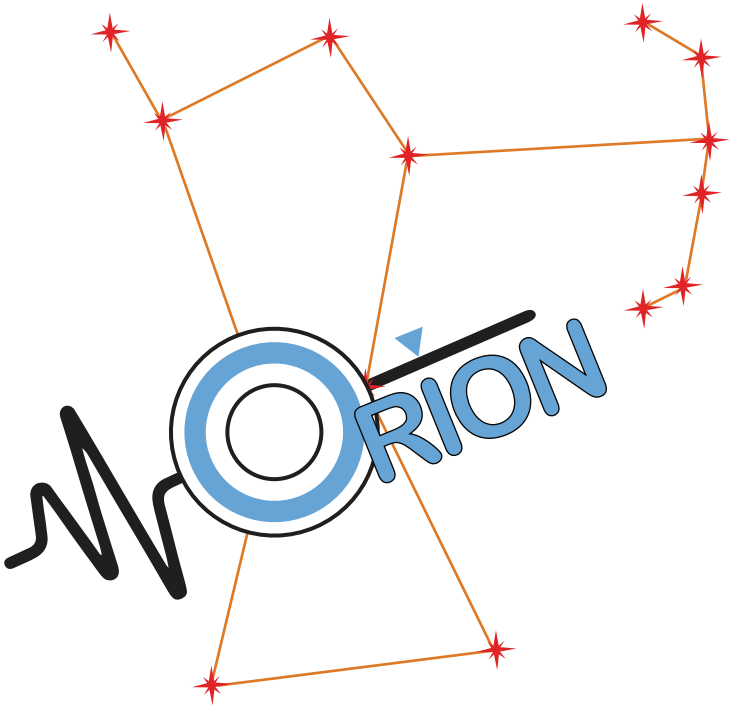
Orion: Operational FoRecastIng fOr INduced Seismicity



Orion Contributors:
Lawrence Livermore National Laboratory
Lawrence Berkeley National Laboratory

October 26, 2022

CONTENTS:

Welcome to the Operational Forecasting of Induced Seismicity (Orion) tool documentation.

CHAPTER

ONE

ABOUT

Orion is developed by Lawrence Livermore National Laboratory (LLNL) and Lawrence Berkeley National Laboratory (LBNL) as part of the United States Department of Energy's SMART and NRAP programs. Orion is built using Python, and is composed of a seismic forecasting engine and an optional GUI. For new users, we recommend that you first read through the User Guide and use the GUI to explore the set of built-in examples

1.1 User Guide

1.1.1 Installation from Source

To install Orion from it's source you must first clone the gitlab repository. You can then install it within an existing Python environment using *pip*:

```
git clone git@gitlab.com:NRAP/orion.git
pip install .
```

Note: On shared machines, we recommend that you install Orion within a virtual Python environment to avoid dependency issues with other tools.

Optional Features (PyCSEP)

Some optional features (fetching ComCat catalogs from the Internet, etc) require `pycsep`, which has some non-Python pre-requisites that may not be present on your system (proj, geos, shapely). To setup your environment, we recommend first following the steps in the PyCSEP Documentation. After this, Orion can be installed via *pip* ` with the optional features:

```
git clone git@gitlab.com:NRAP/orion.git
pip install .[pycsep]
```

1.1.2 Pre-compiled Version (Experimental)

For certain systems, pre-compiled versions of Orion are available on EDX. These files are portable, so they simply need to be downloaded to the local machine and executed. If you encounter any issues running these versions of orion, try running the *debug* version of the executable, which launches a console that will display potential error messages.

1.1.3 Running Orion

To open the Orion GUI, run the following from the command-line:

```
orion_forecast
```

Alternately, Orion can be run without the GUI by specifying the following options:

```
orion_forecast --no_gui --config filename.json
```

As it is running, Orion will maintain a copy of its configuration in the user cache (located at *~/.cache/orion/orion_config.json*). When opening Orion, the code will attempt to resume using this file. If you encounter any error opening Orion, we recommend that you delete the cached file and try again.

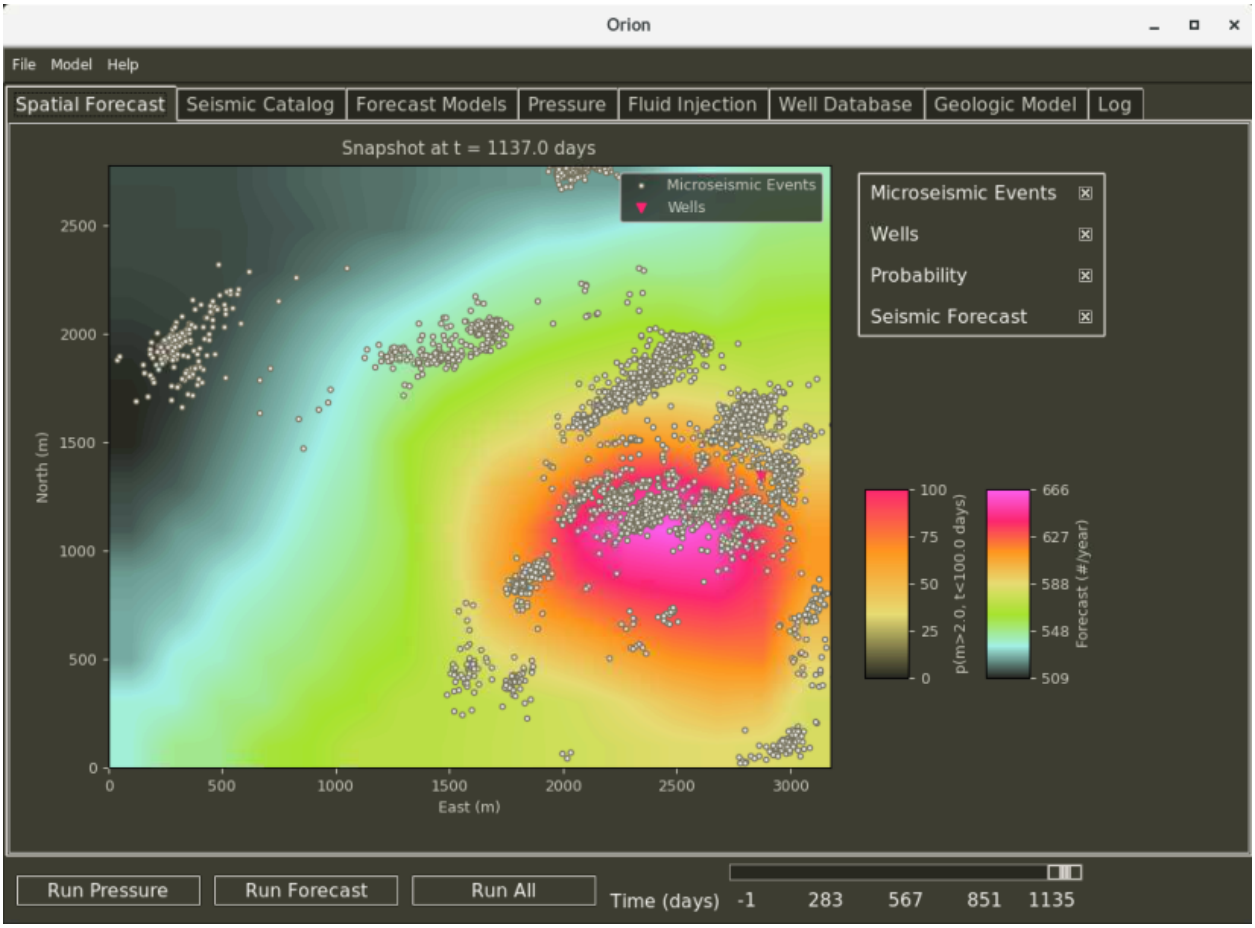
1.1.4 Orion GUI

When running the Orion GUI, a new window will open on the local machine. At the top of the window, there are a set of dropdown menus that can be used to load data, save figures, open the configuration window, and load a set of pre-constructed examples. We recommend that users load the 2014 Cushing Oklahoma example by selecting *Model/Built In/Oklahoma_Cushing_2014*, and then inspect the configuration window by selecting *Model/Configure*.

Once Orion is configured, you can trigger pressure and/or seismic forecast calculations using one of the three buttons at the bottom of the interface. As these calculations complete, they will create and populate the figures in the main body of the interface, which is organized into a set of tabs: *Spatial Forecast*, *Seismic Catalog*, *Forecast Models*, *Pressure*, *Fluid Injection*, *Geologic Model*, and *Log*. For figures that display snapshots of data over time, the active time can be set via the time-slider at the bottom of the interface or via the configuration window.

Spatial Forecast Tab

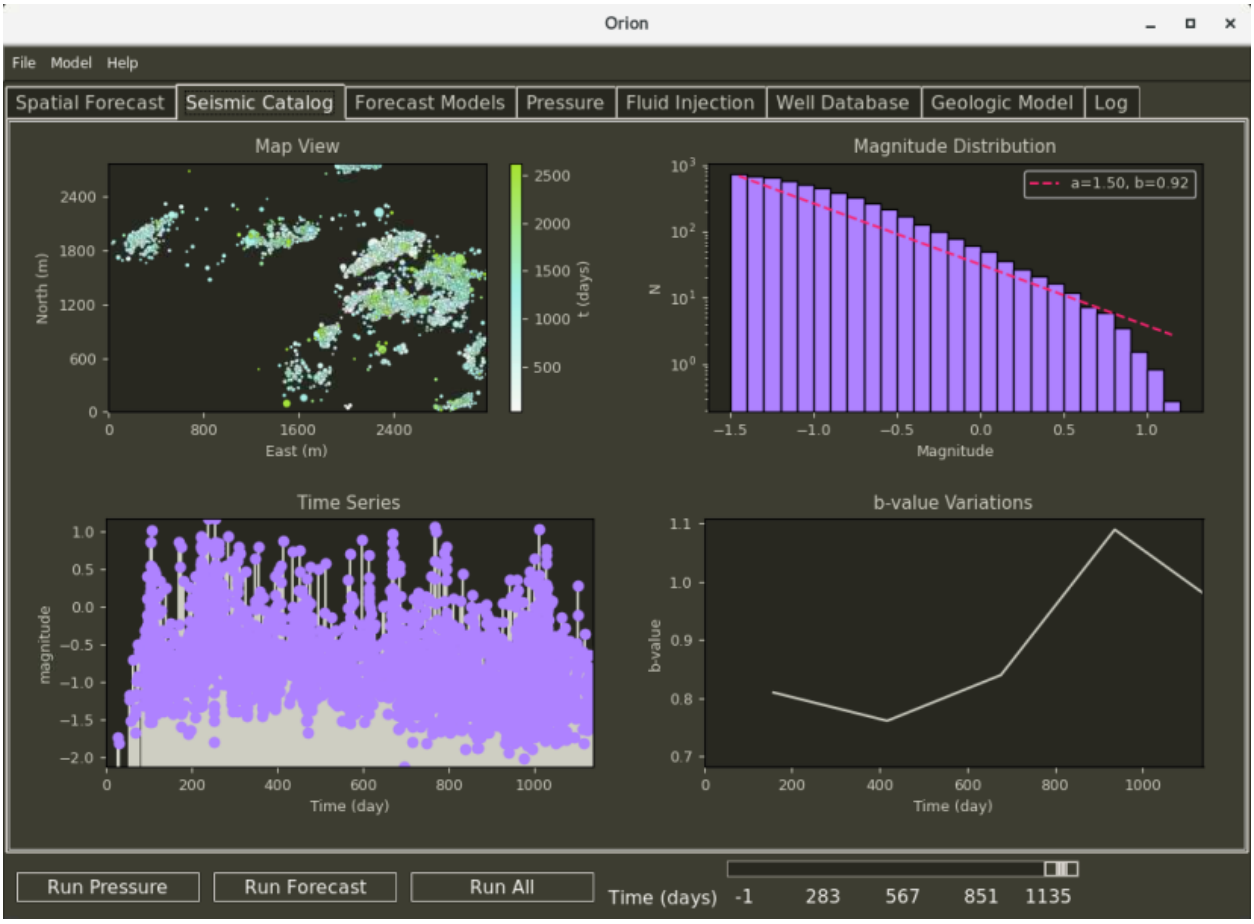
The Spatial Forecast page displays spatial forecast information on the user-defined grid, well locations, and the location of seismic events (up to the active time). To the right of figure, there is a set of checkboxes that can be used to turn on/off the various layers of the plot. The *Seismic Forecast* layer shows the estimated seismic event frequency (number of events per year) at points in the grid, and the *Probability* layer shows the estimated likelihood that an event greater than a target magnitude will occur during the next period of time. The minimum magnitude and time frame can be set in the configuration window under *Forecast Models* (*Time range*, *Dial magnitude*).



Seismic Catalog

The *Seismic Catalog* page displays key information about the active seismic catalog:

- Map view: This shows the location and magnitude of events. The data are projected onto a local UTM grid, with a local origin in the southwest.
- Magnitude distribution: This shows how often different sized seismic events occur. The red trend-line shows the current fit for the Gutenberg-Richter parameters, and the red triangle indicates the smallest magnitude where we have a complete catalog.
- Time series: This shows the size of events over time in days, with the origin set to the first event measured in the catalog.
- b-value variations: This shows how the Gutenberg-Richter b-value changes over time, which believed to be a key indicator of future seismic activity.



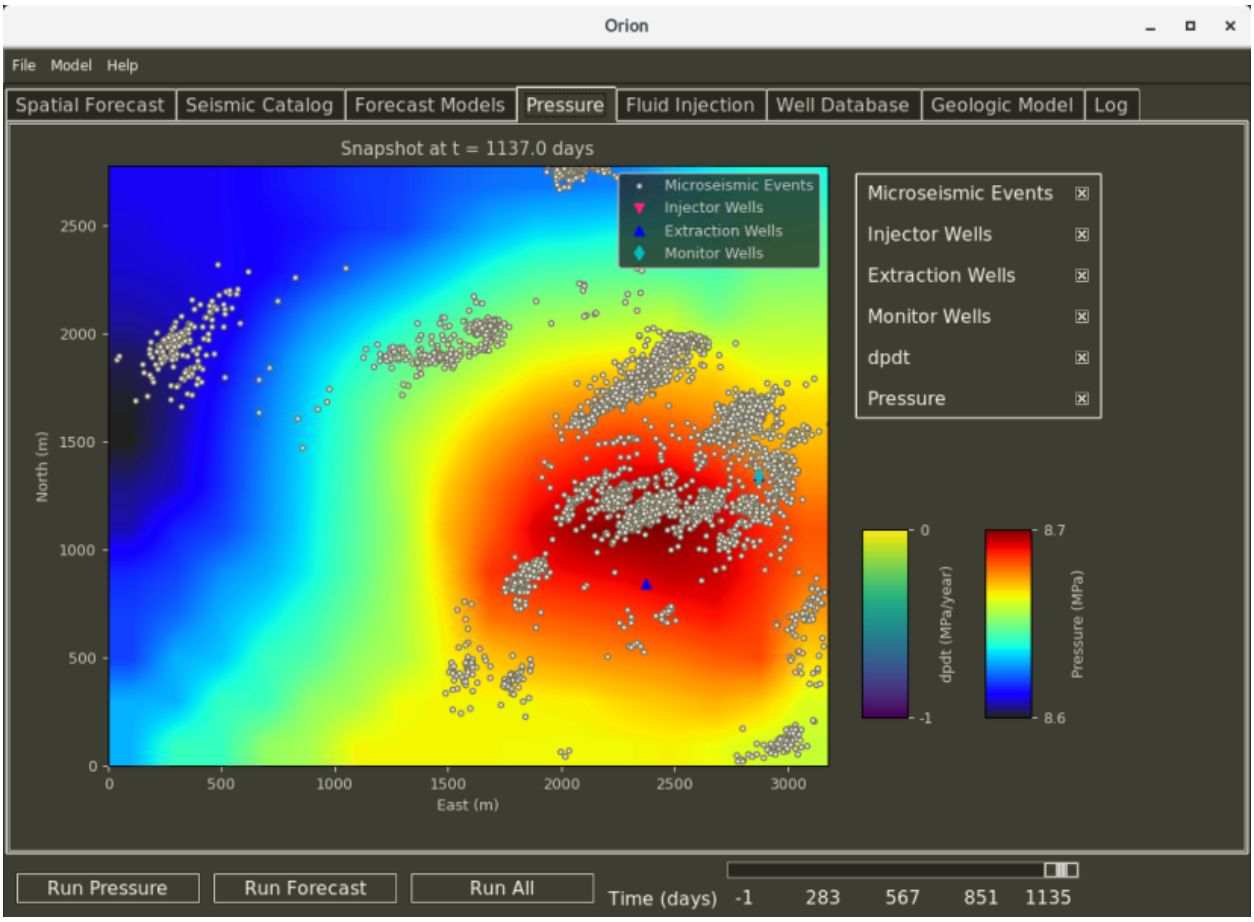
Forecast Models

The *Forecast Models* page displays the results of the seismic forecast models for the entire domain. These include:

- A bar chart showing the estimated likelihood that an event greater than a set of target magnitudes will occur during the next period of time. The magnitude distribution and time frame can be set via the *Configuration* window under *Forecast Models* (*Time range*, *Bar chart bins*)
- A dial plot showing the estimated likelihood for a target magnitude. As before, the parameters for this plot can be set via the *Configuration* window under *Forecast Models* (*Time range*, *Dial magnitude*)
- A time series plot showing the observed number of events, the estimated number of events produced by each forecast model, and an ensemble forecast. Layers can be turned on/off via the checkboxes to the right of the plot.

Pressure

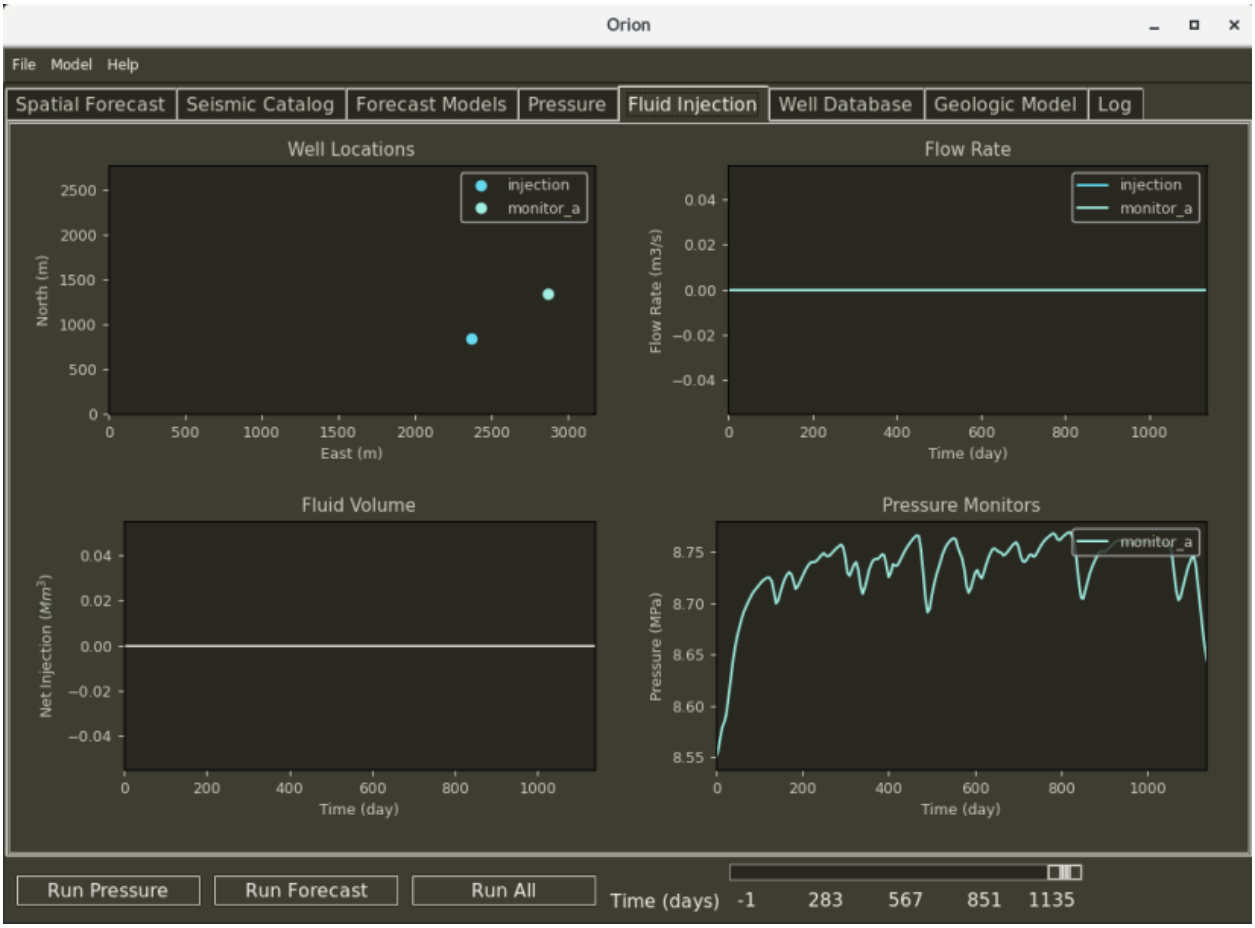
The *Pressure* page displays the results of the active fluid pressure model for the entire domain, including the current pressure and pressurization rate (dpdt). This page also displays the active well locations and the current seismic events.



Fluid Injection

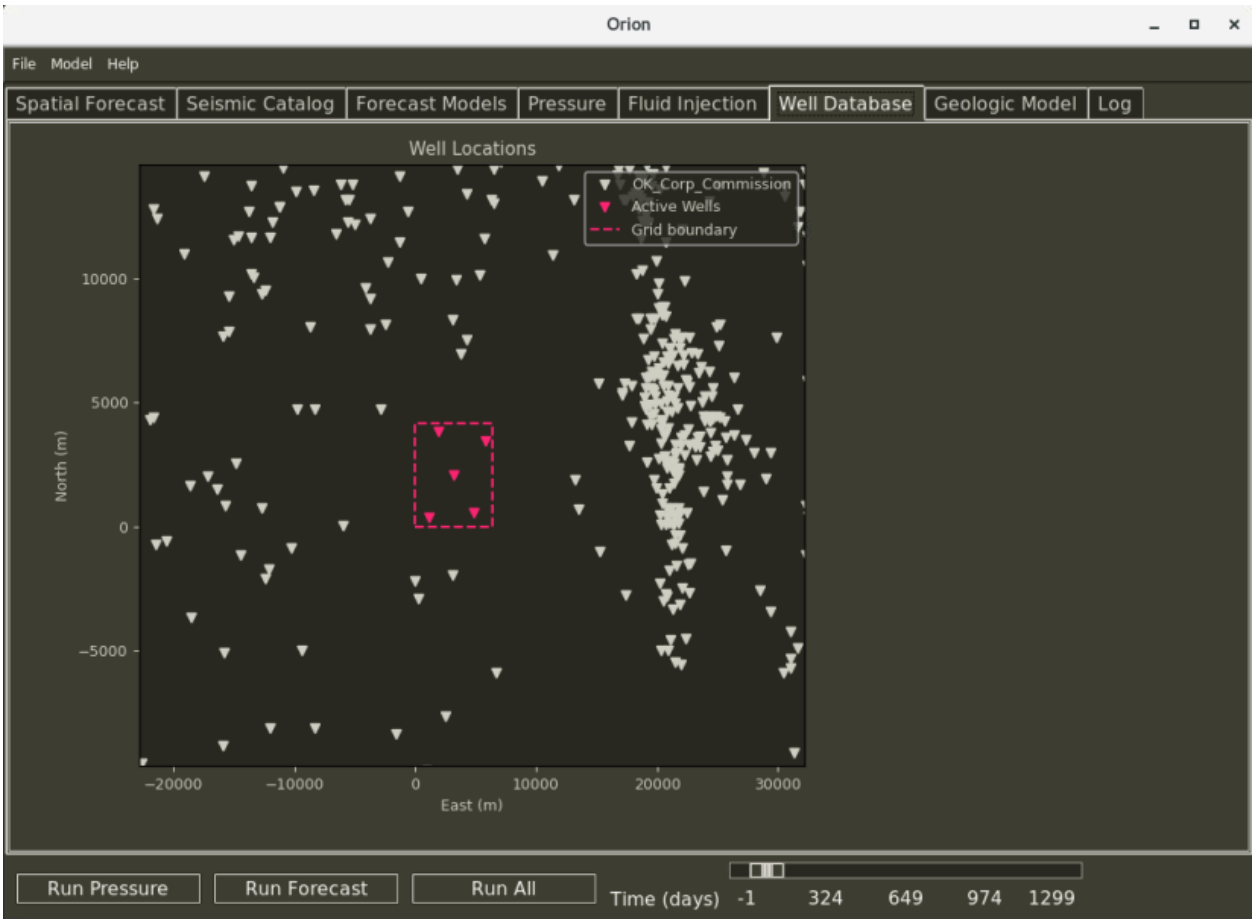
The *Fluid Injection* page includes four figures:

- Well location: The location of the active wells
- Flow rate: The fluid injection rate by well over time
- Fluid volume: The overall fluid volume injected into the reservoir over time
- Pressure: The estimated pressure at the monitor wells over time



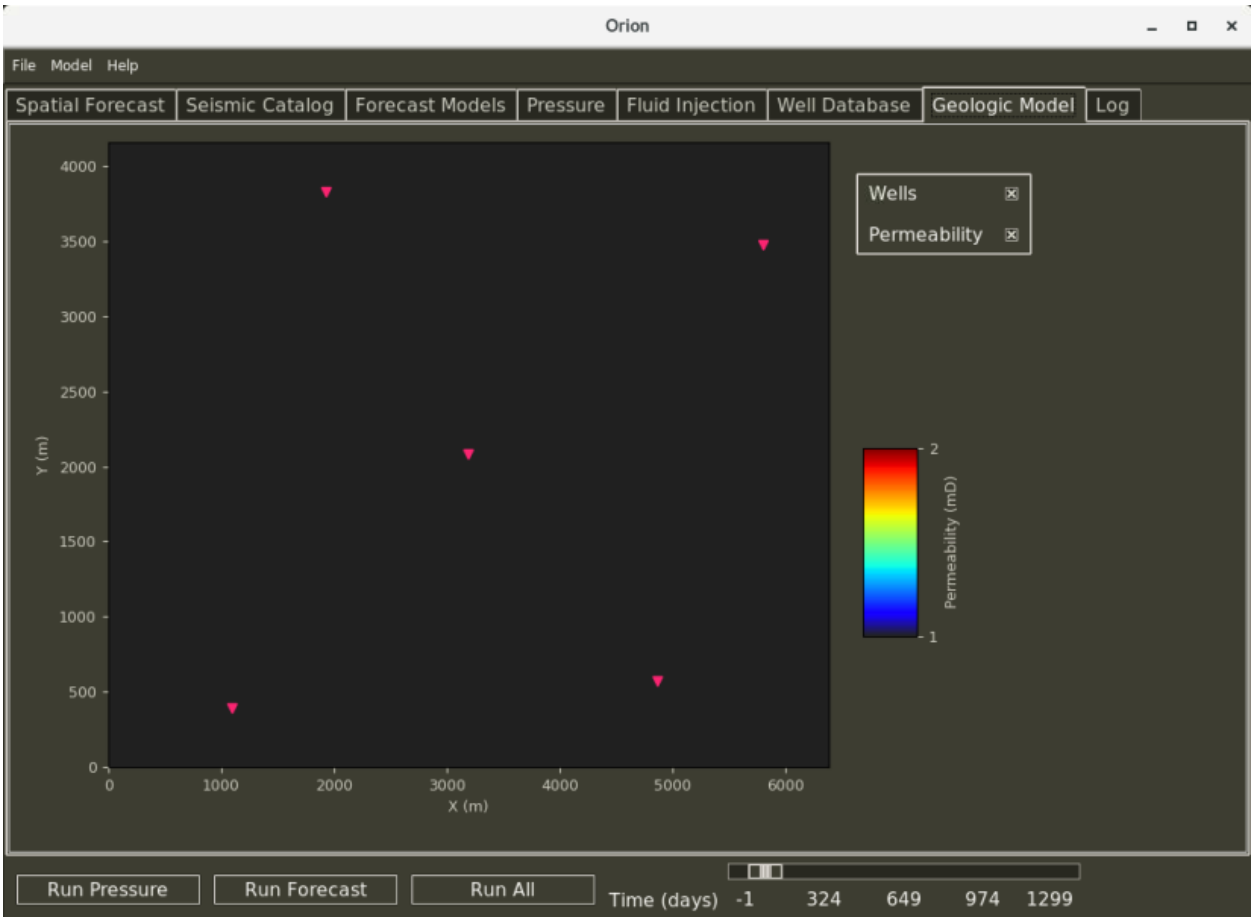
Well Database

The *Well Database* shows the location of wells pulled from external data sources. It also shows the location of active wells in the model, the grid extents, and current seismic activity. The external well database can be updated under *Well Database Configuration*.



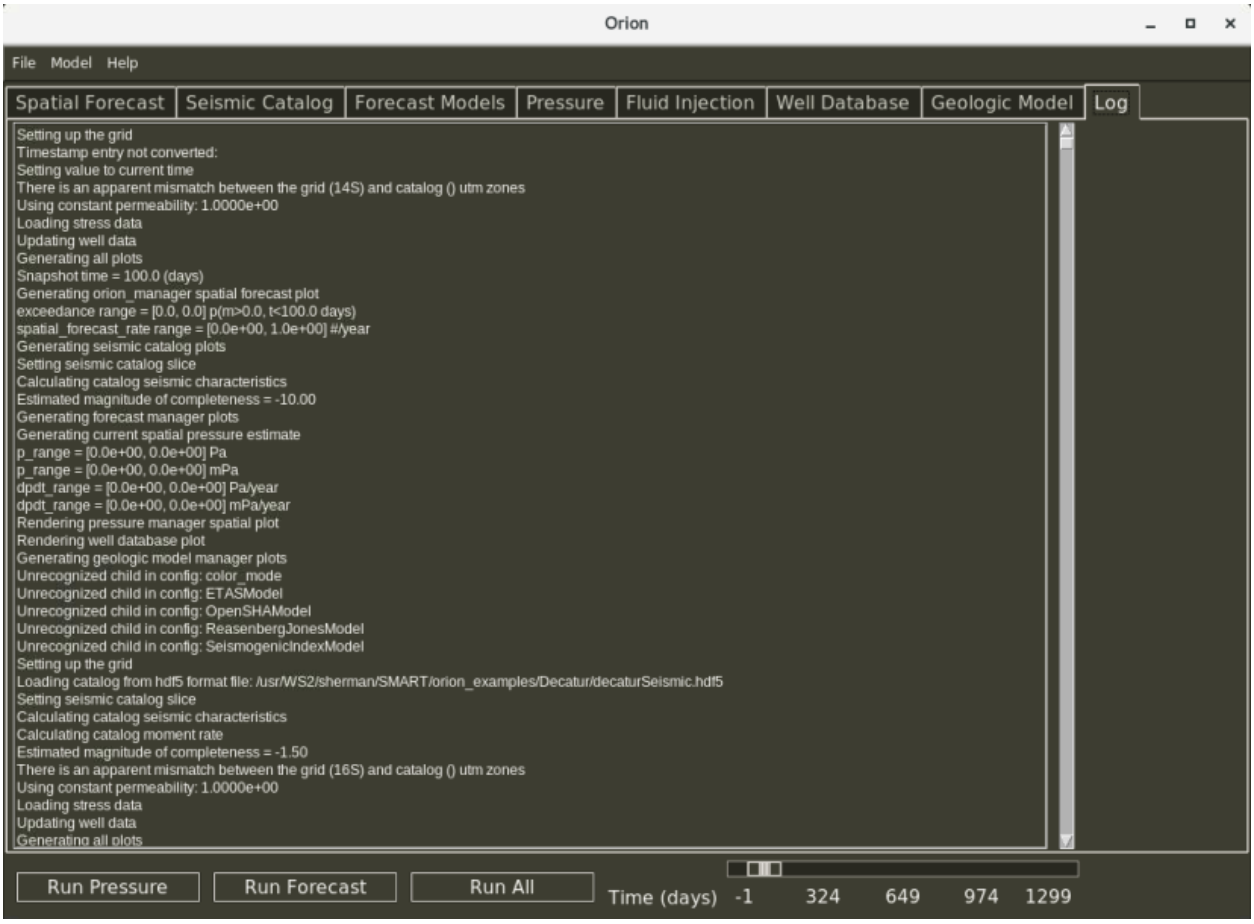
Geologic Model

The *Geologic Model* page includes other key information about the reservoir, including permeability. Note: For now, only the machine learning (ML) based models use these data.



Log

The *Log* page displays messages produced by Orion. The types of messages presented here can be set in *Model/Configure/General*.



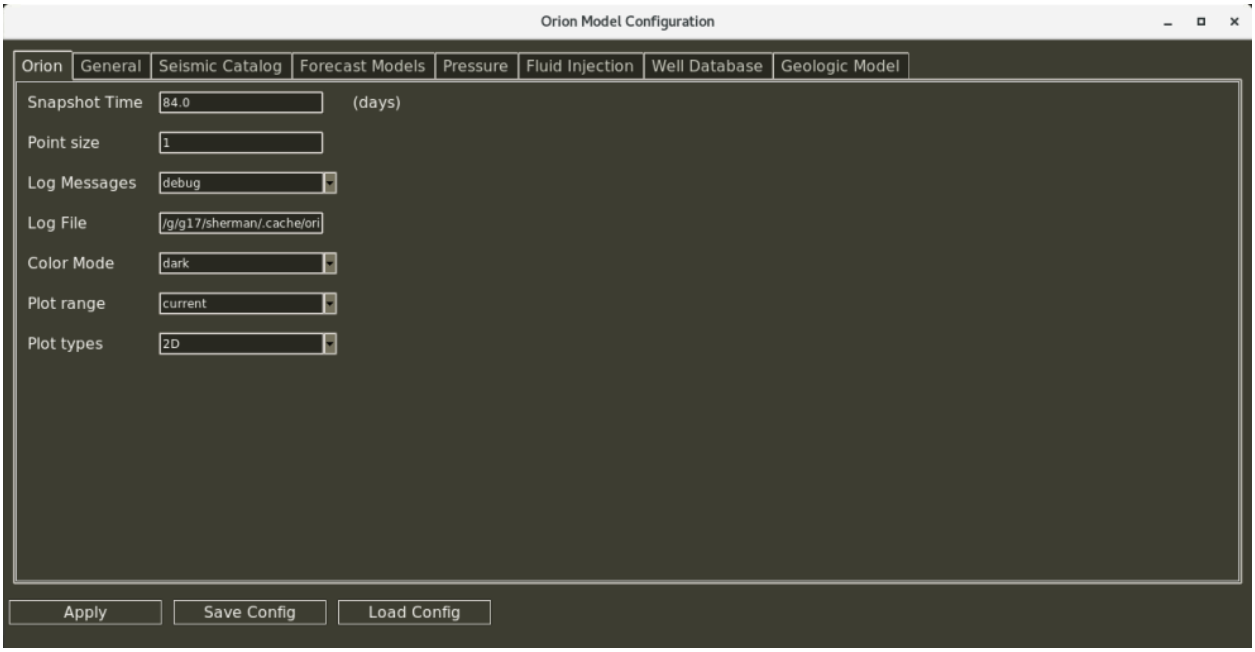
1.1.5 Code Configuration

Orion can be configured either using the GUI (recommended for new users) or via a JSON configuration file. When you exit the code or manually save, Orion will write the configuration JSON file to the user cache (*~/.cache/orion*) or to a user-requested location. When orion_forecast is called with the *–config filename.json* argument, Orion will apply this configuration after launching. If this argument is not given, and a cached configuration file is detected, then Orion will attempt to apply this file instead.

To open the GUI configuration menu, select *Menu/Configure* at the top of the main screen. Like the main Orion window, the main body of the configuration interface contains tabs, which can be selected to display key information. When the configuration window is closed, or the *Apply* button is pressed, the changes will be sent to Orion, and any active figures will be updated. The configuration can also be saved to or loaded from a file by clicking on the *Save Config* and *Load Config* buttons.

Orion Configuration

The default page for the configuration window, *Orion*, contains settings that control the behavior of Orion and the active plots. These include the snapshot time for active plots, the point size to be used for plots, the level of log messages to be produced, and the location of the log file.



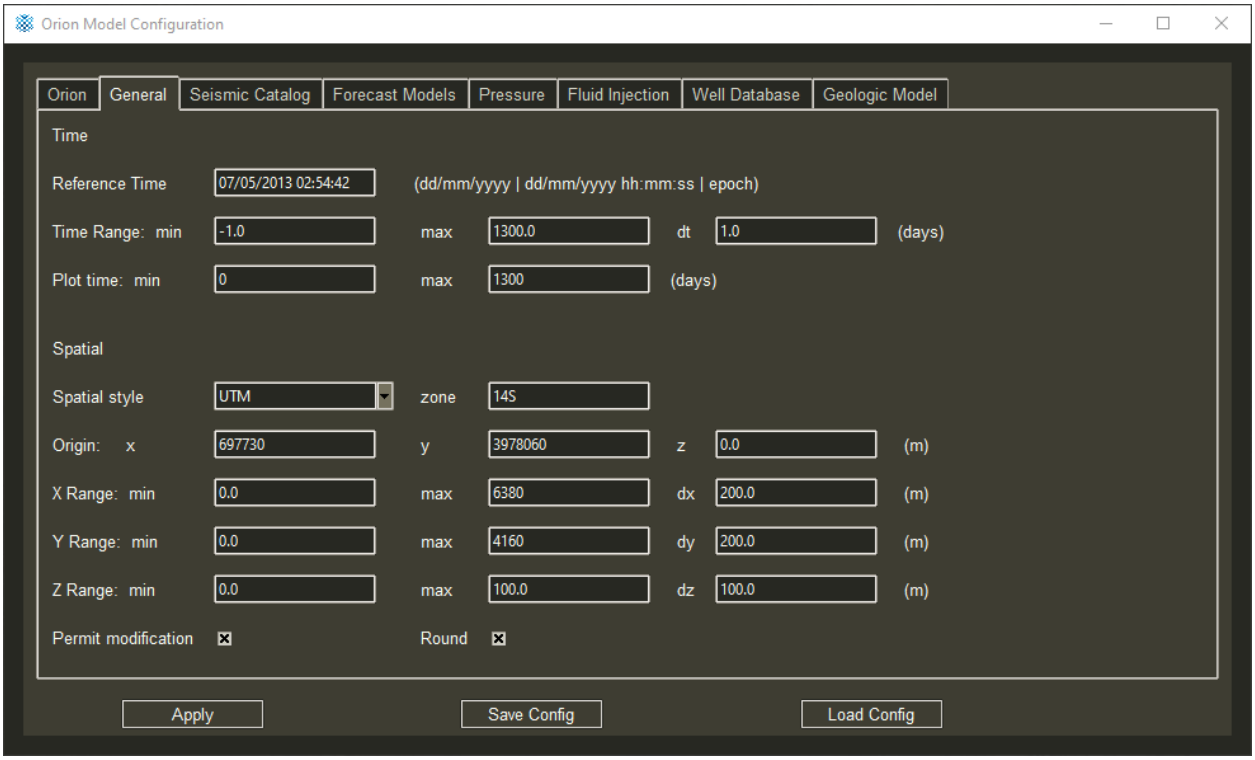
General Configuration

The general page contains settings for the reference time and spatio-temporal grid. Internally, the Orion forecasting engine works with relative timestamps, with $t = 0s$ indicating the *present* time. If the *Reference Time* parameter is left blank, then Orion will use the actual current time in its calculation. Otherwise, this parameter can be set to a value in the past, to allow for testing or *retrospective* forecasting. The *Time range* and *Plot range* parameters define the temporal grid and plot extents for the analysis, and are specified in relative time in days.

The *X Range*, *Y Range*, and *Z Range* parameters are specified relative to the *Spatial Origin* parameter. Other spatial input parameters in Orion are typically given as absolute locations in the UTM grid or in latitude/longitude.

If you are using the UTM option, you may need to set the *zone* (e.g., *14S*) when requesting catalog data from ComCat. This parameter will be automatically calculated if you are using a catalog on the local machine that includes latitude/longitude information.

Note: if the checkbox labeled *Permit Modification* is selected, Orion may attempt to modify the grid dimensions to match the extents of certain input files.

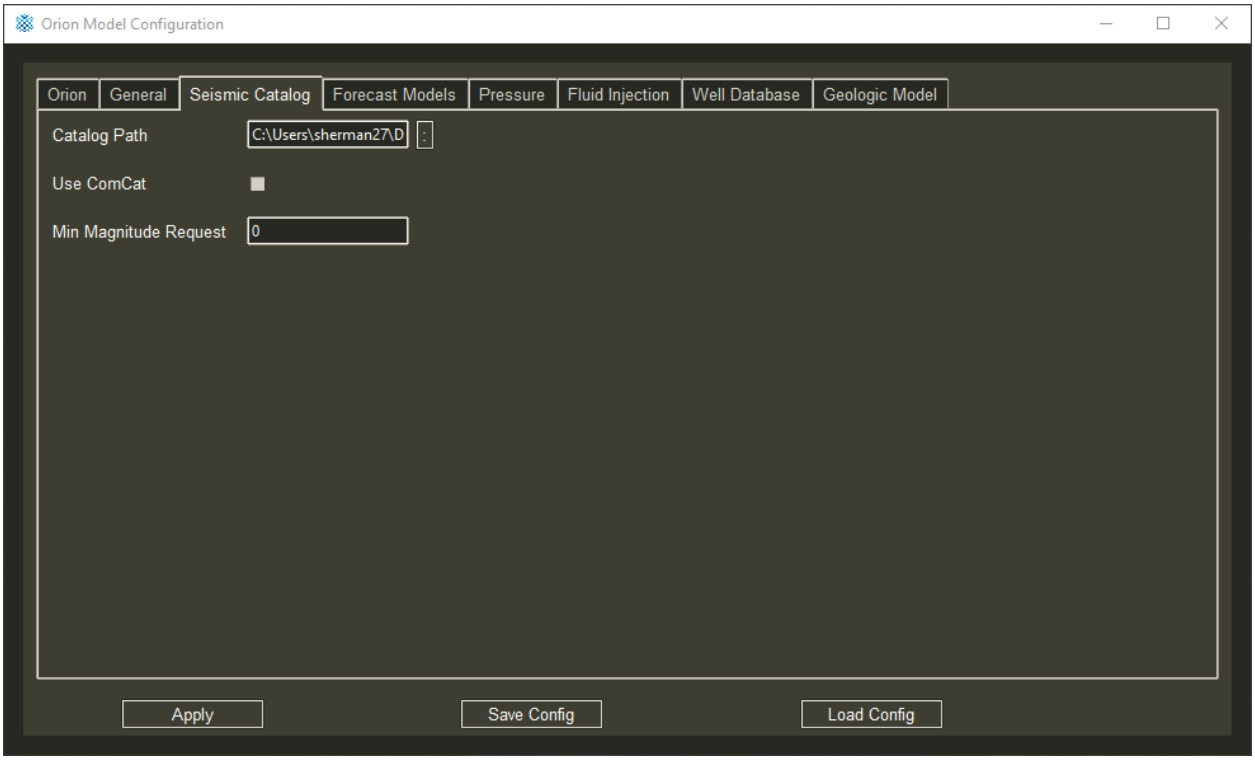


Seismic Catalog Configuration

The Seismic Catalog page contains the location of the target catalog, or instructions to fetch it.

The *Catalog Path* entry can be used to specify the path to a catalog on the local machine. Clicking on the *:* button to the right of this box will open a file explorer, which you can use to navigate to the target file. See *Seismic Catalog Files* for a description of the available formats.

If *Catalog Path* is empty and *Use ComCat* is checked, then Orion will attempt to use `pycsep`` (an optional pre-requisite) to fetch the catalog from the Internet. The bounds of this query in time and space are the same as the Orion grid (see the previous section), and the minimum catalog is set via *Min Magnitude Request*.



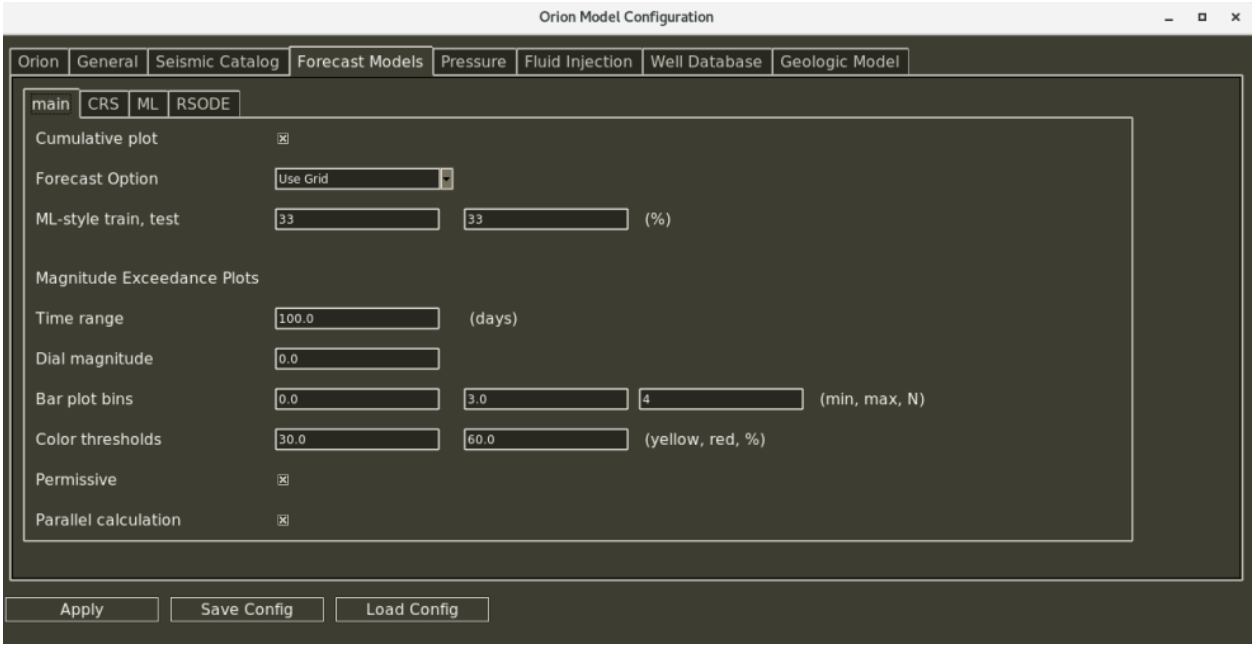
Forecast Model Configuration

The Forecast Models page contains a set of nested set of tabs, which control the general settings for forecast calculations and forecast-specific parameters. The checkbox at the top of the main page can be used to switch between cumulative and instantaneous forecast calculations. The *Forecast Calculation* dropdown box sets the ensemble forecasting method:

- Use Grid: This approach will calculate the child forecasts on the user-defined grid, and then use a linear model to fit the past or retrospective forecast to the actual data.
- ML Style: This approach will split the past data into training, testing, and validation segments similar to how ML methods are trained. It will then use a model to fit the training data, relying on the other segments to estimate the ensemble model accuracy (Note: this method is in development).

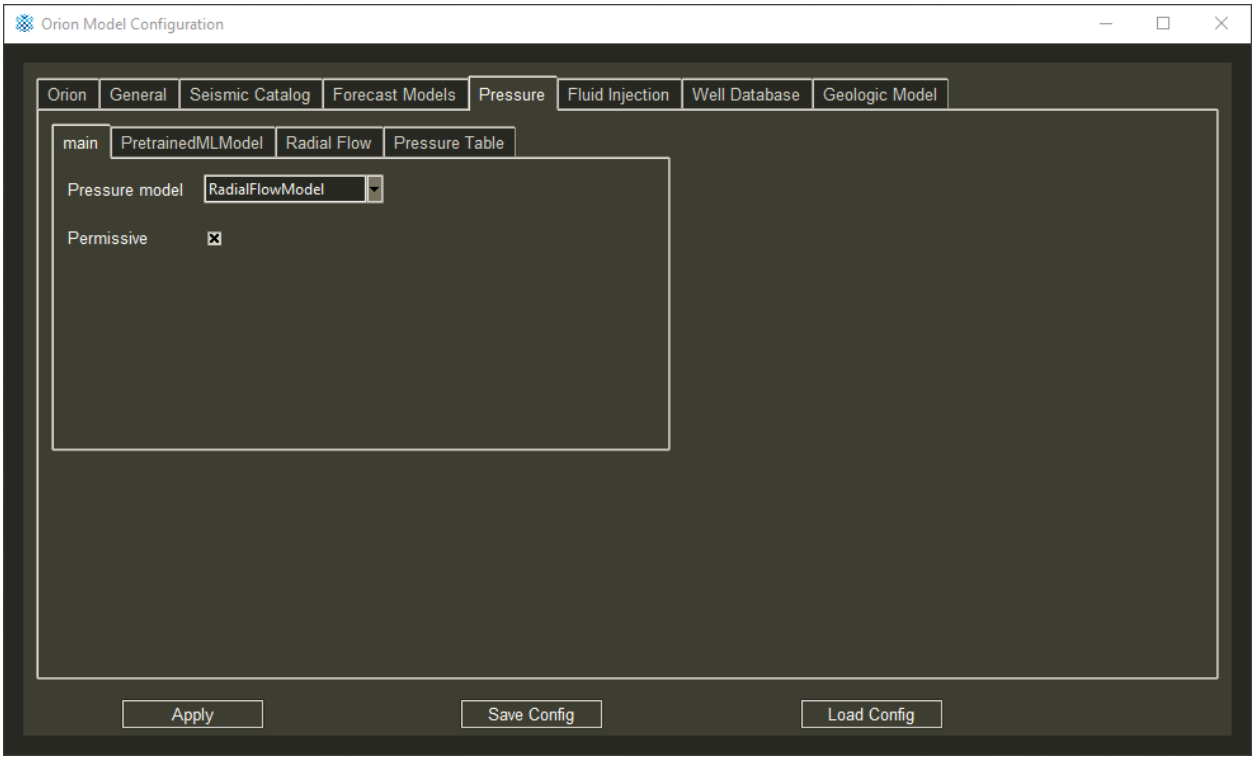
The *Magnitude Exceedance Plots* section includes settings for various probabilistic calculations, which look at the probability of a seismic event exceeding a target magnitude (*Dial Magnitude* or *Bar Chart Bins*) over the next time period (*Time Range*). Some of these plots use a stoplight coloring system, which is configured via *Color Thresholds*.

The two checkboxes at the bottom of the main page can be used to modify the forecast calculation scheme. The *Permissive* option will instruct Orion to catch any error produced by the forecasting model (otherwise the code could exit if it encounters an error). The *Parallel calculations* option will instruct Orion to calculate the active forecast models in parallel, which can help to speed up the calculation and keep the GUI responsive. Note: See the specific forecast model documentation for information on their configuration parameters.



Pressure Model Configuration

The *Pressure* page contains a set of nested set of tabs, which control the general settings for fluid pressure calculations and pressure model-specific parameters. The main page contains a dropdown box *Pressure Model* where you can select the active model to be used in the calculation. This page also includes a *Permissive* checkbox, which will instruct Orion to catch any error produced by the active pressure model (otherwise the code could exit if it encounters an error). Note: See the documentation for specific pressure models for information on their configuration parameters.



Fluid Injection Model Configuration

The *Fluid Injection* page contains a set of nested set of tabs, which hold the configuration for wells in Orion. At the bottom of the page, there are three buttons that will allow you to add a new well, remove the currently selected well, or load wells from a file.

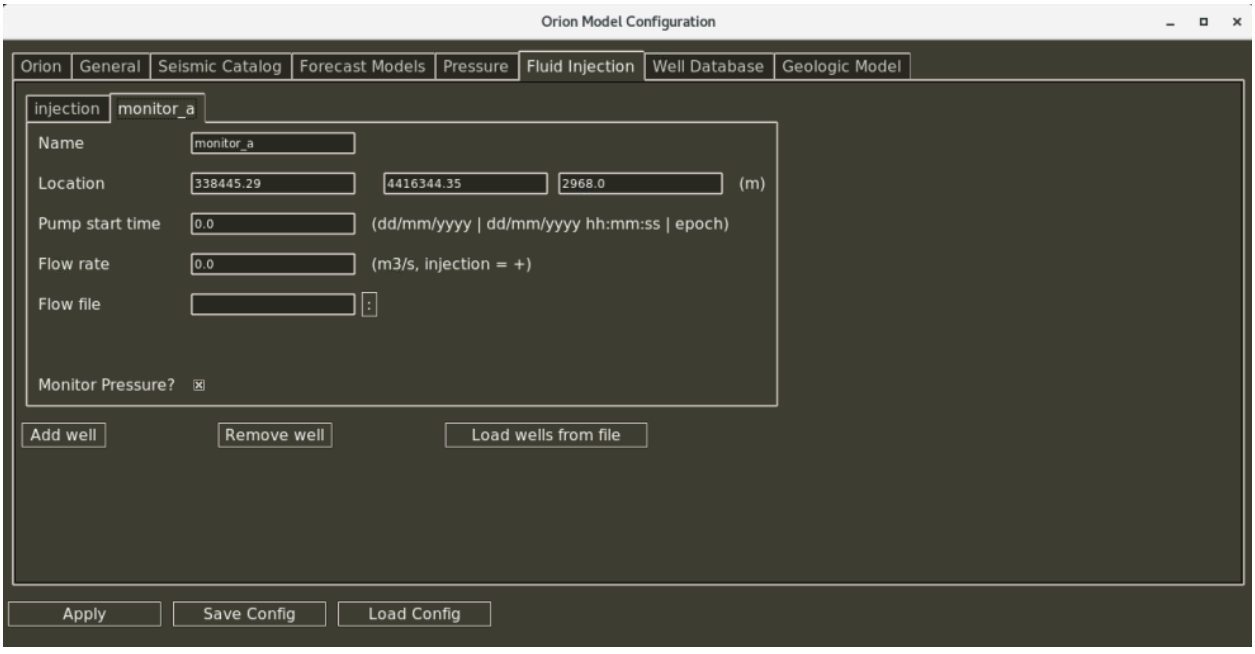
Each well tab contains space for the following information:

- *Name*: The name to be used for the well in plots
- *Location*: The location of the well in UTM (for now only vertical wells are considered)
- *Pump Start Time*: The absolute timestamp indicating when the well was turned on
- *Flow Rate*: The average flow rate for the well (with positive values indicating injection)
- *Flow File*: An optional file that contains time-varying pumping information, which will be used instead of the average flow rate
- *Monitor Pressure*: An optional flag to indicate whether the pressure should be queried at this point

The flow file is a CSV format file, which contains time and flow rate information. The headers indicate which data column corresponds to each value, and the units of each parameter (with exponents indicated via the ** operator). For example, see *orion/data/Oklahoma_Cushing2014/example_flow_file.csv* in the *orion* repository:

```
#epoch, flow_rate
#s, m**3/s
1357002061, 1.0
1357102061, 1.1
1357202061, 0.9
1357302061, 1.0
```

The bulk well file is a CSV format file, with a single-line header, and the following columns: *well_name*, *x*, *y*, *z*, *init_time*, *flow_rate*, *pressure*, *flow_file*.



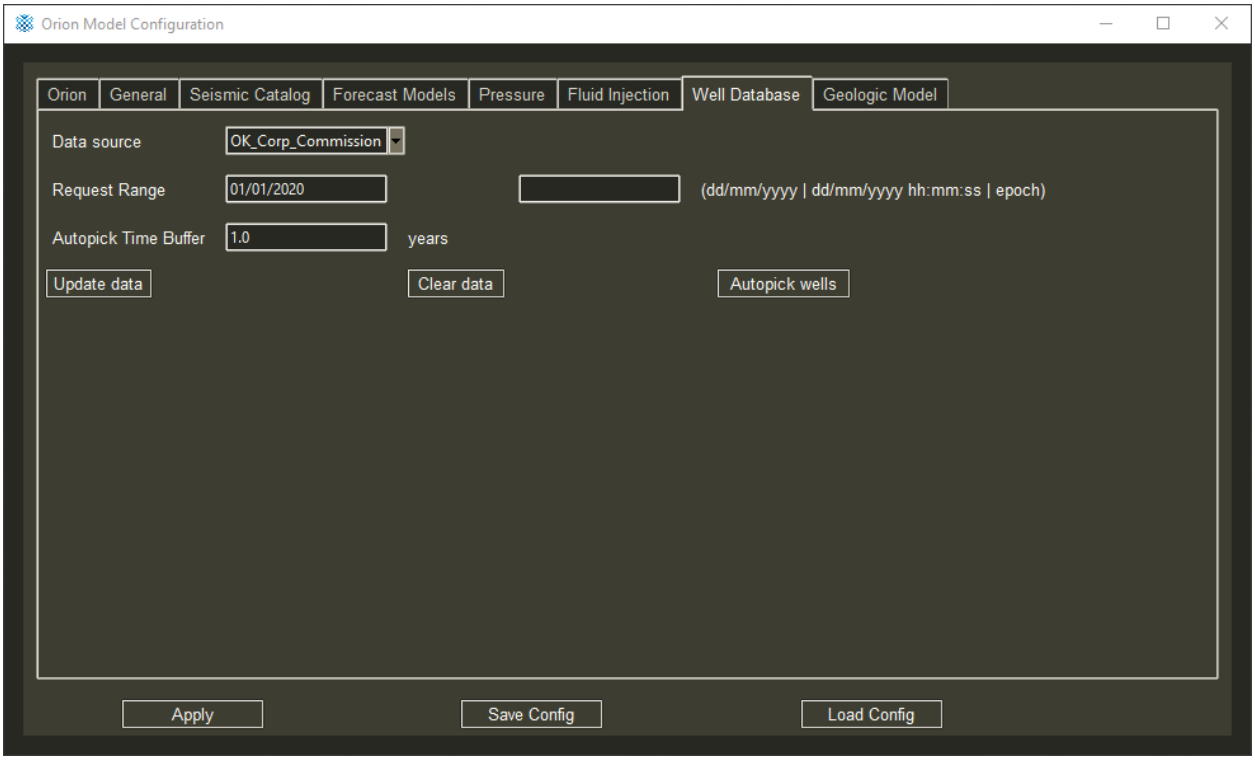
Well Database Configuration

The *Well Database* page contains options to obtain well information from external sources:

- *Data Source*: Select the source of the data (Note: only the Oklahoma Corporation Commission is currently available)
- *Request Range*: Set the start/stop time for the well data request (If either of these are empty, then Orion will choose the maximum/minimum available times for the dataset)
- *Autopick Time Buffer*: For pressure calculations, use well data this amount of time before/after the grid

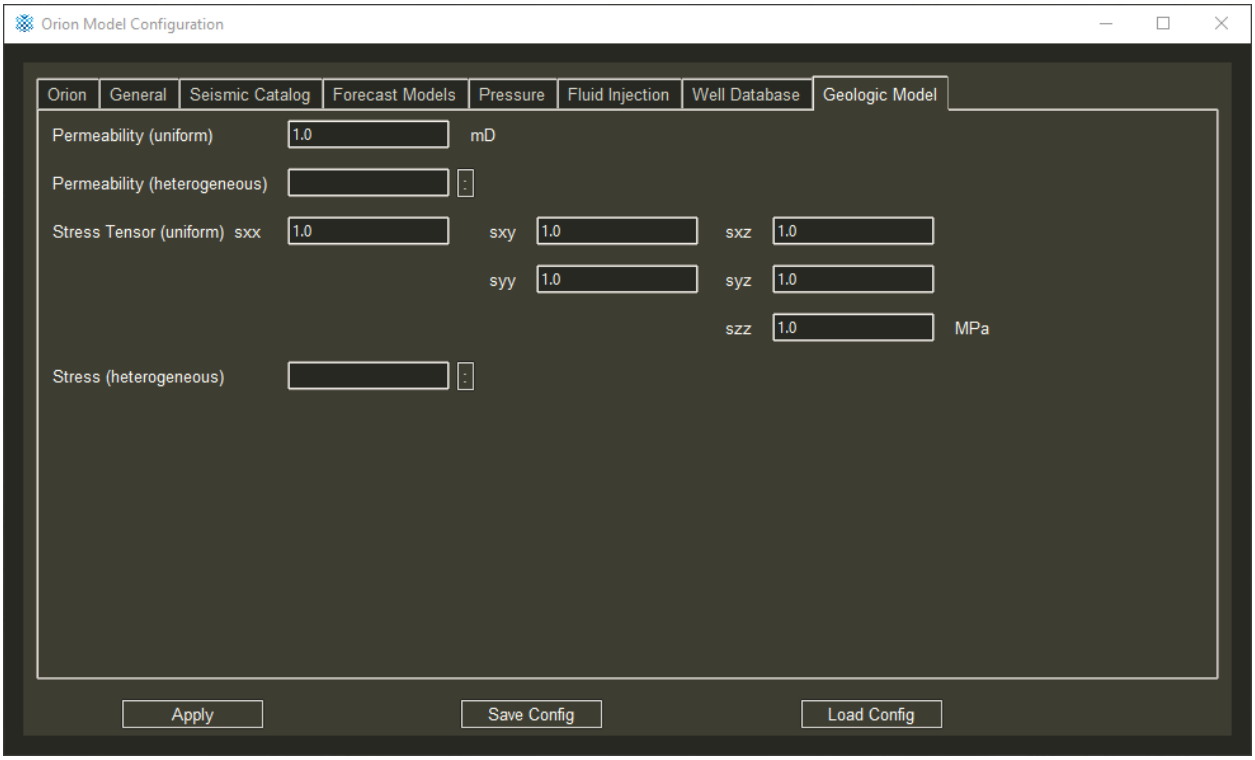
The local well database is stored in the orion cache directory (*~/.cache/orion/*) in an HDF5 format. The buttons at the bottom of the page do the following:

- *Update data*: Request well data using the current configuration and update the database
- *Clear data*: Remove any existing well data
- *Autopick wells*: Add any available wells within the grid to the pressure calculation



Geologic Model Configuration

The *Geologic Model* page contains information about the reservoir, such as permeability and in-situ stress conditions. These include options for values that are uniform across the reservoir, and those that are heterogeneous (see *Table Files*). Note: If a heterogeneous parameter file is specified, these values will override the uniform values.



1.1.6 Pressure Models

The following fluid pressure models are implemented in Orion:

Radial Flow Pressure Model

The radial flow pressure model is based on the transient Theis solution for an infinite, homogeneous reservoir [Ferris et al., 1962]. The solution relies on the following parameters:

- Fluid viscosity, μ
- Fluid permeability, k
- Reservoir storativity, s
- Reservoir thickness, H
- Well locations, x_i, y_i, z_i
- Well pumping start time, t_i
- Well pumping rate, q_i

To estimate the pressure and pressurization rate at points in the space (x, y, z) and time (t), we calculate the transmissivity of the reservoir, T:

$$T = \frac{\rho g k H}{mu}$$

where ρ is the fluid density and g is the gravitational coefficient. The Theis solution is 2D, so the distance from each well to the current point, r_i, is given by:

$$r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$$

The dimensionless well parameter, u_i, is given by:

$$u_i = \frac{r_i^2 s}{4T(t - t_i)}$$

The fluid pressure, p, is given by the superposition of each well contribution:

$$p = \frac{\rho g}{4piT} \left(z + \sum_{i=1}^{N} q_i W(u_i) \right)$$

where N is the number of wells and W is the exponential integral (E_1). Similarly, the pressurization rate $\frac{\partial p}{\partial t}$ is given by:

$$\frac{\partial p}{\partial t} = \frac{\rho g}{4piT} \sum_{i=1}^{N} \frac{q_i}{(t - t_i)} exp(-u_i)$$

For wells that have time-varying flow rates, the well contributions are estimated by separating the time-series into a set of step-wise functions:

$$q_i = q(t_i) - q(t_{i-1})$$

The following image shows the configuration window for the radial flow model:

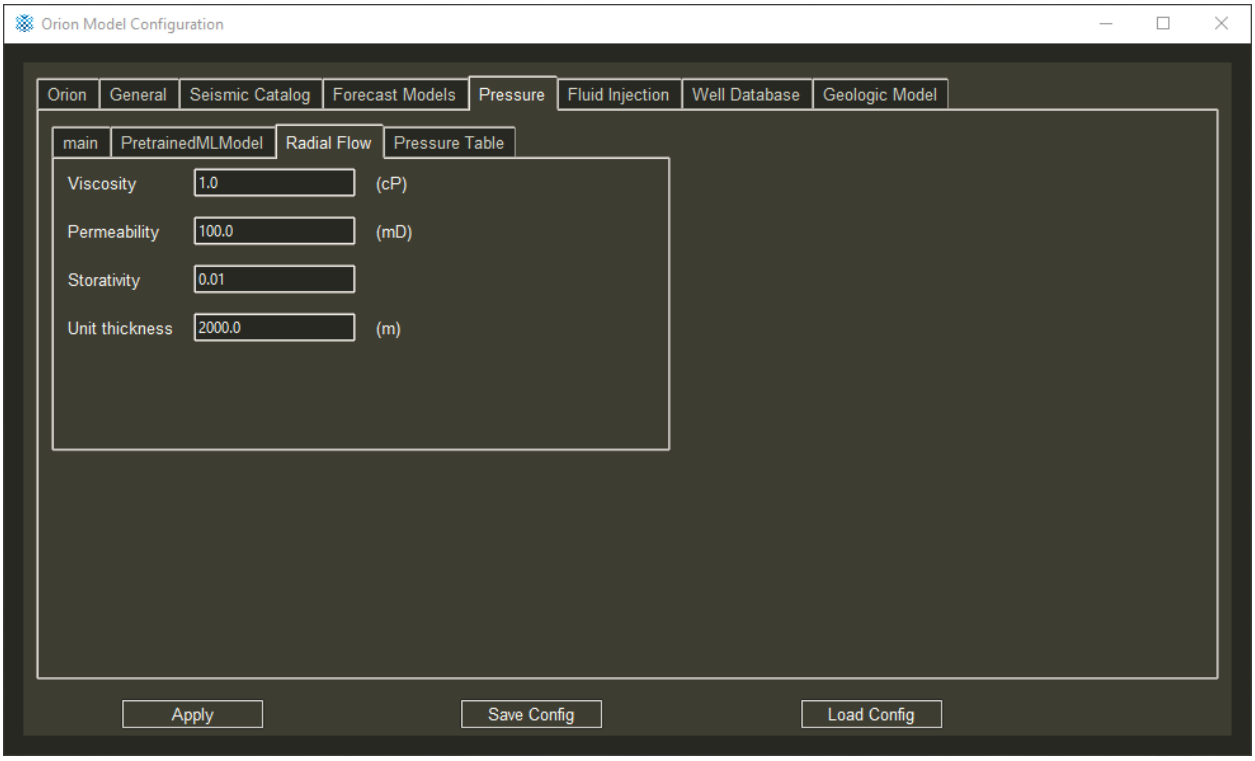
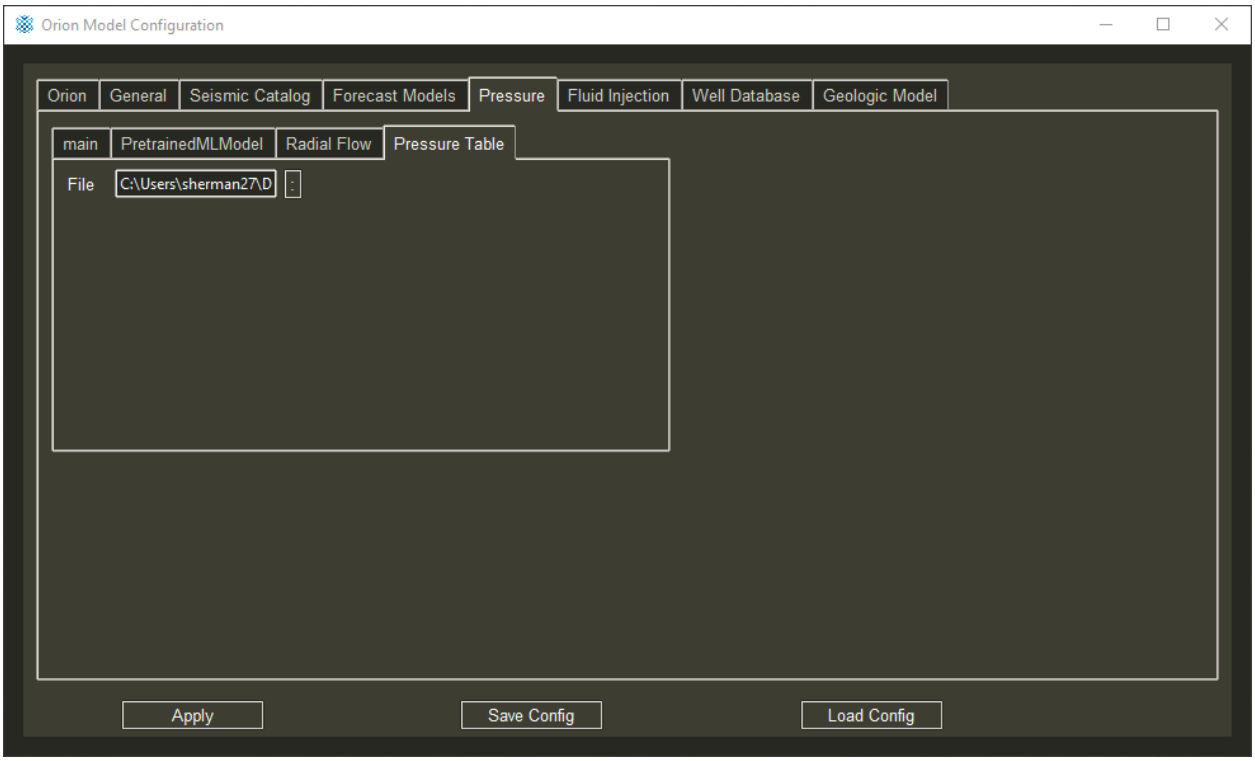


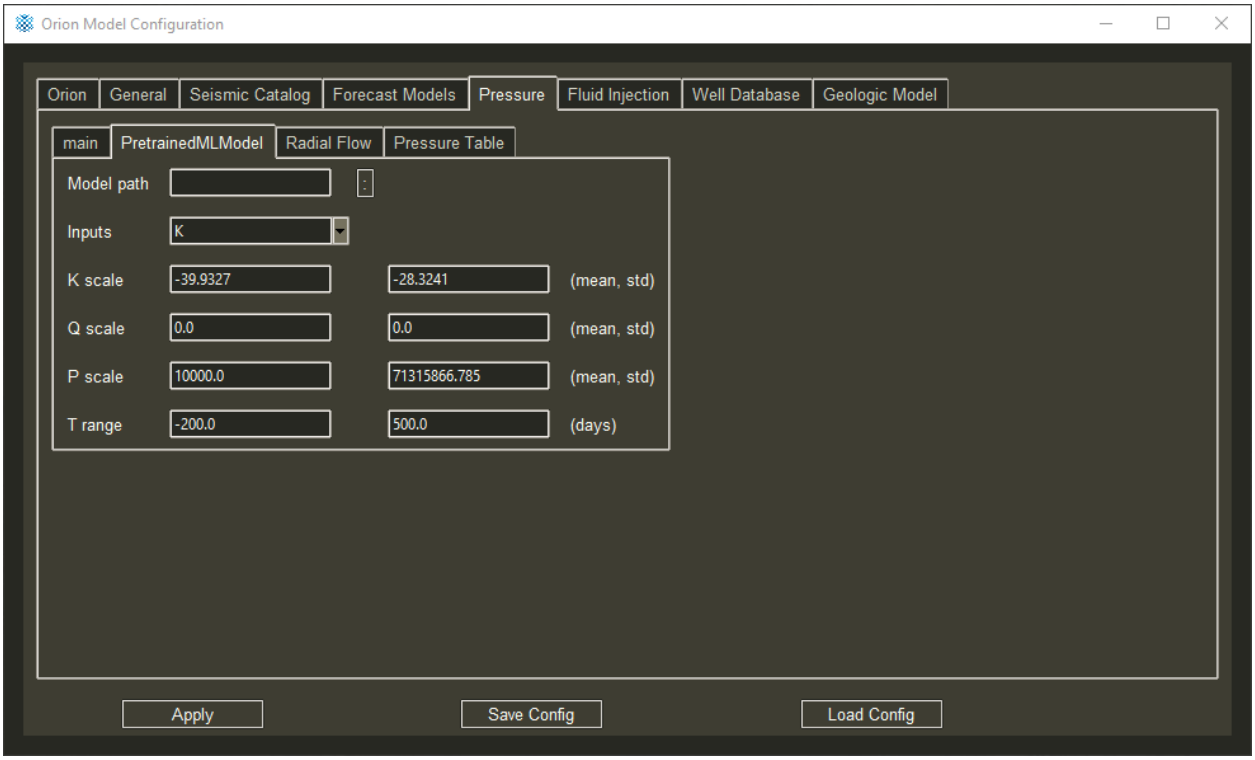
Table Pressure Model

The table-based pressure model is used to import an external pressure model. In the configuration menu, Orion requires the path to a table file (see *Table Files*) on the local machine. The table can be structured (a 4D table with axes corresponding to x, y, z, t) or unstructured (sets of 1D arrays of x, y, z, t), and should contain pressure and/or dpdt estimates. If pressure or dpdt are missing from the tables, Orion will attempt to integrate/differentiate the available data to estimate them from the remaining values.



Pretrained ML Pressure Model

The pretrained ML pressure model is currently designed to work with a 2D ML model developed by Hewei Tang at LLNL. This model requires estimates of permeability, K, across the reservoir and a set of scaling parameters. Note: The model path can either point to a HDF5 format model or a ZIP file containing the model directory (required for some models with custom layers).



1.1.7 Seismic Forecast Models

The following seismic forecast models are implemented in Orion:

Coupled Coulomb Rate-State

The coupled Coulomb Rate-State (CRS) earthquake rate equations describe the evolution of seismicity rate as a function of stressing history [Dieterich, 1994, Dieterich, 2007, Ferris et al., 1962, Hager et al., 2021]. The number of events is simply found by integrating the rates. The solutions for the number of events are listed below the rate solutions for each set of equations.

Input Parameters

The relevant parameters for the CRS model are as follows:

- Nominal coefficient of friction, μ_0
- Biot coefficient, b
- Tectonic normal stressing, σ (MPa)
- Tectonic normal stressing rate, $\dot{\sigma}$ (MPa/year)
- Tectonic shear stressing rate, $\dot{\tau}$ (MPa/year)
- Normal stress dependency, α
- Background seismicity rate, r (events/years)
- Seismicity rate scaling factor, RF (1/MPa)

- Enable clustering effects, (yes/no) [Kroll et al., 2017]

Input Parameter Selection

- Nominal coefficient of friction: Laboratory values of friction for intact rock range from 0 to 1, default values are on the order of 0.6
- Biot coefficient: Value should be consistent with the one used in the geomechanical simulations for computing pressure and/or stress
- Tectonic normal stress: Derived from lithostatic stress at average hypocentral earthquake depth. Default is 30 MPa
- Tectonic normal stressing rate: Estimated based on geodetic surveys and deformation modeling. Default value = 0 MPa/year
- Tectonic shear stressing rate, Estimated based on geodetic surveys and deformation modeling. Default value = 0.00035 MPa/year for mid-continent US
- Normal stress dependency: On the order of 0 to the nominal coefficient of friction
- Background seismicity rate: Estimated from historical seismicity catalogs over time periods with stable magnitude of completeness
- Seismicity rate scaling factor: scaling factor that relates the forecasted number of events to the observed number of events by scaling the reference stressing rate $\dot{S}_r$. In future versions, this parameter will be included in the parameter optimization
- Enable clustering effects, (yes/no): Click yes, when significant mainshock/aftershock sequences are present in the observed seismicity data to enable enhanced parameter fitting.

The instantaneous earthquake rate due to a step change in Coulomb failure stress (CFS) is given by,

$$R = R_0 \exp\left(\frac{\Delta CFS}{a\sigma}\right)$$

where R_0 is the steady-state background seismicity rate before the stress step, a is the rate-state constitutive parameter, and σ is the normal stress. The earthquake rate between stress steps, assuming a constant stressing rate is given by:

$$R(t) = \frac{1}{\frac{\eta}{\dot{S}} + \left(\frac{1}{r} - \frac{\eta}{\dot{S}}\right)\exp\left(\frac{-\dot{S}t}{a\sigma}\right)}$$

$$N(t) = \frac{a\sigma}{\eta}\ln\left[\left(1 + \frac{R_0\eta}{\dot{S}}\right) + \frac{R_0\eta}{\dot{S}}\exp\left(\frac{\dot{S}t}{a\sigma}\right)\right]$$

where η is the ratio of the reference stressing rate, $\dot{S}_r$ to the reference background seismicity rate, r (i.e., $\eta = \dot{S}_r/r$).

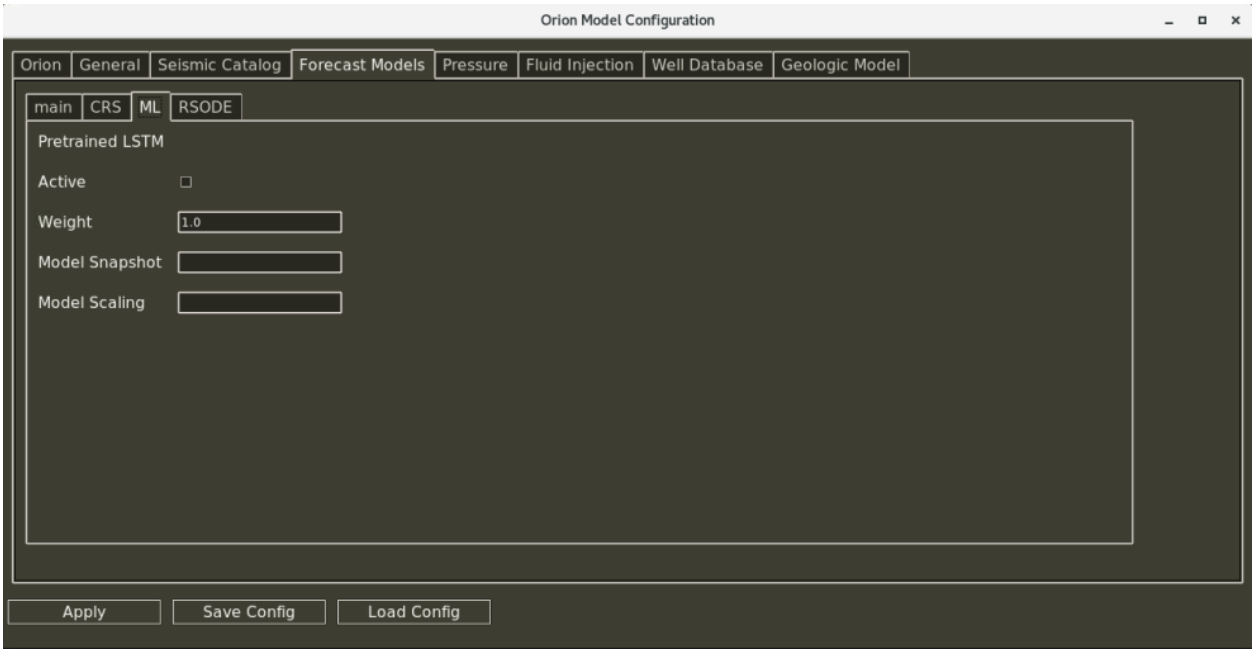
Computing the earthquake rate forecast is quite simple, assuming the pore-pressure and stresssing rate history due to injection is discrete.

- Step 1: Compute the rate at the time of the stress step with Equation 1 of this section
- Step 2: Allow the earthquake rate to evolve with time between the stress steps with Equation 2 of this section
- Step 3: Iterate for the duration of the pressure/stress change time series

Machine learning and numerical optimization techniques are used to solve for the best-fitting parameter values. The free parameters include the ΔCFS, a, σ, r, and $\dot{S}_r$. However, we can restrict the range of the values of these parameters, or allow only a subset of this list to vary during the optimization process. For example, if we believe that our estimate of the pressure change is accurate, this parameter can be held fixed, while we optimize the solution by varying the other parameters.

Pretrained ML Model

Note: this forecast model is under construction, and will be updated in the future.



Rate-and-State ODE

The rate-and-state earthquake nucleation model estimates the number of independent events in response to a change in stress and can be described by the following ordinary differential equation [Dieterich, 1994, Segall & Lu, 2015]:

$$\frac{dR}{dt} = \frac{R}{t_c}\left(\frac{\dot{\tau}}{\dot{\tau_0}} - R\right)$$

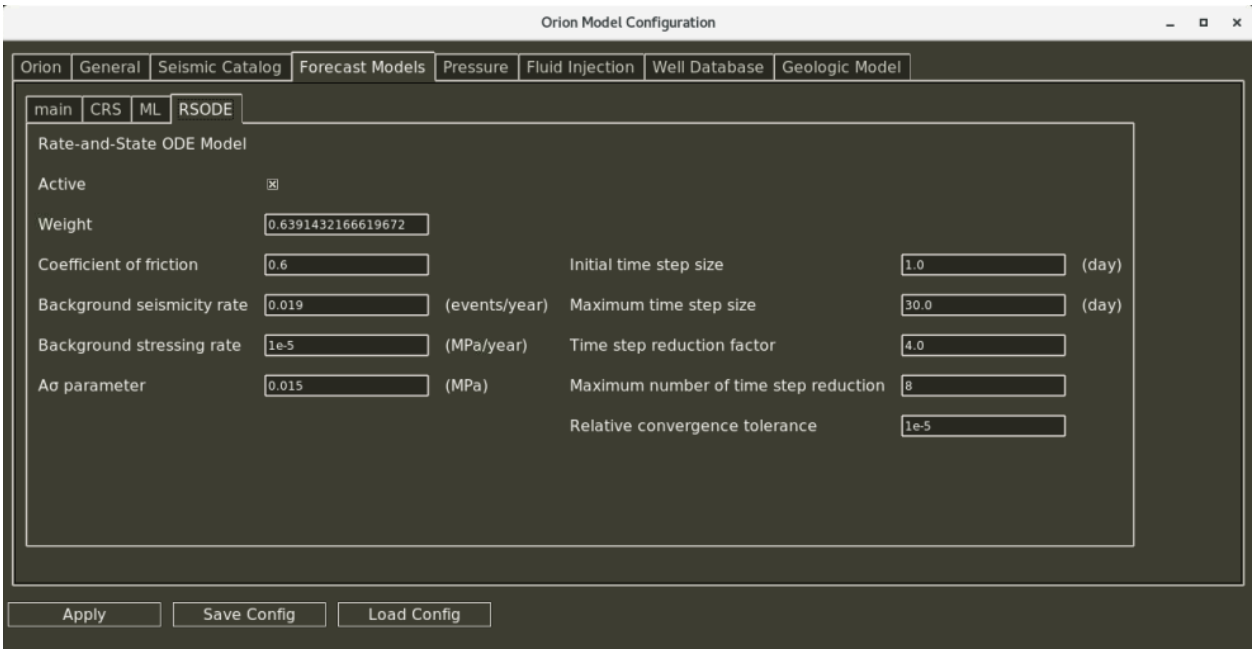
where R is the ratio between the seismicity rate relative to the background rate, $t_c = \frac{A\sigma}{\dot{\tau_0}}$ is the characteristic relaxation time, $\dot{\tau}$ and $\dot{\tau_0}$ are the Coulomb and background stressing rates, respectively. The ordinary differential equation is solved using a fifth order adaptive time step Runge-Kutta-Fehlberg algorithm [Fehlberg, 1969].

Model parameters

- Coefficient of friction, μ
- Background seismicity rate, r_0 (events/year)
- Background stress rate, $\dot{\tau_0}$ (MPa/year)
- Free parameter, $A\sigma$ (MPa)

Solver parameters

- Initial time step size (day): step size for the first iteration. The step size will be either increased or decreased depending on the convergence at the current time step
- Maximum time step size (day): maximum allowed step size, the step size can not be larger than this value
- Time step reduction factor: factor by which the step size is reduced after convergence failure at the current time step
- Maximum number of time step reduction: maximum number of consecutive time step reductions, the solver is stopped if this limit is reached. In this case, the user can either decrease the initial time step size or time step reduction factor, or increase this limit
- Relative convergence tolerance: convergence criterion



1.1.8 Ensemble Forecast Calculation

The performance of individual seismic forecasting method can vary depending on site conditions and/or user configuration. To address this, Orion generates an 'ensemble' or best-guess forecast using one of the following methods:

- Linear Grid: This approach uses a small period of test data in the user-defined grid to estimate the optimal weight and bias for active forecast models. These resultant weights can then be inspected and tuned in the configuration menu.
- ML Style: This method is under development. Our goal is to take a model that is pre-trained on historical data to produce the ensemble forecast. If desired, this model can then be updated to better match site conditions using transfer learning.

1.1.9 Frequently Asked Questions

Please send any question to the Orion development team.

1.2 Examples

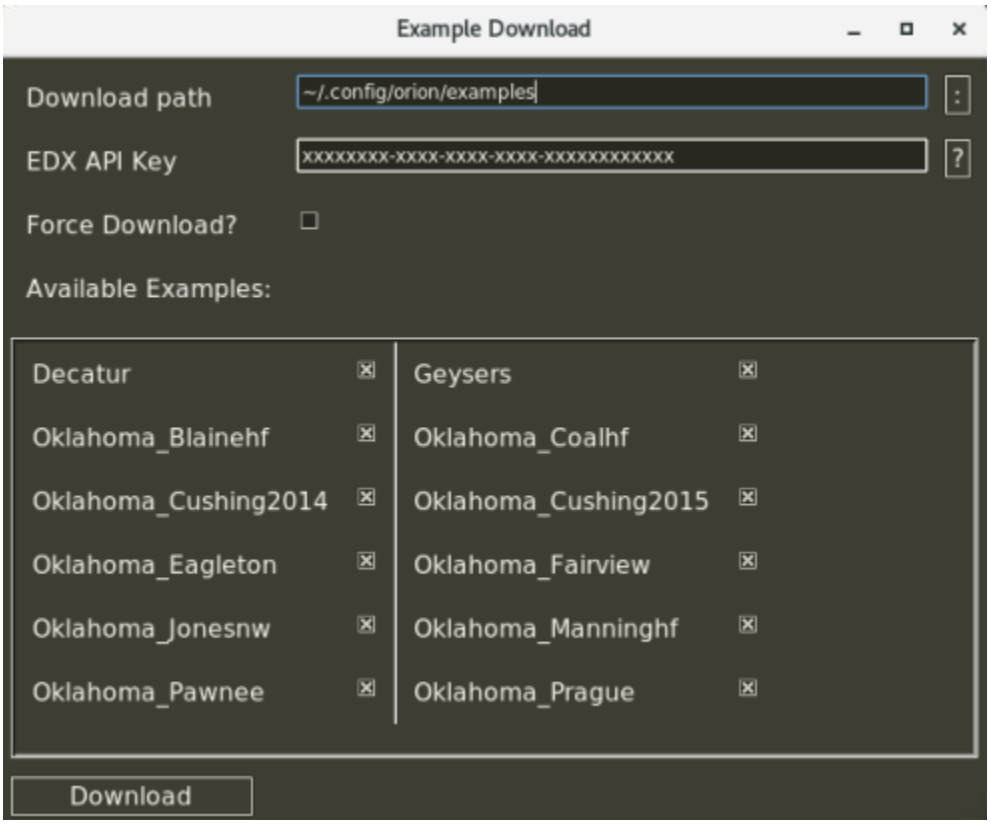
Example data and configurations for Orion are available on the Energy Data eXchange (EDX). Available examples can be loaded by selecting them from the *Model/Examples* dropdown menu.

1.2.1 How to Download Examples

Please note: You will need to have an account on EDX and be a member of the Orion workspace to download examples . To request membership, please contact a member of the Orion development team. Next, you will need to locate your EDX API key by clicking your username in the top-right corner of EDX, and copying the value in EDX-API-KEY. This key is private to your account, so please do not share it with others.

You can bring up the Example Selection menu in Orion by selecting *Model/Download Examples*. This will open up a new window where you can select the location to download examples, enter your EDX API key, and select which examples you would like to download. When ready, you can download the targeted examples by clicking the button at the bottom of the window labeled *Download Examples*. Once complete, the new examples should appear up in the *Model/Examples* dropdown menu Orion will skip downloading any examples that are already present in the target directory unless the *Force Download?* option is selected.

Orion will attempt to remember the example download configuration, and resume where you left it. Like other orion configuration data, these values are stored in the cache directory (default: *~/.cache/orion*). Please note, that if you move the downloaded examples on the local machine or change the download directory, Orion may not be able to find them. If this occurs, try updating the download path in the Example Download window.



1.2.2 Example Format

Orion examples files on EDX use a zip format, with top-level folders corresponding to individual scenarios. Each scenario folder should contain a *config.json* file and any supplementary data files required to run the problem. Alternate configuration files named *config[alternate].json* can be included, and will be appear in the *Examples* menu as *[folder_name]_[alternate]*. Please note: these configuration files use a special keyword *[BUILT_IN_PATH]* to indicate the path of the scenario folder on the local machine.

The active Orion configuration can be saved as a zip-format example by selecting **Model/Save config to file** from the main window, and selecting an output file with a .zip extension. As part of this process, Orion will attempt to collect any required data for the example, and then construct an appropriate configuration file.

1.3 Data Formats

1.3.1 Seismic Catalog Files

Orion can read from a variety of seismic catalog formats, both local and from remote sources such as ComCat. For locally hosted catalogs, we prefer that data be in one of the following csv or hdf5 formats:

csv Catalog Format

csv format catalog files are comma-delimited text files with 1-2 header lines, depending upon their content. Data columns can be placed in any order, and must include epoch, magnitude, depth, and either latitude/longitude or easting/northing. The expected units for these values are seconds for epoch, and meters for depth, easting, and northing. For catalogs that contain easting/northing information, you can supply the utm zone information in the first line of the header; otherwise, Orion will assume that the coordinate system is local.

```
utm_zone,6SU
epoch,magnitude,depth,easting,northing
1367956482,1.6,3.308,703178,3981454
1367958659,1.43,3.4,703150,3981474
```

```
epoch,magnitude,longitude,latitude,depth
1367956482,1.6,-96.74708,35.95636,330
1367958659,1.43,-96.74707,35.95634,340
```

hdf5 Catalog Format

hdf5 format catalog files are expected to have entries for each of the catalog features on the root level. Similar to the csv format, the catalog mush include epoch, magnitude, depth, and either latitude/longitude or easting/northing. The expected units for these values are seconds for epoch, and meters for depth, easting, and northing. For catalogs that contain easting/northing information, the file can also contain an entry for the utm_zone string; otherwise, Orion will assume that the coordinate system is local.

```
{
    'epoch': [1367956482, 1367958659],
    'magnitude': [1.6, 1.43],
    'depth': [330, 348],
    'easting': [703178, 703150],
```

(continues on next page)

(continued from previous page)

```
    'northing': [3981454, 3981474]
}
```

1.3.2 Table Files

Some inputs to Orion use a CSV or HDF5 format that can contain structured or unstructured data:

- Structured: This file/folder is expected to contain children named *x*, *y*, *z*, and/or *t*, which are 1D arrays and define the dimensions of the grid. This file/folder also contains children that contain ND arrays, which match the size of the target grid. Typically, the expected grid order is *xyz*, *xyzt*, or *t*.
- Unstructured: This file/folder is expected to contain children named *x*, *y*, *z*, and/or *t*, which are 1D arrays and define the points in the dataset. This file also contains children that are 1D arrays of the same length, which define properties at the target points.

Note: When constructing these files, you may want to use the HDF5 wrapper in *orion.utilities.hdf5_wrapper*, which simplifies working with these files. The following example shows how to write a simple, structured 3D file in the expected format:

```
import numpy as np
from orion.utilities import hdf5_wrapper

# Setup the data
x = np.linspace(0, 1, 10)
y = np.linspace(0, 2, 20)
z = np.linspace(0, 3, 30)
X, Y, Z = np.meshgrid(x, y, z, indexing='ij')
p = 2 * X + 3 * Y + 4 * Z

# Save to an hdf5 format file
with hdf5_wrapper.hdf5_wrapper('example.hdf5', mode='w') as data:
   data['x'] = x
   data['y'] = y
   data['z'] = z
   data['p'] = p
```

1.4 Developer Guide

1.4.1 Forecast Model Construction

Model File

To create a new forecast model, we recommend making a copy of *orion/forecast_models/seismogenic_index_model.py* and placing it in the *forecast_models* directory. Make sure to choose a descriptive name for the file and the new forecast class name at the top of the file. When it is ready to be included into the manager, you will need to add it to the list of available models in *orion.forecast_models.__init__.list_*.

Model Initialization

A forecast model will have access to the data structures in the *orion.forecast_models.forecast_model_base.ForecastModel* base class (*Forecast Models API*). You can add model-specific attributes via the `orion.forecast_models.forecast_model_base.ForecastModel.__init__()` method:

```
    def __init__(self, **kwargs):
        """
        Forecast model initialization
        """
        # Call the parent's initialization
        super().__init__(**kwargs)

        # Model activation
        self.long_name = ''
        self.short_name = ''
        self.active = True
        self.weight = 1.0

        self.requires_catalog = False

        # Timing
        self.reference_time = 0.0

        # Forecasts
        self.forecast_time = []
        self.forecast_cumulative_event_count = []
        self.spatialNumberForecast = []

        # Gui elements
        # Note: these will point to the class members by name
        self.gui_elements['long_name'] = {'type': 'text',
                                          'position': [0, 0],
                                          'columnspan': 2}
        self.gui_elements['active'] = {'type': 'check',
                                       'label': 'Active',
                                       'position': [1, 0]}
        self.gui_elements['weight'] = {'type': 'entry',
                                       'label': 'Weight',
                                       'position': [2, 0],
                                       'range': [0, 1],
                                       'interval': 0.5,
                                       'resolution': 0.1}
```

These attributes can be called elsewhere within the model via the *self* variable. They can also be accessed in other objects, as attributes of the model instance. At a minimum, each forecast model is expected to set the attributes `self.short_name` and `self.long_name` (these are used in plotting, GUI routines).

Forecast Generation

The *orion.forecast_models.forecast_model_base.ForecastModel.generate_forecast()* method is called by the forecast manager, and is expected to generate the forecast. It has the following arguments:

```
    def generate_forecast(self, grid, seismic_catalog, pressure, wells, geologic_model,␣
↪forecast_range):
        """
        Model forecast run function

        Args:
            grid (orion.managers.grid_manager.GridManager): The Orion grid manager
            seismic_catalog (orion.managers.seismic_catalog.SeismicCatalog): The current␣
↪seismic catalog
            pressure (orion.pressure_models.pressure_model_base.PressureModelBase): The␣
↪current pressure model
            wells (orion.managers.well_manager.WellManager): The well data
            geologic_model (orion.managers.geologic_model_manager.GeologicModelManager):␣
↪The current geological model
            forecast_range (list): Desired forecast range
        """
        raise Exception("This should be overriden by the child class!")
```

- *grid*: This contains information about the requested background grid (spatial, temporal extents, resolution, etc).
- *seismic_catalog*: This contains the seismic catalog for the current time-slice under consideration.
- *pressure*: This is the active pressure model interpolator.
- *geologic_model*: This is the geologic model, which currently contains information such as the permeability field and in-situ stress.
- *forecast_length*: The requested forecast length (beginning at the end of the time slice)

The goal of a forecast model is to set the following attributes:

- *self.forecast_time*: A `numpy.ndarray` of time values (seconds). Note: it is OK if this doesn't match the other models, since they will be re-interpolated onto a common grid
- *self.spatialNumberForecast*: A `numpy.ndarray` of earthquake count values for each grid point and time values.
- *self.forecast_cumulative_event_count*: A `numpy.ndarray` of cumulative earthquake count values.

Seismic Catalog Access

To generate and train seismic forecasts, it is necessary to look at various slices of the seismic catalog. To enable this behavior, there are two approaches available for interacting with the raw catalog data:

1. Using the data accessors (e.g., x = *self.catalog.get_utm_east_slice()*). These will return the current data being considered by the forecast manager.
2. Directly accessing the data in the structure (e.g., x = *self.catalog.utm_east*). This will return the entire catalog, and is not preferred, due to the additional work required.

The target data slice can be updated by calling the `orion.forecast_models.forecast_model_base.ForecastModel.catalog.set_time_slice()` method. When new data is loaded or the time slice is changed, the code will re-calculate the seismic characteristics. The following example shows how to use these structures:

```
# Example data access
# Note: you can see what data lives in this class
#       by looking at this class, and it's base class
#       (forecast_model_base.ForecastModel)

# Note: print statements should start with four spaces
#       to match the forecast manager indentation
#       Also, c-style print statements are preferred
print('    (data handling demo)')

# Seismic catalog holds raw data, shared seismic
# characteristics (a, b, etc.)
print('    Length of catalog = %i' % (self.catalog.N))
print('    Gutenberg-Richter: a = %1.2f, b = %1.2f' % (self.catalog.a_value, self.
↪catalog.b_value))

# Calculate values directly from the base data
mean_x = np.mean(self.catalog.get_utm_east_slice())
mean_y = np.mean(self.catalog.get_utm_north_slice())
mean_z = np.mean(self.catalog.get_depth_slice())
print('    location mean = (%1.2f, %1.2f, %1.2f) m' % (mean_x, mean_y, mean_z))
print('    UTM zone = %i%s' % (self.catalog.utm_zone[0], self.catalog.utm_zone[1]))

# Note: Time values within Orion are given as the Unix epoch
#       (number of seconds since Jan 1, 1970)
#       Also, catalog entries should be sorted by time
start_time_str = datetime.datetime.fromtimestamp(self.catalog.epoch[0]).strftime('%m/%d/
↪%Y')
t_range = self.catalog.epoch[-1] - self.catalog.epoch[0]
print('    Catalog start time: %s' % (start_time_str))
print('    Catalog time: %1.1f days' % (t_range / (60 * 60 * 24)))

# For now, a forecast model is expected to set these two
# values, which represent the forecasted moment rate in time
# Note: It's OK if the time vector doesn't match the other
#       forecast models, since they will be re-interpolated
#       onto a common grid
print('    (setting forecast to random array)')
self.forecast_time = np.linspace(0, 1, 100)
self.forecast_moment_rate = abs(np.random.randn(100))
```

Pressure Model Access

The primary method to interact with a pressure model is through its interpolator (which is passed to *orion.forecast_models.forecast_model_base.ForecastModel.generate_forecast()* as an argument):

```
    def __call__(self, *xargs):
        """
        If the object is called directly, pass the arguments to p
        """
        return self.p(*xargs)
```

For example, to get the estimated pressure at a given location, you could call:

```
# Each of these is in the local coordinate system:
xyz = [1.0, 2.0, 3.0]  # Target location (m)
t = 40.0               # Current time (s)
p = pressure(xyz[0], xyz[1], xyz[2], t)
```

Geologic Model Access

Similar to the pressure model, the primary method to interact with values is through their interpolator. The values that are currently available include:

- *permeability* (mD)
- *sigma_xx* (Pa)
- *sigma_yy* (Pa)
- *sigma_zz* (Pa)
- *sigma_xy* (Pa)
- *sigma_xz* (Pa)
- *sigma_yz* (Pa)

For example, to get the estimated permeability at a given location, you could call:

```
# Each of these is in the local coordinate system:
xyz = [1.0, 2.0, 3.0]  # Target location (m)
k = geologic_model.permeability(xyz[0], xyz[1], xyz[2])
```

Model Documentation

Each method and attribute for the model should be documented. This is done via the Google-format docstrings (denoted by the triple-quotes). These docstrings are parsed and included within the Orion documentation. The following describes the various options for docstrings: .. _Docstrings: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html

Adding an Attribute to the GUI

To add an attribute to the GUI, it needs to be added to the `orion.forecast_models.forecast_model_base.ForecastModel.gui_elements` dict (see the above example). The name of each entry must point to an existing attribute in the forecast model. When the GUI is executing, it will periodically update the target values. At a minimum a GUI definition must include a *"type"* and *"position"*. The following GUI element types are currently supported:

- text: draws the text stored in the target variable
- check: adds a checkbox to set a boolean flag
- slider: adds a sliderbar to set a float value. This type expects that the following attributes are also set: range (a list of two floats representing the lower/upper values), interval (a float indicating the spacing of sliderbar ticks), and resolution (a float indicating the resolution of the slider)
- dropdown: a menu used to select from a list of strings. This type expects that values (a list of strings) is set
- entry: a text-based entry for string/float variables

- entry_with_file_dialog: a text-based entry, which will add a file dialogue button to its right

The position argument denotes the row/column to place the GUI element. Note: if there is already an object there, they may be drawn over each other. Another key attribute is *"label"*. If this is set, it will draw the label text to the left of the desired GUI element.

1.4.2 Data Managers API

manager_base.py

class orion.managers.manager_base.**ManagerBase**(***kwargs*)

Base manager class for ORION

short_name

A short name used to be used in the Orion Gui

Type
str

child_classes

A list of potential children

Type
list

children

Dictionary of initialized children

Type
dict

figures

Dictonary to hold object plot instructions, handles

Type
dict

N

Length of data loaded into manager

Type
int

gui_elements

Dictionary of elements to be added to the gui

Type
dict

cache_root

Location of the cache directory

Type
str

config_type

The style to be applied to the gui configuration (split/unified, default=split)

Type
str

`plot_cmap_range_options`

List of potential color map options

Type
list

`plot_cmap_range`

Active plot color map option

Type
str

`logger`

The orion logger instance

Type
self.logging.Logger

`add_child`(*child_name*)

Method to add a new child to the current object by name

Parameters
child_name (*str*) – The name of the new child

`adjust_figure_axes`()

Apply formatting to the figures on the current object

`close_figures`()

Close the open figures associated with the current manager

`close_figures_recursive`()

Recursively close the figures associated with the current manager and its children

`generate_plots`(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type*)

Generate any plots for the current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **plot_type** (*str*) – Maximum dimensions for plots (default = 2D)

`generate_plots_recursive`(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type='2D'*)

Recursively generate any plots for the current object

Parameters

- **grid** (orion.managers.seismic_catalog.SeismicCatalog) – The Orion grid manager
- **grid** – The current seismic catalog

- **seismic_catalog** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data

get_config_recursive()

Convert the model configuration to a dict

initialize_children()

Create an instance of each object listed in child_classes

load_config(*fname*)

Loads the forecast manager config from a json file

Parameters
fname (*str*) – Name of the target json configuration file

load_data(*grid*)

Load data into the manager

Parameters
grid (orion.managers.grid_manager.GridManager) – The Orion grid manager

process_inputs()

Process any required gui inputs

process_inputs_recursive()

Process this object and its children

reset_figures()

Reset the open figures associated with the current manager

reset_figures_recursive()

Recursively reset the figures associated with the current manager and its children

save_config(*fname=''*)

Saves the manager config as a json file

Parameters
fname (*str*) – Name of the target json configuration file

save_figures(*output_path*, *dpi=400*)

Save figures

Parameters

- **output_path** (*str*) – Path to place output figures
- **dpi** (*int*) – Resolution of the output figures

set_config_recursive(*config*, *ignore_attributes=['log_file']*)

Sets the current object's configuration from a dictionary or json file

Parameters
config (*dict*) – The configuration dictionary

set_reference_time(*reference_time*)

Set the current reference time for Orion

Parameters
reference_time (*float*) – The current reference time (epoch)

setup_figure_axes(*plot_type*)

Setup any requested figure axes for the current object

Parameters
plot_type (*str*) – The target dimension for supported plots (2D or 3D)

setup_figures(*gui_backend=False*)

Open up figure handles

setup_figures_recursive(*gui_backend=False*)

Recursively open figure handles for orion plots This is completed as a separate step from axes and content due to an initialization order requirement in the gui

update_figure_colors(*plot_type*)

Update figure colors that are not set by rcParams.update()

Parameters
plot_type (*str*) – The target dimension for supported plots (2D or 3D)

update_plot_data(*grid*, *seismic_catalog*, *pressure*, *wells*)

Update plot data for current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data

update_plot_data_recursive(*grid*, *seismic_catalog*, *pressure*, *wells*)

Recursively update plot data for the current object

Parameters

- **grid** (orion.managers.seismic_catalog.SeismicCatalog) – The Orion grid manager
- **grid** – The current seismic catalog
- **seismic_catalog** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data

orion_manager.py

class orion.managers.orion_manager.**OrionManager**(***kwargs*)

Primary Orion manager class

Parameters
config_fname (*str*) – An optional json config file name

config_fname

The current config filename

Type
str

`cache_root`

Path to the user's cache directory

Type
str

`cache_file`

The cached config filename

Type
str

`snapshot_time`

Timestamp to draw plot snapshots (days)

Type
float

`ms_point_size`

The point size to use for supported plots

Type
int

`available_log_levels`

The available logging options

Type
list

`log_level`

The current log level

Type
str

`log_file`

The path of the log file

Type
str

`log_file_existing`

The path to a potential pre-existing log file

Type
str

`log_file_handler`

An object that writes the log to a file

Type
logging.FileHandler

`available_themes`

The available color themes

Type
list

theme

The current theme

Type
str

has_pressure_run

A flag indicating whether pressure calculations have been completed

Type
bool

permissive

A flag indicating whether Orion should attempt to catch errors produced from pressure/forecast calculations

Type
bool

available_plot_types

The available plot types

Type
list

active_plot_types

The active plot type

Type
str

apply_theme()

Apply the theme to figures and any attached guis

check_for_cache_file()

Check to see if an orion cache file is present on the user's machine

generate_all_plots()

Generate plots for the orion manager and its children

generate_plots(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type='2D'*)

Generate any plots for the current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **plot_type** (*str*) – Maximum dimensions for plots (default = 2D)

generate_snapshot_plots()

Generate a subset of plots that need to be redrawn when the time slider is updated

load_built_in(*case_name*, *theme_override=''*)

Loads built in data

Parameters

- **case_name** (*str*) – Name of the built-in case_name to load
- **theme_override** (*str*) – Use this theme, rather than the one present in the config file

load_config_file(*config_file*, *theme_override=''*)

Loads a config file

Parameters

- **config_file** (*str*) – Path to the config file
- **theme_override** (*str*) – Use this theme, rather than the one present in the config file

load_data(*grid*)

Loads data sources

Parameters

grid (orion.managers.grid_manager.GridManager) – The Orion grid manager

process_inputs()

Process the log level and file location

run(*run_pressure=True*, *run_forecasts=True*, *status=None*)

Run the Orion manager

save_example(*fname*)

Saves a full example in zip format

Parameters

fname (*str*) – Name of the target file

orion.managers.orion_manager.**run_manager**(*config*)

Runs the orion manager without a gui

Parameters

config (*fname*) – File name for Orion configuration

forecast_manager.py

class orion.managers.forecast_manager.**ForecastManager**(***kwargs*)

A class for managing the various seismic forecasting methods generated via ORION

forecast_data_start

The start time (mm/dd/yyyy) for the analysis (empty = beginning of catalog)

Type

str

forecast_data_stop

The end time (mm/dd/yyyy) for the analysis (empty = end of catalog)

Type

str

`percent_ensemble_train`

The percentage of the catalog to reserve for training the ensemble forecast

Type
int

`percent_ensemble_test`

The percentage of the catalog to reserve for testing the ensemble forecast

Type
int

`train_split_epoch`

Timestamp at the end of the training segment

Type
float

`forecast_split_epoch`

Timestamp at the end of the testing segment

Type
float

`forecast_end_epoch`

Timestamp at the end of the forecast

Type
float

`forecast_length`

The length of the requested forecast (years)

Type
float

`time_range`

The current time slice under considration

Type
list

`forecast_time`

A list of time vectors produced by the forcast models

Type
list

`forecast_cumulative_event_count`

A list of forecast result vectors produced by the forcast models

Type
list

`estimate_magnitude_exceedance_probability`(*grid*, *seismic_catalog*)

Estimates the probability events will exceed a given user-defined magnitude (self.exceedance_dial_plot_magnitude, self.exceedance_bar_plot_magnitudes) during the next time period (self.exceedance_plot_time_input). The calculation is performed for the entire area, and for individual grid cells.

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog

estimate_weights_linear_regression(*seismic_catalog*)

Estimate the decision tree weights

Parameters

seismic_catalog (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog

generate_model_forecasts(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Generate forecasts for the current time slice

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_length** (*float*) – The length of the requested forecast (seconds)

generate_plots(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type*)

Generate any plots for the current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **plot_type** (*str*) – Maximum dimensions for plots (default = 2D)

parse_timing_requests()

Parse the timing requests for forecast training

process_inputs()

Process any required gui inputs

run(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*)

Chooses the forecast manager style.

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog

- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model

run_grid_style(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*)

Runs the forecast manager on the grid, expects results to be gridded

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model

run_ml_style(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*)

Runs the forecast manager and generates an ensemble forecast

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model

update_plot_data(*grid*, *seismic_catalog*, *pressure*, *wells*)

Update plot data for current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data

geologic_model_manager.py

class orion.managers.geologic_model_manager.**GeologicModelManager**(***kwargs*)

A class for managing the geologic model methods within ORION

available_models

A comprehensive list of available forecast models

Type
list

permeability

Permeability interpolation function (3D)

Type
scipy.interpolate.LinearNDInterpolator

permeability_uniform

Value for a uniform permeability field

Type
float

permeability_file

Filename containing structured/unstructured permeability values and grid/locations

Type
float

sigma_xx

stress interpolation function (xx component, 3D)

Type
scipy.interpolate.LinearNDInterpolator

sigma_yy

stress interpolation function (yy component, 3D)

Type
scipy.interpolate.LinearNDInterpolator

sigma_zz

stress interpolation function (zz component, 3D)

Type
scipy.interpolate.LinearNDInterpolator

sigma_xy

stress interpolation function (xy component, 3D)

Type
scipy.interpolate.LinearNDInterpolator

sigma_xz

stress interpolation function (xz component, 3D)

Type
scipy.interpolate.LinearNDInterpolator

sigma_yz

stress interpolation function (yz component, 3D)

Type
scipy.interpolate.LinearNDInterpolator

sigma_xx_uniform

Value for a uniform stress field (xx component)

Type
float

sigma_yy_uniform

Value for a uniform stress field (yy component)

Type
float

sigma_zz_uniform

Value for a uniform stress field (zz component)

Type
float

sigma_xy_uniform

Value for a uniform stress field (xy component)

Type
float

sigma_xz_uniform

Value for a uniform stress field (xz component)

Type
float

sigma_yz_uniform

Value for a uniform stress field (yz component)

Type
float

sigma_file

Filename containing structured/unstructured stress values and grid/locations

Type
float

generate_plots(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type*)

Generate any plots for the current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data

- **plot_type** (*str*) – Maximum dimensions for plots (default = 2D)

load_data(*grid*)

Load permeability and stress data from a file

load_table_files(*fname*)

Load structured or unstructured property values from an hdf5 format file

fname

Target file name

grid_manager.py

class orion.managers.grid_manager.**GridManager**(***kwargs*)

Grid manager class

ref_time_str

Reference time string in dd/mm/yyyy format

Type
str

spatial_type

Style of spatial inputs (default='UTM')

Type
str

t

Time axis (s)

Type
np.ndarray

t_min

Minimum time (s)

Type
float

t_max

Maximum time (s)

Type
float

dt

Target time resolution (s)

Type
float

x

X axis (m)

Type
np.ndarray

x_min
Minimum X value (m)

Type
float

x_max
Maximum X value (m)

Type
float

dx
Target resolution in the X direction (m)

Type
float

y
Y axis (m)

Type
np.ndarray

y_min
Minimum Y value (m)

Type
float

y_max
Maximum Y value (m)

Type
float

dy
Target resolution in the Y direction (m)

Type
float

z
Z axis (m)

Type
np.ndarray

z_min
Minimum Z value (m)

Type
float

z_max
Maximum Z value (m)

Type
float

dz

Target resolution in the Z direction (m)

Type
float

digitize_values(**args*, *include_edges=False*)

Find the bin IDs for a set of points

Parameters
args (*list*) – List of points to digitize (1=t, 3=xyz, 4=xyzt)

Returns
list of 1D arrays containing grid indices

Return type
list

histogram_values(**args*, *include_edges=False*)

Calculate the histogram for a set of points

Parameters
args (*list*) – List of points to calculate histogram (1=t, 3=xyz, 4=xyzt)

Returns
ND histogram of points

Return type
np.ndarray

process_inputs()

Build the x, y, z, and t axes of the target grid

pressure_manager.py

class orion.managers.pressure_manager.**PressureManager**(***kwargs*)

A class for managing the various pressure estimation methods within ORION

available_models

A comprehensive list of available forecast models

Type
list

generate_plots(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type*)

Generate any plots for the current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **plot_type** (*str*) – Maximum dimensions for plots (default = 2D)

seismic_catalog.py

class orion.managers.seismic_catalog.**SeismicCatalog**(***kwargs*)

Structure for holding seismic catalog information

N

Length of the catalog

Type
int

epoch

Event timestamps (seconds)

Type
ndarray

latitude

Event latitudes (degrees)

Type
ndarray

longitude

Event longitudes (degrees)

Type
ndarray

depth

Event depths (m)

Type
ndarray

utm_zone

UTM Zone for projection

Type
int

easting

Eastings in UTM projection or local coordinates (m)

Type
ndarray

northing

Northings in UTM projection or local coordinates (m)

Type
ndarray

magnitude

Event magnitude magnitudes

Type
ndarray

`magnitude_bins`
magnitude magnitude bin edges

Type
ndarray

`magnitude_exceedance`
magnitude magnitude exceedance per bin

Type
ndarray

`a_value`
Gutenberg-Richter a-value

Type
float

`b_value`
Gutenberg-Richter b-value

Type
float

`varying_b_time`
Times for estimating sub-catalog b-values

Type
ndarray

`varying_b_value`
Gutenberg-Richter b-values over time

Type
ndarray

`magnitude_completeness`
Magnitude of completeness for catalog

Type
float

`background_seismicity_rate`
Background seismicity rate

Type
float

`calculate_cumulative_event_count`(*time_bins*)
Count the number of events over time

Parameters
time_bins (*list*) – bin values in time

Returns
The bin centers, event count in each bin

Return type
tuple

calculate_latlon_coordinates()

Convert catalog UTM coordinates to lat/lon

calculate_magnitude_rate(*time_bins*)

Estimate magnitude rate as a function of time

Parameters
time_bins (*list*) – bin values in time

Returns
The bin centers, estimated magnitude rate in each bin

Return type
tuple

calculate_seismic_characteristics(*magnitude_bin_res=0.1*, *time_segments=10*)

Generate various seismic characteristics

Parameters

- **magnitude_bin_res** (*float*) – bin spacing for calculating a, b values
- **time_segments** (*int*) – number of segments to calculate b values over time

calculate_utm_coordinates()

Convert catalog lat/lon coordinates to UTM

convert_coordinates()

Converts utm coordinates to lat/lon or vice-versa if required

generate_plots(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type*)

Generate any plots for the current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **plot_type** (*str*) – Maximum dimensions for plots (default = 2D)

get_catalog_as_dict()

Save key catalog entries to a dict

Returns
A dictionary of catalog data

Return type
dict

get_copy(*time_range=[-1e+99, 1e+99]*, *magnitude_range=[-1e+99, 1e+99]*, *minimum_interevent_time=-1.0*, *seismic_characteristics_dt=-1.0*)

Get a copy of the the catalog that matches the slice configuration

Parameters

- **time_range** (*list*) – list of sub-catalog min/max times

- **magnitude_range** (*list*) – list of sub-catalog min/max event magnitudes
- **minimum_interevent_time** (*float*) – only include events if this amount of time has elapsed since the last
- **seismic_characteristics_dt** (*float*) – timestep for seismic characteristic calculation

get_depth_slice()

Get the catalog depth slice

get_easting_slice()

Get the catalog easting slice

get_epoch_slice()

Get the catalog time slice

get_latitude_slice()

Get the catalog latitude slice

get_longitude_slice()

Get the catalog longitude slice

get_magnitude_rate_data_slice()

Get the estimated catalog magnitude rate, time vector

get_magnitude_slice()

Get the catalog magnitude slice

get_northing_slice()

Get the catalog northing slice

get_relative_time()

Get the catalog relative time

load_catalog_array(***xargs*)

Initialize catalog from pre-loaded arrays. Required arguments include: epoch, magnitude, depth Location entries can include one of the following: * latitude, longitude * easting, northing (local coordinates) * easting, northing, utm_zone

Additional arguments will be placed in the other_data dict

Parameters

- **epoch** (*np.ndarray*) – 1D array of event time in epoch
- **depth** (*np.ndarray*) – 1D array of event depths
- **magnitude** (*np.ndarray*) – 1D array of event magnitudes
- **longitude** (*np.ndarray*) – 1D array of event longitudes
- **latitude** (*np.ndarray*) – 1D array of event latitudes
- **easting** (*np.ndarray*) – 1D array of event eastings
- **northing** (*np.ndarray*) – 1D array of event northings
- **utm_zone** (*str*) – UTM zone string (e.g.: '4SU')

load_catalog_csv(*filename*)

Reads .csv format seismic catalog files The file should have an optional first line with the zone information "utm_zone, zone_id" and a line with variable names separated by commas See load_catalog_dict for required entries

Parameters
filename (*str*) – catalog file name

load_catalog_dict(*data*)

Load the seismic catalog from an dictionary.

Required entries in the catalog include: epoch, magnitude, depth Location entries can include one of the following: * latitude, longitude * easting, northing (local coordinates) * easting, northing, utm_zone

Parameters
data (*dict*) – catalog dictionary

load_catalog_hdf5(*filename*)

Load the seismic catalog from an hdf5 format file. See load_catalog_dict for required entries

Parameters
filename (*str*) – catalog file name

load_catalog_txt(*filename*)

Reads .txt format seismic catalog files (oklahoma catalog)

Parameters
filename (*str*) – catalog file name

load_catalog_zmap(*filename*)

Reads zmap (.dat) format seismic catalog files

Parameters
filename (*str*) – catalog file name

load_data(*grid*)

Load the seismic catalog if necessary

reset_slice()

Sets the catalog time slice to fit the entire catalog

save_catalog_csv(*filename*)

Save the seismic catalog as a .csv format file

Parameters
filename (*str*) – catalog file name

save_catalog_hdf5(*filename*)

Save the seismic catalog to an hdf5 format file

Parameters
filename (*str*) – catalog file name

set_slice(*time_range=[-1e+99, 1e+99]*, *magnitude_range=[-1e+99, 1e+99]*, *minimum_interevent_time=-1.0*, *seismic_characteristics_dt=-1.0*)

Set the catalog time slice

Parameters

- **time_range** (*list*) – list of sub-catalog min/max times
- **magnitude_range** (*list*) – list of sub-catalog min/max event magnitudes

- **minimum_interevent_time** (*float*) – only include events if this amount of time has elapsed since the last
- **seismic_characteristics_dt** (*float*) – timestep for seismic characteristic calculation

orion.managers.seismic_catalog.**gutenberg_richter_a_b**(*magnitude*, *bins*, *dt*, *min_points=10*)

Estimation Gutenbert Richter a, b values using the least-squares method

Parameters

- **magnitude** (*ndarray*) – 1D array of magnitude magnitudes
- **bins** (*narray*) – 1D array of magnitude bins for calculating a,b
- **dt** (*float*) – time range of catalog

Returns

a-value (float), b-value (float), magnitude_completeness (float), magnitude bin centers (ndarray), magnitude exceedance (ndarray)

Return type

tuple

well_manager.py

class orion.managers.well_manager.**WellManager**(***kwargs*)

A class for managing well information

net_volume

Cumulative fluid injection time series (at grid.t)

Type

np.ndarray

net_dqdt

Net fluid injection rate time series (at grid.t)

Type

np.ndarray

add_child(*child_name*)

Adds an instance of orion.managers.well_data.Well when requested by the Orion Gui

Parameters

child_name (*str*) – Name of the new child well

calculate_well_parameters(*grid*)

Calculate well parameters

Parameters

grid (orion.managers.grid_manager.GridManager) – The Orion grid manager

clear_wells()

Remove all wells

generate_plots(*grid*, *seismic_catalog*, *pressure*, *wells*, *plot_type*)

Generates diagnostic plots for the seismic catalog, fluid injection, and forecasts

`get_injector_flag()`

Check to see if wells are on average injectors

Returns
array of flags indicating which wells are injectors

Return type
np.ndarray

`get_monitor_flag()`

Check to see if wells are monitors

Returns
array of flags indicating which wells are monitors

Return type
np.ndarray

`load_data(grid)`

Load child well data

`serialize_well_data(grid)`

Get the serialized well data

Returns
names, x, y, z, t, q

Return type
list

`update_plot_data(grid, seismic_catalog, pressure, wells)`

Update plot data for current object

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data

well_data.py

`class orion.managers.well_data.Well(**kwargs)`

A class for managing well information. Note: this class can handle constant and time-varying data

x

Location of the well in the x-direction (m)

Type
float

y

Location of the well in the y-direction (m)

Type
float

`z`

Location of the well in the z-direction (m)

Type
float

`init_time`

Time when pumping started (s)

Type
float

`flow_rate`

Average well flow rate (m3/s)

Type
float

`pressure`

Average well botom-hole pressure (Pa)

Type
float

`fname`

Filename for time-series well data

Type
str

`old_fname`

Previous filename for time-series data

Type
str

`epoch`

Time-series well data t-vector

Type
np.ndarray

`N`

Length of time series data

Type
int

`load_data`(*grid*)

Load any time series data if necessary

`process_inputs`()

Build the x, y, z, and t axes of the target grid

`read_injection_file`(*fname*)

Read fluid injection data Note: this function currently supports .dat format

Parameters
fname (*str*) – Name of the wellbore data file

1.4.3 Pressure Models API

pressure_model_base.py

class orion.pressure_models.pressure_model_base.**PressureModelBase**(***kwargs*)

Pressure model base class

dpdt(*x*, *y*, *z*, *t*)

Evaluate the pressure dpdt model for a given location(s)

Parameters

- **x** (*float, array*) – X location in the local coordinate system (m)
- **y** (*float, array*) – Y location in the local coordinate system (m)
- **z** (*float, array*) – Z location in the local coordinate system (m)
- **t** (*float, array*) – Time in the local coordinate system (seconds)

Returns
First derivative of pressure with time

Return type
float

p(*x*, *y*, *z*, *t*)

Evaluate the pressure model for a given location(s)

Parameters

- **x** (*float, array*) – X location in the local coordinate system (m)
- **y** (*float, array*) – Y location in the local coordinate system (m)
- **z** (*float, array*) – Z location in the local coordinate system (m)
- **t** (*float, array*) – Time in the local coordinate system (seconds)

Returns
Pressure

Return type
float

run(*grid*, *well_manager*, *geologic_model*)

Runs the pressure model

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **well_manager** (orion.managers.well_manager.WellManager) – The Orion well manager
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model

radial_flow.py

class orion.pressure_models.radial_flow.**RadialFlowModel**(***kwargs*)

Pressure model based off of Theis Solution

viscosity

Fluid viscosity (cP)

Type
float

permeability

Matrix permeability (nD)

Type
float

storativity

Reservoir storativity factor

Type
float

payzone_thickness

Reservoir thickness

Type
float

min_radius

Minimum radius for solution

Type
float

wells_xyz

Well loctions (m)

Type
list

wells_to

Well start times (s)

Type
list

wells_q

Well flow rates (m3/s)

Type
list

min_dt_numerical

Minimum dt value used to avoid FPE's

Type
float

dpdt(*x*, *y*, *z*, *t*, *display_progress=False*)

Evaluate the pressure dpdt model for a given location(s)

Parameters

- **x** (*float, array*) – X location in the local coordinate system (m)
- **y** (*float, array*) – Y location in the local coordinate system (m)
- **z** (*float, array*) – Z location in the local coordinate system (m)
- **t** (*float, array*) – Time in the local coordinate system (seconds)

Returns
First derivative of pressure with time

Return type
float

p(*x*, *y*, *z*, *t*, *display_progress=False*)

Evaluate the pressure model for a given location(s)

Parameters

- **x** (*float, array*) – X location in the local coordinate system (m)
- **y** (*float, array*) – Y location in the local coordinate system (m)
- **z** (*float, array*) – Z location in the local coordinate system (m)
- **t** (*float, array*) – Time in the local coordinate system (seconds)

Returns
Pressure

Return type
float

run(*grid*, *well_manager*, *geologic_model*)

Runs the pressure model

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **well_manager** (orion.managers.well_manager.WellManager) – The Orion well manager
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model

pretrained_ml_model.py

class orion.pressure_models.pretrained_ml_model.**PretrainedMLModel**(***kwargs*)

Pressure model based off of a pre-trained ML network

model_path

Path to the model file (hdf5)

Type
str

`input_type`
Input style string (default = 'K')

Type
str

`permeability_scale_a`
Permeability scale factor a

Type
float

`permeability_scale_b`
Permeability scale factor b

Type
float

`injection_scale_a`
Injection scale factor a

Type
float

`injection_scale_b`
Injection scale factor b

Type
float

`pressure_scale_a`
Pressure scale factor a

Type
float

`pressure_scale_b`
Pressure scale factor c

Type
float

`p_interp`
Pressure interpolator

Type
scipy.interpolate.RegularGridInterpolator

`dpdt`(*x*, *y*, *z*, *t*)
Evaluate the pressure dpdt model for a given location(s)

Parameters

- **x** (*float, array*) – X location in the local coordinate system (m)
- **y** (*float, array*) – Y location in the local coordinate system (m)
- **z** (*float, array*) – Z location in the local coordinate system (m)
- **t** (*float, array*) – Time in the local coordinate system (seconds)

Returns
First derivative of pressure with time

Return type
float

run(*grid*, *well_manager*, *geologic_model*)

Runs the pressure model

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **well_manager** (orion.managers.well_manager.WellManager) – The Orion well manager
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model

1.4.4 Forecast Models API

forecast_model_base.py

class orion.forecast_models.forecast_model_base.**ForecastModel**(***kwargs*)

Base class for seismic forecast models

active

Flag to indicate whether the model is active

Type
bool

weight

Model relative weight

Type
float

requires_catalog

Flag to indicate whether the model needs a catlog

Type
bool

pressure_method

Pressure calculation method

Type
str

forecast_time

Time values for forecast calculation

Type
ndarray

forecast_cumulative_event_count

Forecasted number of events

Type
ndarray

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

seismogenic_index_model.py

class orion.forecast_models.seismogenic_index_model.**SeismogenicIndexModel**(***kwargs*)

Seismogenic Index forecast model

calc_SI_rate(*countFr_diff*, *b_value*, *seismoIdx*, *minMag*)

Old method to calculate the seismicity rate based on the seismogenic index and the fluid volume

Parameters

- **countFr_diff** (*float*) – argument description
- **b_value** (*float*) – argument description
- **seismoIdx** (*float*) – argument description
- **minMag** (*float*) – argument description

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

old_method(*seismic_catalog*)

Existing forecast code

reasenberg_jones_model.py

class orion.forecast_models.reasenberg_jones_model.**ReasenbergJonesModel**(***kwargs*)

Reasenberg-Jones forecast model

active

Flag to indicate whether the model is active

Type
bool

weight

Model relative weight

Type
float

pressure_method

Pressure calculation method

Type
str

forecast_time

Time values for forecast calculation

Type
ndarray

forecast_if_catalog()

Forecasting model if a catalog is active

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

old_method(*seismic_catalog*)

Existing forecast code

orion.forecast_models.reasenberg_jones_model.**calc_RJ**(*fMainMag*, *fMinMag*, *fMaxMag*, *fAvalue*, *fBvalue*, *fPvalue*, *fCvalue*, *fForecastTime*, *fForecastStart*)

Old method to calculate rates of aftershocks and probabilities within time

Parameters

- **fMainMag** (*float*) – main shock magnitude
- **fMinMag** (*float*) – minimum magnitude to forecast rates for
- **fMaxMag** (*float*) – maximum magnitude to forecast rates for
- **dmag** (*float*) – Magnitude Step for integration
- **fAvalue** (*float*) – Generic a-value
- **fBvalue** (*float*) – Generic b-value
- **fPvalue** (*float*) – Generic p-value
- **fCvalue** (*float*) – Generic c-value
- **fForecastTime** (*float*) – Length of forecast window (days)
- **fForcastStart** (*float*) – Start of forcast window after the main shock (days)

Returns
[Forecast Rate, Start Time, Probabilities, mainMag]

Return type
tuple

Example

sForecastRate = calc_RJ(5.5,5, 7, -1.6, 1, 1.2, 0.05, 12, 0)

openSHA_model.py

class orion.forecast_models.openSHA_model.**OpenSHAModel**(***kwargs*)

OpenSHA forecast model

active

Flag to indicate whether the model is active

Type
bool

weight

Model relative weight

Type
float

pressure_method

Pressure calculation method

Type
str

forecast_time

Time values for forecast calculation

Type
ndarray

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

etas_model.py

class orion.forecast_models.etas_model.**ETASModel**(***kwargs*)

ETAS forecast model

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

coupled_coulomb_rate_state_model.py

class orion.forecast_models.coupled_coulomb_rate_state_model.**CRSModel**(***kwargs*)

CRS forecast model

active

Flag to indicate whether the model is active

Type
bool

weight

Model relative weight

Type
float

pressure_method

Pressure calculation method

Type
str

forecast_time

Time values for forecast calculation

Type
ndarray

computeCoulombStressingRate(*pressureRate*)

Effective (time-dependent) normal stress that varies with pressure

Parameters
pressureRate (*np.array*) – pore-fluid pressurization rate time series at a point (MPa)

Returns
Coulomb stressing rate time series

Return type
np.array

computeSigmaEffective(*pressure*)

Effective (time-dependent) normal stress that varies with pressure

Parameters

- **sigma** (*float*) – initial normal stress (Pa)
- **biot** (*float*) – biot coefficient
- **pressure** (*np.array*) – pore-fluid pressure time series at a point (Pa)

Returns
effective normal stress time series

Return type
np.array

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_length*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

instantRateChange(*static_coulomb_stress_change*, *background_rate*, *rate_coefficient*, *sigma_effective*)

Instantaneous change in Rate (R) due to step change in Coulomb stress

Parameters

- **static_coulomb_stress_change** (*float*) – Coulomb stress step (Pa by default)
- **background_rate** (*float*) – long-term background seismicity rate before stress step (seconds by default)
- **rate_coefficient** (*float*) – rate-coefficient in rate-state formulation
- **sigma_effective** (*np.array*) – effective normal stress (Pa; sigma_effective = sigma - biot*pressure)

Returns
Instantaneous change in rate

Return type
float

interseismicNumber(*forecast_time*, *eta*, *coulomb_stressing_rate*, *rate_at_prev_step*, *rate_coefficient*, *sigma_effective*)

Compute expected total number of events during a time of constant stressing rate

Parameters

- **forecast_time** (*np.array*) – np.array of times at which number should be computed (units: same as the time units in sigma_effective; seconds by default)
- **eta** (*float*) – reference stressing rate divided by background rate (steady event rate that would be produced by constant stressing at the reference stressing rate) (units: stress/event with stress in same units as the stress units in sigma_effective; Pa/second by default)
- **coulomb_stressing_rate** (*np.array*) – np.array of constant Coulomb stressing rate (units: same stress units as used in eta, same time units as forecast_time; Pa/second by default)
- **rate_at_prev_step** (*float*) – event rate at the previous time step (units: events/time, same time units as forecast_time (seconds by default))
- **rate_coefficient** (*float*) – rate-coefficient in rate-state formulation
- **sigma_effective** (*np.array*) – effective normal stress (Pa; sigma_effective = sigma - biot*pressure)

Returns
interseismic number

Return type
np.array

interseismicRate(*forecast_time*, *eta*, *coulomb_stressing_rate*, *rate_at_prev_step*, *rate_coefficient*, *sigma_effective*)

Evolution of the Rate during a time of constant stressing rate

Parameters

- **forecast_time** (*np.array*) – Times at which number should be computed (units: same as the time units in sigma_effective; seconds by default)
- **eta** (*float*) – reference stressing rate divided by background rate (steady event rate that would be produced by constant stressing at the reference stressing rate) (units: stress/event with stress in same units as the stress units in sigma_effective; Pa/second by default)
- **coulomb_stressing_rate** (*np.array*) – Constant Coulomb stressing rate (units: same stress units as used in eta, same time units as t; Pa/second by default)

- **rate_at_prev_step** (*float*) – event rate at the previous time step (units: events/time, same time units as forecast_time (seconds by default))
- **rate_coefficient** (*float*) – rate-coefficient in rate-state formulation
- **sigma_effective** (*np.array*) – effective normal stress (Pa; sigma_effective = sigma - biot*pressure)

Returns
interseismic rate

Return type
np.array

numberEvolution(*forecast_time*, *large_event_times_in_forecast*, *static_coulomb_stress_change*, *coulomb_stressing_rate*, *background_rate*, *rate_factor*, *rate_coefficient*, *sigma_effective*)

Calculate the number of events for all times passed into forecast_time. forecast_time and large_event_times_in_forecast must all have the same units of time. for consistency, we will use "seconds" as the standard time unit.

Instaneous stress step vector (static_coulomb_stress_change) must be [length(sigma_effective) - 1] (stressing rate vector) beginning at the time of the first change

Parameters

- **forecast_time** (*np.array*) – Times at which number should be computed (units same as the time units in sigma_effective; seconds by default)
- **large_event_times_in_forecast** (*np.array*) – Times of the stress steps (seconds by default)
- **static_coulomb_stress_change** (*np.array*) – Amplitude of the stress steps (+/- ; Pa by default) must be the same length as large_event_times_in_forecast
- **coulomb_stressing_rate** (*np.array*) – Constant Coulomb stressing rate (units: same stress units as used in eta, same time units as t; Pa/second by default)
- **background_rate** (*float*) – Event rate at time forecast_time=0 (units: events/time, same time units as forecast_time (seconds by default))
- **rate_factor** (*float*) – 1/eta; the background seismicity rate divided by the background stressing rate
- **rate_coefficient** (*float*) – rate-coefficient in rate-state formulation
- **sigma_effective** (*np.array*) – effective normal stress (Pa; sigma_effective = sigma - biot*pressure, same length as forecast_time)

Returns
Number evolution

Return type
np.array

process_inputs()

Process any required gui inputs

rateEvolution(*forecast_time*, *large_event_times_in_forecast*, *static_coulomb_stress_change*, *coulomb_stressing_rate*, *background_rate*, *rate_factor*, *rate_coefficient*, *sigma_effective*)

Calculate the earthquake rate for all times passed into t. t and large_event_times_in_forecast must all have the same units of time. for consistency, we will use "years" as the standard time unit.

Instaneous stress step vector (static_coulomb_stress_change) must be [length(sigma_effective) - 1] (stressing rate vector) beginning at the time of the first change

Parameters

- **forecast_time** (*np.array*) – Times at which number should be computed (units: same as the time units in sigma_effective; seconds by default)
- **large_event_times_in_forecast** (*np.array*) – Times of the stress steps (seconds by default)
- **static_coulomb_stress_change** (*np.array*) – Amplitude of the stress steps (+/- ; Pa by default) must be the same length as large_event_times_in_forecast
- **coulomb_stressing_rate** – (np.array): constant Coulomb stressing rate (units: same stress units as used in eta, same time units as t; Pa/second by default) len(sigma_effective) must equal len(forecast_time)
- **background_rate** (*float*) – Event rate at time forecast_time=0 (units: events/time, same time units as forecast_time (seconds by default))
- **rate_factor** (*float*) – 1/eta; the background seismicity rate divided by the background stressing rate
- **rate_coefficient** (*float*) – rate-coefficient in rate-state formulation
- **sigma_effective** (*np.array*) – effective normal stress (Pa; sigma_effective = sigma - biot*pressure)

Returns
Rate evolution

Return type
np.array

pretrained_lstm_model.py

class orion.forecast_models.pretrained_lstm_model.**PretrainedMachineLearningModel**(***kwargs*)

Pretrained Machine Learning forecast model

active

Flag to indicate whether the model is active

Type
bool

weight

Model relative weight

Type
float

forecast_time

Time values for forecast calculation

Type
ndarray

trained_model_snapshot

File name of the saved model (.hdf5)

Type
str

trained_model_scaling

File name containing model scaling (.hdf5)

Type
str

inputs

List of input parameters

Type
list

outputs

List of output parameters

Type
list

network

ML network instance

Type
Tensorflow.Model

scaling

Dict of mean, std for each input/output parameter

Type
dict

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function

Parameters

- **grid** (orion.managers.grid_manager.GridManager) – The Orion grid manager
- **seismic_catalog** (orion.managers.seismic_catalog.SeismicCatalog) – The current seismic catalog
- **pressure** (orion.pressure_models.pressure_model_base.PressureModelBase) – The current pressure model
- **wells** (orion.managers.well_manager.WellManager) – The well data
- **geologic_model** (orion.managers.geologic_model_manager.GeologicModelManager) – The current geological model
- **forecast_range** (*list*) – Desired forecast range

load_model()

Load the machine learning model in a tensorflow-hdf5 format

rate_and_state_ode_model

class orion.forecast_models.rate_and_state_ode_model.**RSODEModel**(***kwargs*)

generate_forecast(*grid*, *seismic_catalog*, *pressure*, *wells*, *geologic_model*, *forecast_range*)

Model forecast run function.

Parameters

- **grid** (*orion.managers.grid_manager.GridManager*) – The Orion grid manager.
- **seismic_catalog** (*orion.managers.seismic_catalog.SeismicCatalog*) – The current seismic catalog.
- **pressure** (*orion.pressure_models.pressure_model_base.PressureModelBase*) – The current pressure model.
- **wells** (*orion.managers.well_manager.WellManager*) – The well data.
- **geologic_model** (*orion.managers.geologic_model_manager.GeologicModelManager*) – The current geological model.
- **forecast_range** (*list*) – Desired forecast range.

process_inputs()

Convert input units to SI units.

1.4.5 Utilities API

file_io.py

orion.utilities.file_io.**check_table_shape**(*data*, *axes_names=['x', 'y', 'z', 't']*)

Check shape of table arrays

orion.utilities.file_io.**data**

Dictionary of table entries

orion.utilities.file_io.**axes_names**

List of potential axes names

orion.utilities.file_io.**parse_csv**(*fname*)

Parse csv file with headers, units

Parameters

- **fname** (*string*) – Filename
- **header_size** (*int*) – number of header lines

Returns

File results

Return type

dict

orion.utilities.file_io.**read_flowfile**(*filename*)

read flowfile with year month day hour minute second flow pressure constant

@arg filename filename

orion.utilities.file_io.**read_zmap_catalog**(*filename*)

function description

@arg filename filename

function_wrappers.py

class orion.utilities.function_wrappers.**constant_fn**(*value*)

Factory to return a constant function with N arguments

Parameters
value (*float*) – A constant value

Returns
A constant value function

Return type
function

class orion.utilities.function_wrappers.**masked_fn**(*original_fn*, *mask*, *list_arg=False*)

Factory to ignore unnecessary positional arguments to a function

Parameters

- **original_fn** (*scipy.interpolate.LinearNDInterpolator*) – The original interpolation function
- **mask** (*list*) – list of bools indicating which arguments to keep
- **list_arg** (*bool*) – Flag to indicate whether arguments to the function should be converted to a list (default=False)

Returns
The wrapped function

Return type
function

class orion.utilities.function_wrappers.**variable_len_fn**(*original_fn*, *Ndim*, *list_arg=False*)

Factory to handle mismatches between the number of inputs to the original function. If the number of arguments in greater than the original, the additional values will be ignored. If the number of arguments is smaller than the original, they will be padded with zeros

Parameters

- **original_fn** (*scipy.interpolate.LinearNDInterpolator*) – The original interpolation function
- **Ndim** (*int*) – The expected number of arguments to the original function
- **list_arg** (*bool*) – Flag to indicate whether arguments to the function should be converted to a list (default=False)

Returns
The wrapped function

Return type
function

hdf5_wrapper.py

class orion.utilities.hdf5_wrapper.**hdf5_wrapper**(*fname=''*, *target=''*, *mode='r'*)

Class for reading/writing hdf5 files, which behaves similar to a native dict

close()

Closes the database

copy(*output*)

Pack the contents of the current database level onto the target dict

Parameters
output (*dict*) – the dictionary to pack objects into

get_copy()

Copy the entire database into memory

Returns
dict: A copy of the database

items()

Return the key-value pairs for entries at the current level

Returns
tuple: keys, values

keys()

Get a list of groups and arrays located at the current level

Returns
list: a list of strings

link(*k*, *target*)

Link an external hdf5 file to this location in the database

Parameters

- **k** (*str*) – the name of the new link in the database
- **target** (*str*) – the path to the external database

plot_tools.py

orion.utilities.plot_tools.**exceedance_bar_plot**(*ax*, *magnitude*, *probability*, *color_lims=[0.3, 0.6]*)

Build a magnitude exceedance bar plot

Parameters

- **ax** (*matplotlib.pyplot.axis*) – Target figure axis
- **magnitude** (*np.ndarray*) – magnitude magnitudes
- **probability** (*np.ndarray*) – Probability of exceedance
- **color_lims** (*list*) – Locations to place the transition between green/yellow and yellow/red

orion.utilities.plot_tools.**exceedance_dial_plot**(*ax*, *probability*, *color_lims=[0.3, 0.6]*, *ra=0.8*, *rb=0.9*, *rc=1.0*)

Build a magnitude exceedance bar plot

Parameters

- **ax** (*matplotlib.pyplot.axis*) – Target figure axis
- **probability** (*np.ndarray*) – Probability of exceedance
- **color_lims** (*list*) – Locations to place the transition between green/yellow and yellow/red
- **ra** (*float*) – Radius of the pointer
- **ra** – Radius of the inner-dial
- **rc** (*float*) – Radius of the outer-dial

orion.utilities.plot_tools.**multivariatePlot**(*root_axis*, *plots*, *offset_val=0.12*)

Build a multivariate plot

Parameters

- **root_axis** (*matplotlib.pyplot.axis*) – Target figure axis
- **plots** (*dict*) – Dictionary of plot instructions
- **offset_val** (*float*) – Axis offset value (default=0.12)

orion.utilities.plot_tools.**setupColorbar**(*fig*, *ca*, *cax*, *value_range*, *label*)

Setup a colorbar, optimizing the number and format of ticks

Parameters

- **fig** (*matplotlib.figure.Figure*) – Target figure handle
- **ca** (*matplotlib.image.AxesImage*) – AxesImage from plt.imshow
- **cax** (*matplotlib.axes.Axes*) – Location to place colorbar
- **value_range** (*list*) – Target value range
- **label** (*str*) – Colorbar label

statistical_methods.py

orion.utilities.statistical_methods.**poisson_probability**(*forecast_time*, *forecast_number*, *b_value*, *magnitude_completeness*, *magnitude_thresholds*, *forecast_duration*)

Compute the time dependent probability of an event M > magnitude_thresholds in the given time window using b-values, magnitude_completeness, and total number of events from the observed catalog during the fitting period

orion.utilities.statistical_methods.**forecast_time**

time series associated with forecast_number (default unit is seconds)

Type
np.array

orion.utilities.statistical_methods.**forecast_number**

forecasted number of events as a function of time listed in forecast_time

Type
np.array

orion.utilities.statistical_methods.**magnitude_thresholds**

Magnitude of events of interest. Compute probability of M>magnitude_thresholds

Type
np.array

orion.utilities.statistical_methods.**forecast_duration**

forecast_durationation of the probability calculation period (same units a forecast_time)

Type
integer

orion.utilities.statistical_methods.**magnitude_completeness**

Magnitude of completeness of the events in eqs, compute with other means

Type
integer

Returns
np.array of proabilities of an event M>magnitude_thresholds will occur in the time 'forecast_duration' for all magnitude_thresholds

table_files.py

orion.utilities.table_files.**load_table_files**(*data*, *axes_names=['x', 'y', 'z', 't']*)

Load structured or unstructured property values

orion.utilities.table_files.**data**

Dictionary of table entries

orion.utilities.table_files.**axes_names**

List of potential axes names

timestamp_conversion.py

unit_conversion.py

class orion.utilities.unit_conversion.**DictRegexHandler**

This class is used to substitute matched values with those stored in a dict.

class orion.utilities.unit_conversion.**UnitManager**

This class is used to manage unit definitions.

buildUnits()

Build the unit definitions.

regexHandler(*match*)

Split the matched string into a scale and unit definition.

@param match The matching string from the regex.

1.4.6 GUI API

orion_gui.py

class orion.gui.orion_gui.**OrionGUI**(*root*, *manager*)

Main Orion gui

main_buttons

An object to hold the control buttons in the gui

Type
dict

notebook

A notebook holding figure frames

Type
orion.gui.custom_widgets.CompactNotebook

example_dropdown

The menu holding available examples

Type
tkinter.Menu

time_ticks

The number of ticks to use in the time slider

Type
int

time_slider

The primary time slider

Type
ttkwidgets.TickScale

time_var

The time variable associated with the slider

Type
tkinter.DoubleVar

last_time_state

The time state associated with the last gui update

Type
float

last_time_range

The time range associated with the last gui update

Type
list

time_slider_modified

A flag indicating whether the time slider has been modified

Type
bool

`relaunch_config`

A flag indicating whether the config gui should be reopened (typically following updates)

Type
bool

`snapshot_plots_modified`

A flag signaling that snapshot plots should be updated

Type
bool

`all_plots_modified`

A flag signaling that all plots should be updated

Type
bool

`logger`

The orion logging instance

Type
logging.Logger

`logger_frame`

The frame holding the logger outputs

Type
ttk.Frame

`logger_text`

The text object holding logger outputs

Type
tkinter.Text

`logger_index`

The number of logging output lines

Type
int

`logger_handler`

An object that intercepts and records logger outputs

Type
orion.gui.custom_widgets.ListHandler

`button_request_complete`

A flag signaling that button requests are complete

Type
bool

`button_request_queue`

A queue to handle button requests

Type
queue.Queue

orion_manager

An instance of Orion

Type
orion.managers.orion_manager.OrionManager

theme

The active theme

Type
str

add_example_to_menu(*example_name*)

Add an example to the dropdown menu

Parameters
example_name (*str*) – Name of the example

build_figure_frame(*parent*, *ka*)

Build a figure frame

Parameters

- **parent** (orion.managers.manager_base.ManagerBase) – Associated Orion manger
- **ka** (*str*) – Frame key

build_figure_frame_extra_elements(*parent*, *ka*)

Build navigation bars and layer selection elements

Parameters

- **parent** (orion.managers.manager_base.ManagerBase) – Associated Orion manger
- **ka** (*str*) – Frame key

build_splash_page()

Gui splash page creation

button_updater()

This function is periodically evaluated to check for button update requests

create_main()

Gui main window creation

hide_object(*target_object*, *cursor_state*)

Callback function used to hide an object when the cursor is placed over a target

Parameters

- **target_object** (*ttk.Frame*) – The object to show
- **place_args** (*dict*) – A list of arguments to pass to the place method
- **cursor_state** (*bool*) – The state of the cursor

load_built_in(*source*)

Load build in sources

Parameters
source (*str*) – Built in source name

load_config_interactive()

Load a config file using a user prompt

log_updater()

Update the log messages in the gui

manage_button_requests()

Manage the button request queue

open_gui_about()

Set gui configuration options

open_gui_config()

Set gui configuration options

open_gui_example_selection()

Example download options

open_orion_docs()

Open orion documentation

Note: this document requires that you be logged into gitlab and have access to the repository

plot_updater()

This function is periodically evaluated to check for plot update requests

post_load_update()

Updates gui elements after loading a new config file

pre_load_update()

Prepare the gui for loading a config file

quit()

Gui close method

request_all()

Add a pressure/forecast calculation request load to the button request queue

request_data_load()

Add a data request load to the button request queue

request_forecasts()

Add a forecast calculation request load to the button request queue

request_pressure()

Add a pressure calculation request load to the button request queue

run_all()

Update the current configuration and run Orion

run_forecasts()

Update the current configuration and run Orion

run_pressure()

Update the current configuration and run Orion

save_config_interactive()

Save the current configuration to a json file

save_figures()

Save all figures in the gui to a user-selected directory

show_object(*target_object*, *place_args*, *cursor_state*)

Callback function used to show an object when the cursor is placed over a target

Parameters

- **target_object** (*ttk.Frame*) – The object to show
- **place_args** (*dict*) – A list of arguments to pass to the place method
- **cursor_state** (*bool*) – The state of the cursor

theme_updater()

This function is periodically evaluated to check for theme update requests

time_slider_activation(*slider_value*)

Mark that the time slider has been touched

Parameters

slider_value (*float*) – the current slider value

update_config()

Trigger an update of the orion configuration values if the config window is open

update_figure_colors(*parent*, *ka*, *plot_type*)

Update figure colors that aren't set by ttkstyle or rcParams

Parameters

- **parent** (orion.managers.manager_base.ManagerBase) – The parent orion manager
- **ka** (*str*) – The name of the orion manager used in gui definitions
- **plot_type** (*str*) – The target dimension for supported plots (2D, 3D)

update_figure_frames()

Update all figures in the gui

update_time_slider()

Update the time slider intervals

updater(*after=True*)

Updater functions specific to the figure gui

Parameters

after (*bool*) – Flag to indicate whether to schedule another update

orion.gui.orion_gui.**function_toggle_factory**(*variable*, *figure*, *name*, *parent*)

Function factory to handle plot visibility

orion.gui.orion_gui.**launch_gui**(*config_fname*)

Launch the Orion gui

Parameters

config_fname (*str*) – Name of the orion config file

1.5 Acknowledgements

Orion was developed with funding support from the following:

- United States Department of Energy SMART Initiative
- United States Department of Energy National Risk Assessment Partnership (NRAP)

This support is gratefully acknowledged. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

1.6 References

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Dieterich, 1994] Dieterich, J. (1994). A constitutive law for rate of earthquake production and its application to earthquake clustering. *Journal of Geophysical Research: Solid Earth*, *99*(B2), 2601–2618.

[Dieterich, 2007] Dieterich, J. (2007). Applications of rate-and state-dependent friction to models of fault slip and earthquake occurrence. *Earthquake seismology*, *4*, 107–129.

[Fehlberg, 1969] Fehlberg, E. (1969). *Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems*. Vol. 315. National aeronautics and space administration.

[Ferris et al., 1962] Ferris, J. G., Knowles, D. B., Brown, R. H., & Stallman, R. W. (1962). *Theory of aquifer tests*. US Government Printing Office Washington.

[Hager et al., 2021] Hager, B. H., Dieterich, J., Frohlich, C., Juanes, R., Mantica, S., Shaw, J. H., … others. (2021). A process-based approach to understanding and managing triggered seismicity. *Nature*, *595*(7869), 684–689.

[Kroll et al., 2017] Kroll, K. A., Richards-Dinger, K. B., Dieterich, J. H., & Cochran, E. S. (2017). Delayed seismicity rate changes controlled by static stress transfer. *Journal of Geophysical Research: Solid Earth*, *122*(10), 7951–7965.

[Segall & Lu, 2015] Segall, P., & Lu, S. (2015). Injection-induced seismicity: poroelastic and earthquake nucleation effects. *Journal of Geophysical Research: Solid Earth*, *120*(7), 5082–5103.

PYTHON MODULE INDEX

O

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z