Introduction to PyLith v3.0

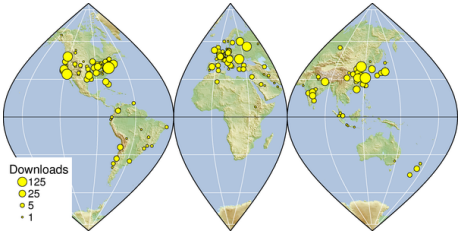
Brad Aagaard

June 10, 2019

PyLith

A modern, community-driven code for crustal deformation modeling

- Developers
  - Brad Aagaard (USGS)
  - Matthew Knepley (Rice University)
  - Charles Williams (GNS Science)
- Combined dynamic modeling capabilities of EqSim (Aagaard) with the quasi-static modeling capabilities of Tecton (Williams)
- Use modern software engineering to develop an open-source, community code
  - Modular design
  - Testing
  - Documentation
  - Distribution
- PyLith v1.0 was released in 2007



Crustal Deformation Modeling

Elasticity problems where geometry does not change significantly

Quasi-static modeling associated with earthquakes

- Strain accumulation associated with interseismic deformation
  - What is the stressing rate on faults X and Y?
  - Where is strain accumulating in the crust?
- Coseismic stress changes and fault slip
  - What was the slip distribution in earthquake A?
  - How did earthquake A change the stresses on faults X and Y?
- Postseismic relaxation of the crust
  - What rheology is consistent with observed postseismic deformation?
  - Can aseismic creep or afterslip explain the deformation?

Crustal Deformation Modeling

Elasticity problems where geometry does not change significantly

Dynamic modeling associated with earthquakes

- Modeling of strong ground motions
  - Forecasting the amplitude and spatial variation in ground motion for scenario earthquakes
- Coseismic stress changes and fault slip
  - How did earthquake A change the stresses on faults X and Y?
- Earthquake rupture behavior
  - What fault constitutive models/parameters are consistent with the observed rupture propagation in earthquake A?

Crustal Deformation Modeling

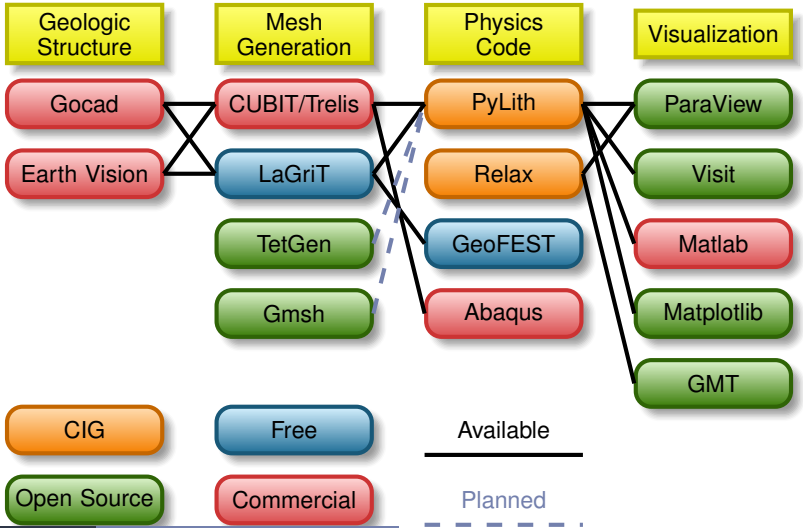
Elasticity problems where geometry does not change significantly

Volcanic deformation associated with magma chambers and/or dikes

- Inflation
  - What is the geometry of the magma chamber?
  - What is the potential for an eruption?
- Eruption
  - Where is the deformation occurring?
  - What is the ongoing potential for an eruption?
- Dike intrusions
  - What is the geometry of the intrusion?
  - What is the pressure change and/or amount of opening/dilatation?

Crustal Deformation Modeling

Overview of workflow for typical research problem



PyLith v3.0

- Multiphysics formulation through point-wise integration kernels
- Higher order spatial and temporal discretizations
- Adaptive time stepping via PETSc TS
- Improved fault formulation for spontaneous rupture (coming in v3.1)
- Many other small changes

Aside: Finite-Element Method

Strong form to weak form

Solve governing equation in integrated sense:

$$\int_{\Omega} \psi_{trial} \cdot PDE \, d\Omega = 0, \tag{1}$$

by minimizing the error with respect to the unknown coefficients.

This leads to equations of the form:

$$\int_{\Omega} \psi_{trial} \cdot f_0(x,t) + \nabla \psi_{trial} \cdot f_1(x,t) \, d\Omega = 0. \tag{2}$$

Governing Equations

We want to solve equations in which the weak form can be expressed as

$$F(t, s, \dot{s}) = G(t, s) \tag{3}$$

$$s(t_0) = s_0 \tag{4}$$

where F and G are vector functions, t is time, and s is the solution vector.

Using the finite-element method and divergence theorem, we cast the weak form into

$$\int_\Omega \vec{\psi}_{trial} \cdot \vec{f}_0(t, s, \dot{s}) + \nabla \vec{\psi}_{trial} : \boldsymbol{f}_1(t, s, \dot{s})\, d\Omega = \int_\Omega \vec{\psi}_{trial} \cdot \vec{g}_0(t, s) + \nabla \vec{\psi}_{trial} : \boldsymbol{g}_1(t, s)\, d\Omega, \tag{5}$$

where $\vec{f}_0$ and $\vec{g}_0$ are vectors, and $\boldsymbol{f}_1$ and $\boldsymbol{g}_1$ are tensors.

Explicit Time Stepping

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

Normally $F(t, s, \dot{s})$ contains the inertial term ($\rho\ddot{u}$).

Therefore, we transform our equation into the form:

$$F^*(t, s, \dot{s}) = \dot{s} = G^*(t, s) \tag{6}$$

$$\dot{s} = M^{-1}G(t, s). \tag{7}$$

Solving the Equations

Explicit time stepping requires a subset of the terms used in implicit time stepping.

- PETSc TS object provides time-stepping and solver implementations
  - Application code provides functions for computing RHS and LHS residuals and Jacobians
- Explicit time stepping
  - Compute RHS residual, $G(t, s)$
  - Compute lumped inverse of LHS, M^{-1}
  - No need to compute LHS residual, because $F(t, s, \dot{s}) = \dot{s}$
- Implicit time stepping (Krylov solvers)
  - Compute RHS residual, $G(t, s)$
  - Compute LHS residual, $F(t, s, \dot{s})$
  - Compute RHS Jacobian, $J_G = \frac{\partial G}{\partial s}$
  - Compute LHS Jacobian, $J_F = \frac{\partial F}{\partial s} + s_{tshift} \frac{\partial F}{\partial \dot{s}}$

Example: Elasticity with Prescribed Slip

Use domain decomposition and Lagrange multipliers to prescribe slip

Implicit time stepping without inertia

$$\vec{s}^T = (\vec{u} \quad \vec{\lambda})^T, \tag{8}$$

$$\vec{0} = \vec{f}(\vec{x}, t) + \boldsymbol{\nabla} \cdot \boldsymbol{\sigma}(\vec{u}) \text{ in } \Omega, \tag{9}$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau}(\vec{x}, t) \text{ on } \Gamma_\tau, \tag{10}$$

$$\vec{u} = \vec{u}_0(\vec{x}, t) \text{ on } \Gamma_u, \tag{11}$$

$$\vec{0} = \vec{d}(\vec{x}, t) - \vec{u}^+(\vec{x}, t) + \vec{u}^-(\vec{x}, t) \text{ on } \Gamma_f, \tag{12}$$

$$\boldsymbol{\sigma} \cdot \vec{n} = -\vec{\lambda}(\vec{x}, t) \text{ on } \Gamma_{f^+}, \tag{13}$$

$$\boldsymbol{\sigma} \cdot \vec{n} = +\vec{\lambda}(\vec{x}, t) \text{ on } \Gamma_{f^-}. \tag{14}$$

Example: Elasticity with Prescribed Slip (cont.)

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

$$0 = \int_{\Omega} \vec{\psi}^{u}_{trial} \cdot \left(\vec{f}(t) + \boldsymbol{\nabla} \cdot \boldsymbol{\sigma}(\vec{u}) \right) \, d\Omega, \tag{15}$$

$$0 = \int_{\Gamma_f} \vec{\psi}^{\lambda}_{trial} \cdot \left(\vec{d}(\vec{x}, t) - \vec{u}^{+}(\vec{x}, t) + \vec{u}^{-}(\vec{x}, t) \right) \, d\Gamma. \tag{16}$$

Using the divergence theorem and incorporating the Neumann boundary and fault interface conditions, we can rewrite the first equation as

$$\begin{aligned} 0 = \int_{\Omega} \vec{\psi}^{u}_{trial} \cdot \vec{f}(t) + \nabla \vec{\psi}^{u}_{trial} : -\boldsymbol{\sigma}(\vec{u}) \, d\Omega + \int_{\Gamma_\tau} \vec{\psi}^{u}_{trial} \cdot \vec{\tau}(\vec{x}, t) \, d\Gamma \\ + \int_{\Gamma_f} \vec{\psi}^{u^+}_{trial} \cdot -\vec{\lambda}(\vec{x}, t) + \vec{\psi}^{u^-}_{trial} \cdot +\vec{\lambda}(\vec{x}, t) \, d\Gamma. \end{aligned} \tag{17}$$

Example: Elasticity with Prescribed Slip (cont.)

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

$$F^u(t, s, \dot{s}) = 0, \tag{18}$$

$$F^\lambda(t, s, \dot{s}) = 0, \tag{19}$$

$$G^u(t, s) = \int_\Omega \vec{\psi}^u_{trial} \cdot \underbrace{\vec{f}(\vec{x}, t)}_{g_0^u} + \nabla \vec{\psi}^u_{trial} : \underbrace{-\boldsymbol{\sigma}(\vec{u})}_{g_1^u} \, d\Omega \tag{20}$$

$$+ \int_{\Gamma_\tau} \vec{\psi}^u_{trial} \cdot \underbrace{\vec{\tau}(\vec{x}, t)}_{g_0^u} \, d\Gamma + \int_{\Gamma_f} \vec{\psi}^{u^+}_{trial} \cdot \underbrace{-\vec{\lambda}(\vec{x}, t)}_{g_0^{u^+}} + \vec{\psi}^{u^-}_{trial} \cdot \underbrace{+\vec{\lambda}(\vec{x}, t)}_{g_0^{u^-}} \, d\Gamma, \tag{21}$$

$$G^\lambda(t, s) = \int_{\Gamma_f} \vec{\psi}^\lambda_{trial} \cdot \underbrace{\left(\vec{d}(\vec{x}, t) - \vec{u^+}(\vec{x}, t) + \vec{u^-}(\vec{x}, t) \right)}_{g_0^\lambda} \, d\Gamma. \tag{22}$$

Example: Elasticity with Prescribed Slip (cont.)

$$J_G^{uu} = \frac{\partial G^u}{\partial u} = \int_\Omega \nabla \vec{\psi}^u_{trial} : \frac{\partial}{\partial u}(-\boldsymbol{\sigma})\, d\Omega = \int_\Omega \nabla \vec{\psi}^u_{trial} : -\boldsymbol{C} : \frac{1}{2}(\nabla + \nabla^T)\vec{\psi}^u_{basis}\, d\Omega$$

$$= \int_\Omega \psi^v_{trial\,i,k} \underbrace{(-C_{ikjl})}_{J^{uu}_{g3}} \psi^u_{basis\,j,l}\, d\Omega \tag{23}$$

$$J_G^{u\lambda} = \frac{\partial G^u}{\partial \lambda} = \int_{\Gamma_{f+}} \vec{\psi}^u_{trial} \cdot \frac{\partial}{\partial \lambda}(-\vec{\lambda})\, d\Gamma + \int_{\Gamma_{f-}} \vec{\psi}^u_{trial} \cdot \frac{\partial}{\partial \lambda}(+\vec{\lambda})\, d\Gamma$$

$$= \int_{\Gamma_f} \psi^{u^+}_{trial\,i} \underbrace{-1}_{J^{u^+\lambda}_{g0}} \psi^\lambda_{basis\,j} + \psi^{u^-}_{trial\,i} \underbrace{+1}_{J^{u^-\lambda}_{g0}} \psi^\lambda_{basis\,j}\, d\Gamma \tag{24}$$

$$J_G^{\lambda u} = \frac{\partial G^\lambda}{\partial u} = \int_{\Gamma_f} \vec{\psi}^\lambda_{trial} \cdot \frac{\partial}{\partial u}\left(\vec{d}(\vec{x},t) - \vec{u^+}(\vec{x},t) + \vec{u^-}(\vec{x},t)\right) d\Gamma$$

$$= \int_{\Gamma_f} \psi^\lambda_{trial\,i} \underbrace{(-1)}_{J^{\lambda u^+}_{g0}} \psi^{u^+}_{basis\,j} + \psi^\lambda_{trial\,i} \underbrace{(+1)}_{J^{\lambda u^-}_{g0}} \psi^{u^-}_{basis\,j}\, d\Gamma \tag{25}$$

(26)

Example: Incompressible Elasticity

Implicit time stepping without inertia

$$\vec{s}^T = (\vec{u} \quad p)^T, \tag{27}$$

$$\vec{0} = \vec{f}(t) + \boldsymbol{\nabla} \cdot \left(\boldsymbol{\sigma}^{dev}(\vec{u}) - p\boldsymbol{I} \right) \text{ in } \Omega, \tag{28}$$

$$0 = \vec{\nabla} \cdot \vec{u} + \frac{p}{K}, \tag{29}$$

$$\boldsymbol{\sigma} \cdot \vec{n} = \vec{\tau} \text{ on } \Gamma_\tau, \tag{30}$$

$$\vec{u} = \vec{u}_0 \text{ on } \Gamma_u, \tag{31}$$

$$p = p_0 \text{ on } \Gamma_p. \tag{32}$$

Example: Incompressible Elasticity (cont.)

Using trial functions $\vec{\psi}^u_{trial}$ and ψ^p_{trial} and incorporating the Neumann boundary conditions:

$$0 = \int_\Omega \vec{\psi}^u_{trial} \cdot \vec{f}(t) + \nabla\vec{\psi}^u_{trial} : \left(-\boldsymbol{\sigma}^{dev}(\vec{u}) + p\boldsymbol{I}\right) d\Omega + \int_{\Gamma_\tau} \vec{\psi}^u_{trial} \cdot \vec{\tau}(t)\, d\Gamma, \tag{33}$$

$$0 = \int_\Omega \psi^p_{trial} \cdot \left(\vec{\nabla} \cdot \vec{u} + \frac{p}{K}\right) d\Omega. \tag{34}$$

Identifying $G(t,s)$, we have

$$0 = \int_\Omega \vec{\psi}^u_{trial} \cdot \underbrace{\vec{f}(t)}_{g^u_0} + \nabla\vec{\psi}^u_{trial} : \underbrace{\left(-\boldsymbol{\sigma}^{dev}(\vec{u}) + p\boldsymbol{I}\right)}_{g^u_1} d\Omega + \int_{\Gamma_\tau} \vec{\psi}^u_{trial} \cdot \underbrace{\vec{\tau}(t)}_{g^u_0} d\Gamma, \tag{35}$$

$$0 = \int_\Omega \psi^p_{trial} \cdot \underbrace{\left(\vec{\nabla} \cdot \vec{u} + \frac{p}{K}\right)}_{g^p_0} d\Omega. \tag{36}$$

Example: Incompressible Elasticity (cont.)

With two fields we have four Jacobians for the RHS associated with the coupling of the two fields.

$$J_G^{uu} = \frac{\partial G^u}{\partial u} = \int_\Omega \nabla \vec{\psi}^u_{trial} : \frac{\partial}{\partial u}(-\boldsymbol{\sigma}^{dev})\, d\Omega = \int_\Omega \psi^u_{trial\, i,k} \underbrace{\left(-C^{dev}_{ikjl}\right)}_{J^{uu}_{g3}} \psi^u_{basis\, j,l}\, d\Omega \tag{37}$$

$$J_G^{up} = \frac{\partial G^u}{\partial p} = \int_\Omega \nabla \vec{\psi}^u_{trial} : \boldsymbol{I} \psi^p_{basis}\, d\Omega = \int_\Omega \psi^u_{trial\, i,k} \underbrace{\delta_{ik}}_{J^{up}_{g2}} \psi^p_{basis}\, d\Omega \tag{38}$$

$$J_G^{pu} = \frac{\partial G^p}{\partial u} = \int_\Omega \psi^p_{trial} \left(\vec{\nabla} \cdot \vec{\psi}^u_{basis}\right) d\Omega = \int_\Omega \psi^p_{trial} \underbrace{\delta_{jl}}_{J^{pu}_{g1}} \psi^u_{basis\, j,l}\, d\Omega \tag{39}$$

$$J_G^{pp} = \frac{\partial G^p}{\partial p} = \int_\Omega \psi^p_{trial} \underbrace{\frac{1}{K}}_{J^{pp}_{g0}} \psi^p_{basis}\, d\Omega \tag{40}$$

Summary of Multiphysics Implementation

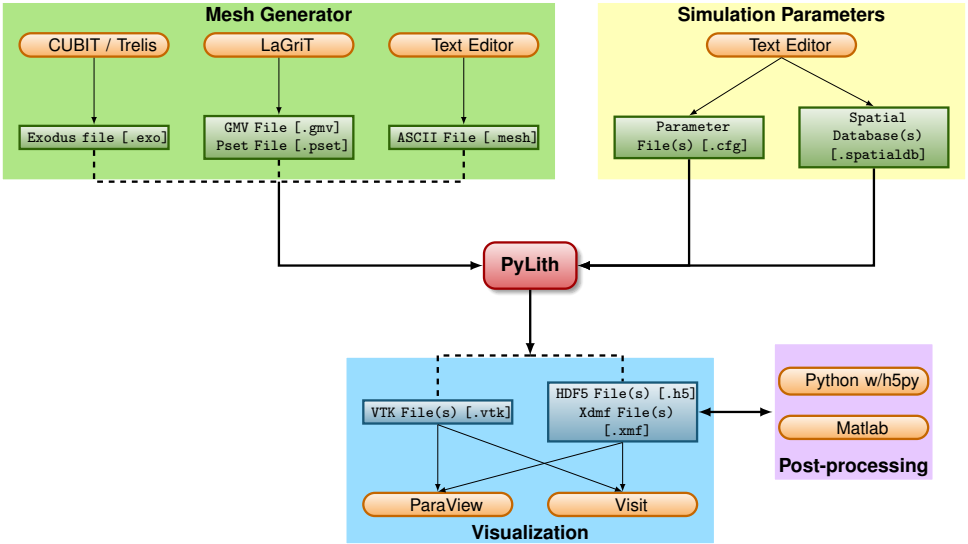
We decouple the element definition from the fully-coupled equation, using pointwise kernels that look like the PDE.

Flexibility The cell traversal, handled by the library, accommodates arbitrary cell shapes. The problem can be posed in any spatial dimension with an arbitrary number of physical fields.

Extensibility The library developer needs to maintain only a single method, easing language transitions (CUDA, OpenCL). A new discretization scheme could be enabled in a single place in the code.

Efficiency Only a single routine needs to be optimized. The application scientist is no longer responsible for proper vectorization, tiling, and other traversal optimization.

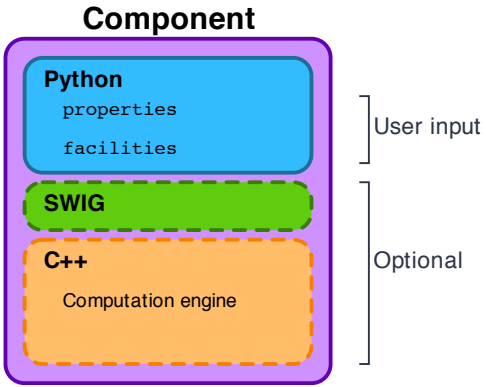
Overview of PyLith Workflow



PyLith as a Hierarchy of Components

Components are the basic building blocks

- Separate functionality into discrete modules (components)
- Alternative implementations use the same interfaces to allow plug-n-play
- Top-level interfaces in Python with computational code in C++
  - Python dynamic typing permits adding new modules at runtime.
  - Users can add functionality without modifying the PyLith code.



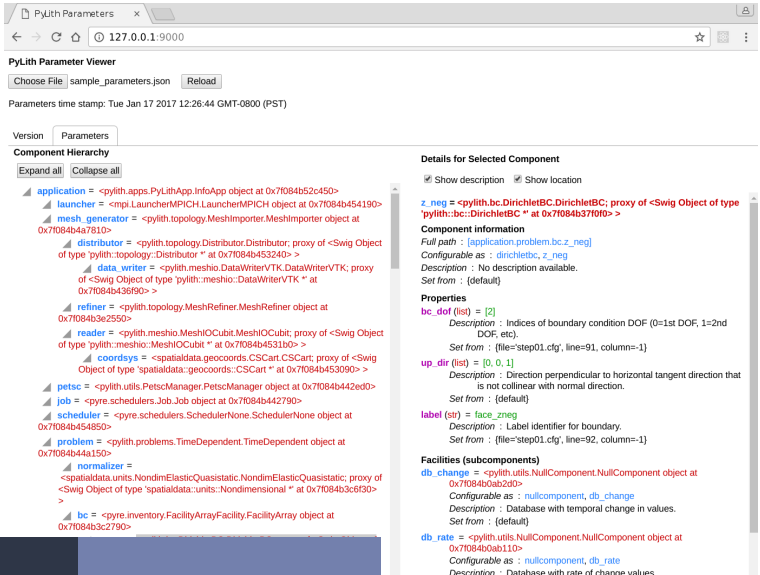
Parameter Files

Simple syntax for specifying parameters for properties and components

```
# Syntax
[pylithapp.COMPONENT.SUBCOMPONENT] ; Inline comment
COMPONENT = OBJECT
PARAMETER = VALUE

# Example
[pylithapp.mesh_generator] ; Header indicates path of mesh_generator in hierarchy
reader = pylith.meshio.MeshIOCubit ; Use mesh from CUBIT/Trelis
reader.filename = mesh_quad4.exo ; Set filename of mesh.
reader.coordsys.space_dim = 2 ; Set coordinate system of mesh.

[pylithapp.problem.solution_outputs.output] ; Set output format
writer = pylith.meshio.DataWriterHDF5
writer.filename = axialdisp.h5

[pylithapp.problem]
bc = [x_neg, x_pos, y_neg] ; Create array of boundary conditions
bc.x_neg = pylith.bc.DirichletTimeDependent ; Set type of boundary condition
bc.x_pos = pylith.bc.DirichletTimeDependent
bc.y_neg = pylith.bc.DirichletTimeDependent

[pylithapp.problem.bc.x_pos] ; Boundary condition for +x
constrained_dof = [0] ; Constrain x DOF
label = edge_xpos ; Name of nodeset from CUBIT/Trelis
db_auxiliary_fields = spatialdata.spatialdb.SimpleDB ; Set type of spatial database
db_auxiliary_fields.label = Dirichlet BC +x edge
db_auxiliary_fields.iohandler.filename = axial_disp.spatialdb ; Filename for database
```

Parameters Graphical User-Interface

`cd parametersgui; ./pylith_paramviewer`



Spatial Databases

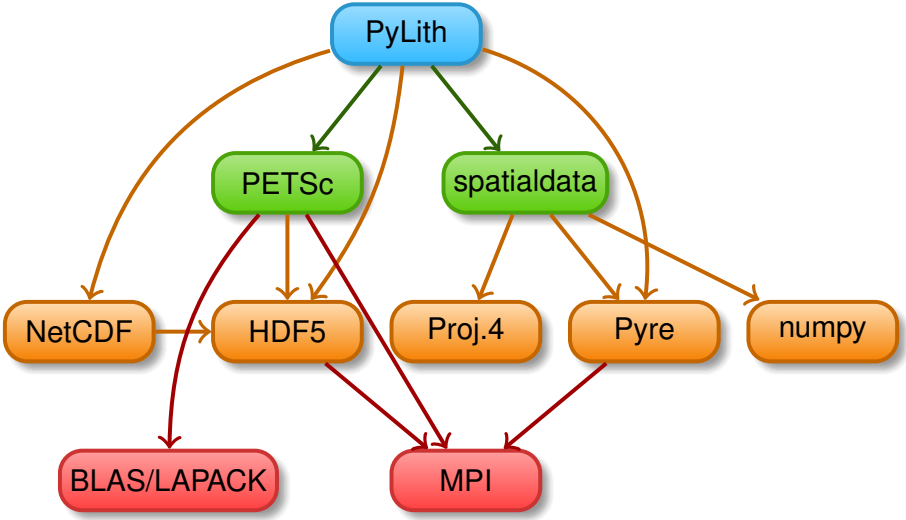
User-specified field/value in space for properties and BC values.

- Examples
  - Uniform value for Dirichlet BC (0-D)
  - Piecewise linear variation in tractions for Neumann BC (1-D)
  - SCEC CVM-H seismic velocity model (3-D)
- Generally independent of discretization for problem
- Available spatial databases

UniformDB	Optimized for uniform value
SimpleDB	Arbitrarily distributed points for variations in 0-D, 1-D, 2-D, or 3-D
SimpleGridDB	Logically gridded points for variations in 0-D, 1-D, 2-D, or 3-D
SCECCVMH	SCEC CVM-H seismic velocity model v5.3
ZeroDispDB	Special case of UniformDB

PyLith Design: Focus on Geodynamics

Leverage packages developed by computational scientists



PyLith Development Follows CIG Best Practices

github.com/geodynamics/best_practices

- Version Control
  - New features are added in separate branches.
  - Use 'master' branch as stable development branch.
- Coding
  - User-friendly specification of parameters at runtime.
  - Development plan, updated annually.
  - Users can add features or alternative implementations without modifying code.
- Portability
  - Build procedure is independent of compilers and optimization flags.
  - Multiple builds (debug/optimized) from same source.
- Documentation and User Workflow
  - Extensive example suite with varying levels of complexity.
  - Changing simulation parameters does not require rebuilding.
  - Displays version information via `--version` command line argument.

Development Tools

Leverage open-source tools for efficient code development.

GitHub	Code repository supporting simultaneous, independent implementation of new features.
Doxygen	Document parameters and purpose of every object and its functions.
CppUnit	Test nearly every function in code during development.
Travis CI	Run tests when code is committed to repository.
gcov	Records which lines of code tests cover.

Testing

Multiple levels of testing facilitates identifying bugs at origin.

unit tests	Serial testing at level of single and multiple functions.
MMS tests	Serial testing with Method of Manufactured Solutions (MMS) to verify implementation of governing equations
full-scale tests	Serial and parallel pass/fail tests of full problems.
benchmarks	Serial and parallel tests for code comparisons, etc.

PyLith v3.0.0beta1 (Jun 10, 2019)

Incomplete, contains bugs, but can do interesting physics

- Features (mesh importing) not changed remain stable.
- Some implemented features have been thoroughly tested.
- Some implemented features have minimal testing.
- A few implemented features have no testing.
- Several major features in v2.2 have not yet been implemented.

PyLith v3.0.0beta1: Governing Equations

Elasticity

- Static and quasi-static problems
- Dynamic problems (with inertia)
- Infinitesimal strains
- Small strain
- Gravitational body forces
- Body forces
- Bulk rheologies (constitutive models)
  - Isotropic, linear elasticity
  - Isotropic, linear Maxwell viscoelasticity
  - Isotropic, linear generalized Maxwell viscoelasticity
  - Isotropic, power-law viscoelasticity
  - Isotropic, Drucker-Prager elastoplasticity

Done
Buggy
In Progress
Coming Later

Incompressible Elasticity

- Static and quasi-static problems
- Infinitesimal strains
- Gravitational body forces
- Body forces
- Bulk rheologies (constitutive models)
  - Isotropic, linear elasticity
  - Isotropic, linear Maxwell viscoelasticity
  - Isotropic, linear generalized Maxwell viscoelasticity
  - Isotropic, power-law viscoelasticity

PyLith v3.0.0beta1: Boundary and Interface Conditions

- Boundary conditions
  - Time-dependent Dirichlet boundary conditions
  - Time-dependent Neumann (traction) boundary conditions
  - Absorbing boundary conditions
- Interface conditions
  - Kinematic (prescribed slip) fault interfaces w/multiple ruptures
  - Dynamic (friction) fault interfaces
    - Static friction
    - Linear slip-weakening
    - Linear time-weakening
    - Dieterich-Ruina rate and state friction w/ageing law

PyLith v3.0.0beta1: Other Features

- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT/Trelis: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Initial conditions
- Output: HDF5 and VTK files
  - Solution over domain
  - Solution over domain boundary
  - Solution interpolated to user-specified points w/station names
  - Solution over materials and boundary conditions
  - State variables (e.g., stress and strain) for each material
  - Fault information (e.g., slip and tractions)

PyLith v3.0.0beta1: Other Features (cont.)

- Automatic conversion of units for all parameters
- Parallel uniform global refinement
- PETSc linear and nonlinear solvers
- Output of simulation progress estimates runtime

How do changes from v2.x to v3.x affect users?

- No changes
  - Meshes
  - Formats of spatial database files
- **Substantial changes**
  - Parameter (cfg) files
  - Names of values in spatial database files

Mesh Generation Tips

There is no silver bullet in finite-element mesh generation

- Hex/Quad versus Tet/Tri
  - Hex/Quad are slightly more accurate and faster
  - Tet/Tri easily handle complex geometry
  - Easy to vary discretization size with Tet, Tri, and Quad cells
  - There is no easy answer
    For a given accuracy, a finer resolution Tet mesh that varies the discretization size in a more optimal way *might* run faster than a Hex mesh
- Check and double-check your mesh
  - Were there any errors when running the mesher?
  - Are the boundaries, etc marked correctly for your BC?
  - Check mesh quality (aspect ratio should be close to 1)

CUBIT/Trelis Workflow

1. Create geometry
   1. Construct surfaces from points, curves, etc or basic shapes
   2. Create domain and subdivide to create any interior surfaces
      - Fault surfaces must be interior surfaces (or a subset) that completely divide domain
      - Need separate volumes for different constitutive *models, not parameters*
2. Create finite-element mesh
   1. Specify meshing scheme
   2. Specify mesh sizing information
   3. Generate mesh
   4. Smooth to fix any poor quality cells
3. Create nodesets and blocks
   1. Create block for each constitutive model
   2. Create nodeset for each BC and fault
   3. **Create nodeset for buried fault edges**
   4. Create nodeset for ground surface for output (optional)
4. Export mesh in Exodus II format (`.exo` files)

CUBIT/Trelis Issues

Keep in mind the scales of the observations you are modeling

- Topography/bathymetry
  - Ignore topography/bathymetry unless you know it matters
  - For rectilinear grid, create UV net surface
  - Convert triangular facets to UV net surface via mapped mesh
- Fault surfaces
  - Building surfaces from contours is usually easiest
  - Include features at the resolution that matters
- Performance
  - Number of points in spline curves/surfaces has huge affect on mesh generation runtime
  - CUBIT/Trelis do not run in parallel
  - Use uniform global refinement in PyLith for large sims ($>$10M cells)

CUBIT/Trelis Best Practices

Issue: Changes in geometry cause changes in object ids
Soln: Name objects and use APREPRO or Python to eliminate hardwired ids wherever possible

Issue: Splines with many points slows down operations
Soln: Reduce the number of points per spline

Issue: Surfaces meet in small angles creating distorted cells
Soln: Trim geometry to eliminate features smaller than cell size

Issue: Difficulty meshing complex geometry with Hex cells
Soln: Use Tet cells even if it requires a finer mesh

Issue: Hex mesh over-samples parts of the domain
Soln: Use Tet mesh and vary discretization within domain

Issue: Extended surfaces create very complex geometry
Soln: Subdivide geometry before webcutting to eliminate overly complex geometry

PyLith Tips

- **Read the PyLith User Manual**
- **Do not ignore error messages and warnings!**
- Use an example/benchmark as a starting point
- Quasi-static simulations
  - Start with a static simulation and then add time dependence
  - **Check that the solution converges at every time step**
- Dynamic simulations
  - Start with a static simulation
  - **Shortest wavelength seismic waves control cell size**
- CIG community forums
  https://community.geodynamics.org/c/pylith
- PyLith User Resources
  https://wiki.geodynamics.org/software:pylith:start

Getting Started

1. **Create a play area for working with examples**

   ```
   cd PATH_TO_PYLITH_DIR
   mkdir playpen
   cp -r src/pylith-2.2.1rc1/examples playpen/
   ```

2. Work through relevant examples
3. Try to complete relevant exercises listed in the manual
4. Modify an example to look like your problem of interest

Overview of Examples

Examples progress from simple to more complex

1. 2d/box
   - Axial compress/extension w/Dirichlet BC
   - Shearing with Dirichlet and Neumann BC
2. 3d/box
   - Same as 2d/box in 3D
3. 2d/strikeslip
   - Variable mesh size in CUBIT/Trelis
   - Prescribed fault slip
   - Dirichlet boundary conditions
4. 2d/reverse
   - Gravitational body forces with linear elasticity
   - Gravitational body forces with incompressible elasticity
   - Prescribed slip on multiple faults

Overview of Examples (cont.)

Examples progress from simple to more complex

6. 2d/subduction
   - Meshing a 2-D cross-section of a subduction zone
   - Prescribed fault slip
   - Afterslip driven by traction changes from coseismic slip
7. 3d/strikeslip (wish list)
   - Meshing intersecting strike-slip faults with complex geometry
   - Prescribed fault slip
8. 3d/subduction
   - Meshing a 3-D sunduction zone with complex geometry
   - Prescribed fault slip