Stochastic Optimization Code Documentation

Alexander C. Hanna

May 10, 2019

Contents

1 Introduction

This stochastic optimization code uses the concepts of object-oriented programming, relational databases, high-throughput computing to implement a general inversion solver that is agnostic to the particular forward model or inversion algorithm being employed.

2 Concepts

The general background knowledge useful for further developing this software is summarized in the following sections.

2.1 Inverse Methods

Concepts from the mathematical discipline of inversion theory are discussed in the following sections. More rigorous and thorough treatments of this material are available in other works [Tarantola, 2004, Menke, 2012] and would be strongly recommended for users intending to develop their own inversion algorithms using this platform.

2.1.1 Parameter Space

The parameter space is an n-dimensional space, where n is the number of unknown parameters in the problem. Thus each point in the parameter space represents a unique model, whose likelihood can be investigated by evaluating the numerical simulation and comparing the results to measured data in order to find its position in the objective space.

The simplest way to define the parameter space is using a uniform distribution over the range of possible values for each of the n parameters, and constructing an n-dimensional hypercube. However, it is often possible to reduce the complexity of the problem by using outside information to define a smaller feasible space with an arbitrary shape. This shape can be deterministic, based on an analytical relationship between two or more parameters, or it can be statistical based on empirical observations.

2.1.2 Objective Space

The goal of the parameter estimation process is to find sets of parameter values that best fit a set of measured data, with the assumption that the parameters that best fit the data are representative of the true subsurface. In the case of reservoir characterization problems however, we often find that the inverse problem is underdetermined or that many contradictory geologic models would explain the data we observe. In addition many separate sets of measured data are often available, meaning the parameter estimation process must satisfy many distinct and sometimes conflicting objectives. These objectives might include minimizing data misfits for the pore pressure, deformation and strain misfits at a number of distinct points in the subsurface representing distinct observation wells. The objective space is therefore an m-dimensional space where each dimension represents one of the m data-fitting objectives. The goal of parameter estimation is therefore to identify the models that minimize the data-model misift of each objective, or are as near as possible to the origin of the objective space.

One approach to working with large, complex objective spaces [Marler and Arora, 2010] is to select a vector of weighting coefficients typically related to the calibrated measurement error of the instrument, and use these weights to combine the data-model misfits into a single value using equation **??**,

$$E = \sum_{i}^{m} E_i \, \omega_i \tag{1}$$

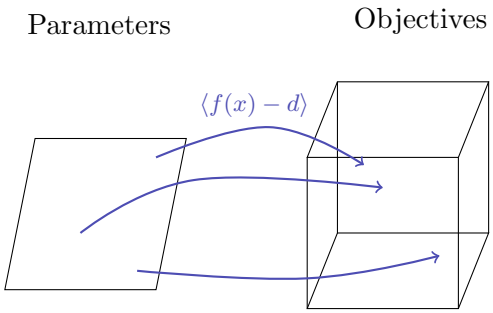


Figure 1: Example of a mapping between a 2-dimensional parameter space and a 3-dimensional objective space.

2.1.3 Pareto Ranking

An alternative approach is to use the concept of Pareto optimality, which allows us to consider each objective's data-model misfit separately. A solution is Pareto optimal when compared with other solutions if no other solution in the group is superior to it in terms of all of the objective functions. For example, in the multiobjective minimization problem presented in Figure 2a, solution 1 has a lower value in terms of objective f_1, but a higher value in objective f_2. Solution 2, by contrast, has a lower value in terms of objective f_2, but a higher value in objective f_1. Therefore neither solution is superior to the other in the Pareto optimality sense. However, solution 3 is not Pareto optimal, because both solutions 1 and 2 have lower f_1 and f_2 values. The group of solutions that are not dominated by any other solution are denoted rank 1. Solutions that are dominated only by rank 1 solutions are denoted rank 2. In Figure 2b, several ranks are identified and color-coded. The rank 1 solutions represent the best approximation of the Pareto front.

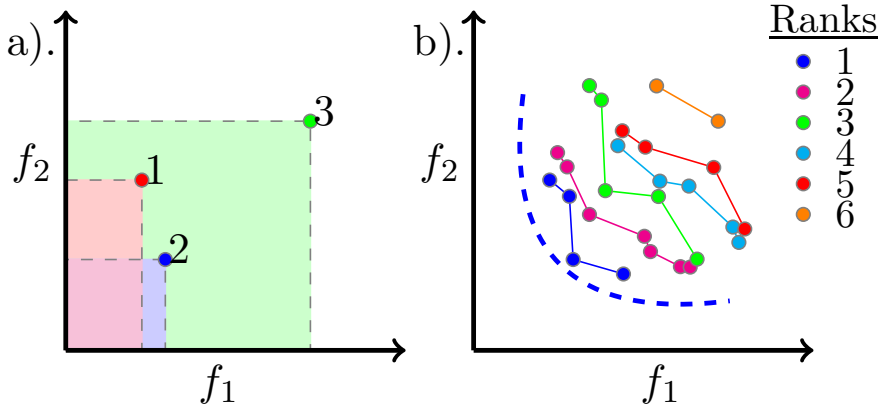


Figure 2: Illustration of Pareto optimality. Blue dashed line represents the 'true' Pareto front. Rank 1 solutions (blue dots) represent the best currently-available approximation of the Pareto front.

2.2 Object-Relational Mapping

Objects in a typical object-oriented program only exist in volatile memory for the duration of the Python script being run, and must be stored in a file for later use if a long-term, ongoing operation on the same dataset. However, serializing a large highly structured set of data structures can be computationally challenging.

Relational databases such as MySQL represent a well-developed, industry standard method of efficiently storing large highly structured datasets in non-volatile memory. However, the programming languages used to read and write to databases (ie SQL) can be difficult and generally some form of application software to mediate between user and database.

Object-relational mapping combines the strengths of object-oriented programming and relational databases by abstracting away most of the data-management tasks and allowing a programmer to work with non-volatile objects that are automatically synchronized and stored as entries in a relational database.

3 Software Architecture

The inversion code has each been implemented in Python using a modular, object-oriented approach that mirrors the structure of the database, with one class definition per database table. Classes and tables are synchronized using SqlAlchemy, a popular object-relational mapping software. This approach allows the use of external Python libraries such as numpy and scipy (math and statistics functions), matplotlib (graphics and visualization), and boto (AWS file transfer and job management).

3.1 Database Design

An abstract class (Optimization) contains general definitions for all the basic operations that any generic inversion algorithm must be able to perform, while each specific inversion algorithm inherits these definitions and adds some distinct code detailing precisely how these operations are to be performed.

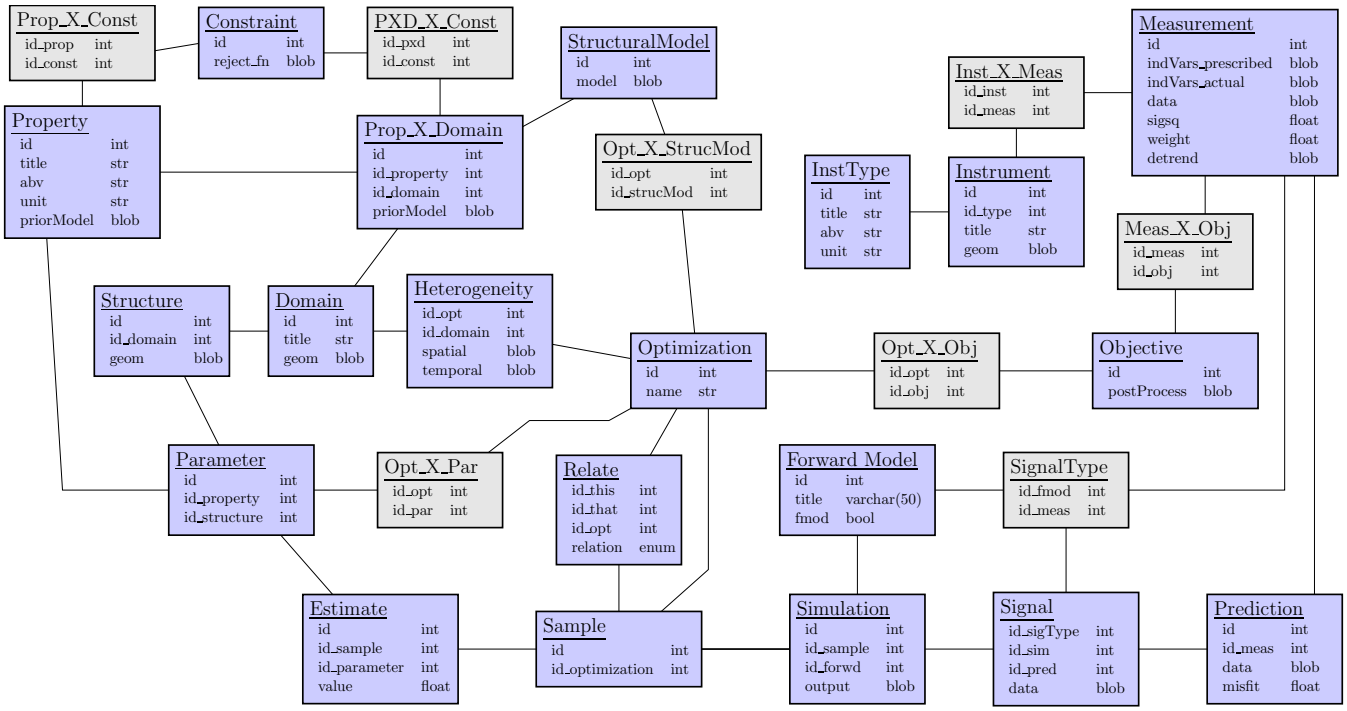


Figure 3: Entity-Relationship diagram for the inversion database.

The physical properties of interest (i.e, density, permeability, compressibility) are listed in the Property table, and the physical domains of interest (i.e., distinct rock layers) are listed in the Domain table. The Prop_X_Domain table defines a particular Property of a particular Domain using a statistical distribution, the prior model. Two or more Prop_X_Domains may be related by the Constraint table, which contains a rejection function that tests whether a particular combination of parameter values is feasible for the forward model. This allows us to define the feasible parameter space as any arbitrary shape rather than simply as an n-dimensional hypercube. The Parameter table lists the unknown parameters in our particular inverse problem. It is conceptually similar to the Prop_X_Domain table, but remains a distinct table to allow for the special case of a tomography or imaging problem, where many parameters may describe a single pixel or voxel of a given structure but all have a common underlying statistical distribution.

The types of instruments installed in the field (i.e., pressure sensor, temperature probe, humidity etc) are listed in the InstrumentType table while specific instruments are listed in the Instrument table with their name and location. In general, each instrument corresponds to a distinct entry in the Objective table, however when there are multiple forward models available to attempt to simulate the behavior of an Instrument, there may be multiple Obective entries for a single Instrument. An Objective may relate to the Measurement table, where synthetic measurement is stored or where actual measurements from the field are periodically uploaded.

3.2 Parallelization

Once these database tables are populated, the inversion process can begin. The basic function of the Optimization object is to create an entry in the Sample table, create one entry in the Estimate table for each Parameter, use these estimates to build a set of input files in a format appropriate for the physics solver being used, then create an entry in the Simulation table and stores the input files in this table. The input files are then uploaded to the Amazon Simple Storage Service (S3) where it can be retrieved by any remote machine with both the appropriate credentials and the S3 object key, a string of characters

which uniquely identifies that particular set of input files. The object key is then posted on the Amazon Simple Queue Service (SQS), another AWS service which maintains the integrity of the simulation queue. A variety of remote machines can then periodically check this queue to see if any new input files are available for evaluation, without the risk that any two compute nodes will be served the same S3 object key and thus duplicating one another's work or causing table locking by attempting to modify the same database entry simultaneously. Each compute node then uses their unique S3 object key to download the appropriate set of input files, uses them to run a (perhaps computationally intensive) physical simulation, and performs any necessary post-processing to extract the expected sensor responses from the larger output file. Since our sensors are generally relatively sparse compared to the scope of the simulated region, the post-processed summary file is generally much smaller. The raw simulation output files are then uploaded to long-term storage in an appropriate S3 bucket (or discarded if the storage cost is judged prohibitive), while the summary file is sent to short-term storage in a separate S3 bucket so it may be later downloaded by the head node. The head node then stores the simulated sensor predictions in the Dataset table. The prediction is compared to the field data stored in the Measurement table to arrive at a prediction-observation misfit.

Once prediction-observation misfits are available, each specific implementation of the Optimization object uses a unique strategy to interpret data-model misfits and assemble progressively better input files. These strategies are described in detail in the following section. Many of these strategies involve relating one Sample to another, so these relationships are defined in the Relate table.

3.3 Compute Nodes

In addition to the head node, a number of compute nodes must be set up in order to run more computationally demanding forward models. The compute nodes run in a continuous loop until the user shuts them down or some internal stopping criteria is reached. Each cycle of this loop has the following tasks.

1. Check for stopping criteria
2. Check SQS for any available input files awaiting simulation
3. Retrieve compressed file containing input files
4. Unpack compressed file
5. Perform any pre-processing necessary before simulation
6. Run simulation
7. Perform any post-processing necessary after simulation
8. Compress output file
9. Push processed and compressed output file to AWS
10. Delete any remaining simulation files that may create confusion during next cyle
11. Notify SQS that simulation is complete

3.4 Algorithms

The specific algorithms implemented so far are described in this section.

3.4.1 Monte Carlo

At each iteration, Monte Carlo uses the prior model to generate a set of parameters to be evaluated. The prior model is a statistical model representing any initial knowledge about the parameter value. It may be a uniform distribution over the range of physically possible values (i.e., porosity between 0% and 100%), or a gaussian distribution centered about what is judged to be the most likely value, with a variance judged to be large enough to be sure of bounding the true value. The Monte Carlo approach is resilient against premature convergence because it is not biased by any results from previous simulations, but also slow to converge because it does not learn from previous simulations to search more thoroughly around areas of interest.

3.4.2 Latin Hypercube

The Latin hypercube [Stein, 1987, Helton and Davis, 2003] is an n-dimensional generalization of the Latin square, a combinatorial puzzle where n different symbols are arranged in a table such that each symbol appears in exactly one column and exactly one row. The Latin hypercube inversion method subdivides the parameter space into a regular grid of arbitrary grid size, and then samples from this grid such that each parameter is allowed to vary with respect to all other parameters. Latin hypercube methods are commonly used in experimental design where a large number of unknowns are being investigated using a limited number of experiments or simulations.

1	2	3	4
2	4	1	3
3	1	4	2
4	3	2	1

Figure 4: Latin Square

3.4.3 Gradient Descent

At the initial iteration, gradient descent uses the prior model to generate a random point within the parameter space. A set of points adjacent to this point is then evaluated in order to estimate the Jacobian of the forward model. The Jacobian [Equation 2] is made up of the partial derivatives of each of the n parameters with respect to each the m objectives,

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \frac{\partial f_m}{\partial x_3} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}, \tag{2}$$

where each gradient is estimated using a central difference Taylor approximation,

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(\mathbf{x} + \delta_j) - f_i(\mathbf{x})}{(\mathbf{x} + \delta_j) - \mathbf{x}}, \tag{3}$$

as the reference model is perturbed by a small value $+\delta$ for each of the n parameters. The Jacobian is then used to determine the down-gradient direction, and an appropriate step is taken in that direction.

This process continues until the gradient is below a given threshold, indicating a local minimum has been found. Once local convergence has been detected, the prior model is again used to generate a random point within the parameter space, and the gradient descent process continues. By taking a series of steps directly down gradient, gradient descent can very quickly find a nearby local minimum. However, for inverse problems where multiple local minimums exist, gradient descent often misses the global minimum. By repeating the gradient descent process using a number of different, random starting points within the parameter space, many local minimums can be found making it more likely that the true global minimum is identified. This method can also be very computationally intensive for high-dimensionality problems, as computing the Jacobian requires $n+1$ function evaluations per step. It is also up to the user to select the step size, as well as the finite difference offset values (δ).

3.4.4 Markov Chain Monte Carlo

The Markov Chain Monte Carlo (MCMC) algorithm also implicitly moves down-gradient, but in a stochastic fashion. At each iteration, the MCMC chain takes a random step in the parameter space, compares the error at the new step location to the error at the initial location, and selects the superior model. If the newer model has a lower error, it is accepted. If the new model has a higher error, it is accepted with a probability proportional to the ratio between the old and new errors (Metropolis-Hastings criteria, [Metropolis et al., 1953]). The algorithm then takes another random step in the parameter space, and continues. Therefore, an MCMC chain is capable of occasionally moving 'up-gradient', allowing it to leave a small local minima and explore the remainder of the parameter space.

The MCMC accept/reject decision requires a single error value to compare each step to its predecessor, whereas a complex multi-objective inversion problem may involve a separate data-model misfit for each instrument installed in the field, potentially meaning dozens or hundreds of objectives. Therefore in this implementation of MCMC, we have included two methods to deal with multiobjective optimization problems. One method uses the weighted sums approach (Equation 1), where each error is weighted by the estimated noise in the data. The other method sorts all simulations run so far by Pareto rank, and uses the respective ranks of the two MCMC candidates as the basis for the accept/reject decision. Examples of these two methods are shown in Figure **??**.

3.4.5 Genetic Algorithms: NSGA-II

Genetic algorithms select the most promising models identified so far, recombine them randomly using a process called crossover, and then perturb the recombined models using a mutation operator. This heuristic approach to parameter estimation mimics the way evolution naturally selects the most capable members of a population for reproduction, gradually shifting the entire population towards the most robust, adaptive gene pool possible. Most genetic algorithms are designed for single-objective optimization, using a single value to represent a models 'fitness', which regulates how the selection, crossover and mutation operators function. However, several multi-objective genetic algorithms exist that use the concept of Pareto optimality (see Figure 2) to govern these operators.

Similar to populations in nature, genetic algorithms require genetic diversity to perform well. The Non-dominated Sorting Genetic Algorithm (NSGA-II) [Deb, 2002] encourages a diverse set of models by imposing a penalty on simulations that are too similar to each other in terms of their position along the Pareto front. It accomplishes this by identifying the models nearest neighbor on either side, along each of the m objective space dimensions. It then computes the 'crowding distance', or average distance to these $2m$ neighbors, and uses this distance as a criteria for the selection operator. This prevents the population from crowding too closely around the first Pareto optimal model that presents itself, but instead investigating

the entire Pareto front as thoroughly and evenly as possible.

The Strength Pareto Evolutionary Algorithm (SPEA2) algorithm [Zitzler and Thiele, 1998, Zitzler et al., 2001] uses the concept of *elitism* to ensure that Pareto optimal solutions are not inadvertently rejected due to the stochastic nature of the selection process. The algorithm maintains an external population of all non-dominated solutions that have been identified so far. At each generation, this external population is combined with the current population and included in the selection operator. A Pareto rank based fitness is assigned, and used in the selection, crossover and mutation operators. Similar to NSGA-II, a k^{th} nearest neighbor approach is used to penalize clustering along the Pareto front.

4 User Interface

4.1 Installation

This optimization software was developed for Ubuntu 14.04, with Python and several readily available Python packages. The necessary dependencies can be installed using the terminal command:

```
$ sudo apt-get install python-numpy python-matplotlib python-scipy python-sqlalchemy python-boto python-mysqldb python-jsonpickle python-paramiko
```

Followed by:

```
$ sudo apt-get install mysql-server mysql-client
```

the MySQL install will prompt to create a password, which does not need to be the same as the system password. Then type the terminal command:

```
$ mysql -u root -p
```

followed by the password defined previously, which will launch the SQL client. Enter the following command:

```
$ CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
```

followed by:

```
$ GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
```

This specifies the MySQL guest user credentials, which also need not be the same as the system password or MySQL password. The next important step is to set up Amazon Web Services (AWS) account, and find the aws_id and aws_key AWS provides. AWS provides temporary cloud storage of input/output files, and maintains the queue of simulations to ensure that no two compute nodes begin running the same simulation.

4.2 Input Files

The input files needed to run a simple inversion are as follows:

The connect_info.py file gives the inversion code access to the MySQL guest user credentials and AWS credentials.

The initialize.py file specifies the inversion parameters, prior models, instruments and data-fitting objectives, and optimization strategies to be employed.

The maintain.py file generally has two steps. The first step checks the AWS queue for any new completed simulations that the compute nodes may have finished, performs any necessary post-processing, and compares those simulation results to the measured data to evaluate their (hopefully improved) data fit. The second step

The maintain.bash file simply runs the maintain.py in a loop until cancelled or some stopping criteria is met. The reason this loop is not incorporated into maintain.py is that memory leak issues tend to crop up when running a very long (weeks) loop in Python, especially where each step of the loop involves dealing with a large, highly structured object-oriented data structure.

The plot.py file produces graphs of the current inversion results.

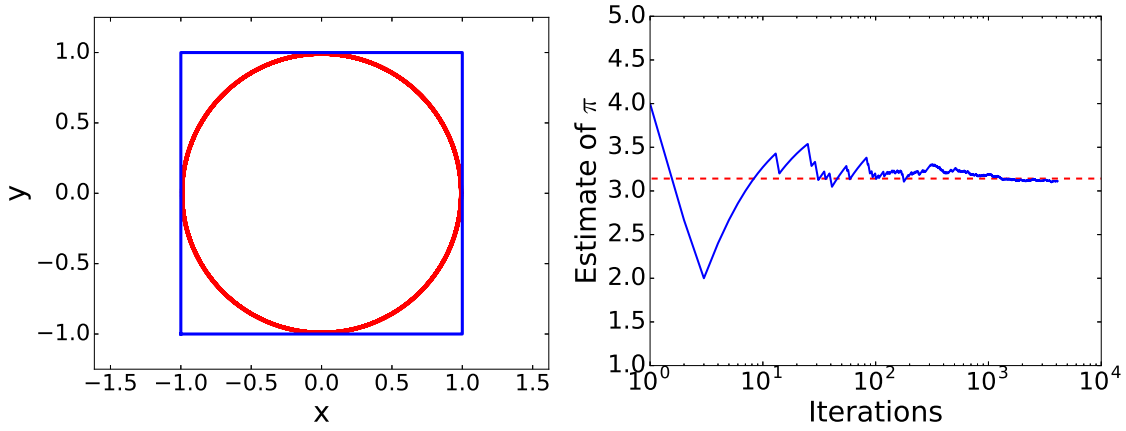
4.3 Examples

Worked examples are discussed in this section.

4.3.1 Calculation of π

Our first example uses the Monte Carlo method to estimate π. A square of length and width [Figure 5] is defined with sides $l = 2, w = 2$. A series of random x, y points are generated within this square, and the distance formula is used to determine whether each point falls within a radius of $r = 1$ of the center [Figure 5b]. The percentage of points falling within the circle is therefore proportional to the area of the circle, and can therefore be used to estimate π. The Monte Carlo algorithm is run for 10,000 iterations.

$$A = \pi r^2$$



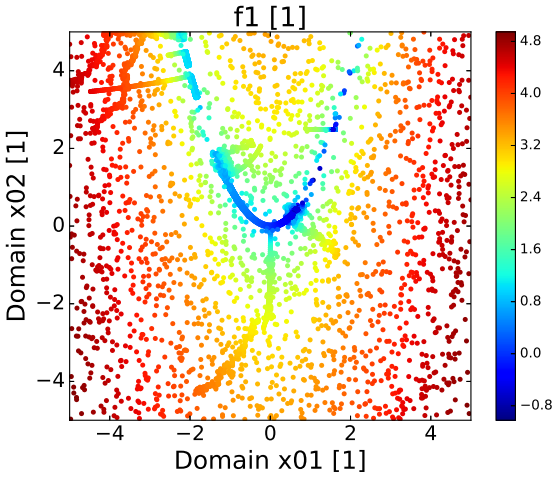
(a) (b)

Figure 5: (a) Circle with radius 1.0 circumscribed by square of sides 2×2. (b) Estimate of π as a function of Monte Carlo iteration.

4.3.2 Rosenbrock

This example uses the Monte Carlo, Gradient Descent, and MCMC algorithms to solve the Rosenbrock problem. A prior distribution is defined over the space $-5 \leq x \leq +5$, $-5 \leq y \leq +5$. Step sizes are selected at $\delta x = 0.05$ and $\delta y = 0.05$. A single 'instrument' and its corresponding forward model is defined, and

denoted f_1. The Monte Carlo algorithm is run for 2,000 iterations. Gradient descent is initialized with 5 random starting locations, and each starting locations is run for 400 steps requiring 3 simulations per step. Markov chain Monte Carlo is initialized with 5 random starting locations and is run for 400 steps requiring 1 simulation per step.



(a)

Figure 6: Scattered plots are produced by the Monte Carlo algorithm, jagged lines are produced by the Markov chain Monte Carlo algorithm and smoothly curved lines are produced by the Gradient Descent algorithm.

4.3.3 Geocentric

This example uses the Latin Hypercube and NSGA-II algorithms to fit a geomechanical model to a set of measured strain signals. Note that this inversion can require tens to hundreds of thousands of core hours in order to converge.

The model uses a program called Geocentric to compute strain responses given a set of subsurface parameters. There are three geologic materials involved: a permeable formation at a depth of 500 m below the surface (thickness 100 m), a higher permeable lens underneath that (thickness 7 m), and a confining unit situated above and below these layers. The three layers are flat and horizontal. The confining unit and formation extend laterally throughout the model domain (16×16 km) while the lens is only a few hundred meters across. A well intersects all three layers and injects water into the lens. The goal of this inversion is to use the resulting strain responses to infer the exact location and size of the lens, as well as the material properties of the lens, formation and confining unit.

There are 15 parameters defined: the permeability, bulk modulus, porosity and Poisson ratios of the permeable lens, formation and confining unit, and the lat/lon centroid and radius of the lens. Three model subdomains are defined: the lens, formation and confining unit. A model constraint is imposed requring that the lens intersect the injection well. A set of instruments are defined, one with 4 components of strain sensor (North-South, East-West, shear and vertical), one with 2 components of tilt (North-South, East-West), and three with pressure sensors installed (Wells 27, 29, 60).

The Latin Hypercube method is used to 50,000 samples of the parameter space. Approximately 50% of these models will include a lens that does not intersect the well and will be discarded. The starting populations for the genetic algorithm is also initialized. After all Latin Hypercube simulations are complete, the genetic algorithm is run for approximately another 20,000-40,000 simulations. Example results are shown below.

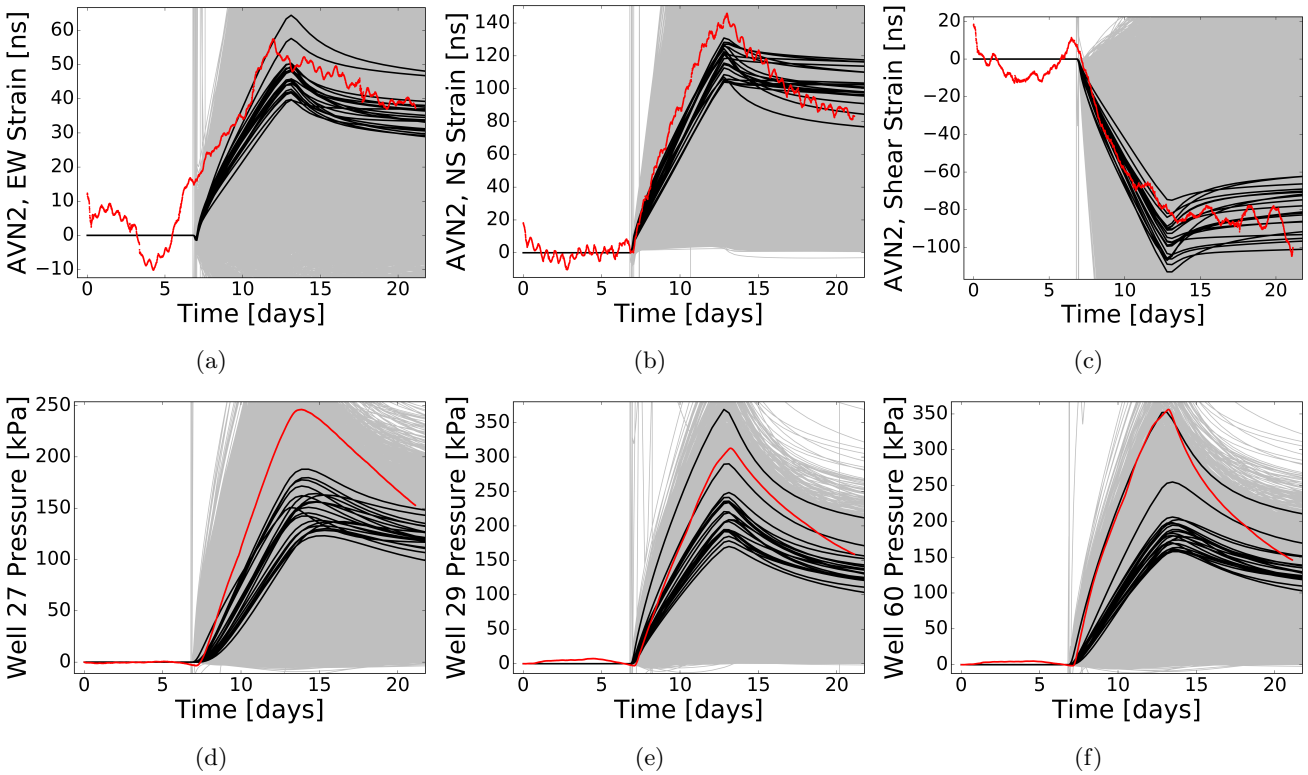


Figure 7: Simulation results from the circular parameterization inversion are compared to their corresponding field measurements (red lines) for three components of strain, and three pressure transducer locations. Simulation results are shown in gray, while best-fitting results are highlighted in black.

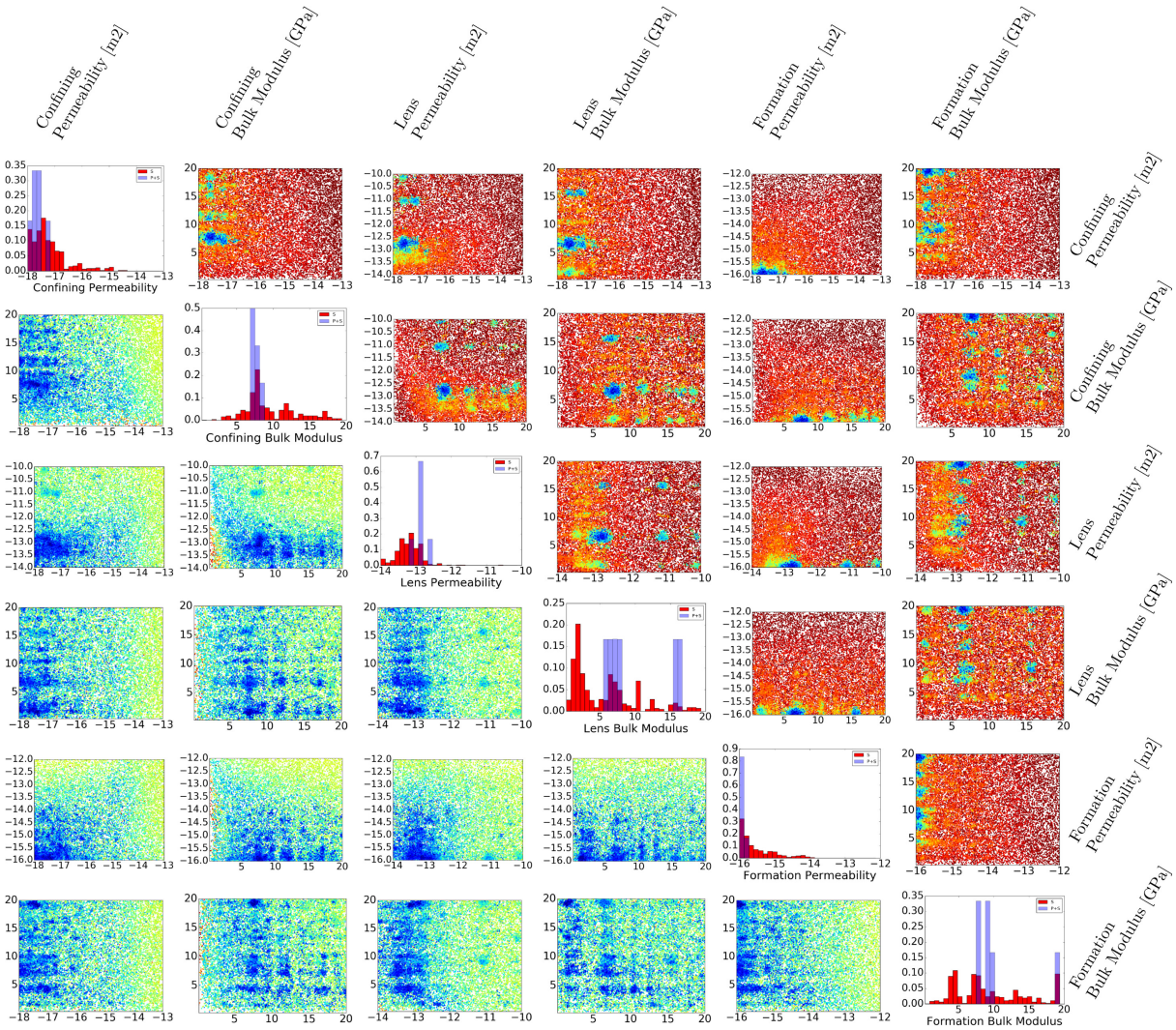


Figure 8: Parameter values for circular parameterization are shown as scatter plots in upper right and lower left corners, where color represents a weighted sum summarizing how well all six datasets (upper right) are fitted, and how well the three strain datasets are fitted (lower left). Blue points represent the best-fit models. Histograms along diagonal show the parameter distribution of best-fitting models, according to both the strain alone (S), and pressure and strain combined (P+S).

References

[Deb, 2002] Deb, K. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[Helton and Davis, 2003] Helton, J. C. and Davis, F. J. (2003). Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability Engineering and System Safety*, 81(1):23–69.

[Marler and Arora, 2010] Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6):853–862.

[Menke, 2012] Menke, W. (2012). *Geophysical data analysis: discrete inverse theory: MATLAB edition*, volume 45. Academic press.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., and Teller, A. H. (1953). Equations of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

[Stein, 1987] Stein, M. (1987). Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151.

[Tarantola, 2004] Tarantola, A. (2004). *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics.

[Zitzler et al., 2001] Zitzler, E., Laumanns, M., and Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland.

[Zitzler and Thiele, 1998] Zitzler, E. and Thiele, L. (1998). An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. Technical Report 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.