Overview of PyLith

What you should have learned by reading the manual, etc. . . . but didn't

Brad Aagaard

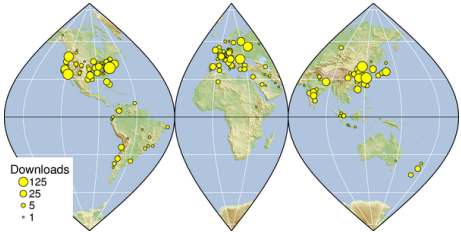
CIG COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

June 26, 2017

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



CIG COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

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

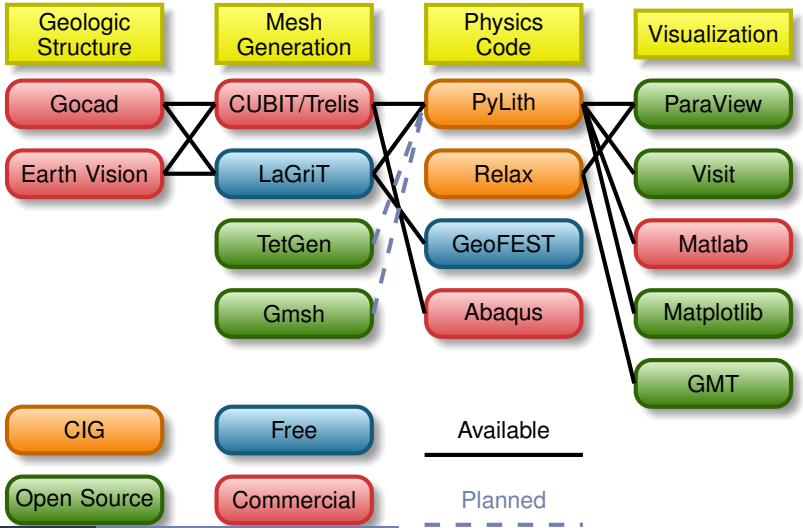
Elasticity problems where geometry does not change significantly

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



Governing Equations

Elasticity equation

$$\sigma_{ij,j} + f_i = \rho \ddot{u} \text{ in } V, \tag{1}$$
$$\sigma_{ij} n_j = T_i \text{ on } S_T, \tag{2}$$
$$u_i = u_i^0 \text{ on } S_u, \text{ and} \tag{3}$$
$$R_{ki}(u_i^+ - u_i^-) = d_k \text{ on } S_f. \tag{4}$$

Multiply by weighting function and integrate over the volume,

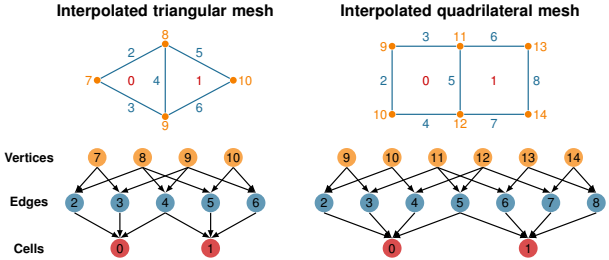
$$-\int_V (\sigma_{ij,j} + f_i - \rho \ddot{u}_i)\phi_i \, dV = 0 \tag{5}$$

After some algebra,

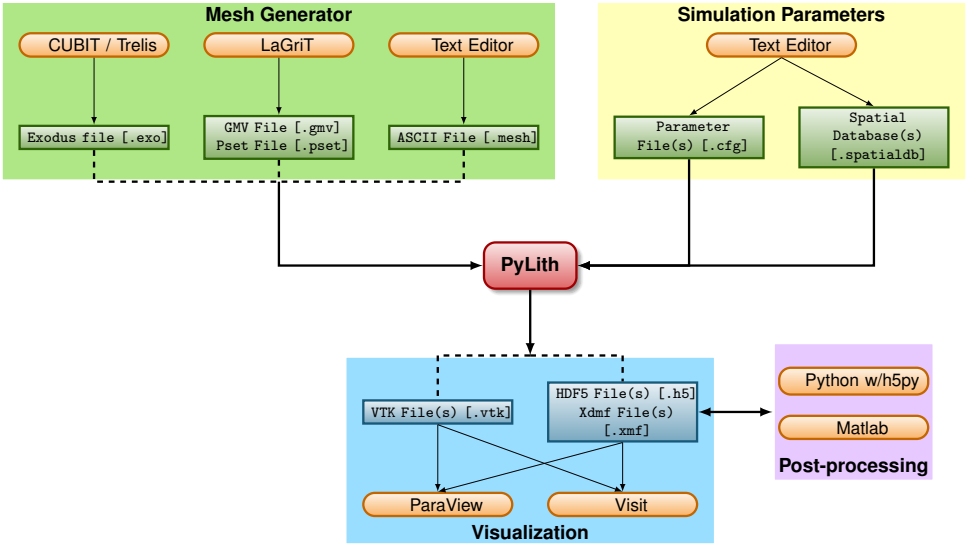
$$-\int_V \sigma_{ij}\phi_{i,j} \, dV + \int_{S_T} T_i \phi_i \, dS + \int_V f_i \phi_i \, dV - \int_V \rho \ddot{u}_i \phi_i \, dV = 0 \tag{6}$$

Discretize Domain Using Finite Elements

PyLith v2.0.0 and later use interpolated meshes



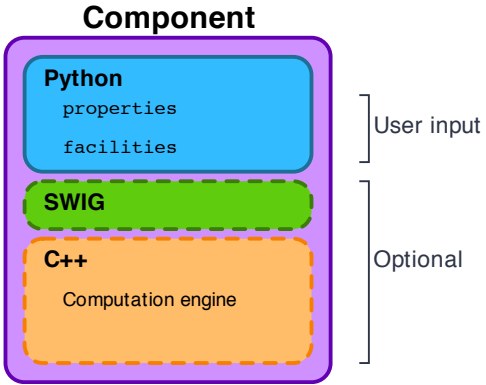
Overview of PyLith Workflow



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.



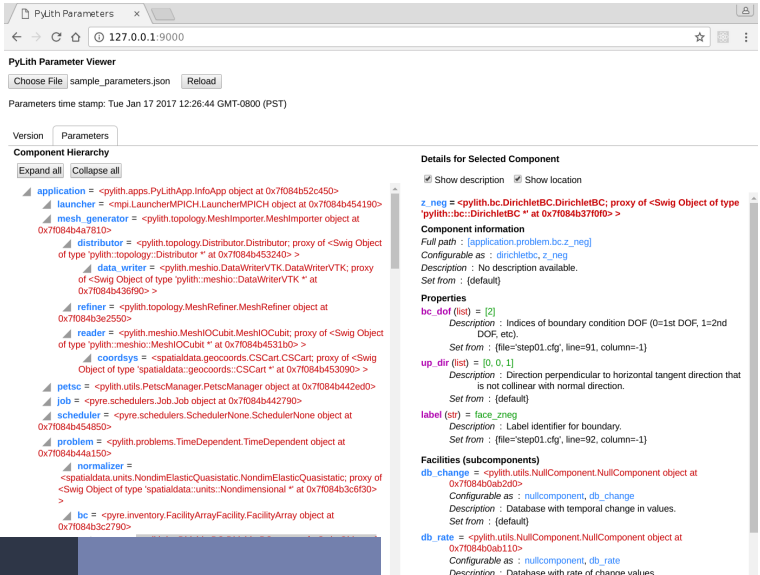
Parameter Files

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

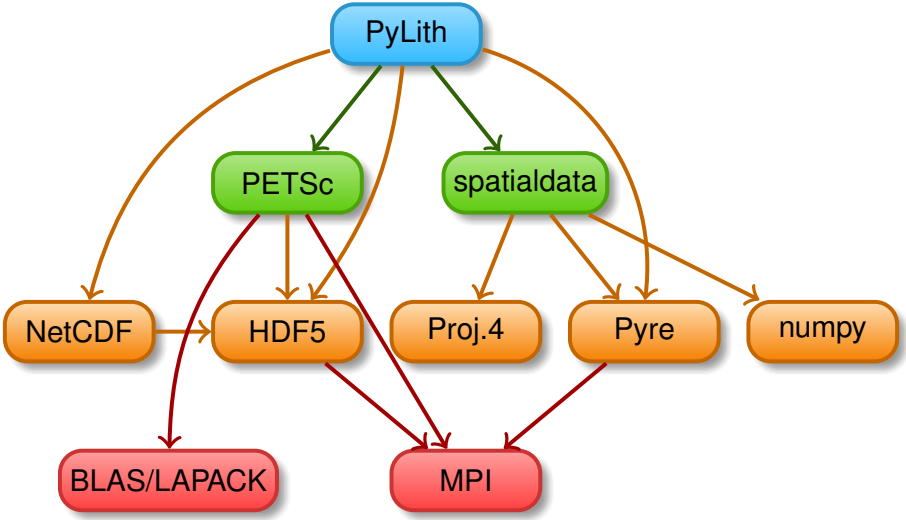
User-specified field/value in space for properties and BC values.

- 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 & Jenkins	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.

full-scale tests Serial and parallel pass/fail tests of full problems using Method of Manufactured Solutions.

benchmarks Serial and parallel tests for code comparisons, etc.

PyLith Current Release: v2.2.1 (Jun 25, 2017)

Same feature as v2.2.0, just new examples and bugfixes

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

Features in PyLith v2.2 (cont.)

- 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
  - Fault interfaces with T intersections
  - Time-dependent point forces
  - Gravitational body forces
- Fault constitutive models
  - Static friction
  - Linear slip-weakening
  - Linear time-weakening
  - Dieterich-Ruina rate and state friction w/ageing law

Features in PyLith v2.2 (cont.)

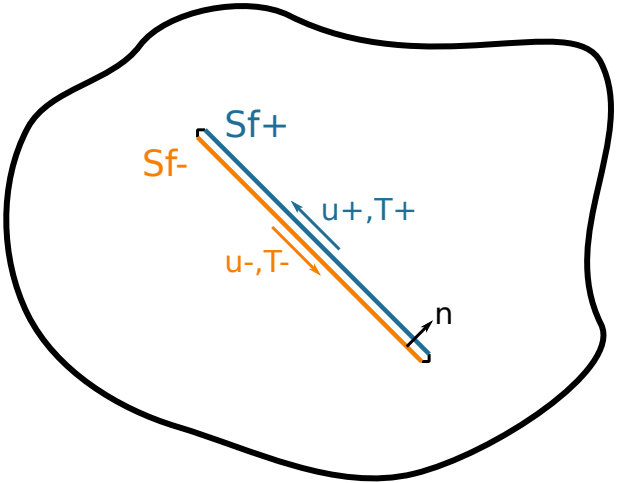
- Automatic and user-controlled time stepping
- Ability to specify initial stress/strain state
- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT/Trelis: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Output: VTK and HDF5 files
  - Solution over volume
  - Solution over surface boundary
  - Solution interpolated to user-specified points w/station names
  - State variables (e.g., stress and strain) for each material
  - Fault information (e.g., slip and tractions)

Features in PyLith v2.2 (cont.)

- Automatic conversion of units for all parameters
- Parallel uniform global refinement
- PETSc linear and nonlinear solvers
  - Custom preconditioner with algebraic multigrid solver
- Output of simulation progress estimates runtime

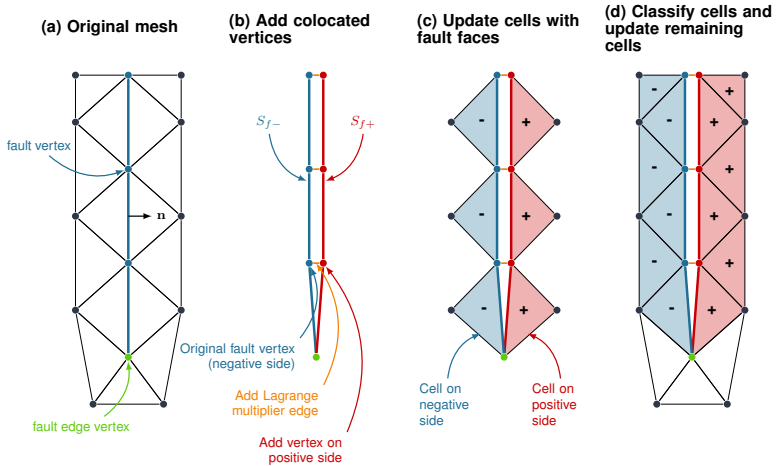
Fault Interface

Fault tractions couple deformation across interface



Implementation: Fault Interfaces

Use zero volume/area cohesive cells to control fault behavior



See Aagaard, Knepley, and Williams, JGR, 2013 doi: 10.1002/jgrb.50217 for more information

Fault Implementation: Governing Equations

Terms in governing equation associated with fault

- Tractions on fault surface are analogous to boundary tractions

$$\ldots + \underbrace{\int_{S_T} \vec{\phi} \cdot \vec{T}\, dS}_{\text{Neumann BC}} \underbrace{- \int_{S_{f+}} \vec{\phi} \cdot \vec{l}\, dS}_{\text{Fault +}} \underbrace{+ \int_{S_{f-}} \vec{\phi} \cdot \vec{l}\, dS}_{\text{Fault -}} \ldots = 0$$

- Constraint equation relates slip to relative displacement

$$\int_{S_f} \vec{\phi} \cdot (\underbrace{\vec{d}}_{\text{Slip}} - \underbrace{(\vec{u}_+ - \vec{u}_-)}_{\text{Relative Disp.}}) dS = 0$$

Fault Slip Implementation

Use Lagrange multipliers to specify slip

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

$$\underline{\mathsf{A}}\vec{u} = \vec{b}$$

  - Fault slip associated with relative displacements across fault

$$\underline{\mathsf{C}}\vec{u} = \vec{d}$$

- System with Lagrange multiplier constraints for fault slip

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

- 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
  - Scalable preconditioner/solver is more complex

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

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 $\Rightarrow$ much faster turnaround
  - Start with an exact solver
  - 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 $\Rightarrow$ much faster turnaround
  - Start with an exact solver
  - 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!**

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
- PyLith User Resources
  http://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