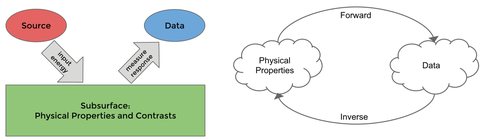
SimPEG: Simulation and Parameter Estimation in Geophysics

Rowan Cockett[1], Lindsey Heagy[1], Seogi Kang[1], Doug Oldenburg[1], Adam Pidlisecky[2], Guðni Rosenkjær[1], Brendan Smithyman[3]

[1]University of British Columbia, [2]University of Calgary, [3]Western University

Why?!

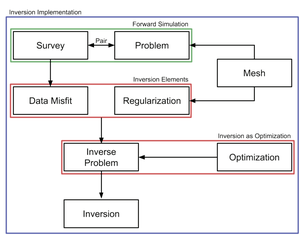
In geophysics, we use inverse theory to mathematically create models of the earth from measured data. It has many moving pieces: physics, discretization, simulation, regularization, optimization, computer science, linear algebra, geology. As geophysicists we care about the combination: how to pull these disciplines together to answer our questions. We are creating a toolbox that allows you to work at a high level and keep your geophysical question in focus.

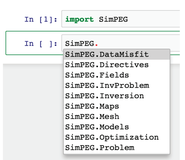


Currently data-sets from the same field area (geology, seismic, electromagnetics, hydrogeology), are often processed separately, producing dissonant results. How do we integrate these geoscience fields? No more black-box software! These do not promote experimentation and investigation. SimPEG is a framework and a mix of tools that span geophysics so that we can communicate effectively and experiment efficiently. A toolbox combined with a framework that allows you to solve your own problems, and creates opportunities for us to work together to better image and understand the subsurface.

Framework

The framework we have developed serves as a blueprint that organizes concepts from inversion methodology and provides a consistent implementation across a variety of geophysical problems. The framework is modular and allows for both software and ideas to be easily shared between researchers and across geophysical problems.






```
reg     = Regularization.Tikhonov(M)
dmis    = DataMisfit.l2_DataMisfit(survey)
opt     = Optimization.InexactGaussNewton(maxIter=8)
invProb = InvProblem.BaseInvProblem(dmis, reg, opt)
beta    = Directives.BetaSchedule()
betaest = Directives.BetaEstimate_ByEig(beta0_ratio=0.75)
inv     = Inversion.BaseInversion(invProb, directiveList=[beta, betaest])

mopt = inv.run(m0)
```

Applications

Package	State	Description	Status
SimPEG	✔	A framework for simulation and gradient based parameter estimation in geophysics.	build passing coverage 83%
simpegEM	🔄	A electromagnetic forward modeling and inversion package for SimPEG.	build passing coverage 88%
simpegMT	🔄	Magnetotellurics forward and inverse codes for SimPEG	build passing coverage 33%
simpegFLOW	🧪	Groundwater (vadose zone) flow equations written in the SimPEG framework.	build passing coverage 91%
simpegDC	🧪	A DC resistivity forward modelling and inversion package for SimPEG.	build passing coverage 91%
simpegPF	🧪	Potential fields codes for SimPEG. Gravity and Magnetics.	build passing coverage 71%
simpegSEIS	🔧	Time and frequency domain forward modeling and inversion of seismic wave.	
simpegGPR	🔧	Forward modelling and inversion of Ground-Penetrating Radar (GPR).	

Electromagnetics

```
import simpegEM as EM
mesh    = Mesh.TensorMesh([40,30,50])
prob    = EM.FDEM.ProblemFDEM_h(mesh, mapping=mapping)
rec     = EM.FDEM.RxFDEM(rxlocs, 'bzr')
src     = EM.FDEM.SrcFDEM_MagDipole([rec], freq=freq, loc=txloc)
survey  = EM.FDEM.SurveyFDEM([src])
prob.pair(survey)
```

EM (f,ω) MT (ω) DC (s)

Governing Equations

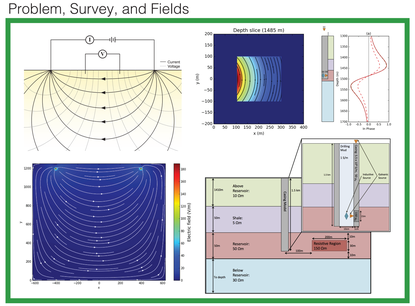
$$\nabla \times \vec{E} + i\omega\vec{B} = 0$$

$$\nabla \times \mu^{-1}\vec{B} - \sigma\vec{E} = \vec{s}$$

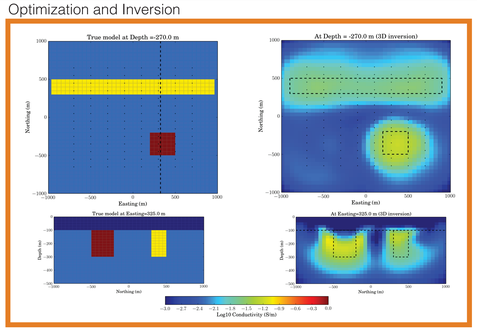
Physical Properties

σ - electrical conductivity
μ - magnetic permeability

Problem, Survey, and Fields



Optimization and Inversion



We use coincident loop Tx-Rx geometry with dbz/dt component measured at 112 stations with 24 time channels ranging from 0.01-2 ms. Plan views of the true conductivity model and the recovered model, as well as section views of the true and the recovered conductivity models. Core cell size: 50×50×20 m, The number of cell: 50×50×48 = 120,000; Reference model: Half-space model with conductivity value, 0.005 S/m; Inexact Gauss-Newton: 13 iterations; CPU Time: 48hrs; Maximum memory usage: 51.2GB; CPU Info: Intel Xeon 2.80 GHz; Memory info: 64 GB.

Fluid Flow

```
from simpegFLOW import Richards
mesh    = Mesh.TensorMesh([40,30,50])
E       = Richards.Empirical.Haverkamp(mesh, **params)
prob    = Richards.RichardsProblem(mesh, mapping=E, timeSteps=[(40,3),(60,3)], **myOptions)
recs    = Richards.RichardsRx(locs, times, 'saturation')
survey  = Richards.RichardsSurvey([recs])
prob.pair(survey)
```

VD (t)

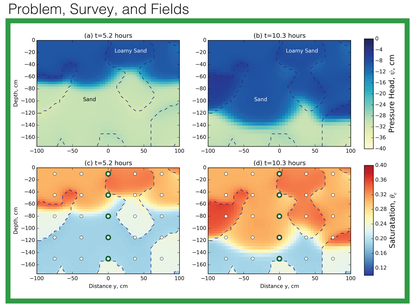
Governing Equations

$$\frac{\partial\theta(\psi)}{\partial t} - \nabla\cdot k(\psi)\nabla\psi - \frac{\partial k(\psi)}{\partial z} = 0$$

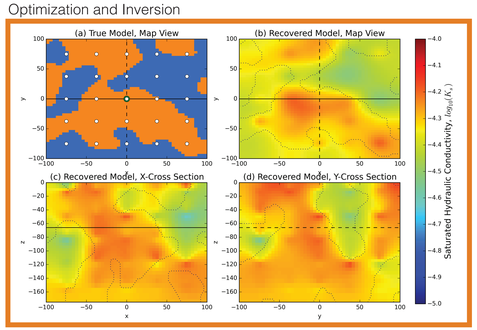
Physical Properties

k - hydraulic conductivity
ϕ - porosity
.. - van Genuchten parameters?

Problem, Survey, and Fields



Optimization and Inversion



Model dimensions of 2.0m×2.0m×2.6m for the x, y and z dimensions, with 1.125e5 cells in space and 40 time steps. In this simulation, the dense Jacobian matrix would have 562.5 million elements. The Jacobian is neither computed nor stored directly. For these experiments we used a single Linux Debian Node on Google Compute Engine (Intel Sandy Bridge, 16 vCPU, 14.4 GB memory) to run the simulations and inversion. The forward problem takes approximately 40 minutes to solve, and the inverse problem took approximately 34.5 hours.

Seismic

```
import simpegSEIS as SEIS
mesh = Mesh.TensorMesh([40,30])
prob = SEIS.ProblemSeis2D(mesh, mapping=mapping, **myOptions)
rec  = SEIS.HelmRx(locs)
src  = SEIS.HelmSrc(rec, freq=100., loc=srcLocs)
survey = SEIS.SurveySeis2D([src])
prob.pair(survey)
```

SM (ω)

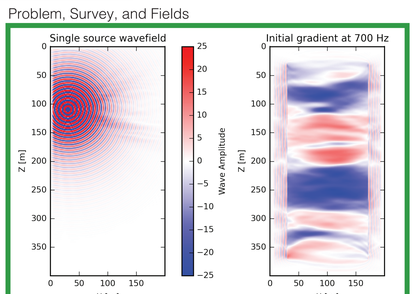
Governing Equations

$$\nabla\cdot\rho\nabla u + \omega^2\kappa u = q$$

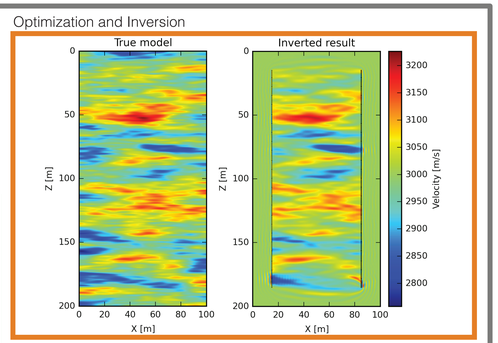
Physical Properties

u - velocity
ρ - density

Problem, Survey, and Fields



Optimization and Inversion

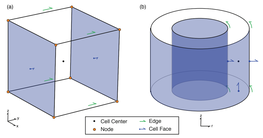


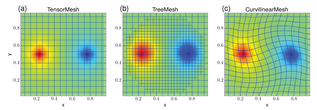
Seismic full waveform inversion in two dimensions from 700–1400 Hz, cross-hole case with 200x400 grid and 0.5 m cell sizes, 171 sources at x=15m, and 171 receivers at x=85m. Inversion starting from a constant model (3000m/s, green). Noise free data with the same kernel (i.e. inverse crime!). Solver is LBFGS (scipy.optimize.fmin_l_bfgs_b), with 10 iterations and no regularization. This uses a parallel implementation on 8 cores w/ MPI on a separate machine through an OpenVPN connection in a different city, with the frontend visualization in an IPython notebook on a laptop.

Finite Volume

We have implemented a staggered mimetic finite volume approach. This approach requires the definitions of variables at either cell-centers, nodes, faces, or edges as described. The mesh implementation has been extended to TensorMesh, CylindricalMesh, QuadTree, OcTree, and CruvilinearMesh. Differential operators, and common plotting procedures are parameters and methods of each mesh type.

```
hx = [3,2,1,1,1,1,1,2,3]
hy = [3,1,1,3]
M = Mesh.TensorMesh([hx, hy])
M.plotGrid(faces=True, nodes=True, centers=True)
```






```
D = M.faceDiv
G = M.cellGrad
# Harmonically average sigma
MsigI = sdInv(sdiag(M.aveF2CC.T*(1/sig)))
A = D*MsigI*G
A[0,0] = 1/M.vol[0] # Remove the null space
Ainv = Solver(A) # Create a default Solver
phi = Ainv * ( - q )
```

Terminology

We have used the terminology of geophysics and inverse theory to drive our decisions in framework development, the layout of modules, and user interactions in SimPEG. As a result of this investment in framework design, we have experienced increased consistency between the discussion of theory and its implementation in Python, which has created a platform for rapid curation of ideas.

What is your model?!

The model, m, and the physical properties are connected through a mapping,and associated derivatives.

```
M = Mesh.TensorMesh([7,51])
vlMap = Maps.Vertical1DMap(M)
expMap = Maps.ExpMap(M)
myMap = expMap * vlMap
m = np.r_[0.1,0.1,2,2.5] # only 5 model parameters!
sig = myMap * m
```

Conductivity $m = \sigma$

Log Conductivity $m = \log(\sigma)$

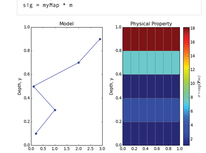
Empirical Relation $m = \sigma_f\,\phi^a\,S_w^b$

Mapping →

Physics

$$\nabla\times\mathbf{E} + i\omega\mathbf{B} = 0$$

$$\nabla\times\mu^{-1}\mathbf{B} - \sigma\mathbf{E} = \mathbf{J}_s$$

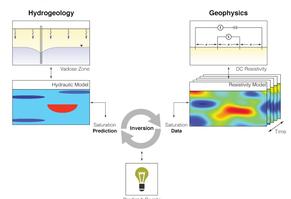



```
class MyPropMap(Maps.PropMap):
    sigma = Maps.Property("Electrical Conductivity", defaultInvProp=True)
    mu    = Maps.Property("Mu", defaultVal=mu_0)

class MyReciprocalPropMap(Maps.PropMap):
    sigma  = Maps.Property("Electrical Conductivity", defaultInvProp=True, propertyLink=('rho',   Maps.ReciprocalMap))
    rho    = Maps.Property("Electrical Resistivity",                       propertyLink=('sigma', Maps.ReciprocalMap))
    mu     = Maps.Property("Mu", defaultVal=mu_0,    propertyLink=('mui',   Maps.ReciprocalMap))
    mui    = Maps.Property("Mu", defaultVal=1.)mu_0, propertyLink=('mu',   Maps.ReciprocalMap))
```

Integrating Information

SimPEG is formulated with the inverse problem as its core focus. We ensure that sensitivities are efficient to compute and are readily available; we presume this will be advantageous for integrated geophysical inversions. Stay tuned... :)



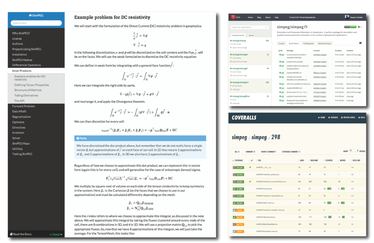
Getting Started

Check out the website and docs:

```
http://simpeg.xyz
```

Install the package:

```
> pip install SimPEG
```

Clone the repos:

```
https://github.com/simpeg
```