PySIT: Seismic Imaging Toolbox for Python

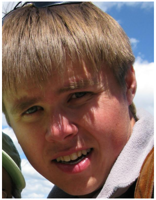
Russell J. Hewett and Laurent Demanet

EAGE 2016, Vienna

Who we are

PySIT
Seismic Imaging Toolbox for Python

- Earth Resources Laboratory (ERL): `https://erlweb.mit.edu`
- Imaging and Computing group: `http://math.mit.edu/icg`
- Main developer: Russell Hewett (Total)



Needed flexible platform for research on...

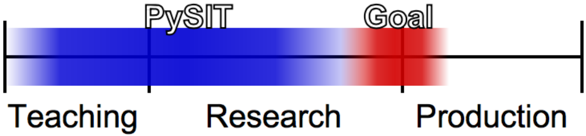
- Wave Equation Solvers
- Objective Functions
- Optimization

...in the context of FWI

Why Python?

- Cross-platform, portability
- Community
- Not MATLAB ($$)
- Open parallel libraries
- General fan of the language

Pedagogical and research platform for seismic inversion



Rapid prototyping new imaging algorithms

- Easy to learn
- Maximize student and intern productivity
- Reproducible research

http://www.pysit.org

Open source under BSD license

- Permissive license
- No legal concerns for industrial use

Open development model

- Version control with Git
- Source code hosted on GitHub
- Explicitly provide mechanisms for handling proprietary research

NumPy [www.numpy.org]

- Core N-D array package

SciPy [www.scipy.org]

- Fundamental library for scientific computing

Matplotlib [www.matplotlib.org]

- Plotting and visualization

ObsPy [www.obspy.org]

- Python framework for seismology
- Data reading, writing, and processing

mpi4py [mpi4py.scipy.org]

- MPI wrapper for Python

Local solvers for salt boundary inversion in FWI

- Willemsen et al. (GJI '16)

Microseismic Event Estimation via FWI

- Minkoff et al. (SEG '15)

Hessian approximations

- Stochastic approximation of FWI Hessian (Willemsen et al.)
- Matrix probing of FWI Inverse Hessian (Hewett et al.)

Uncertainty quantification

- Non-gaussianity of the FWI posterior (Li et al.)

Exercise: 1D full waveform inversion

- Introduces FWI to interns, graduate students, and postdocs
- Achieves a functional FWI in a few hours
- Motivates and teaches both design and structure of PySIT
- Is part of PySIT's documentation

PySIT is also a Pedagogical Tool

Exercise: 1D full waveform inversion

- Introduces FWI to interns, graduate students, and postdocs
- Achieves a functional FWI in a few hours
- Motivates and teaches both design and structure of PySIT
- Is part of PySIT's documentation

Case study: MSRI – Mathematics of Seismic Imaging

- Two weeks and ~ 40 math graduate students
- No imaging experience, moderate programming experience
- Start with 1D exercise . . .
- . . . end with blind 2D inversion problem using PySIT

PySIT is also a Pedagogical Tool

Exercise: 1D full waveform inversion

- Introduces FWI to interns, graduate students, and postdocs
- Achieves a functional FWI in a few hours
- Motivates and teaches both design and structure of PySIT
- Is part of PySIT's documentation

Case study: MSRI – Mathematics of Seismic Imaging

- Two weeks and ~ 40 math graduate students
- No imaging experience, moderate programming experience
- Start with 1D exercise ...
- ...end with blind 2D inversion problem using PySIT on their laptops

Partial List of Features

Dimension independent inversion: 1D, 2D, and 3D

Solvers

- Scalar acoustic wave equation
  - Matrix (numpy) and matrix-free (C++) implementations
  - Leap frog and ODE timestepping
  - Arbitrary spatial accuracy
- + Variable density solvers
- Helmholtz equation
  - Sparse, direct LU with SuperLU + PETSc wrappers

Objective Functions

- Temporal least-squares
- Frequency least-squares

Optimization Algorithms

- Gradient descent, Gauss-Newton, L-BFGS

Visualization

Parallelism (MPI + OpenMP), PMLs, gallery problems, and more

Let the Mathematics Define the API

```
modeling = TemporalModeling(solver)
```

$$u = \mathcal{F}(m) \quad \Leftrightarrow \quad L(m)u = f$$

```
modeling.forward_model(shot, m, ...)
```

$$\delta u = F_{m_0}\delta m \quad \Leftrightarrow \quad L(m_0) = -\frac{\delta L}{\delta m}[\delta m]u_0$$

```
modeling.linear_forward_model(shot, m0, dm, ...)
```

$$\delta m = F^*_{m_0} S^* r \quad \Leftrightarrow \quad \begin{array}{c} \delta m = - < q, \frac{\delta L}{\delta m} u_0 > \\ \text{s.t.} \quad L^*(m_0)q = S^* r \end{array}$$

```
modeling.adjoint_model(shot, m0, r, ...)
```

Let the Mathematics Define the API

```
objective = TemporalLeastSquares(solver)
```

$$J(m) \quad = \quad \frac{1}{2}||d - \mathcal{F}(m)||_2^2$$

```
objective.evaluate(shots, m, ...)
```

$$\nabla J(m_0) \quad = \quad -F_{m_0}^*(d - \mathcal{F}(m_0))$$

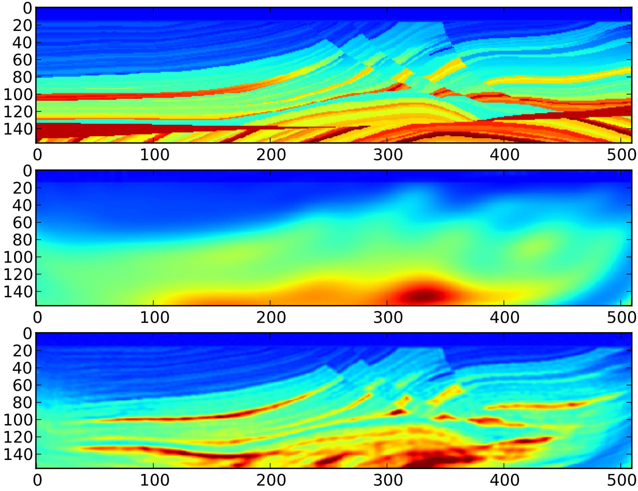
```
objective.compute_gradient(shots, m0, ...)
```

$$D^2 J \delta m \quad = \quad F_{m_0}^* F_{m_0} \delta m - < D^2 \mathcal{F} \delta m, d - \mathcal{F}(m_0) >$$

```
objective.apply_hessian(shots, m0, dm, ...)
```

```
from pysit import *


# Setup a physical problem and acquisition
true_model, initial_model, mesh, domain = marmousi2()

wavelet = RickerWavelet(10.0)
shots = equispaced_acquisition(mesh, wavelet, nshots=5, nrecs=50)

# Setup solver
solver = ConstantDensityAcousticWave(mesh, trange=(0.0, 3.0))

# Populate synthetic data
generate_seismic_data(shots, solver, true_model)

# Define objective function
objective = TemporalLeastSquares(solver)

# Define optimization algorithm
invalg = LBFGS(objective)

# Run inversion for 5 iterations
result = invalg(shots, initial_value, 5)

# Visualize results
vis.plot(result.C, mesh)
```

FWI Results with PySIT



What does the future hold?

- Optimization toward a simpler, more powerful, dependable tool:
  - New features: UQ, hDG, maybe elastic / anisotropic, maybe AD
  - Speed up: encapsulate more C++, maybe switch to Julia
  - Bridge gap between academic research and industrial application
  - Better installer
- Library dependencies can be painful: how to minimize the need for TLC?

http://www.pysit.org