SMART Visualization Platform Prize Challenge, Docker Image Documentation

The instructions below describe the process for accessing the datasets and software necessary to make an application to the Challenge.

1. Install Docker in your preferred development environment, see links below

   - Windows
   - Mac
   - Ubuntu Linux
   - Red Hat Linux

2. Pull a copy of the Docker image and certificate files using the directions at
   https://gitlab.netl.doe.gov/smart/smart-vp-challenge

3. Run Docker image to generate new Docker container

   $ sudo docker run -it registry.gitlab.netl.doe.gov/smart/smart-vp-challenge:latest bash

4. Decompress files

   $ bash run_me_first.bash

The Visualization software product can be developed as desired by the challenger. Submissions may be submitted through DockerHub, GitHub, as an application, or as a link to an ftp (or similar) site. Please refer to the Official Rules for how submissions will be judged. If you are submitting your final product via DockerHub:

1. Create an account in DockerHub and login to the DockerHub registry

   $ sudo docker login

2. Create new private DockerHub repository (*https://hub.docker.com/repository/create*) using your selected username and project name.

3. Find container id of your current container

   $ sudo docker ps -l

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
c3f279d17e0a	SMART/challenge:latest	/bin/bash	7 days ago	Up 25 hours

4. Commit modified docker container to an image

   $ sudo docker commit $container_id username/project_name

5. Push committed docker image to new private DockerHub repository

   $ sudo docker push username/project_name

6. When your submission is complete, please complete the final submission form which was provided to you upon registration and add the following SMART_DOEFE users to your new private repository username/project_name

   - achanna, mkunderwood

Contents of Provided Docker Image

The software platform will use a model-view-adapter (MVA) design pattern in order to provide a layer of abstraction between the teams developing the front-end graphical interface and teams developing the back-end machine learning and subsurface modelling functionalities. The back-end functionalities including the subsurface reservoir simulators, machine learning algorithms, model calibration, uncertainty quantification and decision analysis tools are therefore presented as a black box to the front-end developers. Therefore while the internal workings of this black box will be developed by the SMART Initiative team and will be changing rapidly, the Adapter will provide a simplified interface that changes very rarely and introduces minimal disruption to front-end development.

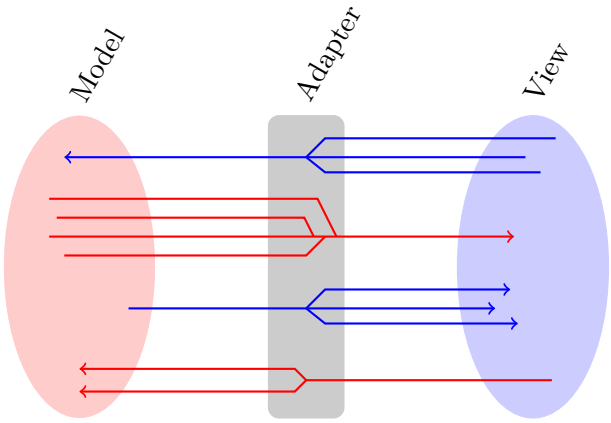


Figure 1: Model-View-Adapter design pattern

Some preliminary "model" functionalities have been implemented and will be improved over the lifetime of this project, and preliminary "view" functionalities are provided here to act as a visualization starting point for front-end developers. A set of static datasets are also provided as examples of the more complex subsurface models that will eventually need to be visualized.

A software platform is also provided, which allows users to dynamically interact with synthetic subsurface models over a span of 50 virtual years. From within a command line interface, the user can set up a synthetic "true" model of the subsurface. They can then place hypothetical wells to forecast the subsurface response to various injection schedules and well locations, and forecast the value of information for various observation locations. They can then choose well locations to drill, and can "advance" the simulation forward in time to collect synthetic data from the "true" subsurface model.

A model calibration process is then run, using this data to make inferences about the "true" subsurface model. This information is then used to develop forecasts and performance metrics, and allows the user to re-evaluate their decisions and install new wells, modify pumping schedules or install new sensors. As this iterative process continues, a decision tree [Figure 2] is gradually built representing the fixed set of choices that have happened in the "past", the more complex set of possible decision that could be made in the future, and the opportunities where uncertainty reduction from a new sensor or a more strategic pumping schedule could better inform a decision that needs to be made in the future.

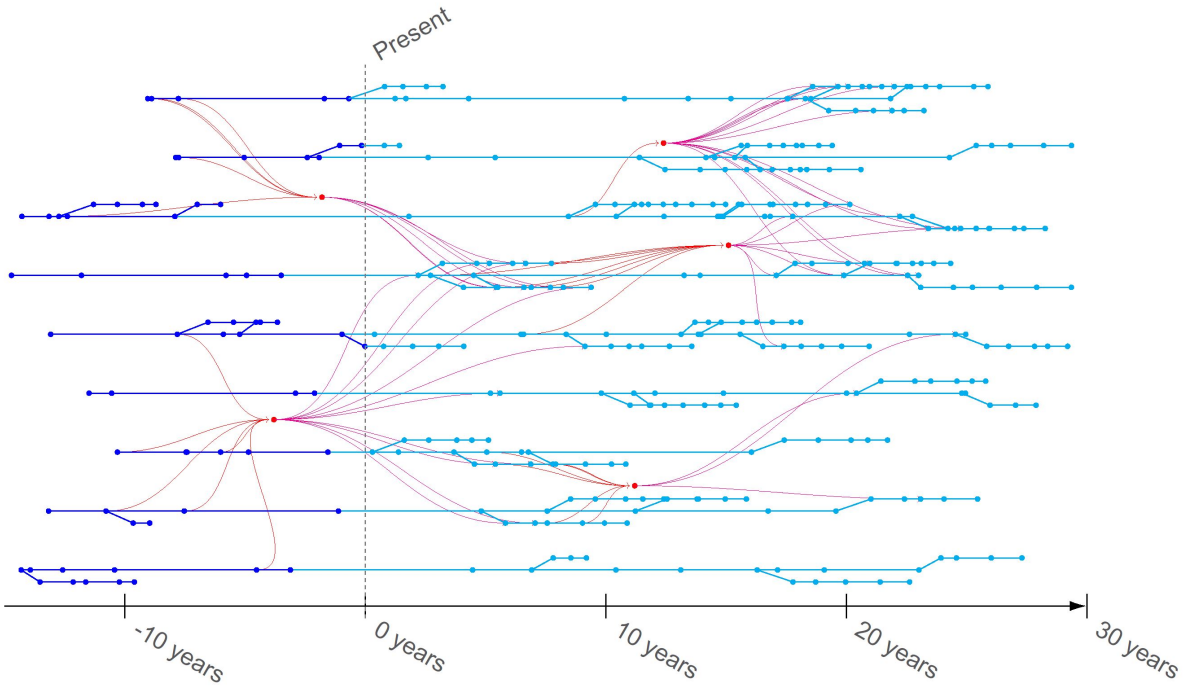


Figure 2: Decision Network where dark blue represents chains of "past"/fixed decisions, light blue represents chains of possible future decisions, and red lines represent the potential for uncertainty reduction.

These datasets are stored in a MySQL database [Figure 3], which is synchronized with a set of Python objects via an object-relational mapper called SQLAlchemy. This Python codebase includes the subsurface model generators, analytical solvers, model calibration routines and visualization functions. These functionalities will later be exposed via a RESTful API (likely in Flask) to allow for remote user interactions with cloud services, likely mediated through JSON or XML data structures. This interface will allow code from a variety of programming languages to interact with the Python functions that produce the datasets.

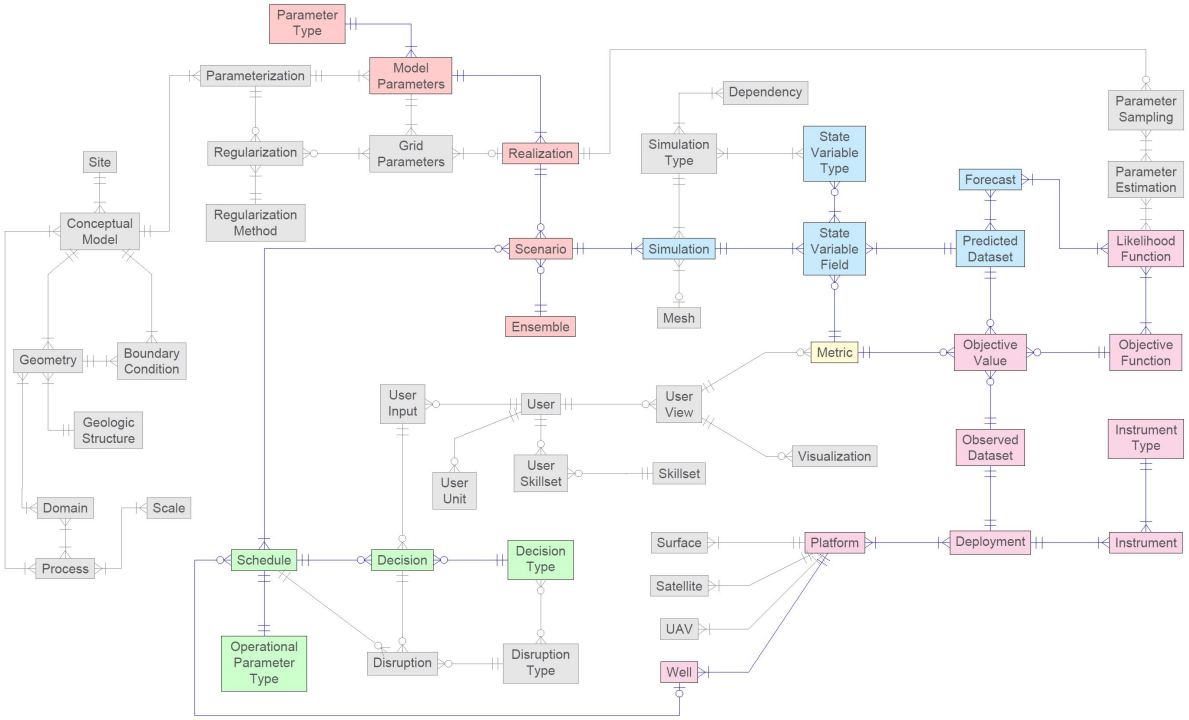
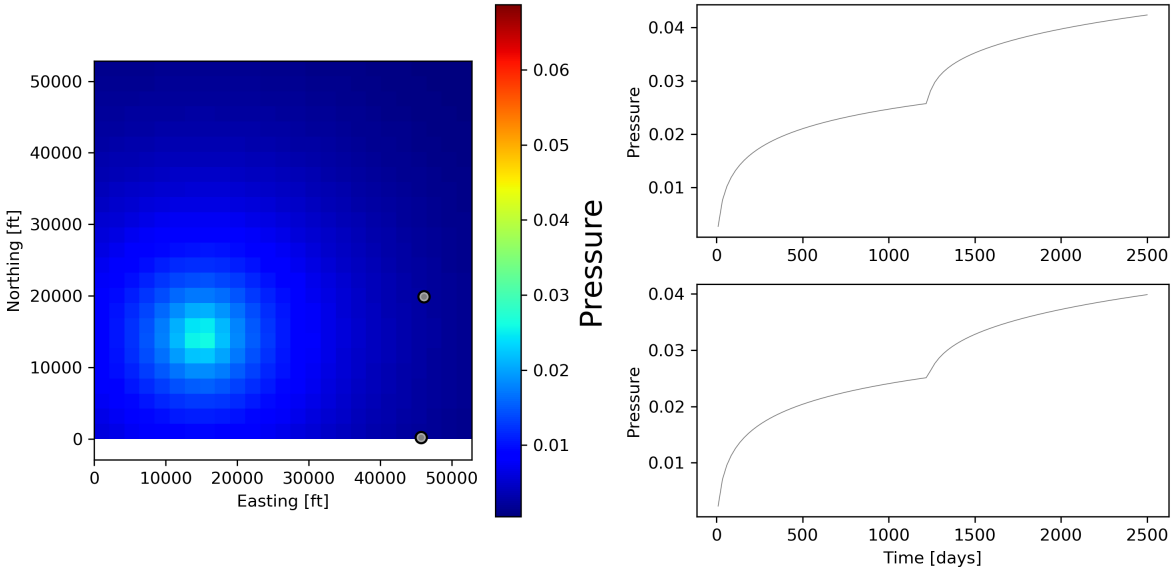


Figure 3: Entity-Relationship diagram shows the structure of the object-relational data structure. Gray boxes represent tables/classes which have not yet been implemented.

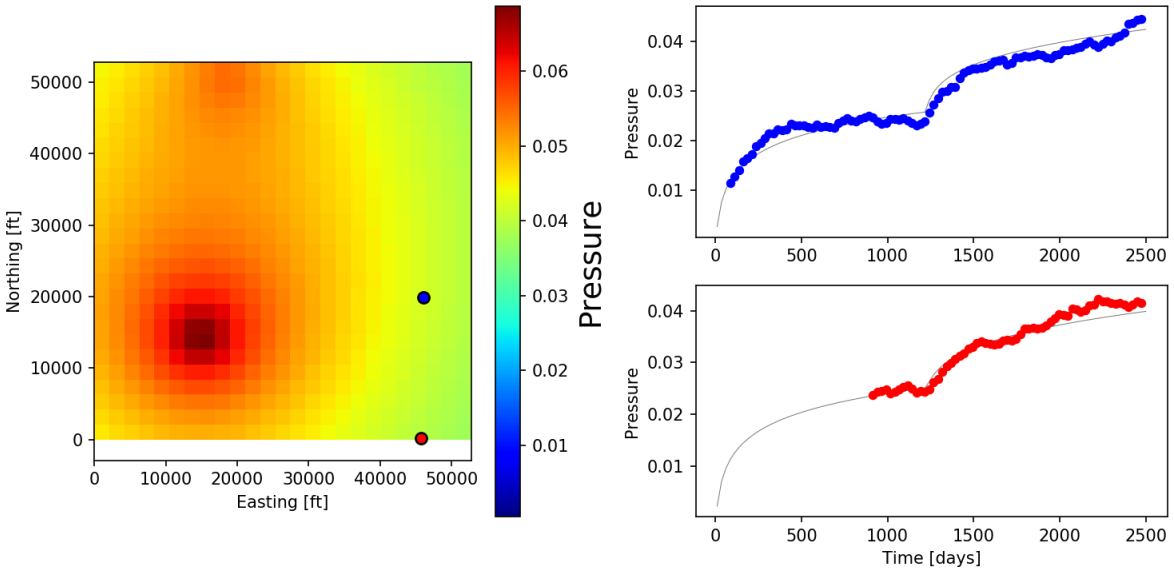
As a preliminary model, a very simple hydrologic system is implemented in Docker using analytical groundwater equations to compute pressure buildup from a well injecting fluid into the subsurface. To run these functions, first start MySQL using the following commands.

```
$ /etc/init.d/mysql start
$ mysql
$ SET @@global.sql_mode= '';
```

The included script (/root/dynamic_datasets/run.py) starts by placing an injection well at a random location, and two wells with pressure sensors installed at random locations. At a point half-way through the experiment, a second pumping well is installed and begins pumping. The first time-step of the reservoir pressurization is shown in map view in the left panel below, and the monitoring well locations are shown as gray dots. In the right panel below, the pressurization predicted by the double-well injection schedule is shown as gray lines.

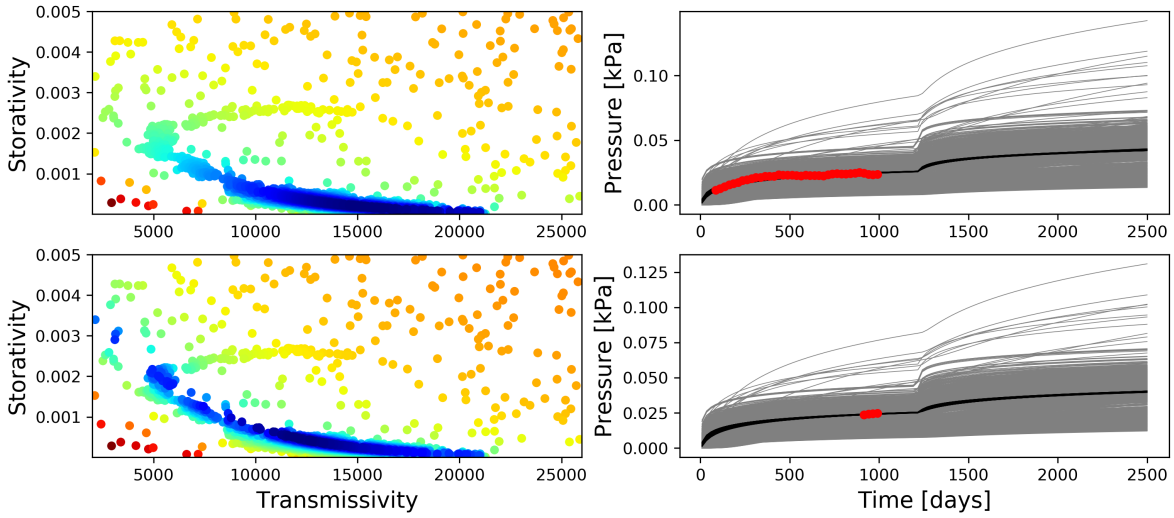


The last time step of the pressurization process is shown in map view in the left panel below, and the "measured" data collected by the sensors is shown in the right panel. The "measured" data includes a random drift and noise component, to mimic the limitations of real field data.



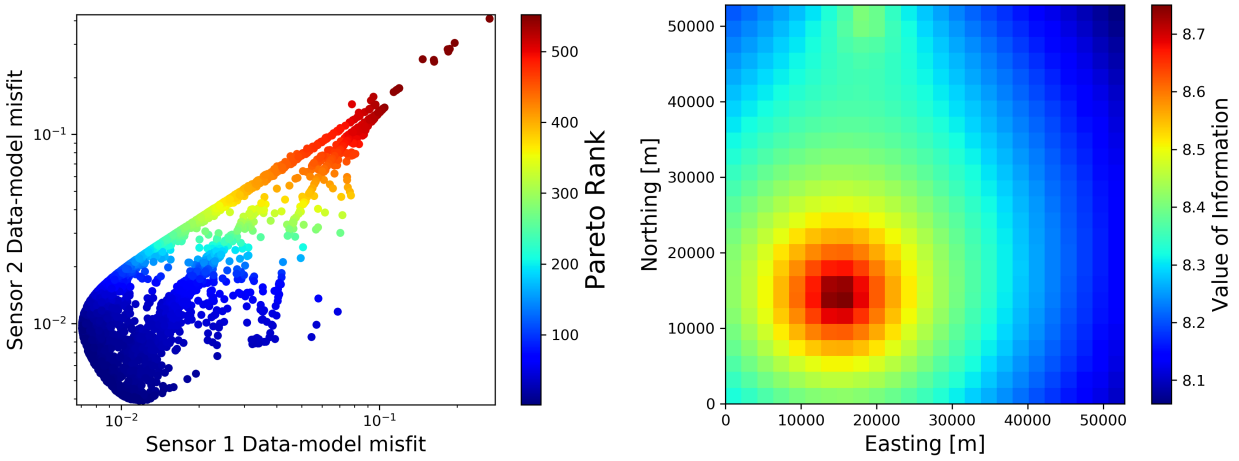
This "measured" data is used to guide a model calibration process, where a series of groundwater models are proposed at random and their predictions are compared to the measured data to decide their likelihood. In this case, each "geologic model" is defined by two numbers, the transmissivity and storativity. These models are represented by points in the left panel below. The pressure signal from each of these models is compared to the "measured" data to define the color of the points, where red points fit the data poorly and blue points fit the data well.

Initially, a set of unrelated, random models or parameter combinations (ie, the Monte Carlo method) are used. These models are represented by the scattered points in the left panel below. Then, a sequence of related models are generated where each model represents a small change from the model before (ie, the Markov Chain Monte Carlo method). With each model change, the predicted pressure signal is compared to the measured pressure signal. If the new model fits the data worse than the previous model, it faces a probability of being discarded and then a different small change is made to the previous model. If the new model fits the data better than the previous model, it is accepted and becomes the new standard to beat. This process repeats for a fixed number of iterations. In the left panel below, the model begin in the upper right corner and gradually wanders to the left until it reaches a parabola-shaped region where it stays for the rest of the process. This indicates that each of the models in that area explain the data approximately equally well. The measured pressures are shown in the right column, compared to the pressure predictions associated with each model. The best-fitting models (ie, the blue dots in the left panel) are highlighted in black.



Since there is measurement noise present, the two datasets will tend to disagree with one another slightly, with some models explaining sensor 1 better, and others explaining sensor 2 better. The figure below shows the tradeoff relationship between fitting sensor 1 and sensor 2. The leading edge near the lower right corner represents the set of models that make some compromise between the two data-fitting objectives.

This collection of compromise models (denoted the Pareto optimal set) are then used to define the value of information of additional sensor locations. Essentially, this value of information step involves looking at the models that agree closely with each other (and with the data) in the places where we do have data, and looking at where in space and time they disagree with each other. We can then produce a map identifying locations where an additional sensor would be expected to provide maximum information about the subsurface. The red areas in the figure below represent high value of information regions, promising areas for additional sensors.

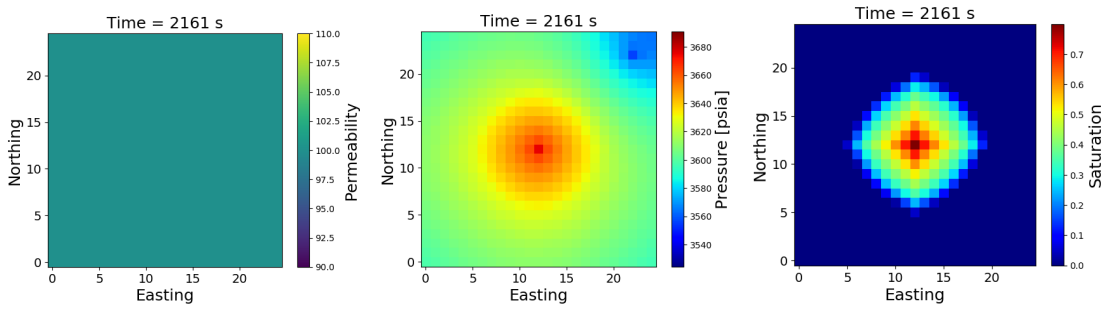


Sensor are then installed in these areas, and the model calibration process is repeated so we can gain a better estimate of the transmissivity and storativity. An ideal visualization experience would allow the user to quickly refer back and forth between these types panels in order to understand the current range of best-fit models, the set of forecasts implied by those models, the set of decision that have already been made in the field and datasets that have already been collected, and the set of future possible decisions that could be made, and how consequences and uncertainties propogate through that decision network.

Static Datasets: A set of static geologic models are provided for visualization purposes. These models do not include the full user interactions described above, but tend to be more physically realistic.

a. CMG Models: A set of 27 stochastic CMG realizations are provided, as well as a Python script (/root/static_datasets/toy_problem/gem-plot.py) which converts these models to numpy arrays and produces visualizations of them. The parameters include

   - 2d static permeability fields
   - 2d, time variant pore pressure fields
   - 2d, time variant CO2 saturation fields



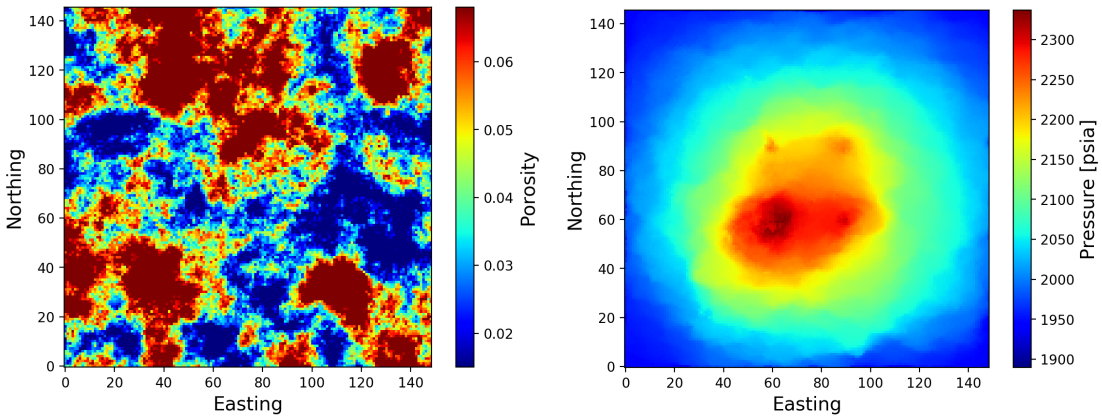
b. EERC Clastic models: A set of 300 stochastic realizations are provided, as well as a Python script to plot them (/root/static$_d$atasets/clastic$_m$odel/gem − plot.py)

   c. 3d static porosity fields

   d. ... 3d, time variant pore pressure fields

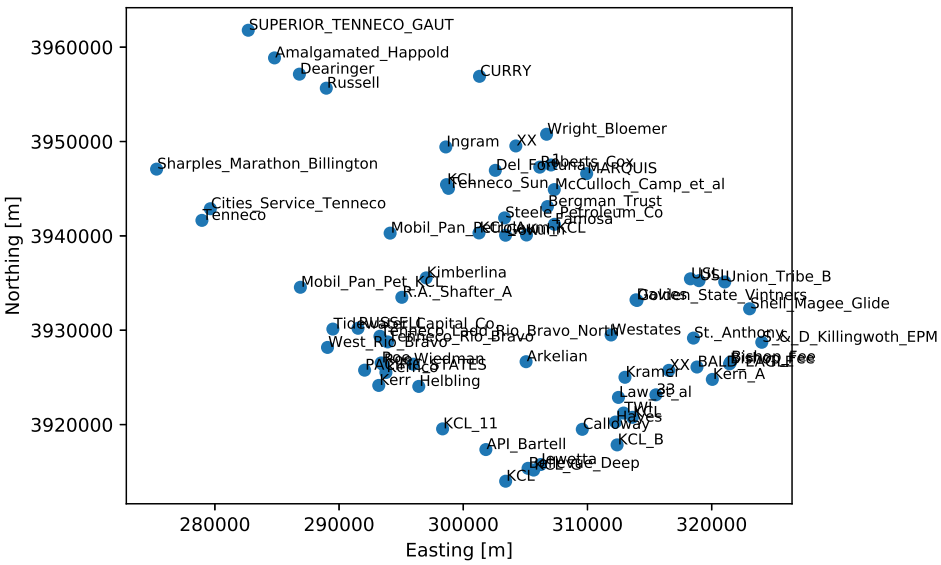
   e. ... 3d, time variant temperature fields

   f. ... 3d, time variant gas saturation fields

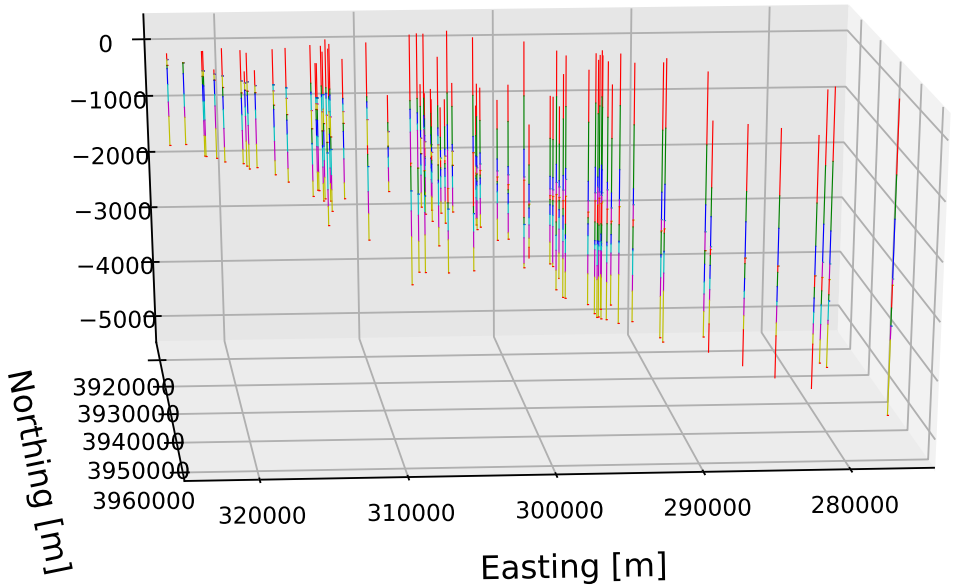


Components of the Kimberlina dataset are included, along with Python code (/root/static_datasets/kimberlina/reservo /root/static_datasets/kimberlina/leak_scenarios/leak_slices.py) to visualize them

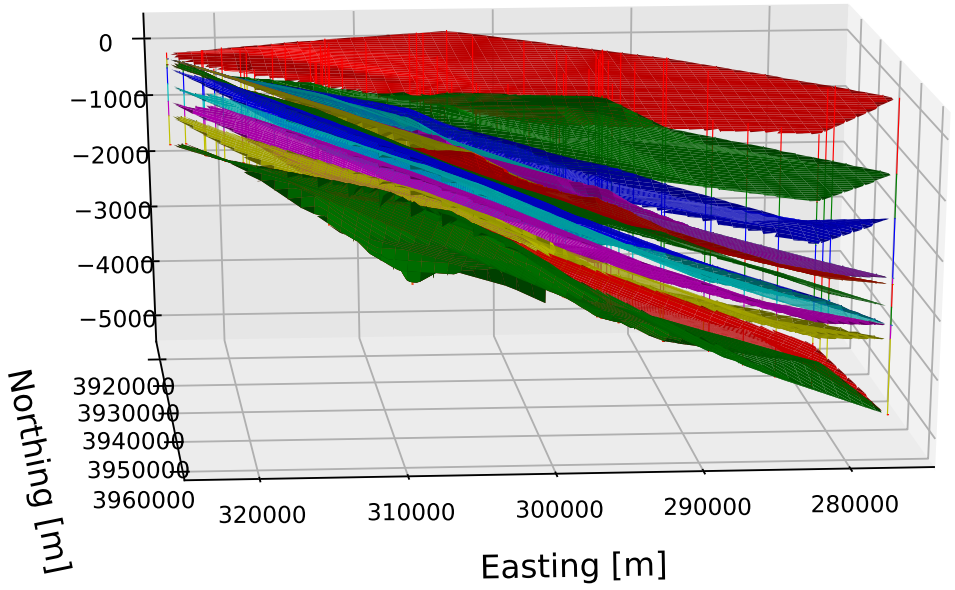
- Python code produces a static 2d map of well locations. An ideal interactive visualization would display well labels next to their point locations, and would include options to change the font size, make the labels transparent, show the labels on mouse scrollover only, and auto-fit the x and y limits to leave enough room to display labels.



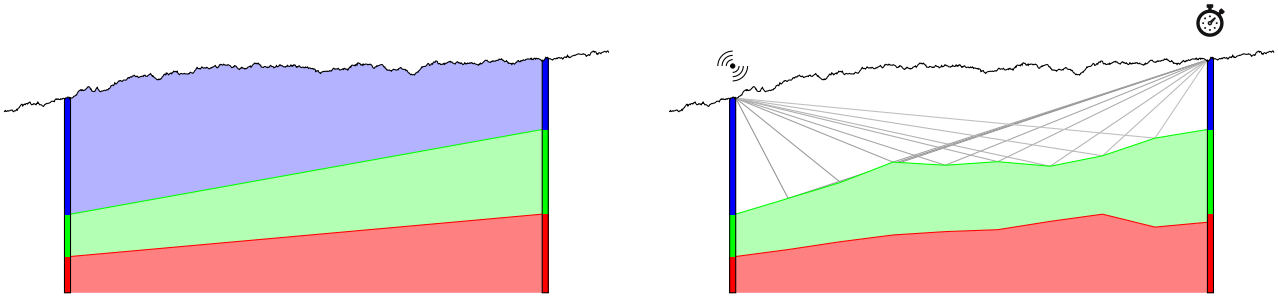
- Python code produces a static 3d visualization of the wells and the locations and thicknesses of the geologic layers through the length of each well.



- Surfaces are also included to represent the geometry of geologic layers, based on linear interpolation of wellbore data.

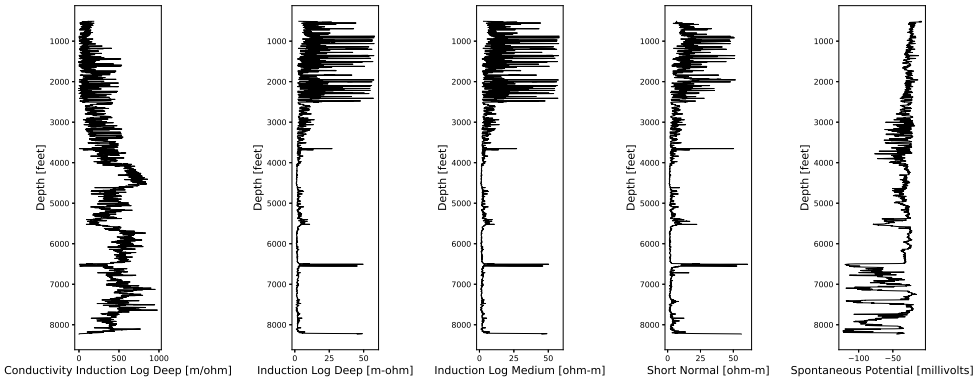


- An ideal visualization experience might include the ability to rotate, zoom, impose vertical exaggeration, select wells and bring up detailed well logs in a separate tab, select a geologic layer and make it transparent or view a detailed geologic description
- We also often have data allowing geophysics-informed interpolation between wells, where rather than simple linear interpolation we transmit some type of signal into the earth and recording its travel-time and how it interacts with the subsurface to infer the shape of rock layers

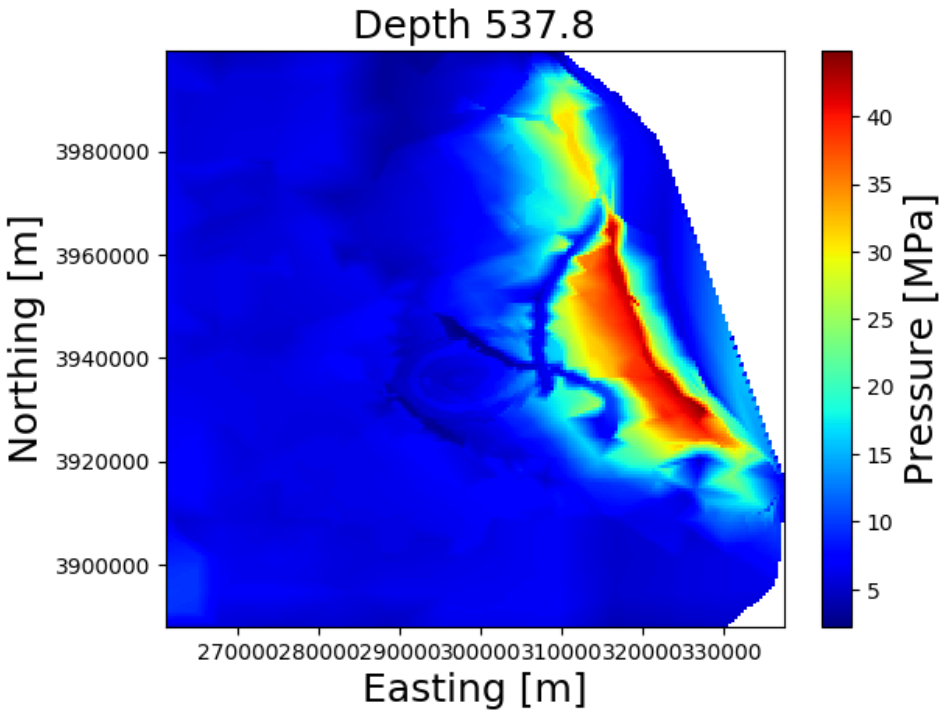


- Note that in some cases geologic layers may be discontinuous, appearing in one wellbore and then pinching out

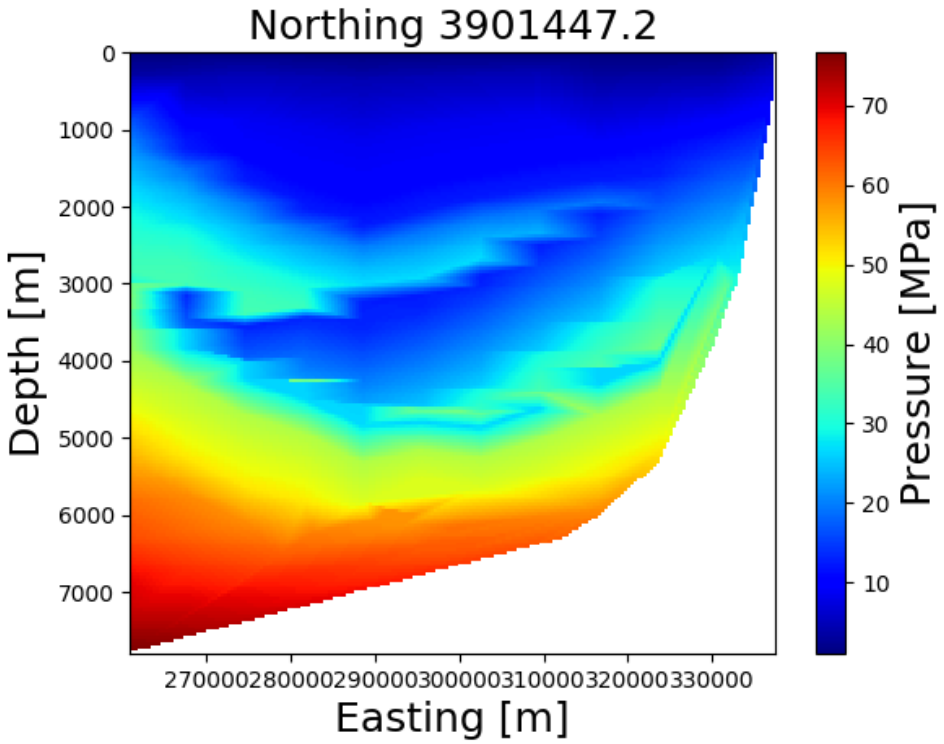
- Other datasets that may be associated with a well are well logs, where a sensor is lowered down a well and measurements are taken as a function of depth



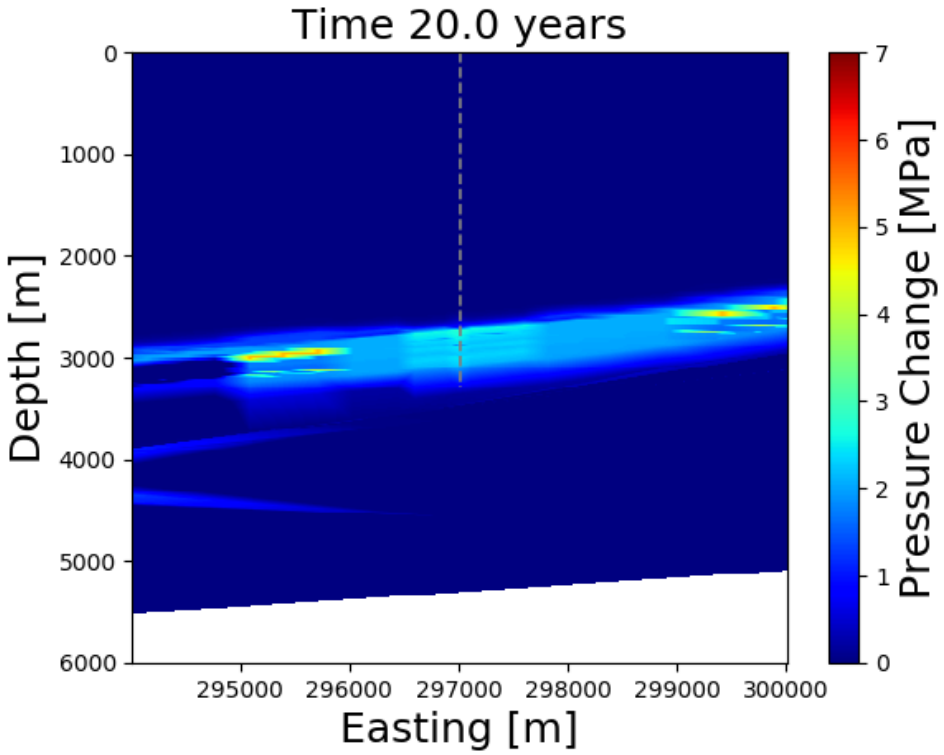
- We also have a set of 3d data cubes representing pressure and CO2 saturation in the Kimberlina site. An ideal visualization experience would display these plumes propagating through time, and visually connect the plume metrics to the insights gleaned from the optimization process. These 3d cubes can be visualized as 2d slices through a 3-dimensional space, for example the figure below shows a single slice in map view (horizontal) at a depth of 537.8 meters. A gif is available inside the Docker image which shows a series of horizontal slices in succession, allowing one to visualize the full 3d shape of the reservoir. A more dynamic visualization might include a button allowing a user to scroll through these slices manually to inspect details of interest, or to define an inclined plane or irregular surface along which to view pressures and saturations.



Similarly, a vertical slice of the 3d reservoir is shown below, and a gif made up of vertical slices is available in the Docker image.



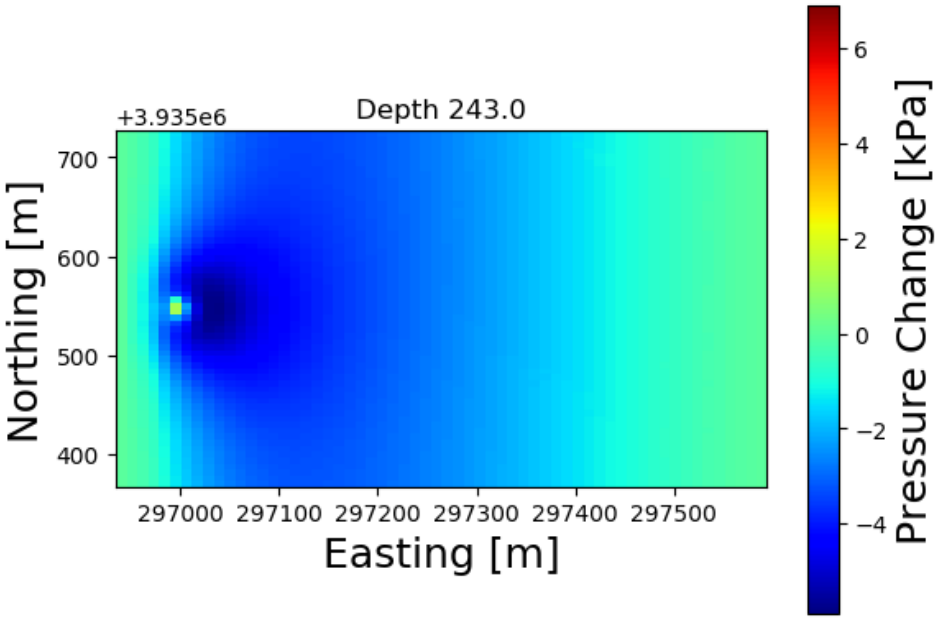
The following slice shows a compartment of the reservoir pressurizing after 20 years of injecting into the Kimberlina well. A time-variant gif of this compartment is included in the Docker image. An ideal visualization might also allow a user to scroll through time slices of the reservoir in addition to horizontal or vertical slices.



While these reservoir simulations depict a large, deep region on the order of 10s of kilometers and several kilometers deep, another set of simulations of potential wellbore leaks are included.

These represent leaks in the shallower subsurface, one the order of hundreds of meters localized immediately about some of the wells shown in the figures above, which may become leakage pathways.

Therefore, a similar set of horizontal, vertical and temporal slices are included for the shallower wellbore leakage scenarios. These datasets use a common coordinate system, but are at different resolutions. An ideal visualization experience would allow the user to quickly switch between these results in order to visually compare between a hypothetical leak or no-leak condition.



A set of example fracture networks are also provided, as well as the code libraries necessary to generate (dfnWorks), simulate (FEHM and PFLOTRAN) and visualize (visit) these datasets.