Subsurface characterization for large scale systems: A Python-based inversion tool for TOUGH2

Amalia Kokkinaki, Jonghyun Harry Lee, Hojat Ghorbanidehno
Eric Darve, Peter Kitanidis

TOUGH2 Symposium 2018

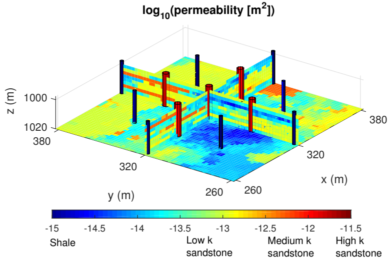
10/9/2018

Outline

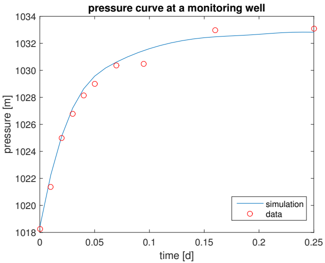
- Introduction
- Geostatistical inverse modeling
- pyPCGA-TOUGH: structure and advantages
- Test case results
- Conclusions

Introduction

Geostatistical Inverse Modeling: a Bayesian framework to estimate grid-scale, spatially distributed model parameters using indirect observations



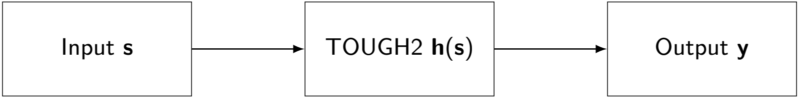
Unknown parameter



Local, noisy observations

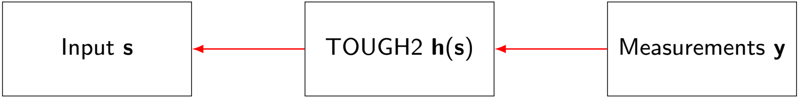
- Subsurface characterization (e.g. hydraulic tomography)
- Real-time estimation (e.g. contaminant tracking)
- Optimization (e.g. pumping schedule estimation)

Introduction: Inverse Problem



- In the **forward problem**, given model parameters, **s**, TOUGH2 predicts the state of the system **y**
- **s** is typically permeability, but could be other rock properties, or boundary conditions
- **y** are the primary variables of the EOS module

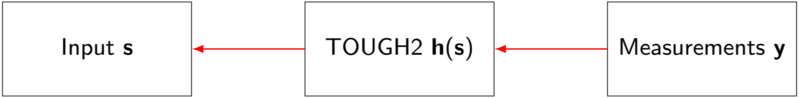
Introduction: Inverse Problem



- In the **forward problem**, given model parameters, **s**, TOUGH2 predicts the state of the system **y**
- **s** is typically permeability, but could be other rock properties, or boundary conditions
- **y** are the primary variables of the EOS module

- In the **inverse** problem, we use measurements of **y** to estimate **s**

Introduction: Inverse Problem



- In the **forward problem**, given model parameters, **s**, TOUGH2 predicts the state of the system **y**
- **s** is typically permeability, but could be other rock properties, or boundary conditions
- **y** are the primary variables of the EOS module

- In the **inverse** problem, we use measurements of **y** to estimate **s**
- TOUGH2 is now used to calculate the sensitivity of measurements to parameters $\mathbf{H} = \frac{\partial \mathbf{y}}{\partial \mathbf{s}}$

Inverse Problem in Hierarchical Bayesian Framework

Consider the measurement equation

$$y = h(s) + v \qquad v \sim \mathcal{N}(0, \mathbf{R})$$

$\mathbf{y} := n_{obs} \times 1$ noisy measurements	pressure, temperature
$h :=$ forward model	TOUGH2
$v :=$ measurement and model error	uncertainty and error
$\mathbf{s} := n_{unknowns} \times 1$	permeability

$$s \sim \mathcal{N}(s_{prior}, \mathbf{Q}_{prior})$$

- Parameters are treated as random variables in a statistical framework (e.g., Gelman, Calin, and Stern, 2013; Kitanidis, 2010, Kitanidis, 1995)
- Use covariances **Q** and **R** to represent variability and uncertainty
- Objective: **A best estimate of unknowns and corresponding uncertainty at each grid cell of the TOUGH2 model, given a set of measurements**

Inverse Problem in Hierarchical Bayesian Framework

Consider the measurement equation

$$y = h(s) + v \qquad v \sim \mathcal{N}(0, \mathbf{R})$$

Using Bayes' rule, the posterior pdf is

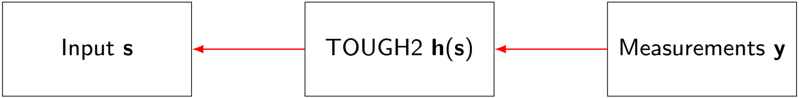
$$p(s|y) \quad \propto \quad \underbrace{p(y|s)}_{\text{Data misfit}} \underbrace{p(s)}_{\text{Prior}}$$

- Data misfit - How well the model reproduces data
- Prior - Prior knowledge of unknown field structure

Best estimate is obtained by maximizing the likelihood of **s** given a set of measurements **y**, using GN optimization:

$$p(\mathbf{s}) \sim \exp\left(\underbrace{-\frac{1}{2}(\mathbf{y} - \mathbf{h}(\mathbf{s}))^\top \mathbf{R}^{-1}(\mathbf{y} - \mathbf{h}(\mathbf{s}))}_{likelihood} \underbrace{-\frac{1}{2}(\mathbf{s} - \mathbf{s}_{prior})^\top \mathbf{Q}_{\text{prior}}^{-1}(\mathbf{s} - \mathbf{s}_{prior})}_{prior} \right)$$

Inverse Problem: the challenges for large systems



For large-scale systems:

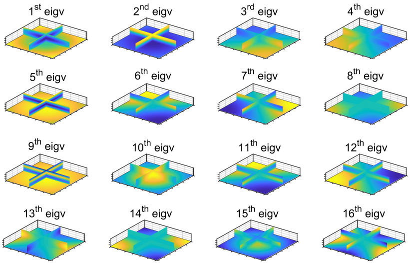
- Typically many unknowns, few measurements $n_{obs} \ll n_{unknowns}$
- Requires $\mathcal{O}(min(n_{obs}, n_{unknowns}))$ TOUGH2 runs or more
- High dimensional matrix operations

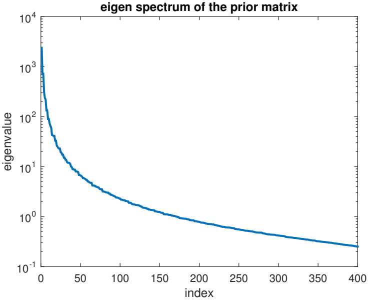
Therefore:

- Fast Linear Algebra is necessary to enable computations and storage
  - Matrix-matrix, matrix-vector multiplications
- Reduce number of forward runs
- Finetuning and evaluation of the inversion design and parameters is critical

pyPCGA: Geostatistical inversion in Python

Principal **C**omponent **G**eostatistical **A**pproach:
A computationally efficient algorithm for geostatistical inversion based on compression of covariance matrices and Jacobian-free evaluation of sensitivity.[1]





[1] Lee and Kitanidis, 2014, Kitanidis and Lee, 2014

pyPCGA: Geostatistical inversion in Python

Compression of the covariance matrix reduces the number of matrix-vector multiplications to $\mathcal{O}(n_{pc})$:

$$\mathbf{Q}_{prior} \approx \mathbf{U}_{n_{pc}} \mathbf{\Sigma}_{n_{pc}} \mathbf{U}_{n_{pc}}^T$$

Calculation of sensitivity matrix requires TOUGH2 runs, black-box style, using the finite difference approach:

$$\mathbf{Hs} = \frac{h(\mathbf{s} + \Delta s) - h(\mathbf{s})}{\Delta s}$$

Computations involving large matrices (**Q**, **H**) utilize fast linear algebra that allows **fully parallelizable, fast matrix-vector multiplications**:

- Fast Fourier Transform (FFT) approach for regular grids[2]
- Fast Multipole Method (FMM) and Hierachical Matrices Approach for unstructured grids[3]

---

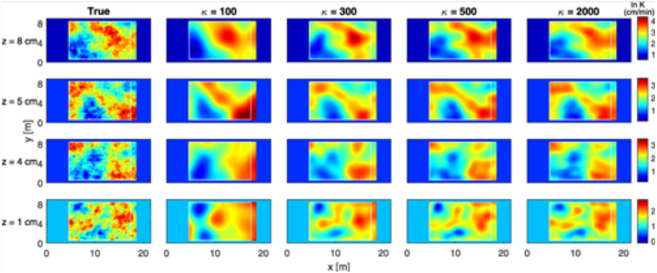
[2] https://github.com/arvindks/kle

[3] https://github.com/ruoxi-wang/PBBFMM3D

pyPCGA: Geostatistical inversion in Python

Computational gain:

- Matrix computations scale linearly with number of unknowns
- $\sim \mathcal{O}(100)$ forward model runs for large domains ($\sim 10^6$ unknowns)
- Parallelization further accelerates inversion



- **Linear scaling makes possible the inversion of domains with millions of unknowns and observations[4].**

[4] Lee et al., 2016

An inversion package for TOUGH2 users

- pyPCGA-TOUGH: An open-source package for geostatistical inversion
  - Compatible with PyTOUGH[5], extension of PyTOUGH for TOUGH2-MP
  - Tutorial-like templates with visualizations, to get started
  - Tool for designing monitoring, and assessing information content of data

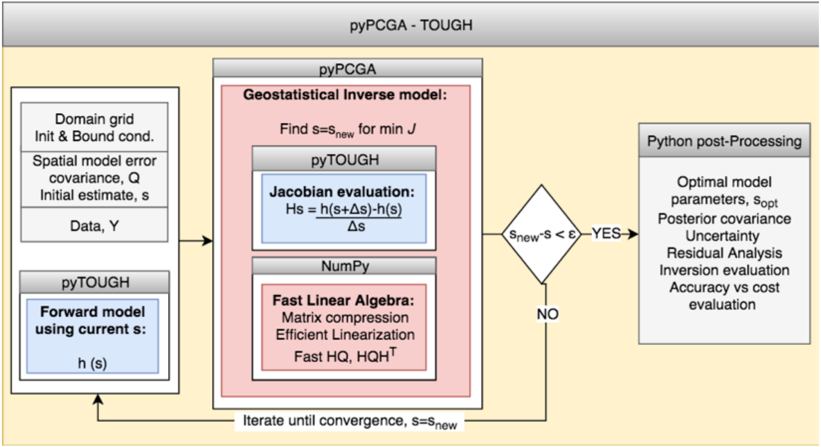
[5] https://github.com/acroucher/PyTOUGH

An inversion package for TOUGH2 users

- pyPCGA-TOUGH: An open-source package for geostatistical inversion
  - Compatible with PyTOUGH[5], extension of PyTOUGH for TOUGH2-MP
  - Tutorial-like templates with visualizations, to get started
  - Tool for designing monitoring, and assessing information content of data
- Framework for powerful statistical estimation for TOUGH2 models
  - Connects with packages for fast linear algebra tools
  - Can be extended with other types of inversion
  - Allows reproducibility and method comparison

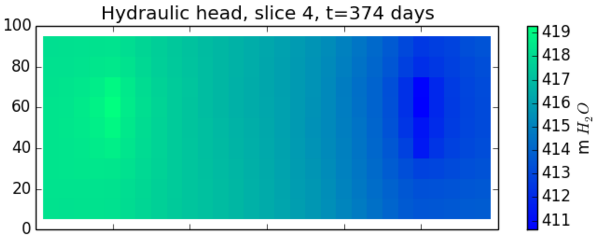
[5] https://github.com/acroucher/PyTOUGH

pyPCGA-TOUGH: Geostatistical inversion for TOUGH2



pyPCGA-TOUGH: Getting started

0. Set up the forward and inverse problem
   - Is the problem physically (numerically) feasible?
   - Are measurements sensitive to parameters?
   - What is a reasonable first guess for the unknowns?
   - Implementation for consistent "obs" for runs with different parameters?



pyPCGA-TOUGH: Getting started

pyPCGA: Python Interface of PCGA algorithm

```
params = {'R': (0.5) ** 2, 'n_pc': 50,
          'maxiter': 10, 'restol': 0.01,
          'matvec': 'FFT', 'xmin': xmin, 'xmax': xmax, 'N': N,
          'prior_std': prior_std, 'prior_cov_scale': prior_cov_scale,
          'kernel': kernel, 'post_cov': "diag",
          'precond': True, 'LM': True,
          'parallel': True, 'linesearch': True,
          'forward_model_verbose': False, 'verbose': False,
          'iter_save': True}
```

pyPCGA-TOUGH: Getting started

pyPCGA: Python Interface of PCGA algorithm

```
params = {'R': (0.5) ** 2, 'n_pc': 50,
          'maxiter': 10, 'restol': 0.01,
          'matvec': 'FFT', 'xmin': xmin, 'xmax': xmax, 'N': N,
          'prior_std': prior_std, 'prior_cov_scale': prior_cov_scale,
          'kernel': kernel, 'post_cov': "diag",
          'precond': True, 'LM': True,
          'parallel': True, 'linesearch': True,
          'forward_model_verbose': False, 'verbose': False,
          'iter_save': True}
```

Integrated with pyTOUGH

```
# TOUGH2 Simulation parameters:
# Table 4.9 page 78 pytough tutorial and Appendix E of TOUGH2 tutorial
# data.parameter is a dictionary
# each parameter can be called as dat.parameter['parameter name']
dat.parameter.update(
    {'max_timesteps': 9000,                # maximum number of time steps
     'tstop': 0.32342126E+08,              # stop time
     'const_timestep': 6,                  # time step length
     'max_timestep':86400,                 # maximum time step size
     'absolute_error': 1,                  # absolute convergence tolerance
     'relative_error': 5.e-6,              # relative convergence tolerance
     'print_interval': 9000,               # time step interval for printing
     'timestep_reduction': 3.,             # time step reduction factor
     'gravity': 9.81,                      # gravitational acceleration
     'default_incons': [100.e4, 10]})      # default initial conditions
    # Pressure in Pa, 100 m water = 10.e5 Pa water, 10 is the temperature in Celcius
dat.start = True
```

pyPCGA-TOUGH: Running the inversion

1. Read inversion parameters

```
prob = PCGA(forward_model, s_init, pts, params, s_true, obs)
```

```
##### PCGA Inversion #####
##### 1. Initialize forward and inversion parameters
------------ Inversion Parameters -------------------------
   Number of unknowns                                  : 10001
   Number of observations                              : 100
   Number of principal components (n_pc)               : 50
   Prior model                                         : def kernel(r): return (prior_std ** 2) * np.
exp(-r)

   Prior variance                                      : 1.600000e-03
   Prior scale (correlation) parameter                 : [200.]
   Posterior cov computation                           : diag
   Posterior variance computation                      : Direct
   Number of CPU cores (n_core)                        : 4
   Maximum GN iterations                               : 10
   machine precision (delta = sqrt(precision))         : 1.000000e-08
   Tol for iterations (norm(sol_diff)/norm(sol))       : 1.000000e-02
   Levenberg-Marquardt (LM)                            : True
   LM solution range constraints (LM_smin, LM_smax)    : None, None
   Line search                                         : True
-----------------------------------------------------------
```

pyPCGA-TOUGH: Running the inversion

1. Read inversion parameters

```
prob = PCGA(forward_model, s_init, pts, params, s_true, obs)
```

```
##### PCGA Inversion #####
##### 1. Initialize forward and inversion parameters
------------ Inversion Parameters -------------------------
   Number of unknowns                                    : 10001
   Number of observations                                : 100
   Number of principal components (n_pc)                 : 50
   Prior model                                           : def kernel(r): return (prior_std ** 2) * np.
exp(-r)

   Prior variance                                        : 1.600000e-03
   Prior scale (correlation) parameter                   : [200.]
   Posterior cov computation                             : diag
   Posterior variance computation                        : Direct
   Number of CPU cores (n_core)                          : 4
   Maximum GN iterations                                 : 10
   machine precision (delta = sqrt(precision))           : 1.000000e-08
   Tol for iterations (norm(sol_diff)/norm(sol))         : 1.000000e-02
   Levenberg-Marquardt (LM)                              : True
   LM solution range constraints (LM_smin, LM_smax)      : None, None
   Line search                                           : True
-----------------------------------------------------------
```

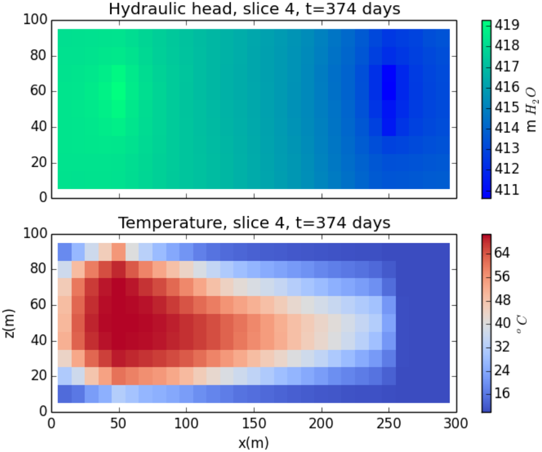
2. Run inversion

```
# run inversion
s_hat, simul_obs, post_diagv, iter_best = prob.Run()
```

```
##### 2. Construct Prior Covariance Matrix
- time for covariance matrix construction (m = 10001) is 0 sec
##### 3. Eigendecomposition of Prior Covariance
- time for eigendecomposition with k = 50 is 0 sec
- 1st eigv : 5.29487, 50-th eigv : 0.00674062, ratio: 0.00127305
##### 4. Start PCGA Inversion #####
-- evaluate initial solution
obs. RMSE (norm(obs. diff.)/sqrt(nobs)): 0.776609, normalized obs. RMSE (norm(obs. diff./sqrtR)/sq
rt(nobs)): 19.4152
```

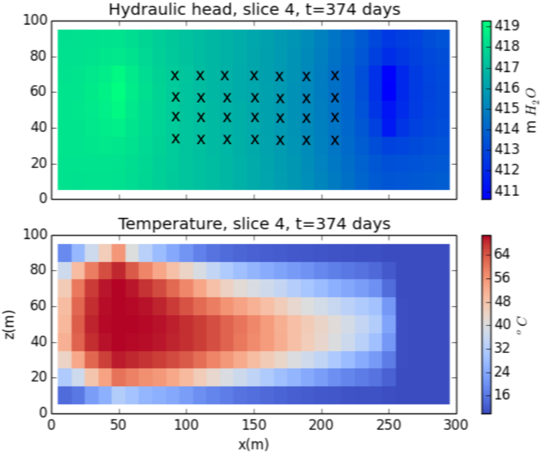
pyPCGA-TOUGH: Test Case

Synthetic case: $300m \times 100m \times 100m$, Log-normal permeability
Boundary conditions: injection-extraction system, warm water injected
Simulation time: 374 days



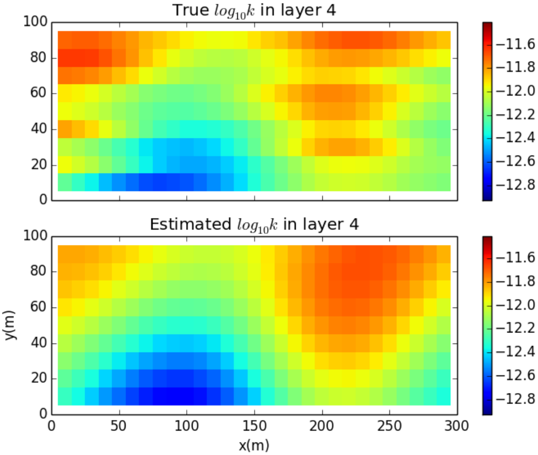
pyPCGA-TOUGH: Test Case

Unknowns: 3000 permeabilities (pmx values)
Measurements: Pressure collected every $\sim$ 5 days at 128 monitoring locations between the injection and extraction well ($n_{press.obs.} = 7400$)



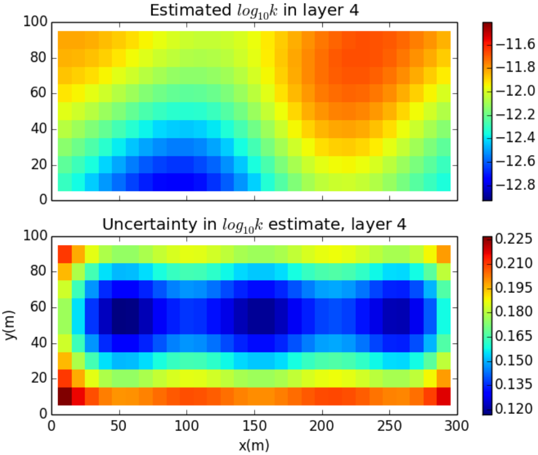
pyPCGA-TOUGH: Test Case

Unknowns: 3000 permeabilities
Measurements: 7400 pressure measurements
Principal components: 30
Time to run (with parallelization): 10 minutes on 36-cores



pyPCGA-TOUGH: Test Case

Unknowns: 3000 permeabilities
Measurements: 7400 pressure measurements
Principal components: 30
Time to run (with parallelization): 10 minutes on 36 cores

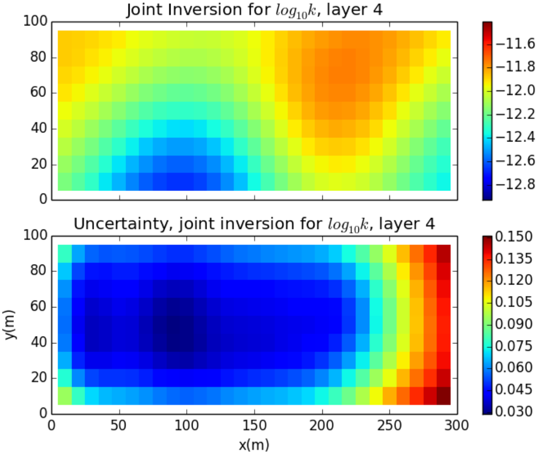


pyPCGA-TOUGH: Test Case 2 (Joint inversion)

Unknowns: 3000 permeabilities

Measurements: pressure and temperature measurements

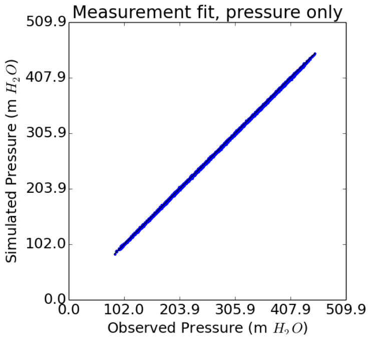
- Joint inversion of P,T data reduces the uncertainty.
- Useful for design of monitoring network.

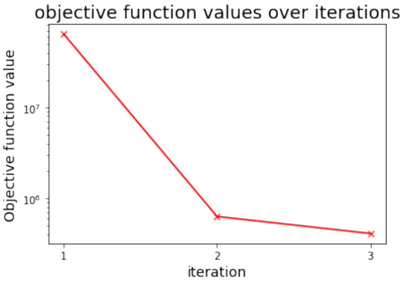


pyPCGA-TOUGH: Test Case

Inversion evaluation: Measurement fit (RMSE), objective function, L2-norm w.r.t. true field, convergence behavior

Post-processing of inversion results allows finetuning of parameters (model error, tolerance, finite difference δ, prior covariance parameters)





pyPCGA-TOUGH: An inversion package for TOUGH2 users

pyPCGA-TOUGH offers an open-source package for geostatistical inversion for TOUGH2-MP models. Package development is ongoing.

Upcoming additions include:

- Extension of PyTOUGH with tools for sensitivity evaluation
- Tutorial templates for inversion using EOS1, EOS3, ECO2N
- Visualization tools fast predictive model validation using cR/Q2 criteria
  - automatic covariance model parameter calibration
- New faster linear algebra for unstructured grids (PBBFMM3D, and HMatrix)
- Level-set and total variation method for sharp boundaries estimation

https://github.com/jonghyunharrylee/pyPCGA
https://github.com/amaliak/pyPCGA-TOUGH

PCGA: method development and applications

- Lee, Kokkinaki and Kitanidis, Fast Large-Scale Joint Inversion for Deep Aquifer Characterization Using Pressure and Heat Tracer Measurements. Transport in Porous Media, 123(3): 533-543, 2018
- Lee, Ghorbanidehno, Farthing, Hesser, Darve, and Kitanidis, Riverine bathymetry imaging with indirect observations. Water Resources Research, 54. https://doi.org/10.1029/2017WR021649, 2018
- Kang, Lee, Fu, Lee, Kitanidis, and Ruben, Improved Characterization of Heterogeneous Permeability in Saline Aquifers from Transient Pressure Data during Freshwater Injection Water Resources Research, 53(5): 4444-458, 2017
- Lee, Yoon, Kitanidis, Werth, and Valocchi, Scalable subsurface inverse modeling of huge data sets with an application to tracer concentration breakthrough data from magnetic resonance imaging Water Resources Research, 52(7), 5213-5231, 2016
- Lee and Kitanidis, Large-scale hydraulic tomography and joint inversion of head and tracer data using the principal component geostatistical approach (PCGA) Water Resources Research, 50(7), 2014
- Kitanidis and Lee, Principal Component Geostatistical Approach for Large-Dimensional Inverse Problem, (2014) *Water Resources Research*, 2014
- Kitanidis, Quasi-linear Geostatistical Theory for Inversing, *Water Resources Research*, 1995

Acknowledgements

- Faculty Research Participation Program at the U.S. Engineer Research and Development Center, Coastal and Hydraulics Laboratory administered by the Oak Ridge Institute for Science and Education through an interagency agreement between the U.S. department of Energy and ERDC.
- NSF EPSCoR Project 'Ike Wai: Securing Hawai'i's Water Future Award OIA: 1557349
- AWS Cloud Credits for Research

Thank you!