Overview of PyLith

Charles Williams
Brad Aagaard
Matthew Knepley

COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

June 18, 2012

What types of problems can we solve with PyLith?

Elasticity problems where geometry does not change significantly

Quasistatic 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?

What types of problems can we solve with PyLith?

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?

What types of problems can we solve with PyLith?

Elasticity problems where geometry does not change significantly

Volcanic deformation associated with magma chambers and/or dikes

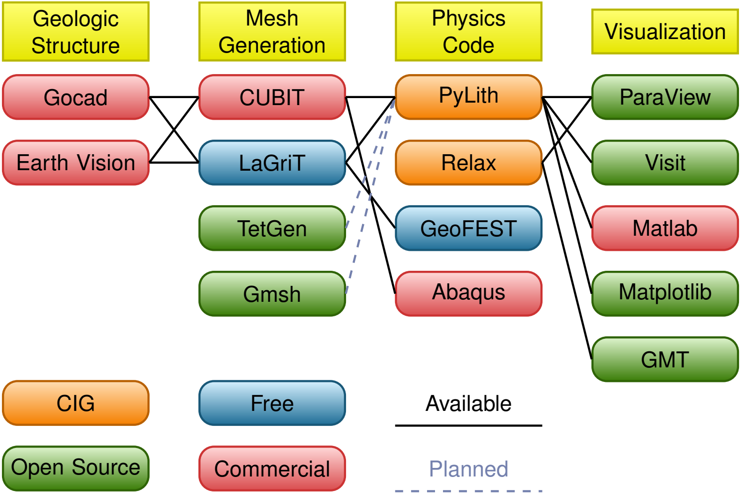
- Inflation
  - What is the geometry of the magma chamber?
  - What is the potential for an eruption?
- Eruption
  - Where is the deformation occurring?
  - What is the ongoing potential for an eruption?
- Dike intrusions
  - What the geometry of the intrusion?

PyLith Background

- Developers
  - Brad Aagaard (USGS, lead developer))
  - Charles Williams (GNS Science, formerly at RPI)
  - Matthew Knepley (Univ. of Chicago, formerly at ANL)
- Combined dynamic modeling capabilities of EqSim (Aagaard) with the quasistatic modeling capabilities of Tecton (Williams)
- Take advantage of recently-developed Sieve package in PETSc to handle mesh topology and related problems (Knepley)
- Use modern software engineering (modular design, testing, documentation, distribution) to develop an open-source, community code

PyLith Background

Overview of workflow for typical research problem



Governing Equations

Elasticity equation

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} - \vec{f} - \nabla \cdot \sigma = \vec{0} \text{ in } V, \quad (1)$$

$$\sigma \cdot \vec{n} = \vec{T} \text{ on } S_T, \quad (2)$$

$$\vec{u} = \vec{u}_0 \text{ on } S_u, \quad (3)$$

$$\vec{d} - (\vec{u}_+ - \vec{u}_-) = \vec{0} \text{ on } S_f. \quad (4)$$

Multiply by weighting function and integrate over the volume,

$$\int_V \vec{\phi} \cdot \left(\nabla \cdot \sigma + \vec{f} - \rho \frac{\partial^2 \vec{u}}{\partial t^2} \right) dV = 0. \quad (5)$$

After some algebra,

$$-\int_V \nabla \vec{\phi} : \sigma \, dV + \int_{S_T} \vec{\phi} \cdot \vec{T} \, dS + \int_V \vec{\phi} \cdot \vec{f} \, dV - \int_V \vec{\phi} \cdot \rho \frac{\partial^2 \vec{u}}{\partial t^2} \, dV = 0 \quad (6)$$

Writing the trial and weighting functions in terms of basis (shape) functions,

$$\vec{u} = \overline{N}_n \cdot \vec{u}_n, \tag{7}$$

$$\vec{\phi} = \overline{N}_m \cdot \vec{a}_m. \tag{8}$$

After some algebra, we obtain

$$-\int_V \nabla \overline{N}_m^T \cdot \sigma \, dV + \int_{S_T} \overline{N}_m^T \cdot \vec{T} \, dS + \int_V \overline{N}_m^T \cdot \vec{f} \, dV - \int_V \rho \overline{N}_m^T \cdot \overline{N}_n \cdot \frac{\partial^2 \vec{u}_n}{\partial t^2} \, dV = \vec{0}. \tag{9}$$

Governing Equations

Using numerical quadrature we convert the integrals to sums over the cells and quadrature points

$$
\begin{aligned}
-\sum_{\text{vol cells}} \sum_{\text{quad pts}} \nabla \overline{N}_m^T \cdot \sigma w_q |J_{\text{cell}}| + \sum_{\text{surf cells}} \sum_{\text{quad pts}} \overline{N}_m^T \cdot \vec{T} w_q |J_{\text{cell}}| \\
+ \sum_{\text{vol cells}} \sum_{\text{quad pts}} \overline{N}_m^T \cdot \vec{f} w_q |J_{\text{cell}}| \\
- \sum_{\text{vol cells}} \sum_{\text{quad pts}} \rho \overline{N}_m^T \cdot \overline{N}_n \cdot \frac{\partial^2 \vec{u}_n}{\partial t^2} w_q |J_{\text{cell}}| = \vec{0} \quad (10)
\end{aligned}
$$

Quasistatic Solution

Neglect inertial terms

Form system of algebraic equations

$$\overline{A}(t)\vec{u}(t) = \vec{b}(t) \tag{11}$$

where

$$\overline{A}(t) = \sum_{\text{vol cells}} \sum_{\text{quad pts}} \frac{1}{4}(\nabla^T + \nabla)\overline{N}_m^T \cdot \mathrm{C} \cdot (\nabla + \nabla^T)\overline{N}_n w_q |J_{\text{cell}}| \tag{12}$$

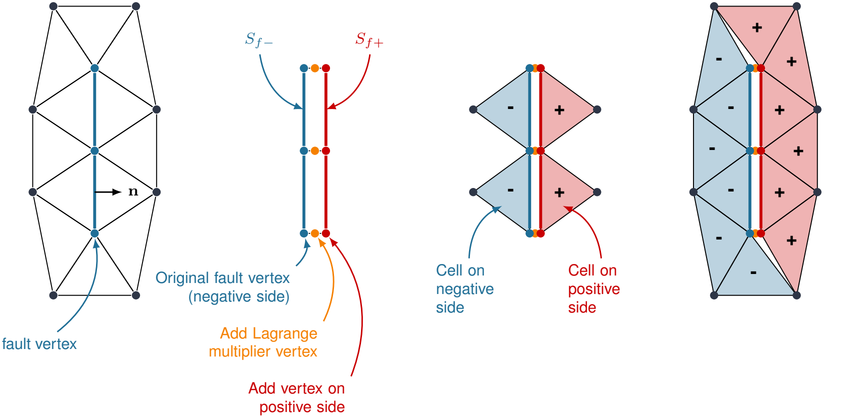
$$\vec{b}(t) = \sum_{\text{surf cells}} \sum_{\text{quad pts}} \overline{N}_m^T \cdot \vec{T} w_q |J_{\text{cell}}| + \sum_{\text{vol cells}} \sum_{\text{quad pts}} \overline{N}_m^T \cdot \vec{f} w_q |J_{\text{cell}}| \tag{13}$$

and solve for $\vec{u}(t)$.

Implementation: Fault Interfaces

Use cohesive cells to control fault behavior

(a) Original mesh **(b) Add colocated vertices** **(c) Update cells with fault faces** **(d) Classify cells and update remaining cells**



Fault Slip Implementation

Use Lagrange multipliers to specify slip

- System without cohesive cells
  - Conventional finite-element elasticity formulation

$$\overline{A}\vec{u} = \vec{b}$$

  - Fault slip associated with relative displacements across fault

$$\overline{C}\vec{u} = \vec{d}$$

- System with cohesive cells

$$\begin{pmatrix} \overline{A} & \overline{C}^T \\ \overline{C} & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ \vec{l} \end{pmatrix} = \begin{pmatrix} \vec{b} \\ \vec{d} \end{pmatrix}$$

- Lagrange multipliers are tractions associated with fault slip
- Prescribed (kinematic) slip
  Specify fault slip ($\vec{d}$) and solve for Lagrange multipliers ($\vec{l}$)
- Spontaneous (dynamic) slip
  Adjust fault slip to be compatible with fault constitutive model

Implementing Fault Slip with Lagrange multipliers

- Advantages
  - Fault implementation is local to cohesive cell
  - Solution includes tractions generating slip (Lagrange multipliers)
  - Retains block structure of matrix, including symmetry
  - Offsets in mesh mimic slip on natural faults
- Disadvantages
  - Cohesive cells require adjusting topology of finite-element mesh

Ingredients for Running PyLith

- Simulation parameters
- Finite-element mesh
  - Mesh exported from LaGriT
  - Mesh exported from CUBIT
  - Mesh constructed by hand (PyLith mesh ASCII format)
- Spatial databases for physical properties, boundary conditions, and rupture parameters
  - SCEC CVM-H or USGS Bay Area Velocity model
  - Simple ASCII files

Spatial Databases

User-specified field/value in space

- Examples
  - Uniform value for Dirichlet (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 Simple ASCII files (0-D, 1-D, 2-D, or 3-D)
  - SCECCVMH SCEC CVM-H seismic velocity model v5.3
  - ZeroDispDB Special case of UniformDB

Features in PyLith 1.7

Enhancements and new features in red

- Time integration schemes and elasticity formulations
  - Implicit for quasistatic problems (neglect inertial terms)
    - Infinitesimal strains
    - Small strains
    - Optional elastic prestep
  - Explicit for dynamic problems
    - Infinitesimal strains
    - Small strains
    - Numerical damping via viscosity
- Bulk constitutive models
  - Elastic model (1-D, 2-D, and 3-D)
  - Linear Maxwell viscoelastic models (2-D and 3-D)
  - Generalized Maxwell viscoelastic models (2-D and 3-D)
  - Power-law viscoelastic model (2-D and 3-D)
  - Drucker-Prager elastoplastic model (2-D and 3-D)

Features in PyLith 1.7 (cont.)

Enhancements and new features in red

- Boundary and interface conditions
  - Time-dependent Dirichlet boundary conditions
  - Time-dependent Neumann (traction) boundary conditions
  - Absorbing boundary conditions
  - Kinematic (prescribed slip) fault interfaces w/multiple ruptures
  - Dynamic (friction) fault interfaces
  - Time-dependent point forces
  - Gravitational body forces
  - Spatial and temporal traction variations for spontaneous rupture
- Fault constitutive models
  - Static friction
  - Linear slip-weakening
  - Linear time-weakening
  - Dieterich-Ruina rate and state friction w/ageing law

Features in PyLith 1.7 (cont.)

Enhancements and new features in red

- Automatic and user-controlled time stepping
- Ability to specify initial stress/strain state
- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Output: VTK and HDF5 files
  - Solution over volume
  - Solution over surface boundary
  - Solution at user-specified locations
  - State variables (e.g., stress and strain) for each material
  - Fault information (e.g., slip and tractions)
- User-friendly interface for generating Green's functions

Features in PyLith 1.7 (cont.)

Enhancements and new features in red

- Automatic conversion of units for all parameters
- Parallel uniform global refinement
- PETSc linear and nonlinear solvers
  - Custom preconditioner with algebraic multigrid solver
  - Ability to use PETSc GPU solvers
- User-specified start time for simulations

PyLith Development

- Long-term priorities (CIG science questions)
  - Multi-cycle earthquake modeling
    - Resolve interseismic, coseismic, and postseismic deformation
    - Elastic/viscoelastic/plastic rheologies
    - Coseismic slip, afterslip, and creep
  - Physics of magmatic systems, geothermal systems, and the cryosphere
  - Models of crustal deformation associated with surface loads
  - Efficient computation of 4-D Green's functions
  - Scaling to 1000 processors
- Short-term priorities
  - Increase user training using virtual workshops
    - CIG/SCEC/NASA/NSF workshop: annual $\rightarrow$ biannual (June 2012)
    - Online training: Building PyLith from source, TBD

- v1.8 (December 2012)
  - Switch to more efficient Sieve implementation
  - Better GPU utilization and additional efficiency improvements
  - Strain hardening/softening for plastic materials
  - Attenuation for dynamic simulations using a generalized Maxwell model
- v2.0+ (2013-2014)
  - Coupling of quasistatic and dynamic simulations
  - Support for incompressible elasticity
  - Heat and fluid flow coupled to elastic deformation
  - Higher order FE basis functions
  - Moment tensor point sources
  - 4-D Green's functions

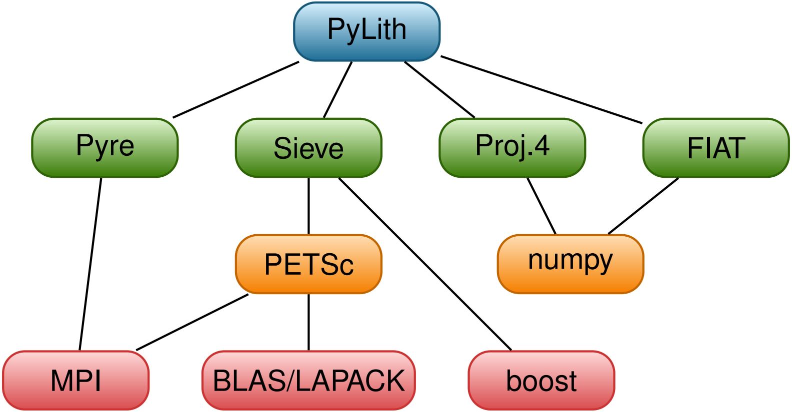
Design Philosophy

Modular, extensible, and smart

- Code should be flexible and modular
- Users should be able to add new features without modifying code, for example:
  - Boundary conditions
  - Bulk constitutive models
  - Fault constitutive models
  - Customized spatial databases
- Input/output should be user-friendly
- Top-level code written in Python (expressive, dynamic typing)
- Low-level code written in C++ (modular, fast)

PyLith Design: Focus on Geodynamics

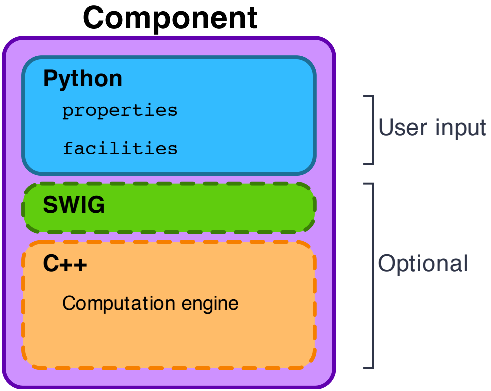
Leverage packages developed by computational scientists



COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

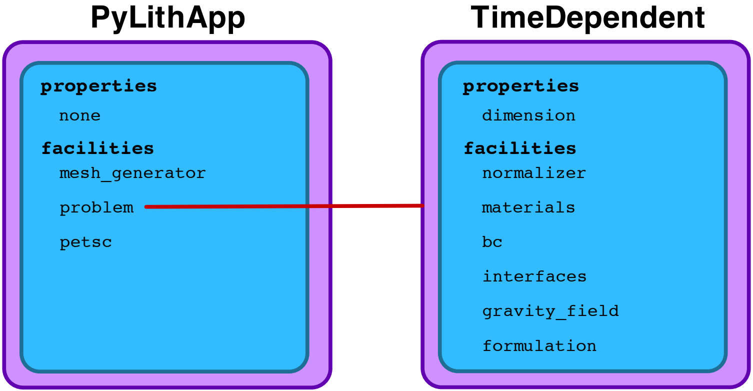
PyLith as a Hierarchy of Components

Components are the basic building blocks



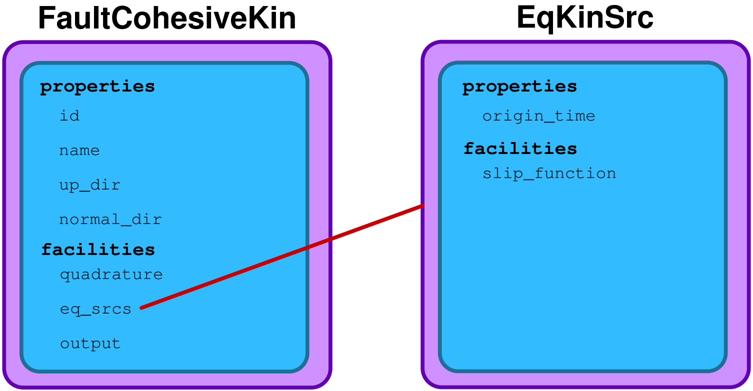
PyLith as a Hierarchy of Components

PyLith Application and Time-Dependent Problem



PyLith as a Hierarchy of Components

Fault with kinematic (prescribed slip) earthquake rupture



PyLith Application Flow

PyLithApp

```
main()
  mesher.create()
  problem.initialize()
  problem.run()
```

TimeDependent (Problem)

```
initialize()
  formulation.initialize()

run()
  while (t < tEnd)
    dt = formulation.dt()
    formulation.prestep(dt)
    formulation.step(dt)
    formulation.poststep(dt)
```

Implicit (Formulation)

```
initialize()

prestep()
  set values of constraints

step()
  compute residual
  solve for disp. incr.

poststep()
   update disp. field
   write output
```

Unit and Regression Testing

Automatically run more than 1800 tests on multiple platforms whenever code is checked into the source repository.

- Create tests for nearly every function in code during development
  - Remove most bugs during initial implementation
  - Isolate and expose bugs at origin
- Create new tests to expose reported bugs
  - Prevent bugs from reoccurring
- Rerun tests whenever code is changed
  - Code continually improves (permits optimization with quality control)
- Binary packages generated automatically upon successful completion of tests
- Additional full-scale tests are run before releases

General Numerical Modeling Tips

Start simple and progressively add complexity and increase resolution

- **Start in 2-D, if possible, and then go to 3-D**
  - Much smaller problems ⇒ much faster turnaround
  - Experiment with meshing, boundary conditions, solvers, etc
  - Keep in mind how physics differs from 3-D
- **Start with coarse resolution and then increase resolution**
  - Much smaller problems ⇒ much faster turnaround
  - Experiment with meshing, boundary conditions, solvers, etc.
  - Increase resolution until solution resolves features of interest
    - Resolution will depend on spatial scales in BC, initial conditions, deformation, and geologic structure
    - Is geometry of domain important? At what resolution?
    - Displacement field is integral of strains/stresses
    - Resolving stresses/strains requires fine resolution simulations
- **Use your intuition and analogous solutions to check your results!**

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?
  - Do all of the nodesets and blocks look correct?
  - Check mesh quality (aspect ratio should be close to 1)
- CUBIT
  - Name objects and use APREPRO or Python for robust scripts
  - Number of points in spline curves/surfaces has huge affect on mesh generation runtime

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 Short-Term Crustal Dynamics mailing list
  `cig-short@geodynamics.org`
- Short-Term Crustal Dynamics wiki
  `http://www.geodynamics.org/cig/community/workinggroups/short/workarea/pylith-wiki`
- CIG bug tracking system
  `http://www.geodynamics.org/roundup`

PyLith Debugging Tools

- `pylithinfo [--verbose] [PyLith args]`
  Dumps all parameters with their current values to text file
- Command line arguments
  - `--help`
  - `--help-components`
  - `--help-properties`
  - `--petsc.start_in_debugger` (run in xterm)
  - `--nodes=N` (to run on N processors on local machine)
- Journal info flags turn on writing progress

  ```
  [pylithapp.journal.info]
  timedependent = 1
  ```

  - Turns on/off info for each type of component independently
  - Examples turn on writing lots of info to stdout using journal flags

Getting Started

- Read the PyLith User Manual
- Work through the examples
  - Chapter 7 of the PyLith manual
  - Input files are provided with the PyLith binary
    `src/pylith/examples`
  - Input files are provided with the PyLith source tarball
    `src/examples`
- Modify an example to look like a problem of interest