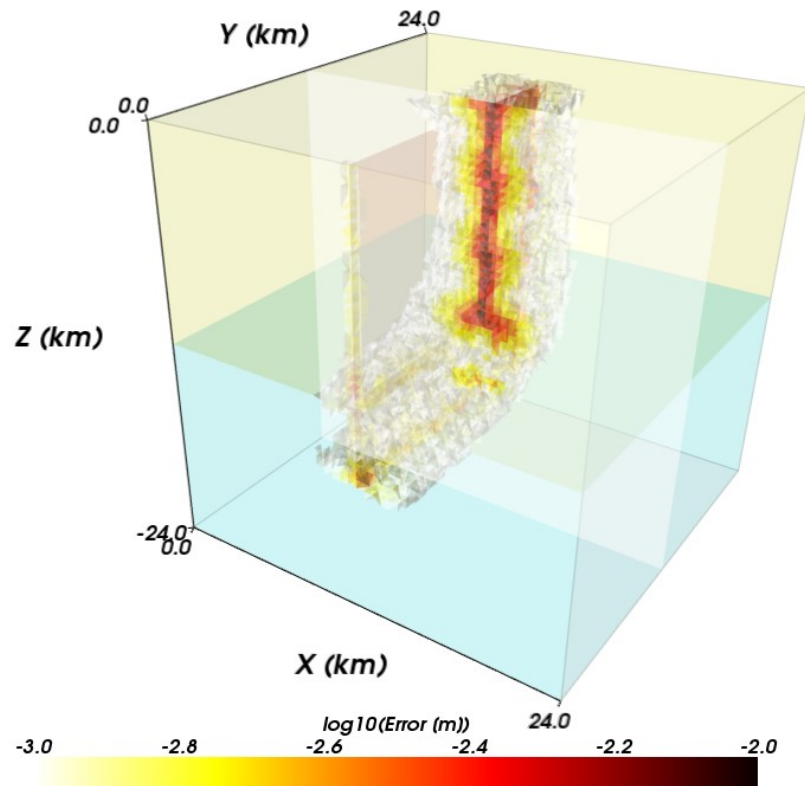


Fig. 4.189: Local error for strike-slip benchmark problem with tetrahedral cells and linear basis functions with a uniform discretization size of 500 m.

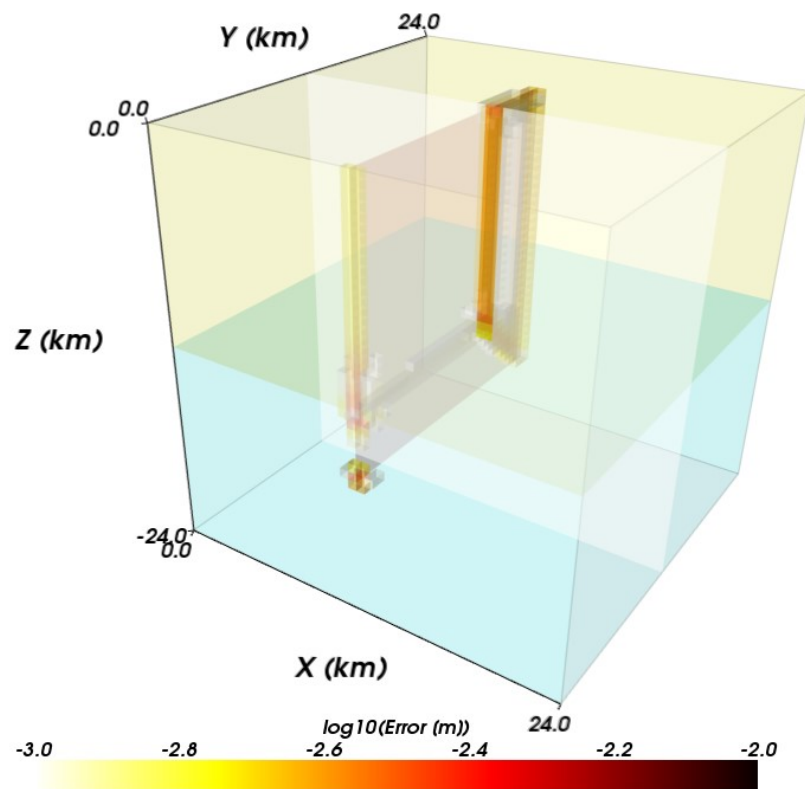


Fig. 4.190: Local error for strike-slip benchmark problem with hexahedral cells and trilinear basis functions with a uniform discretization size of 500 m.

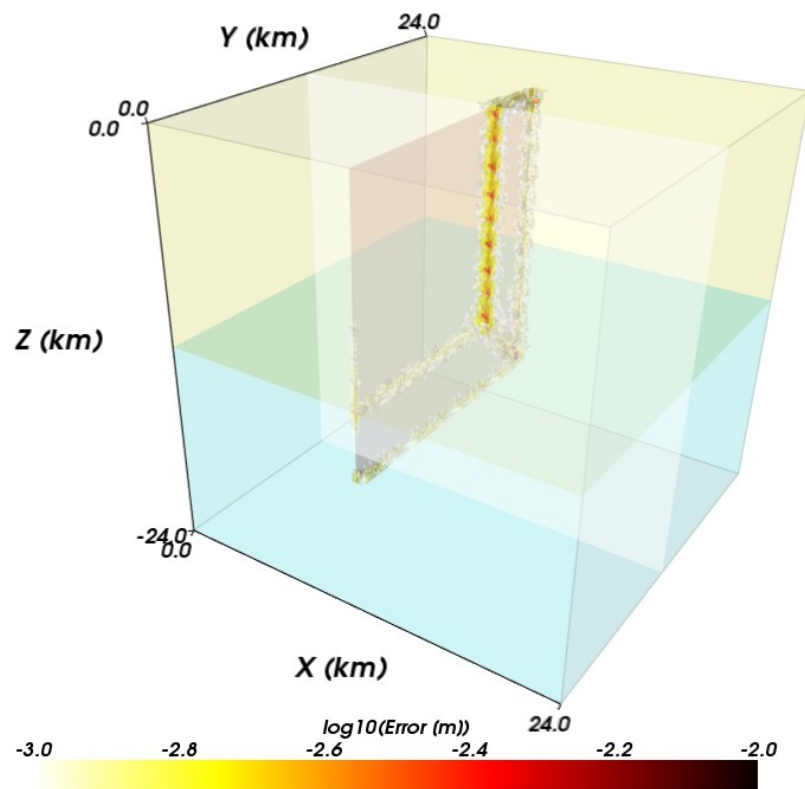


Fig. 4.191: Local error for strike-slip benchmark problem with tetrahedral cells and linear basis functions with a uniform discretization size of 250 m.

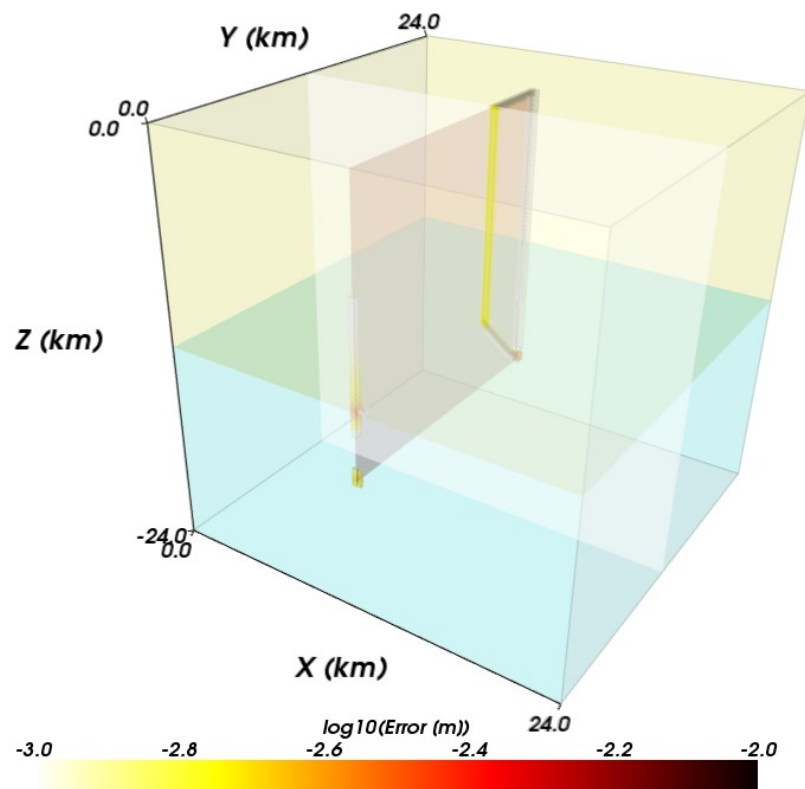


Fig. 4.192: Local error for strike-slip benchmark problem with hexahedral cells and trilinear basis functions with a uniform discretization size of 250 m.

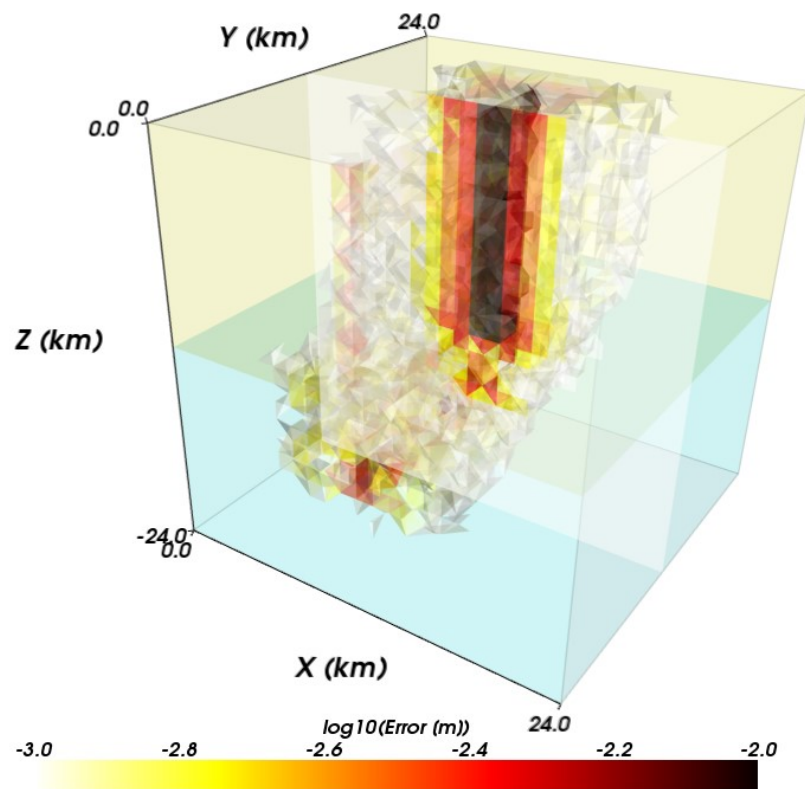


Fig. 4.193: Convergence rate for the strike-slip benchmark problem with tetrahedral cells and linear basis functions and with hexahedral cells with trilinear basis functions.

Performance

Fig. 4.194 summarizes the overall performance of each of the six simulations. Although at a given resolution, the number of degrees of freedom in the hexahedral and tetrahedral meshes are the same, the number of cells in the tetrahedral mesh is about six times greater. However, we use only one integration point per tetrahedral cell compared to eight for the hexahedral cell. This leads to approximately the same number of integration points for the two meshes, but the time required to unpack/pack information for each cell from the finite-element data structure is greater than the time required to do the calculation for each quadrature point (which can take advantage of the very fast, small memory cache in the processor). As a result, the runtime for the simulations with hexahedral cells is significantly less than that for the tetrahedral cells at the same resolution.

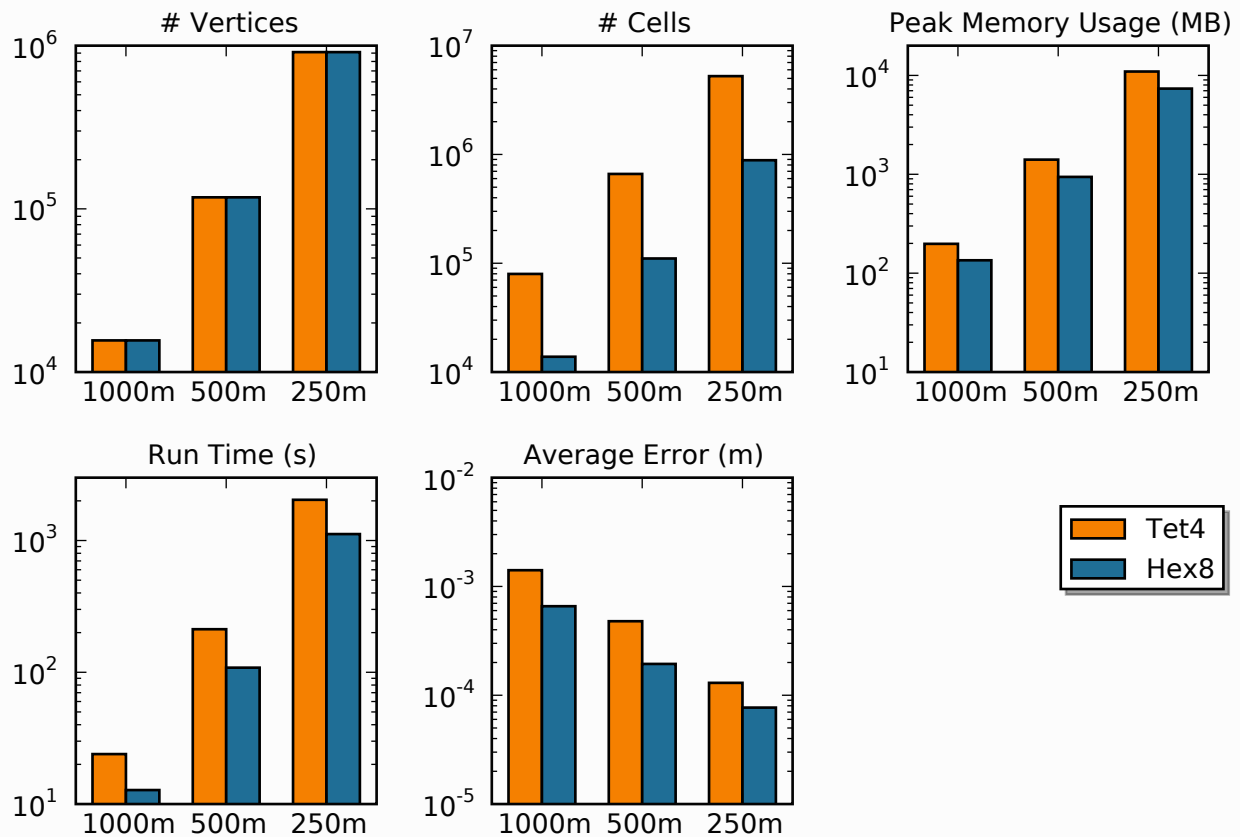


Fig. 4.194: Summary of performance of PyLith for the six simulations of the strike-slip benchmark. For a given discretization size, hexahedral cells with trilinear basis functions provide greater accuracy with a shorter runtime compared with tetrahedral cells and linear basis functions.

Fig. 4.195 compares the runtime for the benchmark (elastic solution only) at 500 m resolution for 1 to 16 processors. The total runtime is the time required for the entire simulation, including initialization, distributing the mesh over the processors, solving the problem in parallel, and writing the output to VTK files. Some initialization steps, writing the output to VTK files, and distributing the mesh are essentially serial processes. For simulations with many time steps these steps will generally occupy only a fraction of the runtime, and the runtime will be dominated by the solution of the equations. Fig. 4.195 also shows the total time required to form the Jacobian of the system, form the residual, and solve the system. These steps provide a more accurate representation of the parallel-performance of the computational portion of the code and show excellent performance as evident in the approximately linear slope of 0.7. A linear decrease with a slope of 1 would indicate strong scaling, which is rarely achieved in real applications.

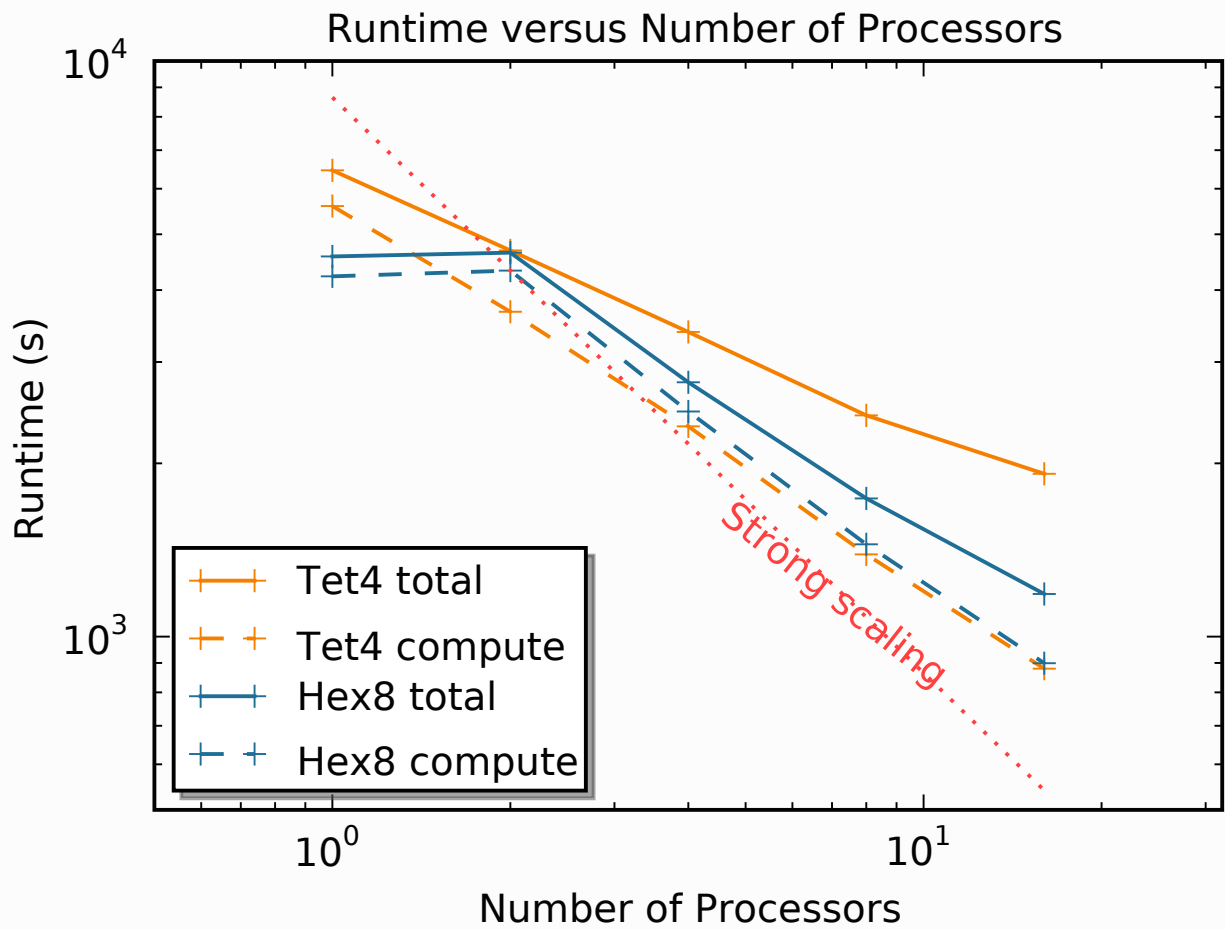


Fig. 4.195: Parallel performance of PyLith for the strike-slip benchmark problem with tetrahedral cells and linear basis functions with a uniform discretization size of 500 m. The total runtime (total) and the runtime to compute the Jacobian and residual and solve the system (compute) are shown. The compute runtime decreases with a slope of about 0.7; a linear decrease with a slope of 1 would indicate strong scaling, which is rarely achieved in any real application.

Savage and Prescott Benchmark

This benchmark problem computes the viscoelastic (Maxwell) relaxation of stresses from repeated infinite, strike-slip earthquakes in 3D without gravity. The files needed to run the benchmark may be found at https://github.com/geodynamics/pylith_benchmarks/tree/master/quasistatic/sceccrustdeform/savageprescott. An analytical solution to this problem is described by Savage and Prescott [Savage and Prescott, 1978], which provides a simple way to check our numerical solution. A python utility code is provided in the `utils` directory to compute the analytical solution. Although this problem is actually 2.5D (infinite along-strike), we solve it using a 3D finite element model.

Problem Description

Fig. 4.196 shows the geometry of the problem, as described by [Savage and Prescott, 1978]. The analytical solution describes the surface deformation due to repeated earthquakes on an infinite strike-slip fault embedded in an elastic layer overlying a Maxwell viscoelastic half-space. The upper portion of the fault (red in the figure) is locked between earthquakes, while the lower portion (blue in the figure) creeps at plate velocity. At regular recurrence intervals, the upper portion of the fault abruptly slips by an amount equal to the plate velocity multiplied by the recurrence interval, thus “catching up” with the lower part of the fault.

There are some differences between the analytical solution and our numerical representation. First, the analytical solution represents the earthquake cycle as the superposition of uniform fault creep and an elementary earthquake cycle. Uniform fault creep is simply the uniform movement of the two plates past each other at plate velocity. For the elementary earthquake cycle, no slip occurs below the locked portion of the fault (blue portion in the figure). On the locked (red) portion of the fault, backslip equal to plate velocity occurs until the earthquake recurrence interval, at which point abrupt forward slip occurs. In the finite element solution, we perform the simulation as described in the figure. Velocity boundary conditions are applied at the extreme edges of the model to simulate block motion, steady creep is applied along the blue portion of the fault, and regular earthquakes are applied along the upper portion of the fault. It takes several earthquake cycles for the velocity boundary conditions to approximate the steady flow due to steady block motion, so we would not expect the analytical and numerical solutions to match until several earthquakes have occurred. Another difference lies in the dimensions of the domain. The analytical solution assumes an infinite strike-slip fault in an elastic layer overlying a Maxwell viscoelastic half-space. In our finite element model we are restricted to finite dimensions. We therefore extend the outer boundaries far enough from the region of interest to approximate boundaries at infinity.

Due to the difficulties in representing solutions in an infinite domain, there are several meshes that have been tested for this problem. The simplest meshes have uniform resolution (all cells have equal dimensions); however, such meshes typically do not provide accurate solutions since the resolution is too coarse in the region of interest. For that reason, we also tested meshes where the mesh resolution decreases away from the center. In the problem description that follows, we will focus on the hexahedral mesh with finer discretization near the fault (`meshes/hex8_6.7km.exo.gz`), which provides a good match with the analytical solution. It will first be necessary to gunzip this mesh so that it may be used by PyLith.

Domain

The domain for this mesh spans the region

$$-1000 \leq x \leq 1000 \text{ km}, \quad (4.509)$$

$$-500 \leq y \leq 500 \text{ km}, \quad (4.510)$$

$$-400 \text{ km} \leq z \leq 0. \quad (4.511)$$

The top (elastic) layer occupies the region $-40 \text{ km} \leq z \leq 0$ and the bottom (viscoelastic) layer occupies the region $-400 \text{ km} \leq z \leq -40 \text{ km}$.

Material properties

The material is a Poisson solid with a shear modulus (μ) of 30 GPa. The domain is modeled using an elastic isotropic material for the top layer and a Maxwell viscoelastic material for the bottom layer. The bottom layer has a viscosity (η) of $2.36682\text{e}+19$ Pa-s, yielding a relaxation time ($2\eta/\mu$) of 50 years.

Fault

The fault is a vertical, left-lateral strike-slip fault. The strike is parallel to the y-direction at the center of the model:

$$x = 0 \text{ km}, \quad (4.512)$$

$$-500 \leq y \leq 500 \text{ km}, \quad (4.513)$$

$$-40 \text{ km} \leq z \leq 0. \quad (4.514)$$

The locked portion of the fault (red section in Fig. 4.196) extends from $-20 \text{ km} \leq z \leq 0$, while the creeping section (blue) extends from $-40 \text{ km} \leq z \leq 0$. Along the line where the two sections coincide ($z = -20 \text{ km}$), half of the coseismic displacement and half of the steady creep is applied (see `finalslip.spatialdb` and `creeprate.spatialdb`).

Boundary conditions

On the bottom boundary, vertical displacements are set to zero, while on the y-boundaries the x-displacements are set to zero. On the x-boundaries, the x-displacements are set to zero, while constant velocities of +/- 1 cm/yr are applied in the y-direction, giving a relative plate motion of 2 cm/year.

Discretization

For the nonuniform hexahedral mesh, the resolution at the outer boundaries is 20 km. An inner region is then put through one level of refinement, so that near the center of the mesh the resolution is 6.7 km. All meshes were generated with Cubit.

Basis functions

We use trilinear hexahedral cells.

Solution

We compute the surface displacements and compare these to the analytical solution in Fig. 4.197.



Fig. 4.196: Problem description for the Savage and Prescott strike-slip benchmark problem.

Running the Benchmark

There are a number of `.cfg` files corresponding to the different meshes, as well as a `pylithapp.cfg` file defining parameters common to all problems. Each problem uses four `.cfg` files: `pylithapp.cfg`, `fieldsplit.cfg` (algebraic multigrid preconditioner), a cell-specific file (e.g., `hex8.cfg`), and a resolution specific file (e.g., `hex8_6.7km.cfg`).

```
# If you have not do so already, checkout the benchmarks from the CIG Git repository.
$ git clone https://github.com/geodynamics/pylith_benchmarks.git
# Change to the quasistatic/sceccrustdeform/savageprescott directory.
$ cd quasistatic/sceccrustdeform/savageprescott
# Decompress the gzipped files in the meshes directory.
$ gunzip meshes/*.gz
# Run one of the simulations.
$ pylith hex8.cfg hex8_6.7km.cfg fieldsplit.cfg
```

Each simulation uses 10 earthquake cycles of 200 years each, using a time-step size of 10 years, for a total simulation time of 2000 years. Ground surface output occurs every 10 years, while all other outputs occur every 50 years.

Once the problem has run, results will be placed in the output directory. These results may be viewed directly using 3-D visualization software such as ParaView; however, to compare results to the analytical solution, some postprocessing is required. First, generate the analytical results by running the `calc_analytic.py` script. This will produce files with displacements and velocities (`analytic_disp.txt` and `analytic_vel.txt`) in the `output` directory that are easy to use with a plotting package, such as matplotlib or Matlab.

Benchmark Results

Fig. 4.197 shows the computed surface displacements for the 10th earthquake cycle compared with the analytical solution. The profile results were obtained as described above, and then all results (analytical and numerical) were referenced to the displacements immediately following the last earthquake. We find very good agreement between the analytical and numerical solutions, even for meshes with uniform refinement. We have not yet explored quantitative fits as a function of mesh resolution. For this benchmark, it is also important to consider the distance of the boundary from the region of interest. Also note that the agreement between analytical and numerical solutions is poor for early earthquake cycles, due to the differences in simulating the problem, as noted above.

SCEC Dynamic Rupture Benchmarks

The SCEC website <https://strike.scec.org/cvws/cgi-bin/cvws.cgi> includes a graphical user interface for examining the benchmark results. Benchmark results for PyLith are available for TPV205-2D (horizontal slice through a vertical strike-slip fault), TPV205 (vertical strike-slip fault with high and low stress asperities), TPV210-2D (vertical slice through a 60-degree dipping normal fault), TPV210 (60-degree dipping normal fault), TPV11, TPV12, TPV13, TPV14-2D and TPV15-2D (horizontal slice through a vertical strike-slip fault with a branch), TPV14, TPV15, TPV24, TPV25 (vertical strike-slip fault with a branch), TPV16 and 17 (vertical strike-slip fault with spatially heterogeneous initial tractions), TPV22 and 23 (vertical strike-slip fault with a stepover), TPV102 (vertical strike-slip fault with rate-state friction).

The benchmark results indicate that triangular and tetrahedral cells generate less numerical noise than quadrilateral or hexahedral cells. The input files in the repository are updated for PyLith v2.0.0, so you will need to modify them if you use another version of PyLith.

4.2.11 Glossary

Pyre

component

Basic building block of a Pyre application. A component may be built-up from smaller building blocks, where simple data types are called properties and data structures and objects are called facilities. In general a component

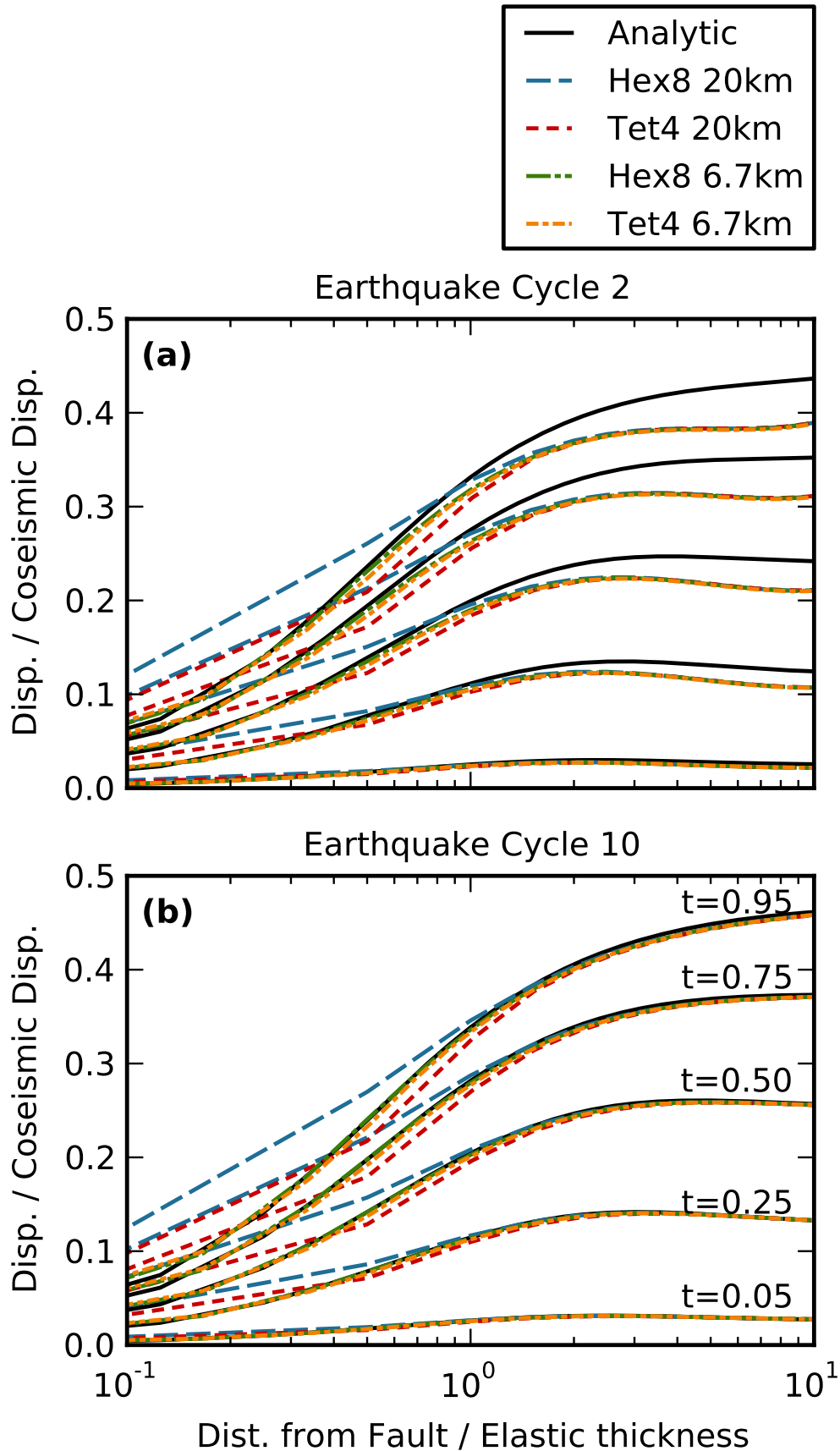


Fig. 4.197: Displacement profiles perpendicular to the fault for a PyLith simulation with hex8 cells and the analytical solution for earthquake cycle 10.

is a specific implementation of the functionality of a facility. For example, SimpleDB is a specific implementation of the spatial database facility. A component is generally composed of a Python object and a C++ object, although either one may be missing. We nearly always use the naming convention such that for an object called Foo the Python object is in file Foo.py, the C++ class definition is in Foo.hh, inline C++ functions are in foo.icc, the C++ class implementation is in Foo.cc, and the SWIG interface file that glues the C++ and Python code together is in Foo.i.

facility

Complex data type (object or data structure) building block of a component. See “component”.

property

Simple data type (string, integer, real number, or boolean value) parameter for a component.

DMPlux

The plex construction is a representation of the topology of the finite-element mesh based upon a covering relation. For example, segments are covered by their endpoints, faces by their bounding edges, etc. Geometry is absent from the plex, and is represented instead by a field with the coordinates of the vertices. Meshes can also be understood as directed acyclic graphs, where we call the constituents points and arrows. See *PETSc Finite-Element Implementation* for additional details.

cell

The highest dimensional elements of a mesh, or mesh entities of codimension zero.

cone

The set of entities which cover any entity in a mesh. For example, the cone of a triangle is its three edges.

dimension

The topological dimension of the mesh, meaning the cell dimension. It can also mean the dimension of the space in which the mesh is embedded, but this is properly called the embedding dimension.

dual space

For any vector space V over a field F , the dual space V^* is the set of all linear functions $\phi : V \rightarrow F$. If V is finite dimensional, then V^* has the same dimension as V . See https://en.wikipedia.org/wiki/Dual_space and https://https://finite-element.github.io/L2_fespaces.html.

face

Mesh elements that separate cells, or mesh entities of codimension one.

field

A parallel section which can be completed, or made consistent, across process boundaries. These are used to represent continuum fields.

mesh

A finite element mesh, used to partition space and provide support for the basis functions.

projection

Interpolation of analytical or discretized functions into the finite-element space. See *Projection* for more information.

section

These objects associate values in vectors to points (vertices, edges, faces, and cells) in a mesh. The section describes the offset into the vector along with the number of values associated with each point.

support

The set of mesh entities which are covered by any entity in a mesh. For example, the support of a triangle is the two tetrahedra it separates.

vertex

The zero dimensional mesh elements.

PyLith

auxiliary field

A field used to specify parameters and state variables for physics. For an elastic materials the auxiliary field contains the elastic properties. For a Dirichlet boundary condition the auxiliary field contains the parameters specifying the solution as a function of space and time.

basis order

Order of the polynomial basis functions used to represent a field.

cohesive cell

A zero volume cell inserted between two cells which share a fault face. They are prisms with a fault face as the base.

derived field

A field that can be computed from the solution field. For example, the infinitesimal strain can be computed from the displacement field.

quadrature order

Order of the quadrature scheme used in numerical integration. The integrals in the weak form are integrated using numerical quadrature. The location of the quadrature points are usually found to achieve a desired order of accuracy in integrated fields.

solution field

A field with subfields for the unknown values that will be determined by solving a system of equations.

spatial database

Data specifying values of one or more fields as a function of space. The topology and resolution of the spatial database is set to match the data and is independent of the finite-element mesh. The `spatialdata` library contains a few different implementations of spatial databases.

4.2.12 File Formats

PyLith Mesh ASCII File

PyLith mesh ASCII files allow quick specification of the mesh information for very small, simple meshes that are most easily written by hand. We do not recommend using this format for anything other than these very small, simple meshes.

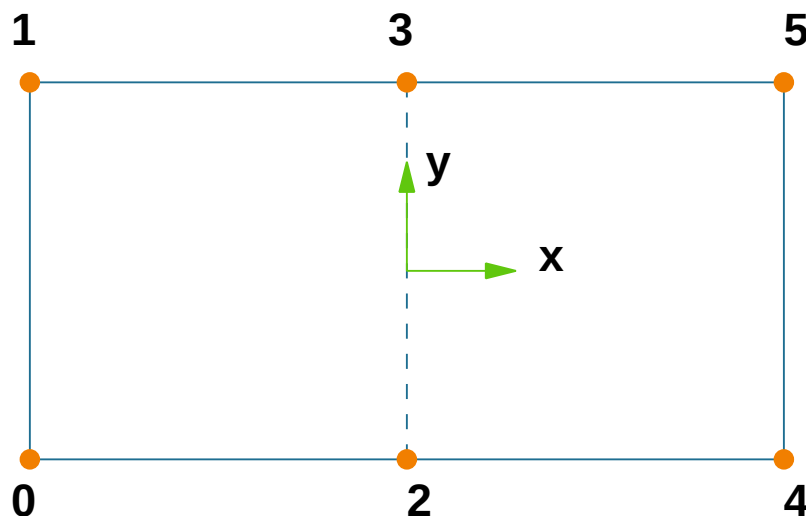


Fig. 4.198: Diagram of mesh specified in the file.

```

// This mesh file defines a finite-element mesh composed of two
// square cells of edge length 2.
//
// Comments can appear almost anywhere in these files and are
// delimited with two slashes (//) just like in C++. All text and
// whitespace after the delimiter on a given line is ignored.
mesh = { // begin specification of the mesh
    dimension = 2 // spatial dimension of the mesh
    // Begin vertex and cell labels with 0. This is the default so
    // this next line is optional
    use-index-zero = true

    vertices = { // vertices or nodes of the finite-element cells
        dimension = 2 // spatial dimension of the vertex coordinates
        count = 6 // number of vertices in the mesh
        coordinates = { // list of vertex index and coordinates
            // the coordinates must coincide with the coordinate
            // system specified in the Mesh object
            // exactly one vertex must appear on each line
            // (excluding whitespace)
            0 -2.0 -1.0
            1 -2.0 +1.0
            2 0.0 -1.0
            3 0.0 +1.0
            4 +2.0 -1.0
            5 +2.0 +1.0
        } // end of coordinates list
    } // end of vertices

    cells = { // finite-element cells
        count = 2 // number of cells in the mesh
        num-corners = 4 // number of vertices defining the cell
        simplices = { // list of vertices in each cell
            // see Section 4.2 for diagrams giving the order for each
            // type of cell supported in PyLith
            // index of cell precedes the list of vertices for the cell
            // exactly one cell must appear on each line
            // (excluding whitespace)
            0 0 2 3 1
            1 4 5 3 2
        } // end of simplices list

        material-ids = { // associated each cell with a material model
            // the material id is specified using the index of the cell
            // and then the corresponding material id
            0 0 // cell 0 has a material id of 0
            1 2 // cell 1 has a material id of 2
        } // end of material-ids list
    } // end of cells

    // This next section lists groups of vertices that can be used
    // in applying boundary conditions to portions of the domain
    vertex-group = { // start of a group

```

(continues on next page)

(continued from previous page)

```

// the name can have whitespace, so no comments are allowed
// after the name
name = vertices_negy
count = 3 // number of vertices in the group
indices = { // list of vertex indices in the group
    // multiple vertices may appear on a line
    0 2 4 // this group contains vertices 0, 2, 4
} // end of list of vertices
} // end of group

// This next section lists groups of cell faces that can be used
// in applying boundary conditions to portions of the domain
face-group = { // start of a group
    // the name can have whitespace, so no comments are allowed
    // after the name
    name = faces_negy
    count = 2 // number of faces in the group
    indices = { // list of cell and vertices on face in the group
        // only 1 face per line
        0 0 2 // cell 0, face with vertices 0 and 2
        1 2 5 // cell 1, face with vertices 2 and 5
    } // end of list of vertices
} // end of group

// additional groups can be listed here

```

Points List File

This file lists the coordinates of the locations where output is requested for the OutputSolnPoints component. The coordinate system is specified in the OutputSolnPoints component.

```

# Comments are limited to complete lines. The default delimiter for comments
# is '#', which can be changed via parameters. Additionally, the delimiter
# separating values can also be customized (default is whitespace).
#
# The first column is the station name. The coordinates of the points are given
# in the subsequent columns.
P0 1.0 -2.0 0.0
P1 2.0 -4.0 -0.1
P2 0.0 +2.0 0.0
P3 2.5 -0.2 -0.2
P4 0.0 2.0 +0.2

```

4.2.13 Appendices

Analytical Solutions

Airy Stress Functions

Airy stress functions provide a simple approach for solving 2D elastoplastic problems with uniform isotropic linearly elastic material properties. They can be useful in creating tests using the Method of Manufactured Solutions.

We start with the equilibrium equation for static elasticity in Cartesian coordinates

$$\begin{aligned}\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + f_x &= 0 \\ \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{xy}}{\partial x} + f_y &= 0,\end{aligned}\tag{4.515}$$

where f_x and f_y are the body force components in the x and y directions, respectively. We assume the body forces can be derived from a potential ψ

$$f_x = -\frac{\partial \psi}{\partial x},\tag{4.516}$$

$$f_y = -\frac{\partial \psi}{\partial y}.\tag{4.517}$$

We choose an Airy stress function $\phi(x, y)$ to trivially satisfy the equilibrium equations so that

$$\begin{aligned}\sigma_{xx} &= \frac{\partial^2 \phi}{\partial y^2} + \psi, \\ \sigma_{yy} &= \frac{\partial^2 \phi}{\partial x^2} + \psi, \\ \sigma_{xy} &= -\frac{\partial^2 \phi}{\partial x \partial y}.\end{aligned}\tag{4.518}$$

We must also satisfy the compatibility equations. For plane strain, we have

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) (\sigma_{xx} + \sigma_{yy}) - \frac{1}{1-\nu} \left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right) = 0.\tag{4.519}$$

Substituting in the expressions for the stress components in terms of the Airy stress function leads to

$$\frac{\partial^4 \phi}{\partial x^4} + \frac{\partial^4 \phi}{\partial x^2 \partial y^2} + \frac{\partial^4 \phi}{\partial y^4} + \frac{1-2\nu}{1-\nu} \left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right) = 0.\tag{4.520}$$

Example

We select a second order polynomial for the Airy stress function with the form

$$\phi = \frac{1}{2}ax^2 + bxy + \frac{1}{2}cy^2.\tag{4.521}$$

and no body forces ($\psi = 0$). By inspection we see that this equation trivially satisfies the equilibrium equation (4.520). Using equation (4.518), we have

$$\sigma_{xx} = c,\tag{4.522}$$

$$\sigma_{yy} = a,\tag{4.523}$$

$$\sigma_{xy} = -b.\tag{4.524}$$

Our stress function corresponds to a uniform stress field.

Let us now consider axial extension of a rectangular block with roller boundary conditions on two sides as shown in Fig. 4.199. We have

$$\tau_x = \tau_0 \text{ on } x = x_1,\tag{4.525}$$

$$u_x = 0 \text{ on } x = x_0,\tag{4.526}$$

$$u_y = 0 \text{ on } y = y_0.\tag{4.527}$$

The other boundaries are free surfaces. Because $\sigma_{yy} = \sigma_{xy} = 0$ and $\sigma_{xx} = \tau_0$, we have $a = b = 0$ and $c = \tau_0$. For plane strain the out of plane stress is given by

$$\sigma_{zz} = \frac{\lambda}{2\lambda + 2\mu} (\sigma_{xx} + \sigma_{yy}) = \frac{\lambda}{2\lambda + 2\mu} \tau_0. \quad (4.528)$$

The strain components for plane strain are

$$\epsilon_{xx} = \frac{1}{2\mu} \left(\sigma_{xx} - \frac{\lambda}{3\lambda + 2\mu} (\sigma_{xx} + \sigma_{yy} + \sigma_{zz}) \right), \quad (4.529)$$

$$\epsilon_{yy} = \frac{1}{2\mu} \left(\sigma_{yy} - \frac{\lambda}{3\lambda + 2\mu} (\sigma_{xx} + \sigma_{yy} + \sigma_{zz}) \right), \quad (4.530)$$

$$\epsilon_{xy} = \frac{1}{2\mu} \sigma_{xy}. \quad (4.531)$$

Substituting in our expressions for the stress components leads to

$$\epsilon_{xx} = \frac{\tau_0}{4\mu} \frac{\lambda + 2\mu}{\lambda + \mu}, \quad (4.532)$$

$$\epsilon_{yy} = -\frac{\tau_0}{4\mu} \frac{\lambda}{\lambda + \mu}, \quad (4.533)$$

$$\epsilon_{xy} = 0. \quad (4.534)$$

The corresponding displacement field is

$$u_x = \frac{\tau_0}{4\mu} \frac{\lambda + 2\mu}{\lambda + \mu} x, \quad (4.535)$$

$$u_y = -\frac{\tau_0}{4\mu} \frac{\lambda}{\lambda + \mu} y. \quad (4.536)$$

Putting everything together the solution to our boundary value problem is

$$\sigma_{xx} = \tau_0, \quad (4.537)$$

$$\sigma_{yy} = 0, \quad (4.538)$$

$$\sigma_{xy} = 0, \quad (4.539)$$

$$\sigma_{zz} = \frac{\lambda}{2\lambda + 2\mu} \tau_0, \quad (4.540)$$

$$\epsilon_{xx} = \frac{\tau_0}{4\mu} \frac{\lambda + 2\mu}{\lambda + \mu}, \quad (4.541)$$

$$\epsilon_{yy} = -\frac{\tau_0}{4\mu} \frac{\lambda}{\lambda + \mu}, \quad (4.542)$$

$$\epsilon_{xy} = 0, \quad (4.543)$$

$$u_x = \frac{\tau_0}{4\mu} \frac{\lambda + 2\mu}{\lambda + \mu} x, \quad (4.544)$$

$$u_y = -\frac{\tau_0}{4\mu} \frac{\lambda}{\lambda + \mu} y. \quad (4.545)$$

4.3 Developer Guide

4.3.1 PyLith design

Design overview

The PyLith software suite is composed of a C++ library, Python modules, a Python application, and a few Python preprocessing and post-processing utilities.

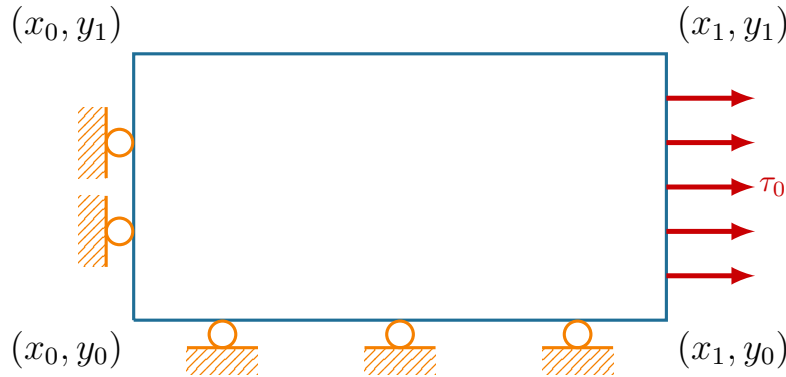


Fig. 4.199: Diagram of axial extension example for the Airy stress function. We constraint the normal degree of freedom on the $x = x_0$ and $y = y_0$ boundaries and apply a uniform normal traction τ_0 on $x = x_1$.

Boundary between Python and C++

We use the Pyre framework (written in Python) to collect all user parameters and to launch the MPI application. As a result, the top-level code is written in Python. In most cases there is a low-level C++ object of the same name with the low-level implementation of the object.

We limit the Python code to collection of the user parameters, some simple checking of the parameters, and passing the parameters to the corresponding C++ objects. Everything else is done in C++. This facilitates debugging (it is easier to track symbols in the C/C++ debugger) and unit testing, and reduces the amount of information that needs to be passed from Python to C++. The PyLith application and a few other utility functions, like writing the parameter file, are limited to Python. All other objects have a C++ implementation. Objects that have user input collect the user input in Python using Pyre and pass it to a corresponding C++ object. Objects that do not have user input, such as the integrators and constraints, are limited to C++.

Important

Consistent inheritance between C++ and Python is important in order for SWIG to generate a Python interface that is consistent with the C++ interface.

PyLith Application Flow

The PyLith application driver performs two main functions. First, it collects all user parameters from input files (e.g., .cfg files) and the command line, and then it performs from simple checks on the parameters. Second, it launches the MPI job.

Once the MPI job launches, the application flow is:

1. Read the finite-element mesh; `pylith.meshio.MeshImporter`.
 1. Read the mesh (serial); `pylith::meshio::MeshIO`.
 2. Reorder the mesh, if desired; `pylith::topology::ReverseCuthillMcKee`.
 3. Insert cohesive cells as necessary (serial); `pylith::faults::FaultCohesive`.
 4. Distribute the mesh across processes (parallel); `pylith::topology::Distributor`.
 5. Refine the mesh, if desired (parallel); `pylith::topology::RefineUniform`.
2. Setup the problem.

1. Preinitialize the problem by passing information from Python to C++ and doing minimal setup `pylith.Problem.preinitialize()`.
2. Perform consistency checks and additional checks of user parameters; `pylith.Problem.verifyConfiguration()`.
3. Complete initialization of the problem; `pylith::problems::Problem::initialize()`.
3. Run the problem; `pylith.problems.Problem.run()`.
4. Cleanup; `pylith.problems.Problem.finalize()`.
 1. Close output files.
 2. Deallocate memory.
 3. Output PETSc log summary, if desired.

In the first step, we list the object performing the work, whereas in subsequent steps we list the top-level object method responsible for the work. Python objects are listed using the `path.class` syntax while C++ objects are listed using `namespace::class` syntax. Note that a child class may redefine or perform additional work compared to what is listed in the parent class method.

Reading the mesh and the first two steps of the problem setup are controlled from Python. That is, at each step Python calls the corresponding C++ methods using SWIG. Starting with the complete initialization of the problem, the flow is controlled at the C++ level.

Time-Dependent Problem

In a time-dependent problem the PETSc TS object (relabelled `PetscTS` within PyLith) controls the time stepping. Within each time step, the `PetscTS` object calls the PETSc linear and nonlinear solvers as needed, which call the following methods of the C++ `pylith::problems::TimeDependent` object as needed: `computeRHSResidual()`, `computeLHSResidual()`, and `computeLHSJacobian()`. The `pylith::problems::TimeDependent` object calls the corresponding methods in the boundary conditions, constraints, and materials objects. At the end of each time step, it calls `problems::TimeDependent::poststep()`.

Code Layout

Directory Structure

The C++, Python, and SWIG Python/C++ interface files all sit in different directories. Similarly, the unit tests, MMS tests, full-scale tests, examples and documentation are also in their own directories.

```
pylith/
├── ci-config # continuous integration testing configuration
├── docker # Dockerfiles for creating Docker images
├── developer # Utilities for developers
├── docs # Documentation source code
├── libsrc # C++ source for library
├── modulesrc # SWIG interfaces files for C++ Python bindings.
├── pylith # Python source code for PyLith modules.
├── applications # Source code for command line programs
├── m4 # Autoconf macros
├── share # Common parameter settings
├── examples # Example suite
├── tests
│   ├── libtests # C++ unit tests
│   └── mms tests # C++ Method of Manufactured solution tests
```

(continues on next page)

(continued from previous page)

```
├─ pytest # Python unit tests
├─ fullscale # Automated full-scale tests
└─ manual # Manual full-scale tests
```

The C++ library, SWIG interface files, and Python modules are organized into several subpackages with common names.

- **bc** Boundary conditions.
- **faults** Faults.
- **feassemble** General finite-element formulation.
- **fekernels** Finite-element pointwise functions (kernels).
- **friction** Fault constitutive models.
- **materials** Material behavior, including bulk constitutive models.
- **meshio** Input and output.
- **problems** General problem formulation.
- **testing** Common testing infrastructure.
- **topology** Finite-element mesh topology.
- **utils** General utilities.

4.3.2 PyLith implementation

Code Structure

Legend for class diagrams

- **Abstract classes** are shown in the yellow boxes.
- **Concrete classes** are shown in the green boxes.
- **Inheritance** is denoted by an arrow.
- **Aggregation** is denoted by the a diamond and arrow.

Application

Problem

Solution

Physics and Finite-Element Objects

We separate the specification of the physics from the finite-element operations. That is, we have one set of objects that specify the physics through materials, boundary conditions, and faults; another set of objects perform the finite-element operations required to solve the equations. [Fig. 4.203](#) illustrates this separation. The user specifies the parameters for the Physics objects, which each create the appropriate integrator and/or constraint via factory methods.

We generalize the finite-element operations into two main classes: `Integrator` and `Constraint`. The `Integrator` is further separated into concrete classes for performing the finite-element integrations over pieces of the domain (`IntegratorDomain`), pieces of the domain boundary (`IntegratorBoundary`), and interior interfaces (`IntegratorInterface`). We implement several kinds of constraints, corresponding to how the values of the constrained degrees of freedom are specified. `ConstraintSpatialDB` gets values for the constrained degrees of freedom

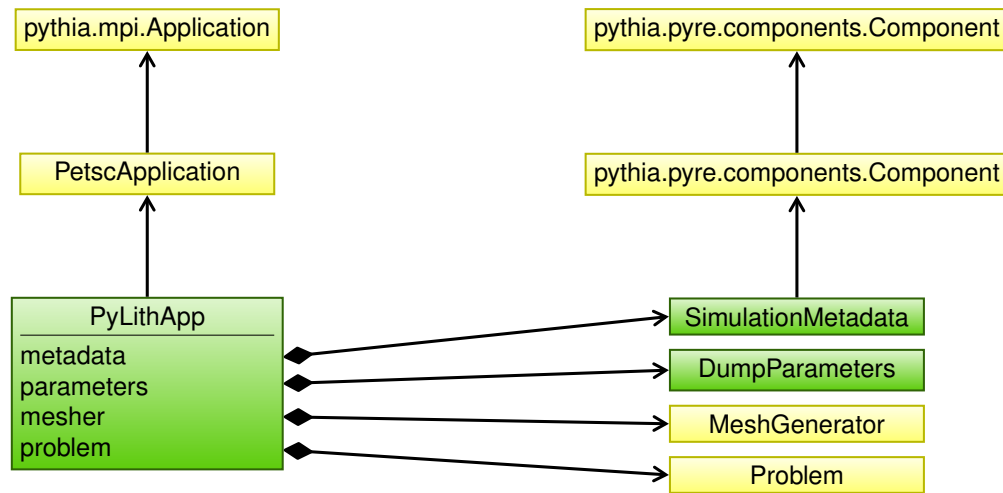
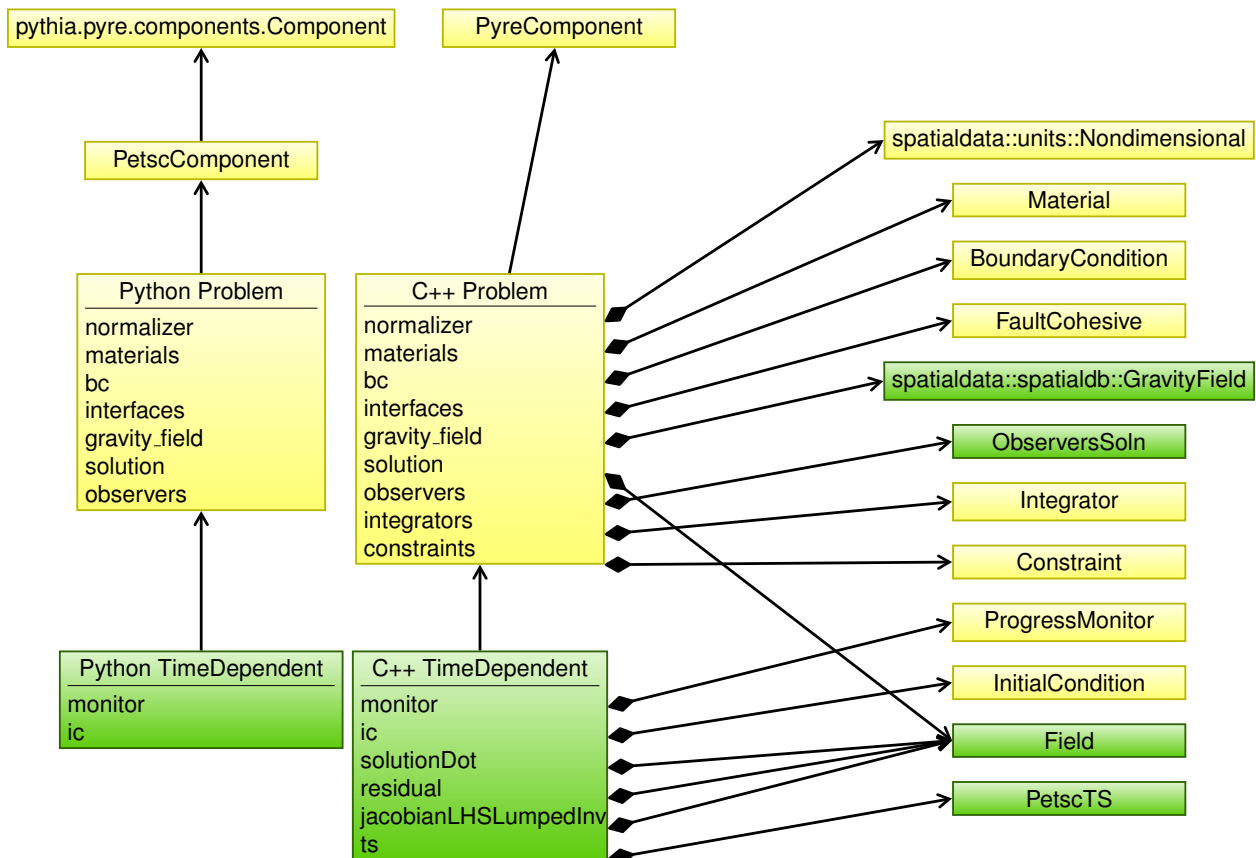
Fig. 4.200: Diagram showing the relationships among objects associated with `PyLithApp`.

Fig. 4.201: Diagram showing the relationships among the Python and C++ Problem objects and their data members.

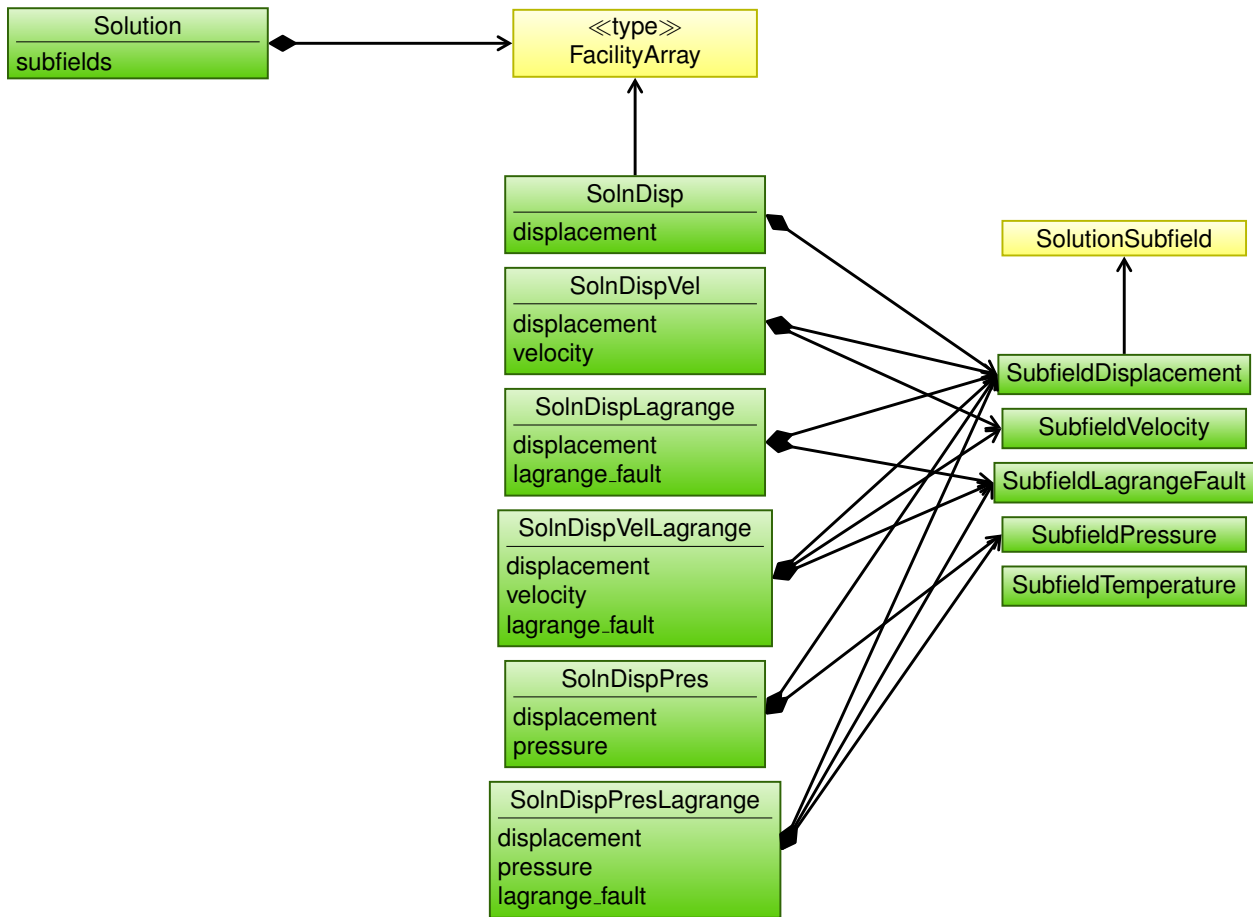


Fig. 4.202: Diagram showing the relationships among the Python and C++ Solution objects and their data members.

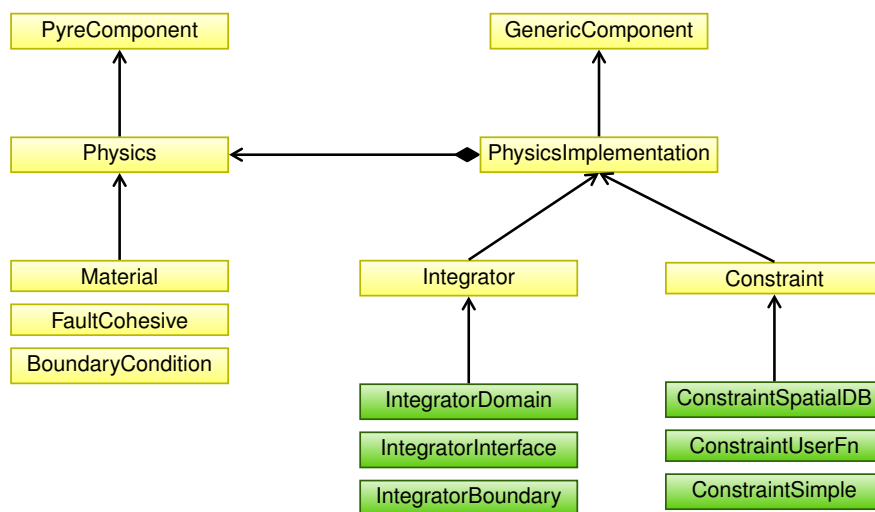


Fig. 4.203: Diagram showing the relationships among objects specifying the physics and the finite-element implementations.

from a spatial database; `ConstraintUserFn` gets the values for the constrained degrees of freedom from a function (this object is widely used in tests); `ConstraintSimple` is a special case of `ConstraintUserFn` with the constrained degrees of freedom set programmatically using a label (this object is used for constraining the edges of the fault).

`Problem` holds the `Physics` objects as materials, boundary conditions, and interfaces. During initialization of `Problem`, each `Physics` object creates any necessary `Integrator` and `Constraint` objects to implement the physics. For example, a material will create an `IntegratorDomain` object that performs integration over that material's cells.

Materials

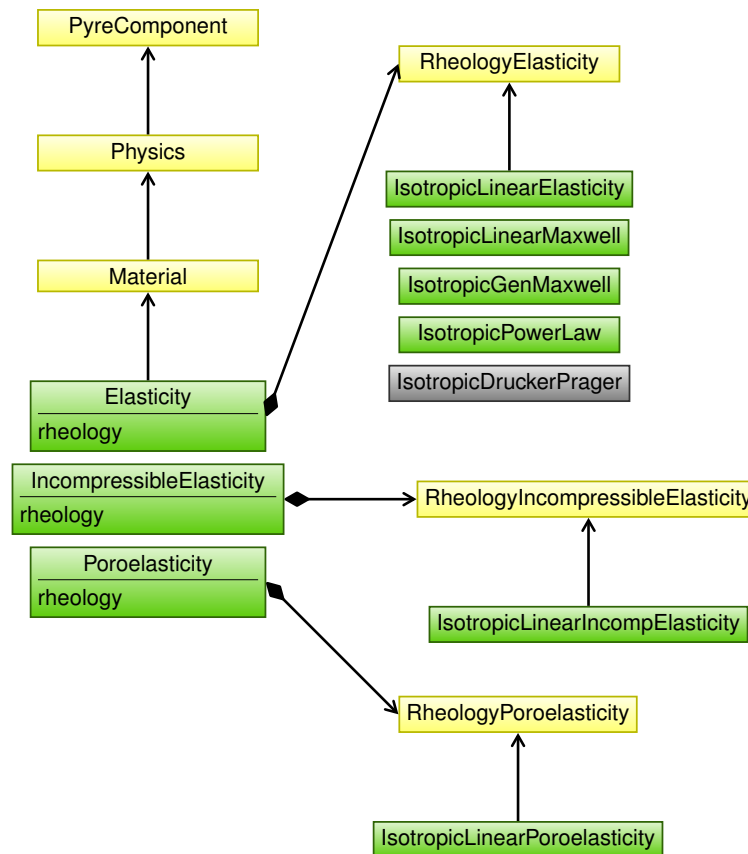


Fig. 4.204: Diagram showing the relationships among objects associated with materials.

Boundary Conditions

Interior Interfaces (Faults)

Mesh Importing

Output

PETSc Finite-Element Implementation

Formulating the weak form of the governing equation in terms of pointwise functions allows the PyLith implementation of the equations to be done at a rather high level. Most of the finite-element details are encapsulated in PETSc routines that compute the integrals and solve the system of equations. In fact, adding materials and boundary conditions requires calling only a few PETSc finite-element functions to register the point-wise functions. A new material may need to add to the library of solution subfields and auxiliary subfields, but adding these fields is also done at a high-level.

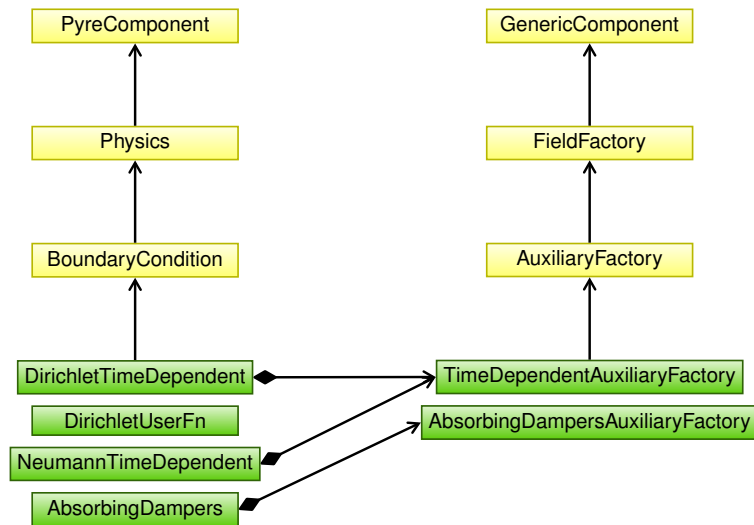


Fig. 4.205: Diagram showing the relationships among objects associated with boundary conditions.

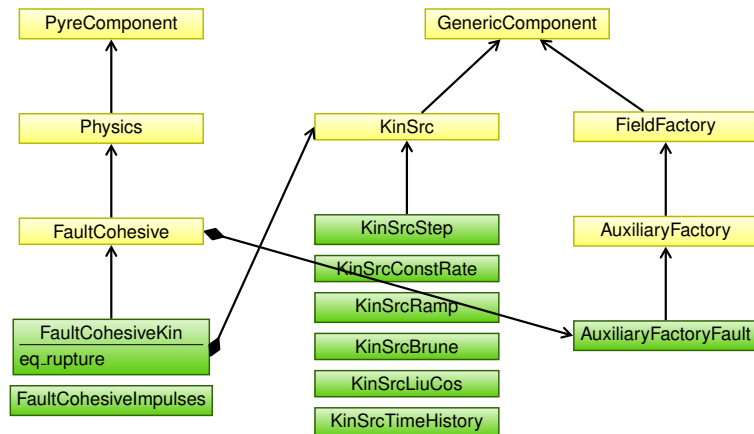


Fig. 4.206: Diagram showing the relationships among objects associated with faults.

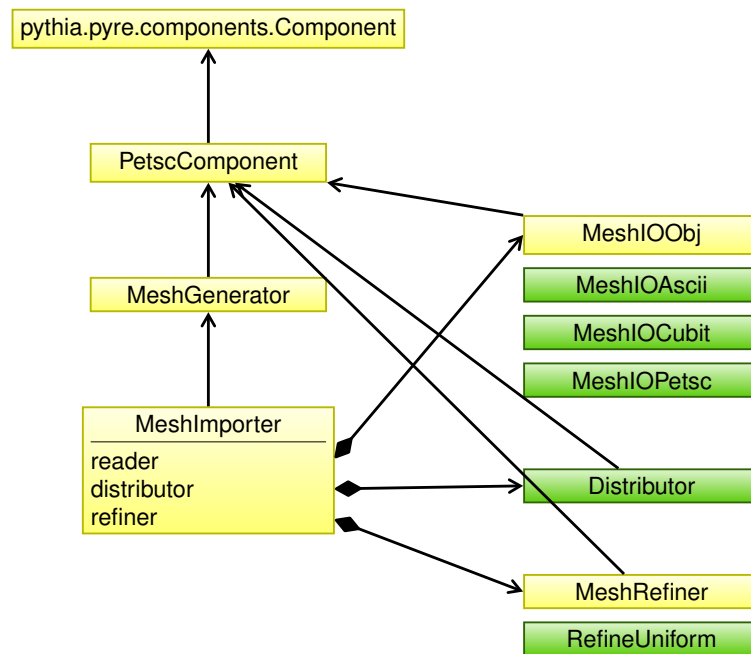


Fig. 4.207: Diagram showing the relationships among objects associated with mesh generation and importing.

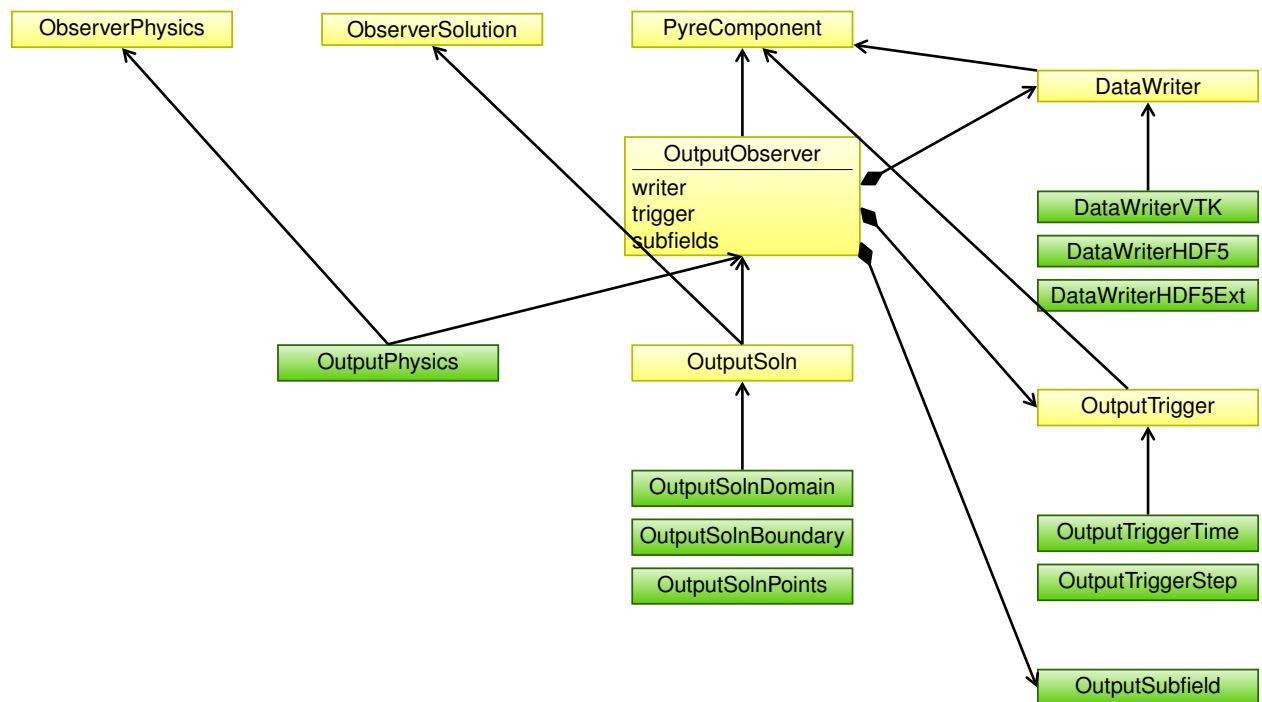


Fig. 4.208: Diagram showing the relationships among objects associated with output.

The remainder of this section discusses three aspects of the finite-element implementation handled by PyLith to give you a peek of what is going on under the hood. Refer to the *Glossary* for definitions of the terminology.

DMplex

The finite-element mesh is stored as a DMplex object. This is a particular implementation of the PETSc Data Management (DM, called PetscDM within PyLith) object. Within a DMplex object, vertices, edges, faces, and cells are all called points. The points are numbered sequentially, beginning with cells, followed by vertices, edges, and then faces. Treating all topological pieces of the mesh the same way, as points in an abstract graph, allows us to write algorithms which apply to very different meshes without change. For example, we can write a finite element assembly loop that applies to meshes of any dimension, with any cell shape, with (almost) any finite element.

Point Depth and Height

In general, vertices are at a *depth* of 0 and cells are at the maximum depth. Similarly, cells are at a *height* of 0 and vertices are at the maximum height. Notice that depth and height correspond to the usual dimension and codimension. Table 4.44 shows the heights and depths of the vertices edges, faces, and cells in a 3D mesh.

Table 4.44: Depth and height of various topological pieces in DMplex.

Point type	Depth	Height
Vertices	0	3
Edges	1	2
Faces	2	1
Cells	3	0

For a boundary mesh, we currently store the full set of points (vertices, edges, faces, and cells). Obviously for 2-D meshes, the boundary mesh doesn’t contain “volume” cells, but just vertices, edges, and faces. This means the “boundary” cells are at a height of 1 and a depth equal to the maximum depth of 1.

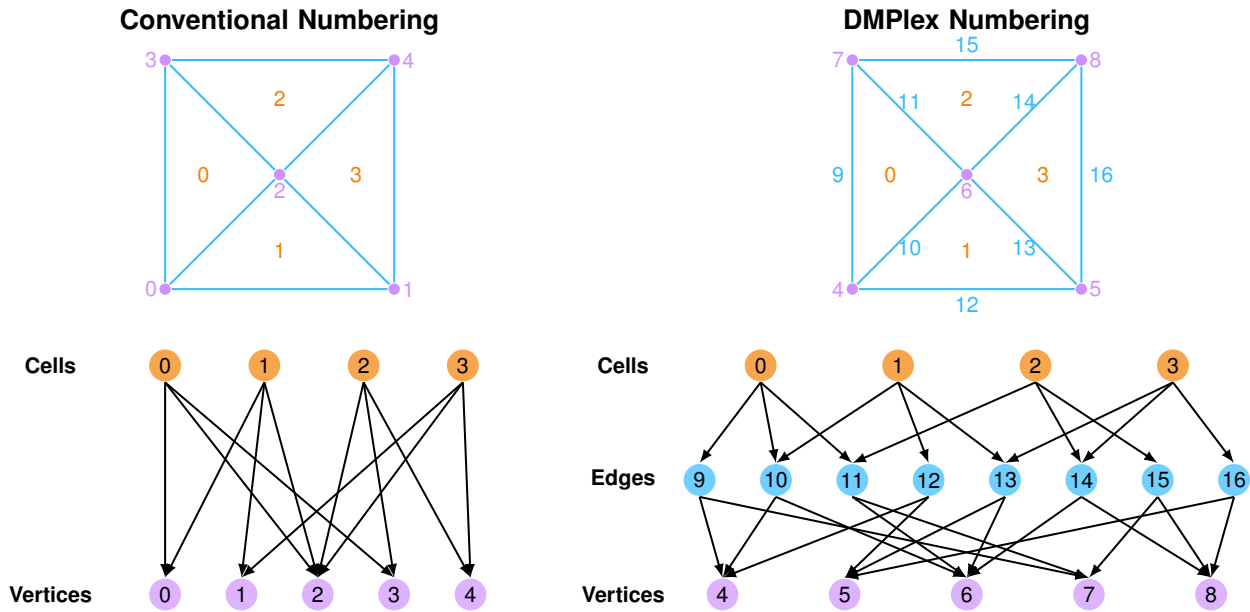


Fig. 4.209: Conventional numbering (left) with vertices and cells numbered independently and DMplex numbering (right) with cells, vertices, edges, (and faces), numbered sequentially.

pylith::topology::Field, PetscSection, and PetscVec

A PyLith `Field` (`pylith::topology::Field`) is composed of a `PetscSection`, which associates the points (vertices, edges, faces, and cells) with the finite-element coefficients, and a `PetscVec`, which is a vector storing the finite-element coefficients. A `Field` may hold a single subfield, such as displacement, or it may hold several subfields, such as the density, shear modulus, and bulk modulus for an isotropic, linear elastic material.

The chart of the `PetscSection` defines the range of points (minimum and maximum) over which the section is defined. For each point in the chart, the section holds the number of degrees of freedom a point has and the offset in the `PetscVec` for its first degree of freedom. The section also holds the number of degrees of freedom and offset for each individual subfield within the section for each point in the chart.

Because the `PetscSection` knows only about points, and not about topology or geometry, it allows us to express the mathematical concepts without getting lost in the details. For example, a P_1 3D vector field assigns 3 degrees of freedom to each vertex. This same section could be used to layout a field over a circle, surface of a cylinder, Mobius strip, sphere, cube, ball, or PyLith mesh. The section separates the layout of data over a mesh from the actual storage of values, which is done by the `PetscVec`. The section tells you how to look up the values, which are associated with a piece of the mesh, inside the vector. For example, you can ask for the offset into the vector for the values associated with an edge in the mesh, and how many degrees of freedom there are on the edge.

The `PetscSection` also includes the information about the constrained degrees of freedom. We refer to a `PetscVec` that includes the values for the constrained degrees of freedom as a *local* vector and a `PetscVec` with the values for the constrained degrees of freedom removed as a *global* vector. Constraints often arise from Dirichlet boundary conditions, which change the basis functions of the approximating space, but can also arise from algebraic relations between unknowns. A local vector is used for assembly of the residual vector and Jacobian matrix, because we need the boundary values in order to compute those integrals. Global vectors are used for the algebraic solver because we do not want solution values fixed by constraints to participate in the solve.

The `Field::view()` method will print the `Field` metadata and the `PetscSection` in a more useful format compared to the corresponding PETSc viewer functions.

Listing 4.329: Output from `pylith::topology::Field::view()` for a solution field with two subfields showing only the sections (layout of the field). The displacement subfield has degrees of freedom on the vertices of the bulk cells. The `lagrange_multiplier_fault` subfield has degrees of freedom on the edges of the cohesive cells. The order of the values (offsets in the `PetscVec`) follows the ordering of the points (cells, vertices, edges, and faces). The local section includes the constrained degrees of freedom, whereas the global section does not. The sections list the point and then the degree of freedom associated with the components of the fields.

```
Viewing field 'DM_0x84000000_1 solution' Solution field.
Subfields:
  Subfield displacement, index: 0, components: displacement_x displacement_y, scale: 1.000, basisOrder: 1, quadOrder: 1, dimension: -1, cellBasis: tensor
  Subfield lagrange_multiplier_fault, index: 1, components: lagrange_multiplier_fault_x lagrange_multiplier_fault_y, scale: 2.25e+10, basisOrder: 1, quadOrder: 1, dimension: 1, cellBasis: tensor
Local Section
Processor 0
  0  --  --  --  -- # cells
  1  --  --  --  --
  2  --  --  --  --
  3  --  --  --  --
  4  --  --  --  --
```

(continues on next page)

(continued from previous page)

```

5  --  --  --  --
6  --  --  --  --
7  --  --  --  --
8  --  --  --  --
9  --  --  --  --
10 --  --  --  --
11 --  --  --  --
12 24 25  --  --  (constrained: 24 25) # vertices
13 26 27  --  --  (constrained: 26 27)
14 28 29  --  --  (constrained: 28 29)
15 30 31  --  --  (constrained: 30 31)
16 32 33  --  --
17 34 35  --  --
18 36 37  --  --
19 38 39  --  --
20 40 41  --  --  (constrained: 40 41)
21 42 43  --  --  (constrained: 42 43)
22 44 45  --  --  (constrained: 44 45)
23 46 47  --  --  (constrained: 46 47)
24  0   1  --  --
25  2   3  --  --
26  6   7  --  --
27  8   9  --  --
28 12  13  --  --
29 14  15  --  --
30 18  19  --  --
31 20  21  --  --
32 --  --  --  -- # edges
33 --  --  --  --
34 --  --  --  --
35 --  --  --  --
36 --  --  --  --
37 --  --  --  --
38 --  --  --  --
39 --  --  --  --
40 --  --  --  --
41 --  --  --  --
42 --  --  --  --
43 --  --  --  --
44 --  --  --  --
45 --  --  --  --
46 --  --  --  --
47 --  --  --  --
48 --  --  --  --
49 --  --  --  --
50 --  --  --  --
51 --  --  --  --
52 --  --  --  --
53 --  --  --  --
54 --  --  --  --
55 --  --  --  --
56 --  --  --  --

```

(continues on next page)

(continued from previous page)

```

57  --  --  --  --
58  --  --  --  --
59  --  --  4   5 # cohesive edges
60  --  -- 10  11
61  --  -- 16  17
62  --  -- 22  23
Global Section
Processor 0
  0  --  --  --  -- # cells
  1  --  --  --  --
  2  --  --  --  --
  3  --  --  --  --
  4  --  --  --  --
  5  --  --  --  --
  6  --  --  --  --
  7  --  --  --  --
  8  --  --  --  --
  9  --  --  --  --
 10  --  --  --  --
 11  --  --  --  --
 12  --  --  --  -- # vertices
 13  --  --  --  --
 14  --  --  --  --
 15  --  --  --  --
 16  24  25  --  --
 17  26  27  --  --
 18  28  29  --  --
 19  30  31  --  --
 20  --  --  --  --
 21  --  --  --  --
 22  --  --  --  --
 23  --  --  --  --
 24  0   1   --  --
 25  2   3   --  --
 26  6   7   --  --
 27  8   9   --  --
 28  12  13  --  --
 29  14  15  --  --
 30  18  19  --  --
 31  20  21  --  --
 32  --  --  --  -- # edges
 33  --  --  --  --
 34  --  --  --  --
 35  --  --  --  --
 36  --  --  --  --
 37  --  --  --  --
 38  --  --  --  --
 39  --  --  --  --
 40  --  --  --  --
 41  --  --  --  --
 42  --  --  --  --
 43  --  --  --  --

```

(continues on next page)

(continued from previous page)

```

44  --  --  --  --
45  --  --  --  --
46  --  --  --  --
47  --  --  --  --
48  --  --  --  --
49  --  --  --  --
50  --  --  --  --
51  --  --  --  --
52  --  --  --  --
53  --  --  --  --
54  --  --  --  --
55  --  --  --  --
56  --  --  --  --
57  --  --  --  --
58  --  --  --  --
59  --  --    4    5 # cohesive edges
60  --  --   10   11
61  --  --   16   17
62  --  --   22   23

```

Listing 4.330: Corresponding output of PetscSectionVew() for the local section of the solution field.

```

PetscSection Object: 1 MPI process
  type not yet set
2 fields
  field 0 "displacement" with 2 components
Process 0:
(  0) dof  0 offset 24 # cells
(  1) dof  0 offset 24
(  2) dof  0 offset 24
(  3) dof  0 offset 24
(  4) dof  0 offset 24
(  5) dof  0 offset 24
(  6) dof  0 offset 24
(  7) dof  0 offset 24
(  8) dof  0 offset 24
(  9) dof  0 offset 12
( 10) dof  0 offset 18
( 11) dof  0 offset 24
( 12) dof  2 offset 24 constrained 0 1 # vertices
( 13) dof  2 offset 26 constrained 0 1
( 14) dof  2 offset 28 constrained 0 1
( 15) dof  2 offset 30 constrained 0 1
( 16) dof  2 offset 32
( 17) dof  2 offset 34
( 18) dof  2 offset 36
( 19) dof  2 offset 38
( 20) dof  2 offset  0
( 21) dof  2 offset  6
( 22) dof  2 offset 12
( 23) dof  2 offset 18

```

(continues on next page)

(continued from previous page)

```

( 24) dof 2 offset 40 constrained 0 1
( 25) dof 2 offset 42 constrained 0 1
( 26) dof 2 offset 44 constrained 0 1
( 27) dof 2 offset 46 constrained 0 1
( 28) dof 2 offset 2
( 29) dof 2 offset 8
( 30) dof 2 offset 14
( 31) dof 2 offset 20
( 32) dof 0 offset 48 # edges
( 33) dof 0 offset 48
( 34) dof 0 offset 48
( 35) dof 0 offset 48
( 36) dof 0 offset 48
( 37) dof 0 offset 48
( 38) dof 0 offset 48
( 39) dof 0 offset 48
( 40) dof 0 offset 48
( 41) dof 0 offset 48
( 42) dof 0 offset 48
( 43) dof 0 offset 0
( 44) dof 0 offset 48
( 45) dof 0 offset 12
( 46) dof 0 offset 48
( 47) dof 0 offset 18
( 48) dof 0 offset 48
( 49) dof 0 offset 48
( 50) dof 0 offset 48
( 51) dof 0 offset 48
( 52) dof 0 offset 48
( 53) dof 0 offset 48
( 54) dof 0 offset 48
( 55) dof 0 offset 48
( 56) dof 0 offset 0
( 57) dof 0 offset 12
( 58) dof 0 offset 18
( 59) dof 0 offset 4 # cohesive edges
( 60) dof 0 offset 10
( 61) dof 0 offset 16
( 62) dof 0 offset 22
field 1 "lagrange_multiplier_fault" with 2 components
Process 0:
( 0) dof 0 offset 24 # cells
( 1) dof 0 offset 24
( 2) dof 0 offset 24
( 3) dof 0 offset 24
( 4) dof 0 offset 24
( 5) dof 0 offset 24
( 6) dof 0 offset 24
( 7) dof 0 offset 24
( 8) dof 0 offset 24
( 9) dof 0 offset 12
(10) dof 0 offset 18

```

(continues on next page)

(continued from previous page)

```
( 11) dof 0 offset 24
( 12) dof 0 offset 26 # vertices
( 13) dof 0 offset 28
( 14) dof 0 offset 30
( 15) dof 0 offset 32
( 16) dof 0 offset 34
( 17) dof 0 offset 36
( 18) dof 0 offset 38
( 19) dof 0 offset 40
( 20) dof 0 offset 2
( 21) dof 0 offset 8
( 22) dof 0 offset 14
( 23) dof 0 offset 20
( 24) dof 0 offset 42
( 25) dof 0 offset 44
( 26) dof 0 offset 46
( 27) dof 0 offset 48
( 28) dof 0 offset 4
( 29) dof 0 offset 10
( 30) dof 0 offset 16
( 31) dof 0 offset 22
( 32) dof 0 offset 48 # edges
( 33) dof 0 offset 48
( 34) dof 0 offset 48
( 35) dof 0 offset 48
( 36) dof 0 offset 48
( 37) dof 0 offset 48
( 38) dof 0 offset 48
( 39) dof 0 offset 48
( 40) dof 0 offset 48
( 41) dof 0 offset 48
( 42) dof 0 offset 48
( 43) dof 0 offset 0
( 44) dof 0 offset 48
( 45) dof 0 offset 12
( 46) dof 0 offset 48
( 47) dof 0 offset 18
( 48) dof 0 offset 48
( 49) dof 0 offset 48
( 50) dof 0 offset 48
( 51) dof 0 offset 48
( 52) dof 0 offset 48
( 53) dof 0 offset 48
( 54) dof 0 offset 48
( 55) dof 0 offset 48
( 56) dof 0 offset 0
( 57) dof 0 offset 12
( 58) dof 0 offset 18
( 59) dof 2 offset 4 # cohesive edges
( 60) dof 2 offset 10
( 61) dof 2 offset 16
( 62) dof 2 offset 22
```

Integration

Integration involves integrals over the domain (materials), over the boundary of the domain (boundary conditions), or over interior interfaces (fault surface). These three operations are done by different PETSc functions and integrator objects.

DMPlexComputeResidual_Internal

Compute the contribution to the LHS or RHS residual for a single material. This function and the functions it calls handle looping over the cells in the material, integrating the weak form for each of the fields, and adding them to the residual. A more appropriate name would be **DMPlexComputeResidualSingle**, and that may be used in the future. This function is called from **IntegratorDomain**.

DMPlexComputeBdResidualSingle

Compute the contribution to the LHS or RHS residual for a single boundary condition. This function and the functions it calls handle looping over faces (3D) or edges (2D) on the boundary, integrating the weak form for each of the fields, and adding them to the residual. This function is called from **IntegratorBoundary**.

DMPlexComputeResidual_Hybrid_Internal

Compute the contribution to the LHS or RHS residual for a single fault surface. This function and the functions it calls handle looping over the cohesive cells, integrating the weak form for each of the fields, and adding them to the residual. This function is called from **IntegratorInterface**.

Projection

Input and output often involve projecting fields to/from the finite-element space. PETSc provides a family of functions for this. We generally use two of these, one for analytical functions and one for discretized fields. Projection may be a misleading term here, since we are not referring to the common L_2 projection, but rather interpolation of the function by functions in our finite-element space.

Let's start with the simple example of Fourier analysis, which most people have experience with. If we want the Fourier interpolant $\tilde{f}$ for a given function f , then we need to determine its Fourier coefficients, f_k , where

$$\tilde{f} = \sum_k f_k e^{ikx}. \quad (4.546)$$

This is straightforward because the basis functions in the Fourier representation are orthogonal,

$$\int_0^{2\pi} e^{-imx} e^{ikx} dx = 2\pi \delta_{km}. \quad (4.547)$$

To find the coefficient f_m , we just multiply by the conjugate of the basis function and integrate,

$$\int_0^{2\pi} e^{-imx} \tilde{f} dx = \int_0^{2\pi} e^{-imx} \sum_k f_k e^{ikx} dx, \quad (4.548)$$

$$= \sum_k f_k \int_0^{2\pi} e^{-imx} e^{ikx} dx, \quad (4.549)$$

$$= \sum_k f_k 2\pi \delta_{km}, \quad (4.550)$$

$$= 2\pi f_m, \quad (4.551)$$

and we have our coefficient f_m ,

$$f_m = \frac{1}{2\pi} \int_0^{2\pi} e^{-imx} \tilde{f} dx. \quad (4.552)$$

The finite element basis ϕ_i is not orthogonal, so we have an extra step. We could take the inner product of f with all the basis functions, and then sort out the dependencies by solving a linear system (the mass matrix), which is what

happens in L_2 projection. However, suppose we have another basis ψ_i of linear functionals which is *biorthogonal* to ϕ_i , meaning

$$\psi_i(\phi_j) = \delta_{ij}. \quad (4.553)$$

We can easily pick out the coefficient of $\tilde{f}$ by acting with the corresponding basis functional. Our interpolant is

$$\tilde{f} = \sum_k f_k \phi_k(x). \quad (4.554)$$

Acting on the interpolant with the biorthogonal basis results in

$$\psi_i(\tilde{f}) = \psi_i \left(\sum_k f_k \phi_k(x) \right). \quad (4.555)$$

Making use of the fact that the finite-element basis ϕ_i is linear, yields

$$\psi_i(\tilde{f}) = \sum_k f_k \psi_i(\phi_k(x)), \quad (4.556)$$

$$\psi_i(\tilde{f}) = \sum_k f_k \delta_{ik}, \quad (4.557)$$

$$\psi_i(\tilde{f}) = f_i. \quad (4.558)$$

We call ϕ_i the *primal* basis, and ψ_i the *dual* basis. We note that if f does not lie in our approximation space spanned by ϕ_i , then interpolation is not equivalent to L_2 projection. This will not usually be important for our purposes.

DMPProjectFunctionLocal

Project an analytical function into the given finite-element space.

DMPProjectFieldLocal

Project a discretized field in one finite-element space into another finite-element space.

Pointwise functions (kernels)

The following code blocks show the function prototypes for pointwise functions for the residual and the Jacobian. We use the same prototype for the boundary and fault interfaces residuals.

Listing 4.331: Function prototype for pointwise functions for integrating the residual over the domain (materials).

```
/** PetscPointFunc prototype.
 *
 * @param[in] dim Spatial dimension.
 * @param[in] numS Number of registered subfields in solution field.
 * @param[in] numA Number of registered subfields in auxiliary field.
 * @param[in] soff Offset of registered subfields in solution field [numS].
 * @param[in] soff_x Offset of registered subfields in gradient of the solution field
 * [numS].
 * @param[in] s Solution field with all subfields.
 * @param[in] s_t Time derivative of solution field.
 * @param[in] s_x Gradient of solution field.
 * @param[in] aoff Offset of registered subfields in auxiliary field [numA]
 * @param[in] aoff_x Offset of registered subfields in gradient of auxiliary field [numA]
 * @param[in] a Auxiliary field with all subfields.
 * @param[in] a_t Time derivative of auxiliary field.
 * @param[in] a_x Gradient of auxiliary field.
```

(continues on next page)

(continued from previous page)

```

* @param[in] t Time for residual evaluation.
* @param[in] x Coordinates of point evaluation.
* @param[in] numConstants Number of registered constants.
* @param[in] constants Array of registered constants.
* @param[out] f [dim].
*/
void
func(const PylithInt dim,
    const PylithInt numS,
    const PylithInt numA,
    const PylithInt sOff[],
    const PylithInt sOff_x[],
    const PylithScalar s[],
    const PylithScalar s_t[],
    const PylithScalar s_x[],
    const PylithInt aOff[],
    const PylithInt aOff_x[],
    const PylithScalar a[],
    const PylithScalar a_t[],
    const PylithScalar a_x[],
    const PylithReal t,
    const PylithScalar x[],
    const PylithInt numConstants,
    const PylithScalar constants[],
    PylithScalar f[]);

```

Listing 4.332: Function prototype for pointwise functions for integrating the Jacobian over the domain (materials).

```

/** PetscPointJac prototype.
*
* This is identical to the PetscPointFunc with the addition of the
* s_tshift argument.
*
* @param[in] s_tshift The multiplier for dF/dS_t.
*/
void
func(const PylithInt dim,
    const PylithInt numS,
    const PylithInt numA,
    const PylithInt sOff[],
    const PylithInt sOff_x[],
    const PylithScalar s[],
    const PylithScalar s_t[],
    const PylithScalar s_x[],
    const PylithInt aOff[],
    const PylithInt aOff_x[],
    const PylithScalar a[],
    const PylithScalar a_t[],
    const PylithScalar a_x[],
    const PylithReal t,
    const PylithReal s_tshift,

```

(continues on next page)

(continued from previous page)

```

const PylithScalar x[],
const PylithInt numConstants,
const PylithScalar constants[],
PylithScalar J[]);

```

Listing 4.333: Function prototype for pointwise functions for integrating the residual over domain boundaries and interior interfaces.

```

/** PetscBdPointFunc prototype.
 *
 * @param[in] dim Spatial dimension.
 * @param[in] numS Number of registered subfields in solution field.
 * @param[in] numA Number of registered subfields in auxiliary field.
 * @param[in] sOff Offset of registered subfields in solution field [numS].
 * @param[in] sOff_x Offset of registered subfields in gradient of the solution field.
 * @param[in] s Solution field with all subfields.
 * @param[in] s_t Time derivative of solution field.
 * @param[in] s_x Gradient of solution field.
 * @param[in] aOff Offset of registered subfields in auxiliary field [numA]
 * @param[in] aOff_x Offset of registered subfields in gradient of auxiliary field [numA]
 * @param[in] a Auxiliary field with all subfields.
 * @param[in] a_t Time derivative of auxiliary field.
 * @param[in] a_x Gradient of auxiliary field.
 * @param[in] t Time for residual evaluation.
 * @param[in] x Coordinates of point evaluation.
 * @param[in] numConstants Number of registered constants.
 * @param[in] constants Array of registered constants.
 * @param[out] f0 [dim].
 */
void
func(const PylithInt dim,
     const PylithInt numS,
     const PylithInt numA,
     const PylithInt sOff[],
     const PylithInt sOff_x[],
     const PylithScalar s[],
     const PylithScalar s_t[],
     const PylithScalar s_x[],
     const PylithInt aOff[],
     const PylithInt aOff_x[],
     const PylithScalar a[],
     const PylithScalar a_t[],
     const PylithScalar a_x[],
     const PylithReal t,
     const PylithScalar x[],
     const PylithReal n[],
     const PylithInt numConstants,
     const PylithScalar constants[],
     PylithScalar f0[]);

```


Listing 4.334: Function prototype for pointwise functions for integrating the residual over domain boundaries.

```
/** PetscBdPointJac prototype.
 *
 * This is identical to the PetscBdPointFunc with the addition of the
 * s_tshift argument.
 *
 * @param[in] s_tshift The multiplier for dF/dS_t.
 */
void
func(const PylithInt dim,
     const PylithInt numS,
     const PylithInt numA,
     const PylithInt sOff[],
     const PylithInt sOff_x[],
     const PylithScalar s[],
     const PylithScalar s_t[],
     const PylithScalar s_x[],
     const PylithInt aOff[],
     const PylithInt aOff_x[],
     const PylithScalar a[],
     const PylithScalar a_t[],
     const PylithScalar a_x[],
     const PylithReal t,
     const PylithReal s_tshift,
     const PylithScalar x[],
     const PylithReal n[],
     const PylithInt numConstants,
     const PylithScalar constants[],
     PylithScalar J[]);
```

The integrators handle calling the appropriate functions for setting the kernels for the integration. The Physics objects (materials, boundary conditions, and faults) tell the integrators what kernels to use. Pointwise functions not used should be set to NULL.

Listing 4.335: Data structure used to specify kernels for integration of the residual over the domain (materials).

```
struct ResidualKernels {
    std::string subfield; ///< Name of subfield
    PetscPointFunc r0; ///< f0 (RHS) or g0 (LHS) function.
    PetscPointFunc r1; ///< f1 (RHS) or g1 (LHS) function.
};
```

Listing 4.336: Data structure used to specify kernels for integration of the Jacobian over the domain (materials).

```
struct JacobianKernels {
    std::string subfieldTrial; ///< Name of subfield associated with trial function (row_
↪ in Jacobian).
    std::string subfieldBasis; ///< Name of subfield associated with basis function_
↪ (column in Jacobian).
```

(continues on next page)

(continued from previous page)

```

PetscPointJac j0; ///< J0 function.
PetscPointJac j1; ///< J1 function.
PetscPointJac j2; ///< J2 function.
PetscPointJac j3; ///< J3 function.
};

```

Listing 4.337: Data structure used to specify kernels for integration of the residual over the domain (materials).

```

struct ResidualKernels {
    std::string subfield; ///< Name of subfield
    PetscBdPointFunc r0; ///< f0 (RHS) or g0 (LHS) function.
    PetscBdPointFunc r1; ///< f1 (RHS) or g1 (LHS) function.
};

```

Listing 4.338: Data structure used to specify kernels for integration of the Jacobian over the domain (materials).

```

struct JacobianKernels {
    std::string subfieldTrial; ///< Name of subfield associated with trial function (row,
    ↪ in Jacobian).
    std::string subfieldBasis; ///< Name of subfield associated with basis function,
    ↪ (column in Jacobian).
    PetscBdPointJac j0; ///< J0 function.
    PetscBdPointJac j1; ///< J1 function.
    PetscBdPointJac j2; ///< J2 function.
    PetscBdPointJac j3; ///< J3 function.
};

```

4.3.3 Rebuilding PETSc and PyLith

For instructions on how to build PyLith from source, please see the [PyLith Installer Documentation](#).

Rebuilding PETSc

Updating and rebuilding PETSc is quite simple once it has been configured and built once before.

```

# Change to PETSc source Directory
cd $PETSC_DIR

# Fetch the updates, pruning deleted branches.
git fetch -p

# Ensure you are on the `knepley/pylith` branch and then pull updates.
git checkout knepley/pylith
git pull

# Reconfigure
arch-pylith-debug/lib/petsc/conf/reconfigure-arch-pylith-debug.py

# Rebuild
make

```

Important

After rebuilding PETSc, you should rebuild PyLith. If there are incompatibilities between the two, then you will normally get compilation errors building PyLith. If there are incompatibilities and you do not rebuild PyLith, then you will usually get a segmentation fault when running PyLith.

Rebuilding PyLith**Overview**

PyLith uses the GNU Build System (often called autotools, which consists of autoconf, automake, and libtool) to configure and build. The configure options and checks are defined in `configure.ac` with additional macros in the `m4` directory. Note that the `m4` directory is a Git submodule corresponding to the `geodynamics/autoconf_cig` Git repository.

Updating your fork

As long as you setup your `main` branch to correspond to `upstream/main`, you should never have to use the `main` branch in your fork. You only use your fork for your feature branches.

Makefiles

The PyLith `configure` script uses `automake` to convert each `Makefile.am` file into a `Makefile`. The organization and content of the `Makefile.am` file depends on whether it is related to the C++ library, SWIG interface files, Python modules, C++ unit tests, Python unit tests, MMS tests, full-scale tests, or examples.

For the C++ library files within the `libsrc` directory, the `libsrc/Makefile.am` contains the implementation files while the header files are listed in the `Makefile.am` file within the underlying directories. For the SWIG interface files within the `modulesrc` directory, `Makefile.am` file contains information on how to build the SWIG Python module. The Python modules use a single `Makefile.am` file in the `pylith` directory. Each test suite (C++ unit tests for each subpackage, Python unit tests, and each suite of full-scale tests) use a single `Makefile.am`. These files define how the tests are built, additional input files that should be included in the source distribution, and temporary files that should be deleted.

Each suite of examples contains a `Makefile.am` that defines the files that are to be included in the source distribution. It also defines which files are created and should be deleted upon `make clean`.

Build Targets

Several build targets are defined to make it easy to rebuild/reinstall PyLith and rerun tests whenever the source code changes. Each target can be run using `make TARGET` where `TARGET` is one of the following:

all

Build all source code.

install

Build and install.

check

Run the entire test suite.

Tip

On a machine with multiple cores, faster builds of the C++ code (library and C++ unit tests) are available using the `-jNTHREADS` command line argument, where `NTHREADS` is the number of threads to use. We usually set the

number of threads equal to twice the number of physical cores.

After modifying code, the C++ library, SWIG modules, and Python code need to be rebuilt and reinstalled before running a PyLith simulation.

Listing 4.339: Rebuilding Pylith C++ library, SWIG modules, and Python modules

```
# Change to top-level PyLith build directory.
cd $PYLITH_BUILDDIR

# Reinstall everything using 16 threads to build library.
make install -j16

# Rebuild and reinstall only the library using 16 threads.
make install -j16 -C libsrc

# Rebuild and reinstall only the SWIG modules
make install -C modulesrc

# Reinstall only the Python modules
make install -C pylith
```

After modifying the C++ code, only the C++ library needs to be rebuilt before running C++ unit tests or MMS tests.

Listing 4.340: Rebuilding Pylith C++ library and rerunning the C++ unit tests

```
# Change to top-level PyLith build directory.
cd $PYLITH_BUILDDIR/build/pylith-debug

# Rebuild the library using 16 threads.
make -j16 -C libsrc

# Rerun the C++ unit tests.
make check -C tests/libtests
```

Similarly, after changing the Python code, only the Python modules need to be reinstalled before running a Python unit test. However, if changes are also made to the C++ code, then the C++ library and modules must be rebuilt and reinstalled before running a Python unit test.

Listing 4.341: Rebuilding PyLith Python modules and rerunning Python unit tests

```
# Change to top-level PyLith build directory.
cd $PYLITH_BUILDDIR/build/pylith-debug

# Reinstall PyLith Python modules.
make install -C pylith

# Rerun Python unit tests.
make check -C tests/pytests
```

4.3.4 Testing

Testing and debugging PyLith can be challenging due its many dependencies and complex interaction with PETSc. Our strategy is to test at a variety of levels to isolate bugs close to their origin while also building a comprehensive suite of tests. We use C++ unit tests for verifying small pieces of code, such as individual C++ class methods, work as intended. We use the Method of Manufactured Solutions for verifying implementation of governing equations and their solution using PETSc. We use full-scale tests for verifying integration of complete simulations in both serial and parallel. When an example, user simulation, or one of the tests indicates a bug exists, we leverage the test suite to find the bugs. In general, the most efficient strategy for debugging is to first try to expose the bug in a serial unit test, followed by an MMS Test, a serial full-scale test, and finally a parallel full-scale test. This may require creating new tests if the bug is not exposed by current tests. The PyLith developers make extensive use of debuggers, such as gdb and lldb, and memory management analysis tools, such as valgrind, to detect and squash bugs. We tools for debugging in *Debugging tools* and tools for evaluating performance in *Performance Tools*.

C++ unit tests

The C++ unit tests target verification at the scale of individual C++ class methods. This isolates bugs very close to their origin, so they are highly effective at catching simple errors, especially memory errors such as invalid reads, writes, and leaks. We use the CppUnit testing framework for constructing and running tests.

Note

We will be transitioning from CppUnit to Catch2 in the near future. Catch2 provides seamless integration with VS Code and more control over running small batches of tests.

For a given C++ class in libsrc/pylith, we create a separate test class in tests/libtests. For example, we create pylith::problems::TestPhysics in tests/libtests/problems/TestPhysics.* to test the pylith::problems::Physics class in libsrc/pylith/problems/Physics.*. For simple classes that can be fully tested with a single test case, we put the class declaration and implementation in a single .cc file. For classes that require testing with multiple alternative test cases, we usually define both the test class and a data class in a header file (for example TestSolutionFactory.hh). The classes are implemented in a corresponding .cc file (for example, TestSolutionFactory.cc) with the test cases defined and implemented in a _Cases.cc file (for example, TestSolutionFactory_Cases.cc).

The test class inherits from CppUnit::TestFixture and defines a list of test methods. The class methods setUp() and tearDown() are run before and after each test, respectively. If a class requires significant initialization, we usually put that in protected methods.

CppUnit macros

- Generating the test suite
 - `CPPUNIT_TEST_SUITE(Classname)`; Create a test suite for class `Classname`.
 - `CPPUNIT_TEST(classMethod)`; Add `classMethod` as a test in the test suite.
 - `CPPUNIT_TEST_SUITE_END()`; Mark end of test suite.
- Implementing tests
 - `CPPUNIT_ASSERT(condition)` Test that `condition==true`.
 - `CPPUNIT_ASSERT_EQUAL(valueExpected, valueTest)`; Test that `valueExpected == valueTest`.
 - `CPPUNIT_ASSERT_DOUBLES_EQUAL(valueExpected, valueTest, tolerance)`; Test that `valueExpected == valueTest` within tolerance.
 - `CPPUNIT_FAIL(msg)`; Force test failure and include `msg` string in test failure message.
 - `CPPUNIT_ASSERT_MESSAGE(msg, condition)`; Include `msg` string in test failure message.
 - `CPPUNIT_ASSERT_EQUAL_MESSAGE(msg, valueExpected, valueTest)`;
 - `CPPUNIT_ASSERT_DOUBLES_EQUAL_MESSAGE(msg, valueExpected, valueTest, tolerance)`;

Within the test implementations, we use `CPPUNIT_ASSERT(condition)` where we would normally use `assert(condition)`. This results in the corresponding test failing and the remaining tests are executed. If an `assert(condition)` fails, the test driver aborts and the remaining tests are not executed.

Listing 4.342: Illustration of a C++ header file for a test class without a separate data class.

```
#if !defined(pylith_problems_testphysics_hh)
#define pylith_problems_testphysics_hh

#include <cppunit/extensions/HelperMacros.h>

#include "pylith/problems/problemsfwd.hh" // HOLDSA Physics
#include "pylith/topology/topologyfwd.hh" // HOLDSA Mesh, Field

// Declaration of class within the pylith namespace.
namespace pylith {
    namespace problems {
        class TestPhysics;
    } // problems
} // pylith

class pylith::problems::TestPhysics : public CPPUNIT::TestFixture {
    CPPUNIT_TEST_SUITE(TestPhysics); // CppUnit macro used to define the `TestPhysics`
    ↪ test suite.

    CPPUNIT_TEST(testSetScales); // CppUnit macro to add test implemented by class
    ↪ method.
    CPPUNIT_TEST(testSetAuxiliaryFieldDB);
    CPPUNIT_TEST(testSetAuxiliarySubfieldDiscretization);
    CPPUNIT_TEST(testObservers);
    CPPUNIT_TEST(testGetKernelConstants);
    CPPUNIT_TEST(testVerifyConfiguration);
```

(continues on next page)

(continued from previous page)

```

CPPUNIT_TEST(testCreateIntegrator);
CPPUNIT_TEST(testCreateConstraint);
CPPUNIT_TEST(testCreateAuxiliaryField);
CPPUNIT_TEST(testCreateDerivedField);

CPPUNIT_TEST_SUITE_END(); // End of test suite declaration.

public:

    // Methods to setup and clean up tests.
    void setUp(void);
    void tearDown(void);

    // Methods that implement tests. The naming convention is testMethodName.
    // All test methods must return void and not have any arguments.
    void testSetScales(void);
    void testSetAuxiliaryFieldDB(void);
    void testSetAuxiliarySubfieldDiscretization(void);
    void testObservers(void);
    void testGetKernelConstants(void);
    void testVerifyConfiguration(void);
    void testCreateIntegrator(void);
    void testCreateConstraint(void);
    void testCreateAuxiliaryField(void);
    void testCreateDerivedField(void);

private:

    pylith::problems::Physics* _physics; // Test subject.
    pylith::topology::Mesh* _mesh; // Mesh for test subject.
    pylith::topology::Field* _solution; // Solution field for test subject.

}; // class TestPhysics

#endif // pylith_problems_testphysics_hh

```

Listing 4.343: Illustration of a C++ implementation file for a test class.

```

// Instantiate the test suite.
CPPUNIT_TEST_SUITE_REGISTRATION(pylith::problems::TestPhysics);

void
pylith::problems::TestPhysics::setUp(void) {
    // We use CPPUNIT_ASSERT() where we normally would use assert().
    _physics = new PhysicsStub(); CPPUNIT_ASSERT(_physics);

    _mesh = new pylith::topology::Mesh(); CPPUNIT_ASSERT(_mesh);
    _solution = new pylith::topology::Field(*_mesh); CPPUNIT_ASSERT(_solution);
} // setUp

void

```

(continues on next page)

(continued from previous page)

```

pylith::problems::TestPhysics::tearDown(void) {
    delete _physics; _physics = NULL;
    delete _solution; _solution = NULL;
    delete _mesh; _mesh = NULL;
} // tearDown

void
pylith::problems::TestPhysics::testSetScales(void) {
    PYLITH_METHOD_BEGIN;

    pylith::scales::Scales scales;
    const PyLithReal lengthScale = 3.0;
    scales.setLengthScale(lengthScale);

    CPPUNIT_ASSERT(_physics);
    _physics->setScales(scales);

    CPPUNIT_ASSERT_DOUBLES_EQUAL(lengthScale, _physics->_scales->getLengthScale(), 1.0e-
↪6);

    PYLITH_METHOD_END;
} // testSetScales

void
pylith::problems::TestPhysics::testSetAuxiliaryFieldDB(void) {
    PYLITH_METHOD_BEGIN;

    spatialdata::spatialdb::UniformDB db;
    db.setLabel("test db");

    CPPUNIT_ASSERT(_physics);
    _physics->setAuxiliaryFieldDB(&db);

    const pylith::feassemble::AuxiliaryFactory* factory = _physics->_
↪getAuxiliaryFactory(); CPPUNIT_ASSERT(factory);
    const spatialdata::spatialdb::SpatialDB* queryDB = factory->getQueryDB(); CPPUNIT_
↪ASSERT(queryDB);
    CPPUNIT_ASSERT_EQUAL(db.getLabel(), queryDB->getLabel());

    PYLITH_METHOD_END;
} // testSetAuxiliaryFieldDB

void
pylith::problems::TestPhysics::testGetKernelConstants(void) {
    PYLITH_METHOD_BEGIN;

    const size_t numConstants = 2;
    const PyLithReal constantsE[numConstants] = { -1.1, 4.4 };

    CPPUNIT_ASSERT(_physics);

```

(continues on next page)

(continued from previous page)

```

_physics->_kernelConstants = pylith::real_array(constantsE, numConstants);

const PyLithReal dt = 2.0;
const pylith::real_array& constants = _physics->getKernelConstants(dt);

CPPUNIT_ASSERT_EQUAL(numConstants, constants.size());
for (size_t i = 0; i < numConstants; ++i) {
    CPPUNIT_ASSERT_DOUBLES_EQUAL(constantsE[i], constants[i], 1.0e-6);
} // for

PYLITH_METHOD_END;
} // testGetKernelConstants

```

Method of Manufactured Solutions

We use the Method of Manufactured Solutions (MMS) to verify the order of accuracy of our solution to the governing equations. See [Code Verification by the Method of Manufactured Solutions](#) for a more detailed discussion of MMS applied to code verification for simulations of partial differential equations. The general formulation is to assume a solution for the governing equations. If the solution is not an exact solution, then plugging it into the governing equation will result in a residual that is a body force. This body force can be included as an additional term in the governing equation so that the solution becomes exact.

In our application of the MMS, we test four features:

1. Representation of the solution in the finite-element space;
2. Residual for the governing equation is zero within some tolerance;
3. A Taylor series expansion of the Jacobian provides the expected order of accuracy; and
4. A finite-difference Jacobian matches the computed Jacobian within some tolerance.

In mathematical terms, the residual test is that the residual computed for a known solution is below some tolerance, ϵ ,

$$\|F(\vec{s}) - G(\vec{s})\| \leq \epsilon, \quad (4.559)$$

where $F(\vec{s})$ is the LHS residual and $G(\vec{s})$ is the RHS residual. In the Taylor series Jacobian test, we verify that

$$\|F(\vec{s} + \epsilon \vec{\delta s}) - F(\vec{s}) - \epsilon J \vec{v}\| < \epsilon^2, \quad (4.560)$$

where $\vec{\delta s}$ is a perturbation in the solution and J is the Jacobian matrix.

The C++ class `pylith::testing::MMSTest` implements these four tests. In implementing an MMS test case, we need to provide:

- Analytical functions for each subfield of the solution;
- Analytical functions for each subfield in the auxiliary fields;
- Dirichlet boundary conditions;
- Finite-element mesh;
- Setup of the problem, including the solution, physics, boundary conditions, and the PETSc TS.

Warning

In setting up a MMS test, remember that the solution should be able to be represented in the finite-element space. If you use a polynomial of order 2 for a subfield of the solution, then you should use a basis order of at least 2 for that subfield of the solution.

Debugging residual errors

When the residual test fails, we generally use the following procedure to diagnose the problem.

1. Verify that the discretization check passes indicating accurate representation of the solution in the finite-element space.
2. Run just the residual test for a single discretization and use the debug journal command line argument to the MMS test driver (`--journal.debug=mms-test`).
3. Verify that the residual kernels show up correctly in the view of the PETSc discretization.
4. Analyze the residual vector to see which degrees of freedom have nonzero terms. Look at the solution section to see what solution subfield and point are associated with those degrees for freedom.
5. Check the pointwise functions for the residual and solution associated with the subfield with nonzero terms.

Debugging Jacobian errors

When one of the Jacobian tests fails, we focus on the finite-difference Jacobian test.

1. Verify that the residual check passes.
2. Run just the Jacobian finite-difference test for a single discretization and use the debug journal command line argument to the MMS test drive (`--journal.debug=mms-test`).
3. Examine any differences between the hand-coded Jacobian and the finite-difference Jacobian.
4. Check the differences against pointwise functions listed in the view of the PETSc discretization. Are any expected functions missing from the list?
5. Isolate the error by dropping corresponding terms from the residual, Jacobian, and solution by editing the pointwise functions until the hand-coded and finite-difference Jacobians agree; that is, drop terms from the governing equation until the MMS tests pass. Then, add the terms back into the governing equation (residual, Jacobian, and solution pointwise functions) one by one, fixing errors as they are detected.

Example

In `tests/mmstests/linearelasticity/nofaults-2d`, we create a suite of MMS tests that all use a single material (Elasticity), a single Dirichlet boundary condition, and a data object to hold test-specific parameters. The `pylith::mmstests::TestLinearElasticity` object sets up the MMS test using data provided for each specific MMS test case. The individual test cases, such as `UniformStrain2D`, provide the problem-specific parameters, such as analytical functions for the solution, auxiliary fields, and boundary conditions.

Warning

The quadrature order must be the same across all solution subfields and auxiliary subfields.

TODO

Add a simple test case that introduces known errors into the discretization, residual, and Jacobian to illustrate these steps.

Running C++ unit tests and MMS tests

Running C++ unit tests

The C++ unit tests (libtests) are implemented using CppUnit and a common test driver, `pylith::testing::TestDriver` in `tests/src/driver_cppunit.cc`. `TestDriver` provides support for command line arguments to control the tests run and set PETSc options. All of the C++ unit test or MMS test executables support the following command line arguments:

- **--help** Show help for command line arguments.
- **--list** List tests run by the executable.
- **--tests=TESTS** Run subset of the tests. TESTS is a comma separated list of tests.
- **--petsc VALUE=ARG** Set PETSc option -VALUE=ARG.
- **--journal.info=CHANNEL** Activate Pythia info journal channel CHANNEL.
- **--journal.debug=CHANNEL** Activate Pythia debug journal channel CHANNEL.
- **--journal.warning=CHANNEL** Activate Pythia warning journal channel CHANNEL.

Listing 4.344: Examples of using command line arguments in running C++ unit tests.

```
$ cd tests/libtests/problems

# List tests
$ ./test_problems --list

# Run all TestObserversPhysics tests.
$ ./test_problems --tests=pylith::problems::TestObserversPhysics

# Run TestObserversPhysics testVerifyObservers and testNotifyObservers tests.
$ ./test_problems --tests=pylith::problems::TestObserversPhysics::testVerifyObservers,
↪pylith::problems::TestObserversPhysics::testNotifyObservers

# Turn on mesh info journal.
$ ./test_problems --journal.info=mesh

# Turn on solution debug journal.
$ ./test_problems --journal.debug=solution
```

Running MMS tests

The Method of Manufactured Solutions (MMS) tests are implemented using Catch2 and a common test driver, `pylith::testing::TestDriver` in `tests/src/driver_catch2.cc`. `TestDriver` provides support for command line arguments to control the tests run and set PETSc options. All of the C++ unit test or MMS test executables support the following command line arguments:

- **--help** Show help for command line arguments.
- **--list-tests** List tests run by the executable.

- **--petsc VALUE=ARG** Set PETSc option -VALUE=ARG.
- **--journal.info=CHANNEL** Activate Pythia info journal channel CHANNEL.
- **--journal.debug=CHANNEL** Activate Pythia debug journal channel CHANNEL.
- **--journal.warning=CHANNEL** Activate Pythia warning journal channel CHANNEL.

Listing 4.345: Examples of using command line arguments in running C++ and MMS tests.

```
$ cd tests/mms-tests/linearelasticity/nofaults-2d

# List tests
$ ./mms-test_linearelasticity_nofaults2d --list-tests

# Run all UniformStrain2D tests.
$ ./mms-test_linearelasticity_nofaults2d [UniformStrain2D]

# Run all UniformStrain2D residual tests
$ ./mms-test_linearelasticity_nofaults2d [UniformStrain2D] [testResidual]

# Turn on application-flow info journal.
$ ./mms-test_linearelasticity_nofaults2d --journal.info=application-flow

# Turn on mms-test debug journal.
$ ./mms-test_linearelasticity_nofaults2d --journal.debug=mms-test
```

Using the debugger

The executables in the build directory are shell script wrappers created by `libtool`. The underlying binary executables are in the `.libs` directory. When using the debugger, pass the binary executable to the debugger. For example, `gdb .libs/test_problems`.

Using valgrind

```
# Run valgrind on the test_problems executable
$ valgrind --log-file=valgrind_problems.log \
  --suppressions=$PYLITH_DIR/share/valgrind-python.supp .libs/test_problems
```

Python unit tests

We use the standard Python `unittest` module to implement the Python unit tests.

In reimplementing the Python unit tests in `tests/pytests`, we have setup bare bones tests that simply check that the objects are instantiated successfully, configure executes without errors, and factory functions return correct objects. These tests are implemented in the `pylith.testing.UnitTestApp` Python object. Tests of individual classes need only specify the class and the factory function. We plan to expand these tests to include verification that information is properly transferred from Python to the underlying C++ object (if it exists).

Full-scale tests

The full-scale tests are constructed in Python, making use of the standard Python `unittest` module. A full-scale test involves running a PyLith simulation and using Python code to check output against a known solution. Whenever possible, we simulate simple boundary value problems with analytical solutions. We run some full-scale tests in parallel on a small number of processes.

The first step is to construct a PyLith simulation (mesh, parameter files, and spatial databases) that solves the desired boundary value problem with a known solution. In constructing the simulation, we create Python scripts that generate the spatial database files. This makes it easy to synchronize any changes between the problem specification and the known solution.

Once we have a PyLith simulation that appears to run correctly, we construct a Python object that inherits from `pylith.testing.FullTestApp`. See `tests/fullscale/linearelasticity/nofaults-2d/TestAxialDisp.py` for an example. This Python object sets up the a PyLith simulation, runs the simulation, and defines methods that check fields in the output files. In most cases we solve the boundary value problems in 2D with both quadrilateral and triangular meshes and in 3D with both tetrahedral and hexahedral meshes.

When multiple full-scale tests use the same meshes, we place them in the same directory and use dictionaries in a `meshes.py` file to define the mesh information.

Command line arguments

The `pylith.testing.FullTestApp` Python application has two optional command line arguments:

- **--verbose (int)** Set verbosity level. Level > 0 will display which fields are being checked.
- **--skip-pylith-run (bool)** Skip running the PyLith simulation and only check the existing output files against the expected results. This argument is useful when debugging the checks.

Using the debugger

To start the `gdb` debugger when running the PyLith application, simply add the command line argument `--petsc.start_in_debugger`. To use an alternative debugger, such as `lldb`, append the name of the debugger executable, for example `--petsc.start_in_debugger=lldb`. By default, PETSc will try to start the debugger in an `xterm`. To use an alternative terminal program, use the command line argument `--petsc.debug_terminal=TERMINAL`. For example for the GNOME terminal, use `--petsc.debug_terminal="gnome-terminal -x"`.

Using valgrind

Running `valgrind` on a the PyLith application, including full-scale tests, requires using an additional argument (`--trace-children=yes`), because the PyLith application forks a subprocess to do the computation.

```
# Run valgrind on the PyLith executable
$ valgrind --trace-children=yes \
  --suppressions=$PYLITH_DIR/share/valgrind-python.supp pylith step01.cfg
```

Example

TODO

Walk through an example of a full-scale test.

4.3.5 Git Workflow

Overview

We use the Git version control system <https://git-scm.com> with a central GitHub repository <https://github.com/geodynamics/pylith> for development. We will refer to this central repository as the `geodynamics/pylith` repository. Only the PyLith maintainers have write access to the `geodynamics/pylith` repository; everyone else is limited to read access. Contributions from the community are incorporated into the `geodynamics/pylith` repository via pull requests.

Currently, the PyLith maintainers use the [Forking Workflow](#). Each developer forks the geodynamics repository and a new feature is added on its own branch in the developer fork. Once it is implemented, tested, and documented, it is merged to the main branch of the geodynamics/pylith repository.

The forking workflow is illustrated in Figures [git-repositories](#) and [git-branch](#). You fork the geodynamics/pylith repository so that you have a copy of the repository for your changes. Each new, independent feature you develop should be done in a separate branch. Large feature contributions should be broken up into multiple phases, where each phase provides a working version that passes all of its tests. Each phase corresponds to a separate branch that is merged back into the geodynamics/pylith repository (usually the main branch) via a [pull request](#) when it is completed.

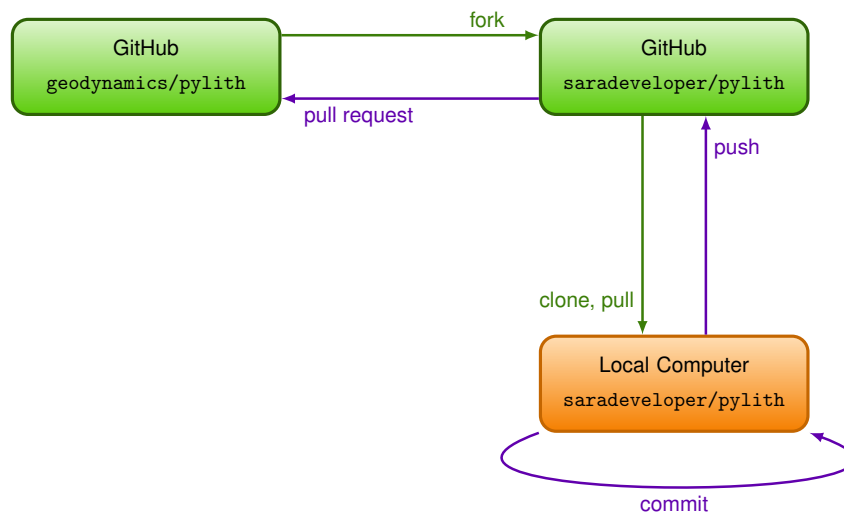


Fig. 4.210: Overview of repositories for the Git forking workflow used in PyLith development. The main repository is geodynamics/pylith at GitHub. Developers create a fork of that repository under their own GitHub account (e.g., saradeveloper), which is cloned to their local computer. Changes and additions to the code are committed to the repository on the local computer, which are pushed to the developer's GitHub account. Once a development task is completed, a developer is encouraged to contribute the changes and additions to the main repository via pull requests.

Git tasks

Setting up your fork

There are two steps to setting up a copy of the PyLith repository that you can use for development under the Forking Workflow:

1. Create a GitHub account.
2. Fork the geodynamics/pylith repository.

This will create a copy of the PyLith repository in your GitHub account. You can make changes to this copy and, when you are ready to contribute changes back to the geodynamics/pylith repository, you will create a pull request.

Important

Create a personal access token for your GitHub account. See [GitHub Docs: Creating a personal access token](#) for more information. This provides additional security and eliminates the need for you to enter your username and password whenever you push to your repository.

We strongly recommend that you setup your GitHub account to use two-factor authentication. See [GitHub Docs: Securing your account with two-factor authentication](#).

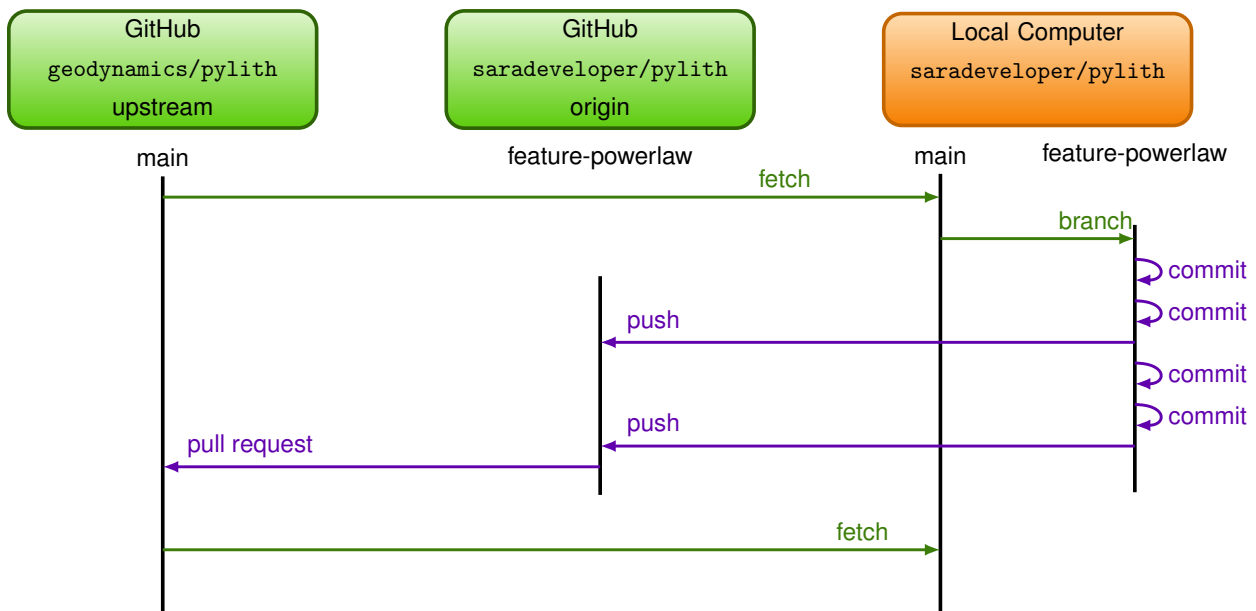


Fig. 4.211: Overview of repositories and branches for the Git forking workflow used in PyLith development. You keep the main branch on your local machine in sync with the main branch in the community repository; we never use the main branch in your GitHub repository. From the main branch on your local machine, you create a feature branch, e.g., `feature-powerlaw`, to complete a single task such as adding a feature, fixing a bug, or making an improvement to the documentation. You should break up the changes into several steps, which are saved locally as commits. The commits are pushed to your repository on GitHub for backup or syncing across multiple computers. Once the task is completed, you submit a pull request to have the changes merged to the main branch in the community repository. Once the pull request is merged, you update your local main branch from the community repository and then push the changes to your repository on GitHub.

You should also sign your commits using GPG or S/MIME. See [GitHub Docs: Managing commit signature verification](#) for more information.

Set the upstream repository

For each clone of your fork (each computer with a local copy of your fork), you need to create a link to the “upstream” geodynamics/pylith repository. This allows you to keep your repository in sync with the community repository. If you are using the PyLith Docker development container and followed all of the installation instructions, then you already completed this step; you can list the current remotes for your fork to verify.

Listing 4.346: Setting upstream repository

```
# List the current remotes for your fork.
git remote -v
# Output
origin https://github.com/YOUR_GITHUB_USERNAME/pylith.git (fetch)
origin https://github.com/YOUR_GITHUB_USERNAME/pylith.git (push)

# Set the link to the remote upstream repository
git remote add upstream https://github.com/geodynamics/pylith.git

# Verify the upstream repository has been added.
git remote -v
# Output:
origin https://github.com/YOUR_GITHUB_USERNAME/pylith.git (fetch)
origin https://github.com/YOUR_GITHUB_USERNAME/pylith.git (push)
upstream https://github.com/geodynamics/pylith.git (fetch)
upstream https://github.com/geodynamics/pylith.git (push)
```

Creating a new feature branch

Listing 4.347: Creating a feature branch

```
# Start from the current development branch (usually "main")
git checkout main

# Make sure it is up to date.
git pull

# Create a new branch from 'main'.
git checkout -b BRANCH-NAME

# Examples
git checkout -b feature-powerlaw-rheology
git checkout -b fix-fault-output
```

Tip

If you are implementing a feature that requires a significant amount of code development, we strongly recommend that you break the implementation into pieces that can each be tested, documented, and integrated into the PyLith main branch. Another approach (equally valid) is to create a series of feature branches implementing the different

phases that all get merged into the main feature branch; the main feature branch would then be merged into the main branch via a pull request.

Tip

Feature branches are a great way to experiment with an implementation. You can create a feature branch and if you decide the implementation is headed in the wrong direction, you can simply create a new feature branch from your original starting point and delete the bad feature branch.

Staging, Committing, and Pushing Changes

The Git `add` and `commit` commands are used to stage and commit changes to a branch. Staging refers to assembling the list of files to include in a commit. A commit adds code changes to the current branch along with a message describing the changes. A commit changes only the current branch on your local machine. In order to update your GitHub repository you need to push your changes. See the Git documentation for details about these commands. There are Git interfaces built into a number of editors and integrated development environments (IDEs), as well as standalone graphical user interfaces to Git.

Tip

Commit messages should explain changes. Keep the first line under 80 characters, if possible, and include an empty line between the first line and any additional lines that provide more detailed explanations.

The commit messages provide important documentation on why code is changed. Your future self and your fellow developers will appreciate good explanations of all changes.

Warning

If you have multiple branches, make sure you are on the correct branch before making commits.

Tip

You should only make commits on your local feature branches. These are also the only branches that you should push to your GitHub repository.

We strongly recommend never pushing branches from the upstream repository to your GitHub repository.

Keeping a feature branch in sync with branch `main`

The PyLith `main` branch may change while you are working on a feature branch. In some cases it might have a new feature that you do not need and does not affect your work, in which case it does not matter whether you incorporate those changes into your feature branch. However, in other cases, there might be an important bugfix or feature that you want to use in your feature branch while you are working on it. The recommended way to incorporate these changes is to rebase your feature branch relative to the PyLith `main` branch.

Rebasing essentially replays your commits on top of the commits in the other branch. With interactive rebasing you can also rewrite the commit history of a feature branch by reordering, dropping, and/pr combining (squashing) commits. See [Git: Rewriting History](#) for more information.

Important

Make sure all of your local changes have been committed or [stashed](#).

Danger

Rebasing and force pushing can cause irreversible damage to your repository. We recommend practicing rebasing using a toy repository before attempting to rebase with real code.

The primary steps to rebasing with respect to the PyLith main branch are:

1. Commit or stash any changes so that you do not have any modified files.
2. Start the interactive rebasing by running `git rebase -i main`.
3. Edit the commit history as desired.
4. Fix any conflicts during the rebase.
5. Verify the rebase is correct by running tests, etc.
6. Force push the branch using `git push --force`.

Danger

After you have rebased but before doing a forced push, running `git status` will report something like:

```
On branch saradeveloper/my-great-new-feature
Your branch and 'origin/saradeveloper/my-great-new-feature' have diverged,
and have 56 and 16 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
```

Do not run `git pull!!!!` This is one of the few times that you should *not* do what git suggests. Doing so will try to merge your original history into your new one. Instead, you should verify that the new commit history is correct and then do a force push by running `git push --force`.

Tip

If anytime *during* the rebasing process, you make a mistake or decide you want to abort the rebasing process, simply run `git rebase --abort`. This will return the repository to the state immediately before the rebasing process.

Creating a pull request

Once you have completed implementing, testing, and documenting a new feature on a branch in your fork and wish to contribute it back to the `geodynamics/pylith` repository, you open a pull request. See [GitHub Docs: About pull requests](#) for more information. When you create a pull request, GitHub will run a series of tests using a temporary copy with the merged changes using GitHub actions. The GitHub webpage for the pull request will have a **Checks** tab showing the progress of the tests and any errors. One or more of the PyLith maintainers will review your pull request and may request changes. You can continue to make commits to your branch; the pull request will be automatically updated whenever you push the changes to your GitHub repository.

 Tip

Be sure the checkbox `Allow edits and access to secrets by maintainers` is checked so maintainers can make corrections.

 Tip

To become familiar with making pull requests, we recommend starting with a small, simple change. This may be as little as fixing a typo in the documentation or a comment. Create a feature branch for the change, push it to your repository, and then make a pull request.

Adding remotes for accessing other PyLith forks

When collaborating with other people working on PyLith, it is helpful to be able to checkout branches from their forks. You can add their fork as an additional “remote” repository.

Listing 4.348: Adding an additional remote repository to track branches in other forked repositories.

```
# Add remote
git remote add FORK-NAME https://github.com/GITHUB_USERNAME/pylith.git
# Example:
git remote add saradeveloper https://github.com/saradeveloper/pylith.git

# Show remotes
git remote -v

# Fetch the information for the remote
git fetch FORK-NAME
# Example:
git fetch saradeveloper

# Checkout remote branch
git checkout -b saradeveloper/feature-powerlaw-rheology

# Push to remote branch (requires write access)
git push FORK-NAME BRANCH
# Example:
git push saradeveloper feature-powerlaw-rheology
```

Git Quick Reference

Break commit into multiple commits

```
git rebase -i main
# Set commit to 'edit', exit, and save.
git reset HEAD~
# Make changes.
git rebase --continue
```

Update from upstream

1. Set remote (one time)

```
# Show current remotes
git remote -v

# Add remote upstream
git remote add upstream UPSTREAM_REPO

# Verify addition of upstream remote
git remote -v
```

2. Update upstream repo

```
git fetch upstream -p
```

Rebase branch off main

```
# Make sure main is current
git pull main

# Check out branch you want to rebase
git checkout BRANCH

# Rebase branch with respect to `main`
git rebase -i main
```

Branches

- Checkout branch and set tracking: `git checkout -b $BRANCH --track $REMOTE/$BRANCH`
- Compare current heads: `git diff branch1..branch2`
- Compare with respect to common ancestor: `git diff main...feature`
- Compare commits: `git log branch1..branch2`
- Compare specific file: `git diff main..feature -- FILE`
- Compare specific file with working tree: `git diff main-- FILE`

Delete branches

- Local: `git branch -d BRANCH`
- Remote: `git push origin --delete BRANCH`

Rename local branch

```
git branch -m OLD NEW
```

Show merged/unmerged branches

```
git branch --merged $BRANCH
git branch --no-merged $BRANCH
```

4.3.6 Debugging

Debugging tools

VS Code integration

You can use the VS Code debugger with C++ unit tests (libtests), MMS Tests, and PyLith simulations.

C++ unit test

1. Click on **Run** and **Debug** in the left sidebar of the VS Code window.
2. At the top of the panel, select the test suite to run.
3. Click on the green triangle next to the name of the selected test suite.

MMS test

1. Click on **Testing** in the left sidebar of the VS Code window.
2. Click on the **Debug Test** icon that pops up when you mouse over the name of the test you want to run.

Note

Once we migrate the C++ unit tests to Catch2, they will also show up in list of tests under **Testing**.

PyLith simulation

We provide a configuration in `launch.json` to attach the VS Code debugger to a running process, such as a PyLith simulation.

1. To attach the debugger, start the PyLith simulation while adding the command line arguments `--petsc.stop_for_debugger --petsc.pause_debugger=10` (10 corresponds to the number of seconds the program will wait for you to attach the debugger before continuing).
2. Note the process id that is printed to stdout.
3. Select **Run** and **Debug** in the left sidebar of the VS Code window.
4. Select **Attach to process** at the top of the panel.
5. Select the `mpinemesi` process with the process id that matches the one printed to stdout.

Tip

These PETSc command line arguments used to attach the debugger are provided in `$PREFIX/share/settings/attach_debugger.cfg` where `$PREFIX` is the directory where PyLith is installed.

Debugger quick reference

Please see the gdb and lldb documentation for detailed instructions. Here we illustrate some common basic commands.

Listing 4.349: Debugging with gdb

```
# Set breakpoint at line 150 of Material.cc
(gdb) b Material.cc:150

# Set breakpoint at exception throw
(gdb) catch throw

# Show arguments for the current frame
(gdb) info args

# Show local variables for the current frame
(gdb) info locals

# Show the contents of a local variable: p VARIABLE
(gdb) p numFields

# Show the contents of local array: p POINTER[0]@SIZE
# Print array of 4 values pointed to by variable values
(gdb) p values[0]@4

# Print stack trace
(gdb) backtrace
```

Listing 4.350: Debugging with lldb

```
(lldb) b Material.cc:150

# Set breakpoint at exception throw
(lldb) break set -E C++

# Show local variables
(lldb) frame variable

# Show the contents of a local variable: frame variable VARIABLE
(lldb) frame variable numFields
# Alternatively
(lldb) p numFields

# Show the contents of an array of values: parray SIZE POINTER
# Show the contents of array of 10 values pointed to by the variable values.
(lldb) parray 10 values
```

Listing 4.351: Printing PETsc and PyLith objects (same for gdb and lldb)

```
# Print PETSc section "section"
(gdb) call PetscSectionView(section, 0)
```

(continues on next page)

(continued from previous page)

```
# Print PETSc DS "ds"
(gdb) call PetscDSView(ds, 0)

# Print PyLith mesh "mesh"
# Formats:
#   "::ascii_info_detail" - ASCII to stdout
#   "::mesh.txt:ascii_info_detail" - ASCII to mesh.txt
#   "::mesh.tex:ascii_latex" - LaTeX to mesh.tex
#   "vtk:mesh.vtu:vtu_vtu" - VTK to mesh.vtu
(gdb) call mesh.view "::ascii_info_detail"

# Print PyLith field "field"
# Options values:
#   0: metadata only
#   1: Metadata and section
#   2: Metadata and PETSc vector
#   3: Metadata, section, and PETSc vector
call field.view("MY LABEL", pylith::topology::Field::ViewOptions(1))
```

Valgrind quick reference

Valgrind is a useful tool for finding memory leaks, use of uninitialized variables, and invalid reads and writes to memory. When running valgrind there are three very useful command line arguments:

- **--log-filename=FILENAME** Send output to FILENAME. This does not work when running the PyLith application because each new process wipes out the log file.
- **----suppressions=FILE** Omit errors matching given patterns when reporting errors. Valgrind often reports lots of errors arising from the way OpenMPI and Python handle memory allocation and deallocation. We usually use the Python suppression file `share/valgrind-python.supp` when running valgrind.
- **--trace-children=yes** Continue tracing errors in subprocesses. This is important when running valgrind on the PyLith executable, as the actual computation is done in a forked process.

Viewing fields

In addition to using the debugger to inspect code and variables, it is often helpful to print fields to stdout or inspect where a computed field does not match the expected field. Turning on this type of output is usually done by activating debug journals via command line arguments.

Viewing a field will print the subfield metadata, the layout of the field, and the field values. See *pylith::topology::Field*, *PetscSection*, and *PetscVec* for how to interpret the layout of a field.

Listing 4.352: Output from `Field::view()` for solution field with displacement and fluid pressure subfields.

```
Viewing field 'solution' Solution Field.
Subfields: # Order of subfields is given by the index, not the order listed.
Subfield displacement, index: 0, components: displacement_x displacement_y, scale: 1000, basisOrder: 1, quadOrder: 1
Subfield fluid_pressure, index: 2, components: fluid_pressure, scale: 0.1, basisOrder: 1, quadOrder: 1
Subfield velocity, index: 1, components: velocity_x velocity_y, scale: 100, basisOrder: 1, quadOrder: 1
```

(continues on next page)

(continued from previous page)

```

    dimensionalize flag: 0
DM Object: 1 MPI processes
    type: plex
DM_0xe6a550_38 in 2 dimensions:
    0-cells: 5
    1-cells: 8
    2-cells: 4
Labels:
    boundary_bottom: 1 strata with value/size (1 (3))
    boundary: 1 strata with value/size (1 (8))
    material-id: 1 strata with value/size (24 (4))
    depth: 3 strata with value/size (0 (5), 1 (8), 2 (4))
PetscSection Object: 1 MPI processes
    type not yet set
3 fields
    field 0 with 2 components # displacement vector field
Process 0:
(  0) dim 0 offset  0
(  1) dim 0 offset  0
(  2) dim 0 offset  0
(  3) dim 0 offset  0
(  4) dim 2 offset  0 constrained 1 # y degree of freedom is constrained
(  5) dim 2 offset  5 constrained 1 # y degree of freedom is constrained
(  6) dim 2 offset 10
(  7) dim 2 offset 15
(  8) dim 2 offset 20
(  9) dim 0 offset 25
( 10) dim 0 offset 25
( 11) dim 0 offset 25
( 12) dim 0 offset 25
( 13) dim 0 offset 25
( 14) dim 0 offset 25
( 15) dim 0 offset 25
( 16) dim 0 offset 25
    field 1 with 2 components # velocity vector field
Process 0:
(  0) dim 0 offset  0
(  1) dim 0 offset  0
(  2) dim 0 offset  0
(  3) dim 0 offset  0
(  4) dim 2 offset  2
(  5) dim 2 offset  7
(  6) dim 2 offset 12
(  7) dim 2 offset 17
(  8) dim 2 offset 22
(  9) dim 0 offset 25
( 10) dim 0 offset 25
( 11) dim 0 offset 25
( 12) dim 0 offset 25
( 13) dim 0 offset 25
( 14) dim 0 offset 25
( 15) dim 0 offset 25

```

(continues on next page)

(continued from previous page)

```

( 16) dim 0 offset 25
field 2 with 1 components # pressure scalar field
Process 0:
( 0) dim 0 offset 0
( 1) dim 0 offset 0
( 2) dim 0 offset 0
( 3) dim 0 offset 0
( 4) dim 1 offset 4
( 5) dim 1 offset 9
( 6) dim 1 offset 14
( 7) dim 1 offset 19
( 8) dim 1 offset 24
( 9) dim 0 offset 25
(10) dim 0 offset 25
(11) dim 0 offset 25
(12) dim 0 offset 25
(13) dim 0 offset 25
(14) dim 0 offset 25
(15) dim 0 offset 25
(16) dim 0 offset 25
Proc 0 local vector # 5 nondimensionalized values per point: displacement (2), velocity_
→(2), pressure (1)
Vec Object: unknown 1 MPI processes
type: seq
-0.999 # offset 0, point 4, x-displacement
-4.2 # offset 1, point 4, y-displacement
-9.99 # offset 2, point 4, x-velocity
-9.99 # offset 3, point 4, y-velocity
-9990. # offset 4, point 4, pressure
-0.999 # offset 5, point 5, x-displacement
0.6
-9.99
-9.99
-9990.
-0.999 # offset 10, point 6, x-displacement
0.
-9.99
-9.99
-9990.
-0.999 # offset 15, point 7, x-displacement
-0.6
-9.99
-9.99
-9990.
-0.999 # offset 20, point 8, x-displacement
4.2
-9.99
-9.99
-9990.

```

Viewing differences

In tests in which we compare a computed field against one from an analytical solution using `DMPlexComputeL2DiffLocal()` and the fields do not agree, it is generally helpful to determine which pieces do not agree. The `DMPlex` object contains an internal switch to print the point-by-point differences while computing the norm. This switch can be activated using the `--petsc dm_plex_print_l2=1` command line argument in the C++ and MMS tests.

Listing 4.353: Debugging output from `DMPlexComputeL2DiffLocal()`

```
Cell 0 Element Solution for Field 0 # displacement vector field
| -0.999 | # Values of solution field variable at vertices of cell 0
| -4.2 |
| -0.999 |
| 0. |
| -0.999 |
| -0.6 |
elem 0 field 0 diff 0. # Differences at quadrature points with respect to field_
→given by analytical function
elem 0 field 0 diff 6.27226e-32
elem 0 field 0 diff 2.24281e-33
elem 0 field 0 diff 8.97125e-33
elem 0 field 0 diff 0.
elem 0 field 0 diff 0.
elem 0 field 0 diff 2.24281e-33
elem 0 field 0 diff 0.
Cell 0 Element Solution for Field 1 # velocity field vector field
| -9.99 |
| -9.99 |
| -9.99 |
| -9.99 |
| -9.99 |
| -9.99 |
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
elem 0 field 1 diff 0.
Cell 0 Element Solution for Field 2 # pressure scalar field
| -9990. |
| -9990. |
| -9990. |
elem 0 field 2 diff 0.
elem 0 field 2 diff 0.
elem 0 field 2 diff 0.
elem 0 field 2 diff 0.
elem 0 diff 7.61795e-32
Cell 1 Element Solution for Field 0
| -0.999 | # Values of solution field variable at vertices of cell 1
| 0.6 |
```

(continues on next page)

(continued from previous page)

```

| -0.999 |
| 0. |
| -0.999 |
| -4.2 |
elem 1 field 0 diff 0.
elem 1 field 0 diff 3.92016e-33
elem 1 field 0 diff 2.24281e-33
elem 1 field 0 diff 1.4354e-31
elem 1 field 0 diff 0.
elem 1 field 0 diff 0.
elem 1 field 0 diff 2.24281e-33
elem 1 field 0 diff 0.
Cell 1 Element Solution for Field 1
| -9.99 |
| -9.99 |
| -9.99 |
| -9.99 |
| -9.99 |
| -9.99 |
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
elem 1 field 1 diff 0.
Cell 1 Element Solution for Field 2
| -9990. |
| -9990. |
| -9990. |
elem 1 field 2 diff 0.
elem 1 field 2 diff 0.
elem 1 field 2 diff 0.
elem 1 field 2 diff 0.
elem 1 diff 1.51946e-31
... # Output continues for values in other cells

```

Running tests in CI Docker containers

If one or more of the CI test runners report an error or someone reports an error using a specific Linux distribution, it is often convenient to run a CI test runner interactively on your local machine using Docker. Currently, we maintain Docker images containing all of the PyLith dependencies for the following Linux distributions:

- debian-stable
- debian-testing
- ubuntu-20.04
- ubuntu-22.04
- ubuntu-23.04
- fedora-37

- fedora-38
- centos-7
- rockylinux-8
- rockylinux-9

You need to checkout the PyLith branch that you want to test and be in the top-level PyLith source directory.

```
# Change to the top-level PyLith source directory.
cd $TOP_SRCDIR/pylith

# Set the name of the Linux distribution to use to the BASE_IMAGE environment
# variable.
BASE_IMAGE=debian-stable

# Build a new Docker image with the PyLith source code. The base image will
# be downloaded as necessary. The "--target build" command line argument
# will run "make install" but not "make check". To stop at the configure step,
# use "--target src".
$ docker build \
  -t pylith-debug \
  --build-arg BASE_IMAGE=registry.gitlab.com/cig-pylith/pylith_installer/testenv-$BASE_
  IMAGE \
  --target build \
  -f docker/pylith-testenv .

# Run the Docker image interactively. This allows you to run "make check" or
# run tests manually, potentially making use of the gdb debugger and valgrind.
$ docker run -ti --rm pylith-debug /bin/bash
```

4.3.7 Performance

Performance Tools

New in v4.1.0.

Overview

We use PETSc event logging to instrument the PyLith code for evaluating runtime performance. This allows us to simultaneously evaluate the performance of the PyLith C++ code as well as the PETSc code. In PyLith v2, we logged the floating point operations for the residual and Jacobian evaluation. We have not yet included these in PyLith v3 and later because we now use kernels that are called many, many times and event logging in this methods would introduce overhead and reduce performance.

Evaluating Performance

 Tip

We can get a general ideal of runtime performance bottlenecks with a debugging build of the code, but fine-tuning performance should be done using an optimized build.

Quick View

Run a PyLith simulation adding the command line argument `--petsc.log_view=ascii:FILENAME`. This will dump performance information in ASCII format to `FILENAME`.

Graphical View

Run a PyLith simulation adding the command line argument `--petsc.log_view=:FILENAME_LOGVIEW:ascii_flamegraph`. This will dump performance information to `FILENAME_LOGVIEW` that can be viewed using flame graph tools, such as [FlameGraph](#).

Viewing Performance with FlameGraph

1. Run `PATH_TO_FLAMEGRAPH/flamegraph.pl FILENAME_LOGVIEW > FILENAME_PLOT.svg`
2. Load `FILENAME_PLOT.svg` into your favorite svg viewer, such as a web browser.

 Note

You can install [FlameGraph](#) by cloning the repository or downloading and unpacking a release.

Logging

PyLith includes `pylith::utils::EventLogger` as a high-level interface for event logging with PETSc. We create local `Events` classes within the C++ implementation files (*.cc) to create variables that hold the event ids. This streamlines logging events within the C++ code. We identify events using variables rather than strings, which means that typos in names are detected at compile time rather than at runtime. Refer to the `_Problem::Events` class near the top of `libsrc/pylith/problems/Problem.cc` for an example of how we implement event logging.

We name events in PyLith C++ code following the template `PL:CLASS:METHOD` (C++ namespace style but reducing `::` to `:` to reduce the length of the string) in which we are logging method `METHOD` in C++ class `CLASS`. In the PyLith Python code, we follow the template `PL.CLASS.METHOD`.

4.3.8 Contributing to PyLith**Documentation**

We write the PyLith documentation in Markedly Structured Text (a Markdown version of reStructured Text) and generate the manual using Sphinx. The manual is automatically built for the main branch and posted at <https://pylith.readthedocs.io>.

Building the documentation

You can build a local copy of the documentation using Sphinx.

Prerequisites

You do not have to install PyLith to generate a local copy of the documentation. Nevertheless, you must have Python and the following Python packages installed:

- **sphinx** (v3.5 or later)
- **myst-parser** (0.14.0 or later)
- **pydata-sphinx-theme** (0.6.2 or later)
- **sphinx-copybutton**
- **sphinxcontrib.bibtex**
- **sphinx_design**

These packages are all included in the PyLith development Docker container.

Generating the documentation

Use the `build.sh` script in the `docs` directory to generate the documentation. The default format is `html`, which will place the files in the `_build/html` directory. PDF and epub formats can also be generated.

```
cd docs

# Generate the documentation as html.
./build.sh

# Generate the documentation as a PDF.
./build.sh latex
cd _build/latex && make

# Generate the documentation as an epub.
./build.sh epub
```

MyST Quick reference

Style guide

1. Use Markedly Structured Text (MyST), not reStructured Text (rST).
2. **Place each sentence on its own single line.**

Headings

Use `# Heading 1` only at the top of each page.

Heading 2

Heading 3

Heading 4

Refer to *Heading 2*.

Admonitions

General admonition as warning

Text goes here.

Attention

This is an attention admonition.

Danger

This is a danger admonition.

Error

This is an error admonition.

Important

This is an important admonition.

Note

This is a note admonition.

Tip

This is a tip admonition.

Warning

This is a warning admonition.

See also

This is a seealso admonition.

i TODO

This is a custom TODO admonition.

Lists**Itemized lists**

- Level 1a
 - Level 2a
 - * Level 3a
 - * Level 3b
 - Level 2b
- Level 1b
- Level 1c

Definition lists**Term 1**

Definition of term 1

Term 2

Definition of term 2

Field lists**field 1**

Description of field 1

field 2

Description of field 2

Code blocks

Listing 4.354: C++ code block.

```
int
main(int argc, char* argv[]) {
    // Emphasized lines corresponding to body of main().
    return 0;
}
```

Listing 4.355: Python code block.

```
def square(x):
    return x**2
```

Listing 4.356: Interactive shell.

```
$ ls
a b c
```


Listing 4.357: Bash code block.

```
# Comment
for i in "a b c"; do
    print $i
done
```

Listing 4.358: Config code block.

```
# Comment
[pylithapp]
journal.device = color-console

[pylithapp.petsc]
ksp_rtol = 1.0e-3
```

Tables

Please see [Table 4.45](#). Table labels cannot contain more than one dash, so we use colons to separate the words in the label. This is likely a bug.

Table 4.45: Table caption

Header 1	Header 2	Header 3
right aligned	centered	left aligned
more data, more data	yet more data	even more data

Figures

Please see [Fig. 4.212](#). Figure labels cannot contain more than one dash, so we use colons instead. This is likely a bug.



Fig. 4.212: This is the figure caption. Vector graphics should be provided in **both** PDF (latex output) and SVG (html output) formats. Raster graphics should be provided in either PNG or JPG formats (whichever is more compact).

Math

For unlabeled equations you can use AMSTeX environments, such as `equation`, `gather` and `align`. Unfortunately, Sphinx does not currently support labeled equations using AMSTeX environments. Labeled equations must use the Markdown syntax as in [equation \(4.566\)](#).

Warning

A group of equations only has a single label, so if you need to label individual equations, place each one in its own math block.

Here is a single equation:

$$F = ma \quad (4.561)$$

Here is a group of equations:

$$f_1(x, y) = a_1x + b_1y + c_1 \quad (4.562)$$

$$f_2(x, y) = a_2x + b_2y + c_2 \quad (4.563)$$

Here is a group of aligned equations:

$$f_1(x, y) = a_1x + b_1y + c_1 \quad (4.564)$$

$$f_2(x, y) = a_{xx}x^2 + a_{xy}xy + a_{yy}y^2 + a_x x + a_y y + a_0 \quad (4.565)$$

Here is a labeled equation (equation (4.566)):

$$F = ma \quad (4.566)$$

Citations

Traditional citation [[Aagaard *et al.*, 2007](#)]. Citation as noun Bathe [[1995](#)].

Table of contents

```
# Table of contents tree with a maximum depth of 1
:::{toctree}
---
maxdepth: 1
---
one.md
two.md
:::
```

```
# Table of contents tree with no maximum depth specified.
:::{toctree}
one.md
two.md
:::
```

Contributing to the documentation

In browsing the HTML documentation there is a one-to-one correspondence between an HTML page and its Markdown page. You can look at the URL of the HTML page and the corresponding Markdown page will be in the docs directory with the `.html` suffix replaced by `.md`.

Workflow

1. Update your local main branch
2. Create feature branch

3. Implement feature
4. Build and install
5. Implement associated tests
6. Build tests
7. Run/debug tests
8. Add appropriate documentation
9. Create pull request
10. Maintainers review pull request
11. Revise pull request
12. Maintainers merge pull request

Note

You will likely need to update your main branch during this process to keep it in sync with the main branch in `geodynamics/pylith`.

Creating a new example

The workflow for creating a new example is similar to the workflow for creating a new feature. A new example is added using a feature branch with additions to the `examples` directory and documentation.

Checklist

1. Example files
 1. Example files: `README.md`, `Makefile.am`, parameter files, spatial databases, mesh generation scripts.
 2. Add directory in `examples/Makefile.am`
 3. Update `configure.ac` to include example `Makefile`.
2. User Guide
 1. Include sections: overview, meshing, common, steps, exercises
 2. Update `update-synopsis.sh` (generate list of features from example parameter files) to include example directory.
 3. Use include directive in example Markdown file to add list of features, etc in synopsis file.
 4. Add example to table and toc in `examples/index.md`.
 5. Add files to `docs/Makefile.am`
 6. Check that all Markdown files have each sentence on its own single line.

Coding Style

There are a number of standard coding styles for programming languages, notably PEP8 for Python. For PyLith, we try to be consistent in naming conventions across Python and C++ while following a subset of the conventions used in PETSc and PEP8 with documentation styles consistent with Doxygen.

■ Important

We use 4 spaces for indentation. Configure your editor to use spaces instead of tabs.

General guidelines

- Naming conventions
 - Use self-documenting names.
 - Avoid single letter variables. Choose meaningful names that will be found via searches across single or multiple files (e.g., `grep`).
 - Class names are generally nouns and methods are verbs.
 - Class names use upper camel case, e.g., `TimeDependent`.
 - Public method names use camel case, e.g., `computeRHSResidual()`.
 - Protected and private method names use camel case preceded by an underscore, e.g., `_setFEKernelsRHSResidual()`.
 - In C++ data members are private and use camel case preceded by an underscore, e.g., `_gravityField`.
 - In Python data members are public and use camel case, e.g., `self.gravityField`.
 - Local variables use camel case, e.g., `numIntegrators`.
- Comments
 - Include standard banner at the very beginning of every file.
 - For every class method, describe its function and include a description for each argument. For Python this is done in the docstring of the method, and for C++ this is done in a doxygen style comment immediately before the method declaration in the header file.
 - Document nontrivial algorithms and any assumptions.
- Error checking
 - PyLith should never crash without an error message.
 - All user errors should be trapped as early as possible and reported with an informative error message by throwing an appropriate exception. If possible, suggest ways to correct the error.
 - Messages for internal errors should indicate the location in the code where the error was trapped.
 - All pointers should be checked for NULL values before use. Usually we use `assert()` to do this.
 - Check the return values for all calls to functions in external libraries. For PETSc functions, we use `PyLithCallPetsc(PetscFunction(args));`.
- Testing
 - All C++ methods should be covered by unit tests.
 - All governing equations should be covered by Method of Manufactured solution tests.
 - All functionality should be covered by full-scale tests.

Formatting source files

We use `autopep8` and `uncrustify` to format Python and C/C++ source files, respectively. The corresponding configuration files are `autopep8.cfg` and `uncrustify.cfg` in the `developer` directory. The Python script `developer/format_source.py` is a handy utility for calling `autopep8` and `uncrustify` with the appropriate arguments and formatting multiple files.

Tip

We highly recommend using an integrated development environment, such as Visual Studio Code, that allow `uncrustify` and `autopep8` to automatically format all C/C++ and Python source files.

Listing 4.359: Formatting Python and C++ source code using `format\_source.py`. `autopep8` and `uncrustify` must be in the current path.

```
# Format a C++ file.
developer/format_source.py --cplusplus=libsrc/pylith/materials/Material.cc

# Format all C++ files in the 'libsrc/pylith/materials' directory.
developer/format_source.py --cplusplus=libsrc/pylith/materials/*.cc

# Format a Python file.
developer/format_source.py --python=pylith/materials/Material.py

# Format all Python files in the 'pylith/materials' directory.
developer/format_source.py --python=pylith/materials/*.py
```

Error Checking

Our philosophy is that PyLith should never crash without an error message. If it encounters a fatal error, then it should generate an appropriate error message and abort. In C++ we throw `std::runtime_error` exceptions for errors resulting from user input and `std::logic_error` exceptions for internal inconsistencies or logic errors. In Python we use standard exception objects.

Additional protections against crashing include: using asserts to verify pointers are non-null before using them and using the `PyLithCallPetsc()` macro to check the return value for *every* call to a PETSc function.

Listing 4.360: Example of using `assert()`

```
assert(_solution); // Verify _solution is not NULL.

// Initialize integrators.
const size_t numIntegrators = _integrators.size();
for (size_t i = 0; i < numIntegrators; ++i) {
    assert(_integrators[i]); // Verify _integrators[i] is not NULL.
    _integrators[i]->initialize(*_solution);
} // for
```

Tip

When we build the code for production runs, we usually configure with `CPPFLAGS=-DNDEBUG` to remove `assert()` calls.

Listing 4.361: Example of using PyLithCallPetsc() macro.

```
PyLithCallPetsc(TSGetTimeStep(ts, &dt));
```

In combination with the above procedures, we also make use of the Pyre journals to display warnings and errors to facilitate debugging. The journals provide the file name and line number along with the message. By default, Pyre journals for errors are turned on and those for warnings and debugging are turned off. The header file `utils/journals.hh` defines several useful macros for reporting warnings and errors.

Listing 4.362: Example of using Pyre journals and standard exceptions.

```
switch (bitUse) {
case 0x1:
    _bcKernel = pylith::fekernels::TimeDependentFn::initial_scalar;
    break;
case 0x2:
    _bcKernel = pylith::fekernels::TimeDependentFn::rate_scalar;
    break;
case 0x0:
    PYLITH_COMPONENT_WARNING("Dirichlet BC provides no constraints.");
    break;
default:
    PYLITH_FIREWALL(pylith::InternalLogicError, pylith::journals::internal,
        "Unknown combination of flags for Dirichlet BC terms "
        "<< "(useInitial="<<_useInitial<< ", useRate="<<_useRate<<").");
} // switch
```

C/C++ style

Object Declaration Files

C++ object declaration (header) files use the `.hh` suffix. C header files use the `.h` suffix.

Important

All declarations of class methods should include a description of what the method does and a description of each argument and the return value if it is not void.

Object Implementation Files

C++ object implementation files use the `.cc` suffix. Inline implementation files use the `.icc` suffix and are included from the definition (header) files. C implementation files use the `.c` suffix.

To facilitate debugging and error messages, we use the following macros:

PYLITH_METHOD_BEGIN

This macro allows line numbers of source files to be included in PETSc error messages. Use this macro at the beginning of all methods using any PETSc routines as well as most other methods. We don't use this macro in destructors because many of them are called *after* `PetscFinalize`. We also do not use this macro in trivial or inline methods that do not call any PETSc routines.

PYLITH_METHOD_END

Use the macro at the end of all methods that begin with `PYLITH_METHOD_BEGIN` and return void.

PYLITH_RETURN_END

Use this macro at the end of all methods that begin with `PYLITH_METHOD_BEGIN` and return non-void values.

PyLithCallPetsc()

Use this macro in *every* call to a PETSc function to check the return value.

PYLITH_JOURNAL_DEBUG

Use this macro immediately after `PYLITH_METHOD_BEGIN` in methods of all objects that inherit from `GenericComponent`.

PYLITH_COMPONENT_DEBUG

Use this macro immediately after `PYLITH_METHOD_BEGIN` in methods of all objects that inherit from `PyreComponent`. Non-abstract classes should call `PyreComponent::setName()` in the constructor. We recommend using a static data member for the name with the lowercase name matching the Pyre component, e.g., “timedependent” for the C++ `TimeDependent` object.

Python style

Python files use the `.py` suffix. All classes, methods, and functions should be documented using docstrings.

Adding New Governing Equations and/or Bulk Rheologies**Overview**

There are four basic tasks for adding new physics in the form of a governing equation:

1. Select the fields for the solution. This will control the form of the partial differential equation and the terms in the residuals and Jacobians.
2. Derive the pointwise functions for the residuals and Jacobians. Determine flags that will be used to indicate which terms to include.
3. Determine which parameters in the pointwise functions could vary in space as well as any state variables. We bundle all state variables and spatially varying parameters into a field called the auxiliary field. Each material has a separate auxiliary field.
4. Parameters that are spatially uniform are treated separately from the parameters in the auxiliary field.

`Material` is responsible for the terms in the governing equations associated with the domain (i.e., volume integrals in a 3D domain and surface integrals in a 2D domain). A separate object implements the bulk rheology for a specific governing equation. [Fig. 4.213](#) shows the objects used to implement multiple rheologies for the elasticity equation: an isotropic, linear elastic rheology for incompressible elasticity, and an isotropic, linear elastic rheology for poroelasticity. The `Elasticity` object describes the physics for the elasticity equation, including the pointwise functions and flags for turning on optional terms (such as inertia) in the governing equation, and `RheologyElasticity` defines the interface for bulk elastic rheologies.

Python

- Define solution subfields.
 - All subfields in the solution field are `SolutionSubfield` objects (see [Fig. 4.214](#)). PyLith already includes several solution subfields:
 - * `SubfieldDisplacement` Displacement vector field.
 - * `SubfieldVelocity` Velocity vector field.
 - * `SubfieldLagrangeFault` Lagrange multiplier field for fault constraints.
 - * `SubfieldPressure` Fluid pressure or mean stress scalar field.

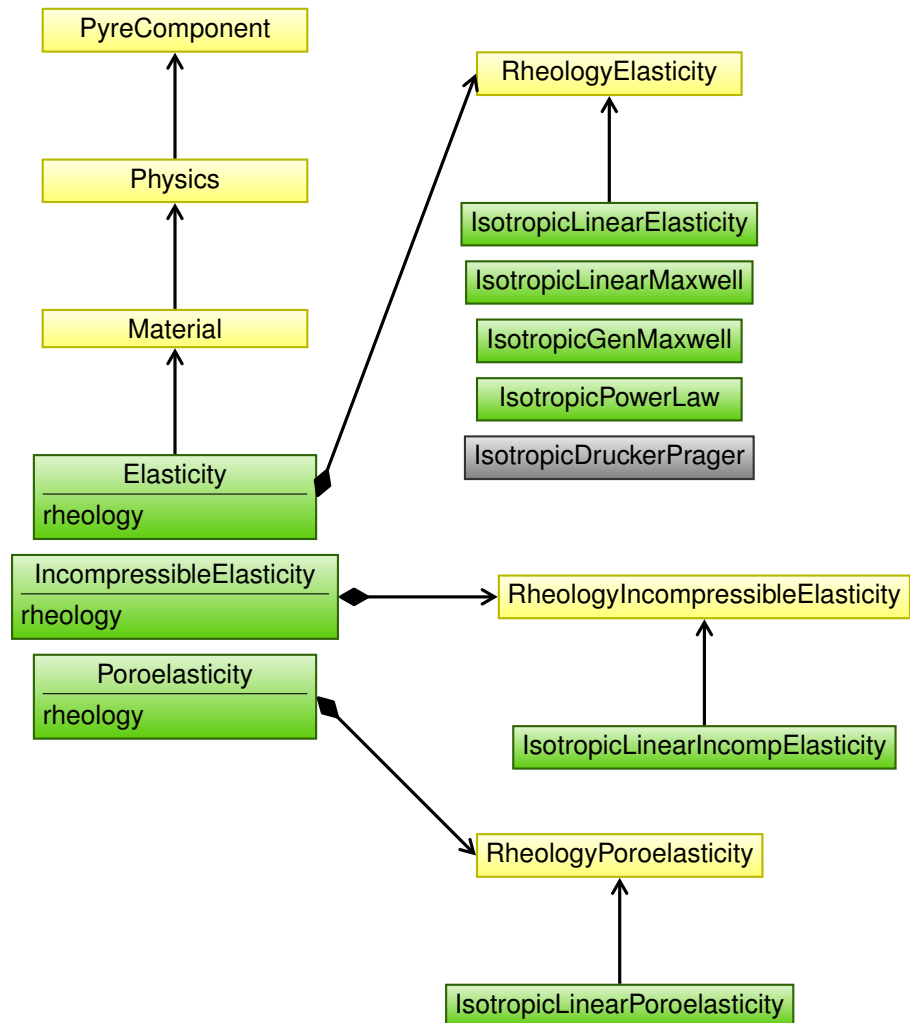


Fig. 4.213: Class diagram for the implementation of governing equations and bulk rheologies. Each governing equation implementation inherits from the abstract `Material` class and bulk rheologies inherit from the abstract rheology class specific to that governing equation.

- * SubfieldTemperature Temperature scalar field.
- PyLith includes solution field containers with predefined subfields:
 - * SolnDisp Solution composed of a displacement field.
 - * SolnDispVel Solution composed of displacement and velocity fields.
 - * SolnDispPres Solution composed of displacement and mean stress (pressure) fields.
 - * SolnDispLagrange Solution composed of displacement and Lagrange multiplier fields.
 - * SolnDispPresLagrange Solution composed of displacement, mean stress (pressure), and Lagrange multiplier subfields.
 - * SolnDispVelLagrange Solution composed of displacement, velocity, and Lagrange multiplier subfields.
- Define auxiliary subfields.

The auxiliary subfields for a governing equation are defined as facilities in a Pyre Component. For example, the ones for Elasticity are in `AuxFieldsElasticity`. The order of the subfields is defined *not* by the order they are listed in the Pyre component, but by the order they are added to the auxiliary field in the C++ object. The auxiliary subfields bulk rheologies are defined in the same way.

Important

A single auxiliary field will be created for each material; it contains the auxiliary subfields from both the governing equation and the bulk rheology.

- Flags to turn on/off terms in governing equation.

For the elasticity equation, we sometimes do not include body forces or inertial terms in our simulations. Rather than implement these cases as separate materials, we simply include flags in the material to turn these terms on/off. The flags are implemented as Pyre properties in our material component.

C++

- Define auxiliary subfields.

We build the auxiliary field using classes derived from `pylith::feassemble::AuxiliaryFactory`. The method corresponding to each subfield specifies the name of the subfield, its components, and scale for nondimensionalizing. We generally create a single auxiliary factory object for each governing equation, but not each bulk constitutive model, because constitutive models for the same governing equation often have many of the same subfields. For example, most of our bulk constitutive models for the elasticity contain density, bulk modulus, and shear modulus auxiliary subfields.

Important

Within the concrete implementation of the material and bulk rheology objects, we add the subfields to the auxiliary field. The order in which they are added determines the order they will be in the auxiliary field. You will need to know this order when you implement the pointwise functions. See [Fig. 4.215](#) for more information on the layout of the auxiliary field.

- Implement the pointwise functions.

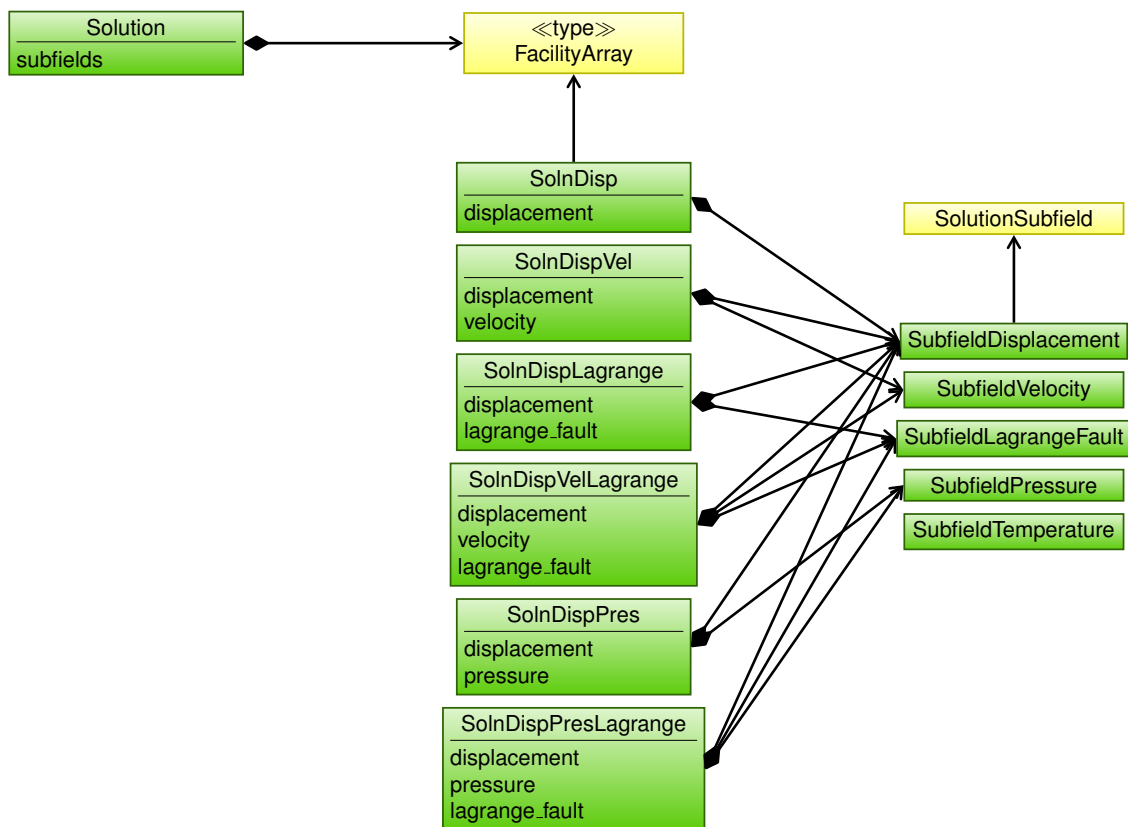


Fig. 4.214: Class diagram for the solution field, solution subfields, and pre-defined containers of solution subfields.

The pointwise functions for the residuals, Jacobians, and projections follow nearly identical interfaces. In most cases you should implement the terms in the governing equation in 3D with 2D plane strain using the more general 3D version. While this is slightly less efficient, it results in less code to maintain.

- Set the pointwise functions.

We set the pointwise functions for the RHS and LHS residuals and Jacobians, taking into consideration which optional terms of the governing equation have been selected by the user.

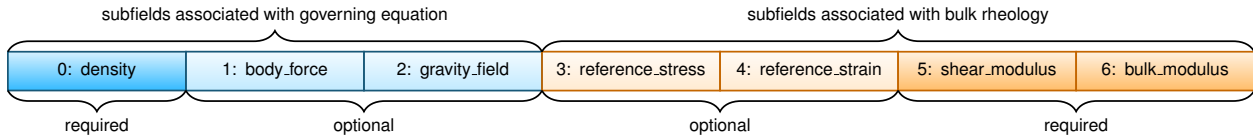


Fig. 4.215: Layout of material auxiliary subfields. The subfields include those for both the governing equation and the bulk rheology. The required subfields are at the ends with the optional fields in the middle. This allows the same pointwise functions to be used for some cases with and without the optional subfields.

4.3.9 Integrated development environments

Visual Studio Code

Visual Studio Code (VS Code) provides a extensive integrated development environment for writing code, source control, building, and running tests. The PyLith repository is setup to include local settings for VS Code for building, running tests, and debugging.

Setting up VS Code

The PyLith settings for VS Code make use of the following environment variables, which are automatically set in the PyLith development Docker container. If you are working in VS Code with local source code, then we recommend setting these environment variables as part of activating the PyLith virtual environment.

Environment variable	Decription
PETSC_DIR	Directory for PETSc
PETSC_ARCH	Build label for PETSc debugging configuration
PYLITH_BUILDDIR	Top-level directory where we build PyLith ¹
PYLITH_DIR	Directory containing CIG-related dependencies ¹
PYLITHDEPS_DIR	Directory containing external dependencies ¹
PYTHON_INCDIR	Directory containing Python header files ¹
MPI_INCDIR	Directory containing MPI header files ¹
PROJ_INCDIR	Directory containing Proj header files ¹
CATCH_INCDIR	Directory containing Catch2 header files ¹

Starting VS Code

If you have PyLith source code on your local machine, then it is best to start VS Code from a terminal in which you have activated the PyLith virtual environment. This helps VS Code know the location of dependencies.

```
# Change to top-level PyLith source directory
cd src/pylith
```

(continues on next page)

¹ Used in VS Code tasks.

(continued from previous page)

```
# Start VS Code and open the top-level directory
code .
```

If you are connecting to the PyLith development Docker container, then you can start VS Code using a terminal or the application icon. Select **File->Open folder...** and navigate to the top-level PyLith source directory; this ensures VS Code loads the settings from `.vscode` in the top-level PyLith source directory.

Extensions

We strongly recommend the following extensions to leverage all of the default settings included in the PyLith repository.

- C/C++
- Python
- Uncrustify
- C++ TestMate
- Test Explorer UI
- markdownlint
- MyST-Markdown

Other extensions that you may find useful include:

- Dev Containers
- Live Share
- autoconf
- Code Spell Checker
- Markdown all in One
- Material Icon Theme
- Remote-SSH
- Git Graph
- Git Lens

4.4 References

BIBLIOGRAPHY

- [AWK07] B. Aagaard, C. Williams, and M. Knepley. Pylith: a finite-element code for modeling quasistatic and dynamic crustal deformation. In *Eos Trans. Fall Meet. Suppl.*, volume 88. AGU, 2007. Abstract T21B-0592.
- [AHH01a] Brad T. Aagaard, John F. Hall, and Thomas H. Heaton. Characterization of near-source ground motions with earthquake simulations. *Earthquake Spectra*, 17(2):177–207, May 2001.
- [AHH01b] Brad T. Aagaard, Thomas H. Heaton, and John F. Hall. Dynamic earthquake ruptures in the presence of lithostatic normal stresses: implications for friction models and heat production. *Bulletin of the Seismological Society of America*, 91(6):1765–1796, December 2001.
- [Bat95] K.-J. Bathe. *Finite-Element Procedures*. Prentice Hall, Upper Saddle River, New Jersey, 1995. 1037 pp.
- [Bru70] James N. Brune. Tectonic stress and spectra of seismic shear waves from earthquakes. *Journal of Geophysical Research*, 75:4997–5009, September 10 1970.
- [Che16] Alexander H.-D. Cheng. *Poroelasticity*. Springer International Publishing, 2016. doi:10.1007/978-3-319-25202-5.
- [CH23] Colin Cotter and David Ham. *Finite Elements analysis and implementation*. Imperial College, London, 2023. URL: <https://finite-element.github.io/>.
- [DC93] E. Detournay and Alexander H.-D. Cheng. Fundamentals of poroelasticity. In Charles Fairhurst, editor, *Analysis and Design Methods*, pages 113–171. Pergamon, Oxford, 1993. doi:10.1016/B978-0-08-040615-2.50011-3.
- [DCC13] Boyang Ding, Alexander H-D Cheng, and Zhanglong Chen. Fundamental solutions of poroelastodynamics in frequency domain based on wave decomposition. *Journal of Applied Mechanics*, 2013.
- [Far21] Patrick E. Farrell. *Finite Element Methods for PDEs*. Oxford, 2021. URL: <https://people.maths.ox.ac.uk/farrell/femvideos/notes.pdf>.
- [HWJ12] Hayes, G. P., D. J. Wald, and R. L. Johnson. Slab1.0: a three-dimensional model of global subduction zone geometries. *Journal of Geophysical Research*, 2012. doi:10.1029/2011JB008524.
- [KK87] S. H. Kirby and A. K. Kronenberg. Rheology of the lithosphere: selected topics. *Reviews of Geophysics*, 25:1219–1244, 1987.
- [KB87] M. Kojic and K.-J. Bathe. The ‘effective stress-function’ algorithm for thermo-elasto-plasticity and creep. *Int. J. Num. Meth. Eng.*, 24:1509–1532, 1987.
- [LAH06] P. Liu, R. J. Archuleta, and S. H. Hartzell. Prediction of broadband ground-motion time histories: Hybrid low/high- frequency method with correlated random source parameters. *Bulletin of the Seismological Society of America*, 96(6):2118–2130, December 2006. doi:10.1785/0120060036.

- [NBHA23] J.M. Nevitt, B.A. Brooks, J.L. Hardebeck, and B.T. Aagaard. 2019 M7.1 Ridgecrest earthquake slip distribution controlled by fault geometry inherited from Independence dike swarm. *Nature Communications*, 2023. doi:[10.1038/s41467-023-36840-2](https://doi.org/10.1038/s41467-023-36840-2).
- [Oka92] Y. Okada. Internal deformation due to shear and tensile faults in a half-space. *Bulletin of the Seismological Society of America*, 82(2):1018–1040, 1992.
- [Pat94] W. S. B. Paterson. *The Physics of Glaciers*. Elsevier Science Ltd., Oxford, third edition edition, 1994. 480 pp.
- [Pre68] J. H. Prentice. Dimensional problem of the power law in rheology. *Nature*, 217:157, 1968. doi:[10.1038/217157a0](https://doi.org/10.1038/217157a0).
- [SS86] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi:[10.1137/0907058](https://doi.org/10.1137/0907058).
- [SP78] J. C. Savage and W. H. Prescott. Asthenosphere readjustment and the earthquake cycle. *Journal of Geophysical Research*, 83:3369–3376, 1978.
- [Tay03] R. L. Taylor. *FEAP—A Finite Element Analysis Program*. version 7.5 theory manual edition, 2003. 154 pp.
- [UMG22] C. Uphoff, D.A. May, and A.-A. Gabriel. Electronic supplement to A discontinuous Galerkin method for sequences of earthquakes and aseismic slip on multiple faults using unstructured curvilinear grids. *Geophysical Journal International*, 2022. doi:[10.5281/zenodo.5796104](https://doi.org/10.5281/zenodo.5796104).
- [WC03] Z. L. Wu and Y. T. Chen. Definition of seismic moment at a discontinuity interface. *Bulletin of the Seismological Society of America*, 93(4):1832–1834, August 2003. doi:[10.1785/0120020234](https://doi.org/10.1785/0120020234).
- [ZT00] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method*. Volume Volume 2: Solid Mechanics. Butterworth-Heinemann, Oxford, fifth edition edition, 2000.
- [NOAANCfEInformation22] NOAA National Centers for Environmental Information. *ETOPO 2022 15 Arc-Second Global Relief Model*. 2022. doi:[10.25921/fd45-gt74](https://doi.org/10.25921/fd45-gt74).