Thunor Core Documentation

Release 0.1.26+5.g76a5a28.dirty

Alex Lubbock

May 25, 2021
Thunor Core is a Python package for managing and viewing high throughput screen data. It can calculate and visualize both single-timepoint viability calculations, and the multi-timepoint drug-induced proliferation rate (DIP rate) metric, which is a dynamic measure of drug response.

For further information on Thunor, related projects (including a web interface, Thunor Web), and further help see the Thunor website.

Contents:
To install Thunor, you can use pip:

```
pip install thunor
```

Please note that Python 3 is required (not compatible with Python 2.7).
Thunor (pronounced THOO-nor) is a free software platform for managing, visualizing, and analyzing high throughput cell proliferation data, which measure the dose-dependent response of cells to one or more drug(s).

This repository, Thunor Core, is a Python package which can be used for standalone analysis or integration into computational pipelines.

A web interface is also available, called Thunor Web. Thunor Web has a comprehensive manual, which goes into further detail about the curve fitting methods, types of plots available and other information you may find relevant.

Please see the Thunor website for additional resources, and a link to our chat room, where you can ask questions about Thunor.

2.1 Start Jupyter Notebook

Run jupyter notebook with the following argument:

```
jupyter notebook --NotebookApp.iopub_data_rate_limit=1.0e10
```

The data rate limit needs to be increased or init_notebook_mode() throws an error. This is a plotly requirement.

2.2 Check Thunor Core is available

```python
# If the import doesn't work, uncomment the following two lines, or "pip install thunor"
import os, sys
sys.path.insert(0, os.path.abspath('../'))

import thunor
```
2.3 Load a file

First, specify a file to load. Here, we use an example dataset from the thunor package itself.

```
hts007_file = '../thunor/testdata/hts007.h5'
```

Load the file using `read_hdf` (for HDF5 files), `read_vanderbilt_hts` (for CSV files), or another appropriate reader.

```
from thunor.io import read_hdf
hts007 = read_hdf(hts007_file)
```

We'll just use a subset of the drugs, to make the plots manageable.

```
hts007r = hts007.filter(drugs=['cediranib', 'everolimus', 'paclitaxel'])
```

```
hts007r.drugs

[('cediranib'), ('everolimus'), ('paclitaxel')]
```

```
hts007r.cell_lines

['BT20', 'HCC1143', 'MCF10A-HMS', 'MCF10A-VU', 'MDAMB231', 'MDAMB453', 'MDAMB468', 'SUM149']
```

2.4 Calculate DIP rates and parameters

These two operations can be done in two lines of code (plus imports). Note that you may see `RuntimeWarning` messages, which indicates that some dose response curves were not able to be fitted. This can happen if the cells do not stop proliferating in response to drug, the response is not closely approximated by a log-logistic curve, or the data are very noisy.

```
from thunor.dip import dip_rates
from thunor.curve_fit import fit_params

ctrl_dip_data, expt_dip_data = dip_rates(hts007r)
fp = fit_params(ctrl_dip_data, expt_dip_data)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/thunor/checkouts/latest/thunor/curve_fit.py:157: RuntimeWarning: invalid value encountered in log
  return c + (d - c) / (1 + np.exp(b * (np.log(x) - np.log(e))))
/home/docs/checkouts/readthedocs.org/user_builds/thunor/checkouts/latest/thunor/curve_fit.py:225: RuntimeWarning: invalid value encountered in double_scalars
  1 / self.hill_slope)
```
2.5 Setting up plots

Each of the plot_X functions returns a plotly Figure object which can be visualised in a number of ways. Here, we use the offline iplot function, which generates a plot for use with Jupyter notebook. We could also generate plots using the plot function in standalone HTML files. See the plotly documentation for more information on the latter approach.

```python
from thunor.plots import plot_drc, plot_drc_params, plot_time_course, plot_ctrl_dip_by_plate, plot_plate_map
```

2.6 Plot Types

2.6.1 Plot DIP rate curves

```python
plot_drc(fp)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

2.6.2 Plot DIP parameters

```python
plot_drc_params(fp, 'auc')
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

2.6.3 Filtering fit params

The fp object is a pandas data frame, so we can filter it before plotting. Some examples:

```python
fit_params_BT20_pac = fp[fp.index.isin(['BT20'], level='cell_line') & 
fp.index.isin(['paclitaxel'], level='drug')]
plot_drc(fit_params_BT20_pac)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html
2.6.4 Plot time course

Time course plot for paclitaxel on BT20 cells:

```python
[12]: plot_time_course(
    hts007.filter(drugs=['paclitaxel'], cell_lines=['BT20'])
)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

2.6.5 Quality control check: plot DIP rate ranges by cell line and plate (box plot)

```python
[13]: plot_ctrl_dip_by_plate(ctrl_dip_data)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

2.6.6 Quality control check: plot DIP rate as a plate heat map

```python
[14]: plate_data = hts007.plate('HTS007_149-28A', include_dip_rates=True)

[15]: plot_plate_map(plate_data, color_by='dip_rates')
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html
3.1 I/O, file reading and writing, core formats (thunor.io)

class thunor.io.HtsPandas(doses, assays, controls)
   High throughput screen dataset
   Represented internally using pandas dataframes
   Parameters
   • doses (pd.DataFrame) – DataFrame of doses
   • assays (pd.DataFrame) – DataFrame of assays
   • controls (pd.DataFrame) – DataFrame of controls

cell_lines
   List of cell lines in the dataset
   Type list

drugs
   List of drugs in the dataset
   Type list

assay_names
   List of assay names in the dataset
   Type list

dip_assay_name
   The assay name used for DIP rate calculations, e.g. “Cell count”
   Type str

doses_unstacked()
   Split multiple drugs/doses into separate columns

filter(cell_lines=None, drugs=None, plate=None)
   Filter by cell lines and/or drugs
   “None” means “no filter”
   Parameters
   • cell_lines (Iterable, optional) – List of cell lines to filter on
   • drugs (Iterable, optional) – List of drugs to filter on
   • plate (Iterable, optional) –
Returns  A new dataset filtered using the supplied arguments

Return type  \texttt{HtsPandas}

\texttt{plate}(\texttt{plate\_name}, \texttt{plate\_size}=384, \texttt{include\_dip\_rates}=False)

Return a single plate in PlateData format

Parameters

- \texttt{plate\_name} (\texttt{str}) – The name of a plate in the dataset
- \texttt{plate\_size} (\texttt{int}) – The number of wells on the plate (default: 384)
- \texttt{include\_dip\_rates} (\texttt{bool}) – Calculate and include DIP rates for each well if True

Returns  The plate data for the requested plate name

Return type  \texttt{PlateData}

\texttt{class thunor.io.PlateData}(\texttt{width}=24, \texttt{height}=16, \texttt{dataset\_name}=None, \texttt{plate\_name}=None, \texttt{cell\_lines}=[], \texttt{drugs}=[], \texttt{doses}=[], \texttt{dip\_rates}=[])

A High Throughput Screening Plate with Data

\texttt{exception thunor.io.PlateFileParseException}

\texttt{class thunor.io.PlateMap}(**\texttt{kwargs})

Representation of a High Throughput Screening plate

Parameters  \texttt{kwargs} (\texttt{dict}, optional) – Optionally supply “width” and “height” values for the plate

\texttt{col\_iterator}()

Iterate over the column numbers in the plate

Returns  Iterator over the column numbers (1, 2, 3, etc.)

Return type  \texttt{Iterator of int}

\texttt{property num\_wells}

Number of wells in the plate

\texttt{classmethod plate\_size\_from\_num\_wells}(\texttt{num\_wells})

Calculate plate size from number of wells, assuming 3x2 ratio

Parameters  \texttt{num\_wells} (\texttt{int}) – Number of wells in a plate

Returns  Width and height of plate (numbers of wells)

Return type  \texttt{tuple}

\texttt{row\_iterator}()

Iterate over the row letters in the plate

Returns  Iterator over the row letters (A, B, C, etc.)

Return type  \texttt{Iterator of str}

\texttt{well\_id\_to\_name}(\texttt{well\_id})

Convert a Well ID into a well name

Well IDs use a numerical counter from left to right, top to bottom, and are zero based.

Parameters  \texttt{well\_id} (\texttt{int}) – Well ID on this plate

Returns  Name for this well, e.g. A1

Return type  \texttt{str}
**well_iterator()**
Iterator over the plate’s wells

**Returns**  Iterator over the wells in the plate. Each well is given as a dict of ‘well’ (well ID), ‘row’ (row character) and ‘col’ (column number)

**Return type**  Iterator of dict

**well_list()**
List of the plate’s wells

**Returns**  The return value of `well_iterator()` as a list

**Return type**  list

**well_name_to_id(well_name, raise_error=True)**
Convert a well name to a Well ID

**Parameters**

- **well_name** *(str)* – A well name, e.g. A1
- **raise_error** *(bool)* – Raise an error if the well name is invalid if True (default), otherwise return -1 for invalid well names

**Returns**  Well ID for this well. See also `well_id_to_name()`

**Return type**  int

**thunor.io.read_hdf(filename_or_buffer)**
Read a HtsPandas dataset from Thunor HDF5 format file

**Parameters**

- **filename_or_buffer** *(str or object)* – Filename or buffer from which to read the data

**Returns**  Thunor HTS dataset

**Return type**  HtsPandas

**thunor.io.read_vanderbilt_hts(file_or_source, plate_width=24, plate_height=16, sep=None, _unstacked=False)**
Read a Vanderbilt HTS format file

See the wiki for a file format description

**Parameters**

- **file_or_source** *(str or object)* – Source for CSV data
- **plate_width** *(int)* – Width of the microtiter plates (default: 24, for 384 well plate)
- **plate_height** *(int)* – Width of the microtiter plates (default: 16, for 384 well plate)
- **sep** *(str)* – Source file delimiter (default: detect from file extension)

**Returns**  HTS Dataset containing the data read from the CSV

**Return type**  HtsPandas

**thunor.io.write_hdf(df_data, filename, dataset_format='fixed')**
Save a dataset to Thunor HDF5 format

**Parameters**

- **df_data** *(HtsPandas)* – HTS dataset
- **filename** *(str)* – Output filename
dataset_format (str) – One of ‘fixed’ or ‘table’. See pandas HDFStore docs for details

thunor.io.write_vanderbilt_hgs (df_data, filename, plate_width=24, plate_height=16, sep=None)

Read a Vanderbilt HTS format file
See the wiki for a file format description

Parameters

• df_data (HtsPandas) – HtsPandas - HTS dataset
• filename (str or object) – filename or buffer to write into
• plate_width (int) – plate width (number of wells)
• plate_height (int) – plate height (number of wells)
• sep (str) – Source file delimiter (default: detect from file extension)

3.2 DIP calculations and statistics (thunor.dip)

thunor.dip.adjusted_r_squared (r, n, p)
Calculate adjusted r-squared value from r value

Parameters

• r (float) – r value (between 0 and 1)
• n (int) – number of sample data points
• p (int) – number of free parameters used in fit

Returns Adjusted r-squared value
Return type float

thunor.dip.ctrl_dip_rates (df_controls)
Calculate control DIP rates

Parameters df_controls (pd.DataFrame) – Pandas DataFrame of control cell counts from a thunor.io.HtsPandas object

Returns Fitted control DIP rate values
Return type pd.DataFrame

thunor.dip.dip_rates (df_data, selector_fn=<function tyson1>)
Calculate DIP rates on a dataset

Parameters

• df_data (thunor.io.HtsPandas) – Thunor HTS dataset
• selector_fn (function) – Selection function for choosing optimal DIP rate fit (default: tyson1)

Returns Two entry list, giving control DIP rates and experiment (non-control) DIP rates (both as Pandas DataFrames)
Return type list

thunor.dip.expt_dip_rates (df_doses, df_vals, selector_fn=<function tyson1>)
Calculate experiment (non-control) DIP rates
Parameters

- **df_doses** *(pd.DataFrame)* – Pandas DataFrame of dose values from a *thunor.io HtsPandas* object
- **df_vals** *(pd.DataFrame)* – Pandas DataFrame of cell counts from a *thunor.io HtsPandas* object
- **selector_fn** *(function)* – Selection function for choosing optimal DIP rate fit (default: *tyson1()*)

Returns  Fitted DIP rate values

Return type  pd.DataFrame

```
thunor.dip.tyson1(adj_r_sq, rmse, n)
```

Tyson1 algorithm for selecting optimal DIP rate fit

Parameters

- **adj_r_sq** *(float)* – Adjusted r-squared value
- **rmse** *(float)* – Root mean squared error of fit
- **n** *(int)* – Number of data points used in fit

Returns  Fit value (higher is better)

Return type  float

### 3.3 Viability calculations and statistics *(thunor.viability)*

```
thunor.viability.viability(df_data, time_hrs=72, assay_name=None, include_controls=True)
```

Calculate viability at the specified time point

Viability is calculated as the assay value over the mean of controls from the same plate, cell line, and time point

Parameters

- **df_data** *(HtsPandas)* – HTS dataset
- **time_hrs** *(float)* – Time in hours to use for viability. The closest time point in each well to the one specified is used.
- **assay_name** *(str, optional)* – The assay name to use for viability calculation, or None to use the default proliferation assay
- **include_controls** *(bool)* – Return the control values for reference as a the second entry in a two-tuple, if True

Returns  A DataFrame containing the viability results and a Series containing the control values, if requested (None is returned as the second return value otherwise)

Return type  pd.DataFrame, pd.Series or None

3.3. Viability calculations and statistics *(thunor.viability)* 13
3.4 Dose Response Curve Fitting (thunor.curve_fit)

exception thunor.curve_fit.AAFitWarning
exception thunor.curve_fit.AUCFitWarning
exception thunor.curve_fit.DrugCombosNotImplementedError
   This function does not support drug combinations yet

class thunor.curve_fit.HillCurve
   Base class defining Hill/log-logistic curve functionality

   null_response_fn (axis=None, dtype=None, out=None, keepdims=<no value>)
   Compute the arithmetic mean along the specified axis.

   Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.

   Parameters

   • a (array_like) – Array containing numbers whose mean is desired. If a is not an array, a conversion is attempted.

   • axis (None or int or tuple of ints, optional) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.

   New in version 1.7.0.

   If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.

   • dtype (data-type, optional) – Type to use in computing the mean. For integer inputs, the default is float64; for floating point inputs, it is the same as the input dtype.

   • out (ndarray, optional) – Alternate output array in which to place the result. The default is None; if provided, it must have the same shape as the expected output, but the type will be cast if necessary. See ufuncs-output-type for more details.

   • keepdims (bool, optional) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

   If the default value is passed, then keepdims will not be passed through to the mean method of sub-classes of ndarray, however any non-default value will be. If the sub-class’ method does not implement keepdims any exceptions will be raised.

   Returns m – If out=None, returns a new array containing the mean values, otherwise a reference to the output array is returned.

   Return type ndarray, see dtype parameter above

   See also:

   average Weighted average

   std, var, nanmean, nanstd, nanvar
Notes

The arithmetic mean is the sum of the elements along the axis divided by the number of elements.

Note that for floating-point input, the mean is computed using the same precision the input has. Depending on the input data, this can cause the results to be inaccurate, especially for float32 (see example below). Specifying a higher-precision accumulator using the dtype keyword can alleviate this issue.

By default, float16 results are computed using float32 intermediates for extra precision.

Examples

```python
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([2., 3.])
>>> np.mean(a, axis=1)
array([1.5, 3.5])
```

In single precision, mean can be inaccurate:

```python
>>> a = np.zeros((2, 512*512), dtype=np.float32)
>>> a[0, :] = 1.0
>>> a[1, :] = 0.1
>>> np.mean(a)
0.54999924
```

Computing the mean in float64 is more accurate:

```python
>>> np.mean(a, dtype=np.float64)
0.55000000074505806 # may vary
```

class thunor.curve_fit.HillCurveLL2(popt)

classmethod fit_fn(x, b, e)

Two parameter log-logistic function (“Hill curve”)

Parameters

- x (np.ndarray) – One-dimensional array of “x” values
- b (float) – Hill slope
- e (float) – EC50 value

Returns

Array of “y” values using the supplied curve fit parameters on “x”

Return type

np.ndarray

classmethod initial_guess(x, y)

Heuristic function for initial fit values

Uses the approach followed by R’s drc library: https://cran.r-project.org/web/packages/drc/index.html

Parameters

- x (np.ndarray) – Array of “x” (dose) values
- y (np.ndarray) – Array of “y” (response) values

3.4. Dose Response Curve Fitting (thunor.curve_fit)
Returns: Four-valued list corresponding to initial estimates of the parameters defined in the `ll4()` function.

Return type: list

```python
class thunor.curve_fit.HillCurveLL3u(popt)
    Three parameter log logistic curve, for viability data
```

classmethod `fit_fn(x, b, c, e)`
Three parameter log-logistic function (“Hill curve”)

Parameters:
- `x` (*np.ndarray*) – One-dimensional array of “x” values
- `b` (*float*) – Hill slope
- `c` (*float*) – Maximum response (lower plateau)
- `e` (*float*) – EC50 value

Returns: Array of “y” values using the supplied curve fit parameters on “x”

Return type: *np.ndarray*

```python
classmethod initial_guess(x, y)
    Heuristic function for initial fit values
```

Uses the approach followed by R’s drc library: https://cran.r-project.org/web/packages/drc/index.html

Parameters:
- `x` (*np.ndarray*) – Array of “x” (dose) values
- `y` (*np.ndarray*) – Array of “y” (response) values

Returns: Four-valued list corresponding to initial estimates of the parameters defined in the `ll4()` function.

Return type: list

```python
static null_response_fn(_)
    Compute the arithmetic mean along the specified axis.
```

Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. *float64* intermediate and return values are used for integer inputs.

Parameters:
- `a` (*array_like*) – Array containing numbers whose mean is desired. If `a` is not an array, a conversion is attempted.
- `axis` (*None or int or tuple of ints, optional*) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. New in version 1.7.0.
  If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.
- `dtype` (*data-type, optional*) – Type to use in computing the mean. For integer inputs, the default is *float64*; for floating point inputs, it is the same as the input `dtype`.
- `out` (*ndarray, optional*) – Alternate output array in which to place the result. The default is *None*; if provided, it must have the same shape as the expected output, but the type will be cast if necessary. See *ufuncs-output-type* for more details.
• **keepdims** *(bool, optional)* – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

If the default value is passed, then *keepdims* will not be passed through to the *mean* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class’ method does not implement *keepdims* any exceptions will be raised.

**Returns**  
- m – If *out=None*, returns a new array containing the mean values, otherwise a reference to the output array is returned.

**Return type**  
*ndarray*, see dtype parameter above

**See also:**  
- *average*  
- *std, var, nanmean, nanstd, nanvar*

**Notes**

The arithmetic mean is the sum of the elements along the axis divided by the number of elements.

Note that for floating-point input, the mean is computed using the same precision the input has. Depending on the input data, this can cause the results to be inaccurate, especially for *float32* (see example below). Specifying a higher-precision accumulator using the *dtype* keyword can alleviate this issue.

By default, *float16* results are computed using *float32* intermediates for extra precision.

**Examples**

```python
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([2., 3.])
>>> np.mean(a, axis=1)
array([1.5, 3.5])
```

In single precision, *mean* can be inaccurate:

```python
>>> a = np.zeros((2, 512*512), dtype=np.float32)
>>> a[0, :] = 1.0
>>> a[1, :] = 0.1
>>> np.mean(a)
0.54999924
```

Computing the mean in *float64* is more accurate:

```python
>>> np.mean(a, dtype=np.float64)
0.55000000074505806 # may vary
```

**class**  
*thunor.curve_fit.HillCurveLL4*(popt)*

**aa**(min_conc, max_conc)  
Find the activity area (area over the curve)
Parameters

- \texttt{min\_conc}(float) – Minimum concentration to consider for fitting the curve
- \texttt{max\_conc}(float) – Maximum concentration to consider for fitting the curve

Returns Activity area value

Return type float

\texttt{auc}(\texttt{min\_conc})

Find the area under the curve

Parameters \texttt{min\_conc}(float) – Minimum concentration to consider for fitting the curve

Returns Area under the curve (AUC) value

Return type float

\texttt{ec}(\texttt{ec\_num}=50)

Find the effective concentration value (e.g. IC50)

Parameters \texttt{ec\_num}(int) – EC number between 0 and 100 (response level)

Returns Effective concentration value for requested response value

Return type float

classmethod \texttt{fit\_fn}(x, b, c, d, e)

Four parameter log-logistic function (“Hill curve”)

Parameters

- \texttt{x}(\texttt{np.ndarray}) – One-dimensional array of “x” values
- \texttt{b}(float) – Hill slope
- \texttt{c}(float) – Maximum response (lower plateau)
- \texttt{d}(float) – Minimum response (upper plateau)
- \texttt{e}(float) – EC50 value

Returns Array of “y” values using the supplied curve fit parameters on “x”

Return type \texttt{np.ndarray}

\texttt{ic}(\texttt{ic\_num}=50)

Find the inhibitory concentration value (e.g. IC50)

Parameters \texttt{ic\_num}(int) – IC number between 0 and 100 (response level)

Returns Inhibitory concentration value for requested response value

Return type float

classmethod \texttt{initial\_guess}(x, y)

Heuristic function for initial fit values

Uses the approach followed by R’s drc library: https://cran.r-project.org/web/packages/drc/index.html

Parameters

- \texttt{x}(\texttt{np.ndarray}) – Array of “x” (dose) values
- \texttt{y}(\texttt{np.ndarray}) – Array of “y” (response) values

Returns Four-valued list corresponding to initial estimates of the parameters defined in the \texttt{ll4()} function.
Return type  list
class  thunor.curve_fit.HillCurveNull (popt)

tahunor.curve_fit.ValueWarning

tahunor.curve_fit.aa_obs (responses, doses=None)
Activity Area (observed)

Parameters

• responses (np.array or pd.Series) – Response values, with dose values in the
  Index if a Series is supplied

• doses (np.array or None) – Dose values - only required if responses is not a
  pd.Series

Returns  Activity area (observed)

Return type  float

tahunor.curve_fit.fit_drc (doses, responses, response_std_errs=None, fit_cls=<class
  'thunor.curve_fit.HillCurveLL4'>, null_rejection_threshold=0.05,
  ctrl_dose_test=False)
Fit a dose response curve

Parameters

• doses (np.ndarray) – Array of dose values

• responses (np.ndarray) – Array of response values, e.g. viability, DIP rates

• response_std_errs (np.ndarray, optional) – Array of fit standard errors for
  the response values

• fit_cls (Class) – Class to use for fitting (default: 4 parameter log logistic “Hill” curve)

• null_rejection_threshold (float, optional) – p-value for rejecting curve fit against no effect “flat” response model by F-test (default: 0.05). Set to None to skip test.

• ctrl_dose_test (boolean) – If True, the minimum dose is assumed to represent control values (in DIP rate curves), and will reject fits where E0 is greater than a standard deviation higher than the mean of the control response values. Leave as False to skip the test.

Returns  A HillCurve object containing the fit parameters

Return type  HillCurve

tahunor.curve_fit.fit_params (ctrl_data, expt_data, fit_cls=<class
  'thunor.curve_fit.HillCurveLL4'>, ctrl_dose_fn=<function
  <lambda>>)
Fit dose response curves to DIP rates or viability data

This method computes parameters including IC50, EC50, AUC, AA, Hill coefficient, and Emax. For a faster
version, see fit_params_minimal().

Parameters

• ctrl_data (pd.DataFrame or None) – Control DIP rates from dip_rates() or
  ctrl_dip_rates(). Set to None to not use control data.

• expt_data (pd.DataFrame) – Experiment (non-control) DIP rates from
  dip_rates() or expt_dip_rates(), or viability data from viability()

• fit_cls (Class) – Class to use for curve fitting (default: HillCurveLL4())

3.4. Dose Response Curve Fitting (thunor.curve_fit)
• **ctrl_dose_fn** (*function*) – Function to use to set an effective “dose” (non-zero) for controls. Takes the list of experiment doses as an argument.

**Returns**  DataFrame containing DIP rate curve fits and parameters

**Return type**  pd.DataFrame

```python
def ctrl_dose_fn(doses):
    # Function to set an effective "dose" for controls.
    # Takes the list of experiment doses as an argument.
    pass
```

**thunor.curve_fit.fit_params_from_base**  (base_params, 

```python
def fit_params_from_base(base_params, ctrl_resp_data=None, 
                        expt_resp_data=None, 
                        ctrl_dose_fn=<function <lambda>>, 
                        custom_ic_concentrations=frozenset([]), 
                        custom_ec_concentrations=frozenset([]), 
                        custom_e_values=frozenset([]), 
                        custom_e_rel_values=frozenset([]), 
                        include_aa=False, 
                        include_auc=False, 
                        include_hill=False, 
                        include_emax=False, 
                        include_einf=False, 
                        include_response_values=True)
```

Attach additional parameters to basic set of fit parameters

**thunor.curve_fit.fit_params_minimal**  (ctrl_data, 

```python
def fit_params_minimal(ctrl_data, expt_data, fit_cls=<class 'thunor.curve_fit.HillCurveLL4'>, 
                       ctrl_dose_fn=<function <lambda>>, 
                       fit_cls=<class 'thunor.curve_fit.HillCurveLL4'>)
```

Fit dose response curves to DIP or viability, and calculate statistics

This function only fits curves and stores basic fit parameters. Use *fit_params()* for more statistics and parameters.

**Parameters**

- **ctrl_data**  (*pd.DataFrame or None*) – Control DIP rates from dip_rates() or ctrl_dip_rates(). Set to None to not use control data.
- **expt_data**  (*pd.DataFrame*) – Experiment (non-control) DIP rates from dip_rates() or expt_dip_rates()
- **fit_cls**  (*Class*) – Class to use for curve fitting (default: *HillCurveLL4()*)
- **ctrl_dose_fn**  (*function*) – Function to use to set an effective “dose” (non-zero) for controls. Takes the list of experiment doses as an argument.

**Returns**  DataFrame containing DIP rate curve fits and parameters

**Return type**  pd.DataFrame

```python
def fit_params_minimal(ctrl_data, expt_data, fit_cls=<class 'thunor.curve_fit.HillCurveLL4'>, 
                       ctrl_dose_fn=<function <lambda>>, 
                       fit_cls=<class 'thunor.curve_fit.HillCurveLL4'>)
```

**thunor.curve_fit.is_param_truncated**  (df_params, param_name)

Checks if parameter values are truncated at boundaries of measured range

**Parameters**

- **df_params**  (*pd.DataFrame*) – DataFrame of DIP curve fits with parameters from *fit_params()

- **param_name**  (*str*) – Name of a parameter, e.g. ‘ic50’

**Returns**  Array of booleans showing whether each entry in the DataFrame is truncated

**Return type**  np.ndarray

```python
def is_param_truncated(df_params, param_name)
```

---

20  Chapter 3. Thunor Core Modules Reference
3.5 Plots and visualization (thunor.plots)

exception thunor.plots.CannotPlotError

thunor.plots.plot_ctrl_cell_counts_by_plate(df_controls, title=None, subtitle=None, template='none')

Parameters
- df_controls (pd.DataFrame) – Control well cell counts
- title (str, optional) – Title (or None to auto-generate)
- subtitle (str, optional) – Subtitle (or None to auto-generate)
- template (str) – Name of plotly template (https://plot.ly/python/templates/)

Returns A plotly figure object containing the graph
Return type plotly.graph_objs.Figure

thunor.plots.plot_ctrl_dip_by_plate(df_controls, title=None, subtitle=None, template='none')

Parameters
- df_controls (pd.DataFrame) – Control well DIP values
- title (str, optional) – Title (or None to auto-generate)
- subtitle (str, optional) – Subtitle (or None to auto-generate)
- template (str) – Name of plotly template (https://plot.ly/python/templates/)

Returns A plotly figure object containing the graph
Return type plotly.graph_objs.Figure

thunor.plots.plot_drc(fit_params, is_absolute=False, color_by=None, color_groups=None, title=None, subtitle=None, template='none')

Plot dose response curve fits

Parameters
- fit_params (pd.DataFrame) – Fit parameters from thunor.curve_fit.fit_params()
- is_absolute (bool) – For DIP rate plots, use absolute (True) or relative (False) y-axis scale. Ignored for viability plots.
- color_by (str or None) – Color the traces by cell lines if ‘cl’, drugs if ‘dr’, or arbitrarily if None (default)
- color_groups (dict or None) – If using color_by, provide a dictionary containing the color groups, where the values are cell line or drug names
- title (str, optional) – Title (or None to auto-generate)
- subtitle (str, optional) – Subtitle (or None to auto-generate)
- template (str) – Name of plotly template (https://plot.ly/python/templates/)

Returns A plotly figure object containing the graph
Return type plotly.graph_objs.Figure
thunor.plots.plot_drc_params(df_params, fit_param, fit_param_compare=None, fit_param_sort=None, title=None, subtitle=None, aggregate_cell_lines=False, aggregate_drugs=False, multi_dataset=False, color_by=None, color_groups=None, template='none', **kwargs)

Box, bar, or scatter plots of DIP rate fit parameters

Parameters

- **df_params** (pd.DataFrame) – DIP fit parameters from thunor.dip.dip_params()
- **fit_param** (str) – Fit parameter name, e.g. ‘ic50’
- **fit_param_compare** (str, optional) – Second fit parameter name for comparative plots, e.g. ‘ec50’
- **fit_param_sort** (str, optional) – Fit parameter name to use for sorting the x-axis, if different from fit_param
- **title** (str, optional) – Title (or None to auto-generate)
- **subtitle** (str, optional) – Subtitle (or None to auto-generate)
- **aggregate_cell_lines** (bool or dict, optional) – Aggregate all cell lines (if True), or aggregate by the specified groups (dict of cell line names as values, with group labels as keys)
- **aggregate_drugs** (bool or dict, optional) – Aggregate all drugs (if True), or aggregate by the specified groups (dict of drug names as values, with group labels as keys)
- **multi_dataset** (bool) – Set to true to compare two datasets contained in fit_params
- **color_by** (str or None) – Color by cell lines if “cl”, drugs if “dr”, or arbitrarily if None (default)
- **color_groups** (dict or None) – Groups of cell lines of drugs to color by
- **template** (str) – Name of plotly template (https://plot.ly/python/templates/)
- **kwargs** (dict, optional) – Additional keyword arguments

Returns A plotly figure object containing the graph

Return type plotly.graph_objs.Figure

thunor.plots.plot_drug_combination_heatmap(ctrl_resp_data, expt_resp_data, title=None, subtitle=None, template='none')

Plot heatmap of drug combination response by DIP rate

Two dimensional plot (each dimension is a drug concentration) where squares are coloured by DIP rate value.

Parameters

- **ctrl_resp_data** (pd.DataFrame) – Control DIP rates from thunor.dip.dip_rates()
- **expt_resp_data** (pd.DataFrame) – Experiment (non-control) DIP rates from thunor.dip.dip_rates()
- **title** (str, optional) – Title (or None to auto-generate)
- **subtitle** (str, optional) – Subtitle (or None to auto-generate)
- **template** (str) – Name of plotly template (https://plot.ly/python/templates/)
3.5. Plots and visualization (thunor.plots)
3.6 Miscellaneous “helper” functions (thunor.helpers)

thunor.helpers.format_dose(num, sig_digits=12, array_as_string=None)

Format a numeric dose like 1.2e-9 into 1.2 nM

Parameters

• num (float or np.ndarray) – Dose value, or array of such

• sig_digits (int) – Number of significant digits to include

• array_as_string (str, optional) – Combine array into a single string using the supplied join string. If not supplied, a list of strings is returned.

Returns Formatted dose values

Return type str or list of str

thunor.helpers.plotly_to_dataframe(plot_fig)

Extract data from a plotly figure into a pandas DataFrame

Parameters plot_fig (plotly.graph_objs.Figure) – A plotly figure object

Returns A pandas DataFrame containing the extracted traces from the figure

Return type pd.DataFrame

3.7 Conversion tools for external formats and databases (thunor.converters)

thunor.converters.convert_ctrp(directory='.', output_file='ctrp_v2.h5')

Convert CTRP v2.0 data to Thunor format

CTRP is the Cancer Therapeutics Response Portal, a project which has generated a large quantity of viability data.

The data are freely available from the CTD2 Data Portal:

The required files can be downloaded from their FTP server:

You’ll need to download and extract the following file:

• “CTRPv2.0_2015_ctd2_ExpandedDataset.zip”

Please note that the layout of wells in each plate after conversion is arbitrary, since this information is not in the original files.

Please make sure you have the “tables” python package installed, in addition to the standard Thunor Core requirements.

You can run this function at the command line to convert the files; assuming the two files are in the current directory, simply run:
This script will take several minutes to run, please be patient. It is also resource-intensive, due to the size of the
dataset. We recommend you utilize the highest-spec machine that you have available.

This will output a file called (by default) ctrp_v2.h5, which can be opened with thunor.io.
read_hdf(), or used with Thunor Web.

Parameters

- **directory** *(str)* – Directory containing the extracted CTRP v2.0 dataset
- **output_file** *(str)* – Filename of output file (Thunor HDF5 format)

```python
thunor.converters.convert_ctrp()
```

Convert GDSC data to Thunor format

GDSC is the Genomics of Drug Sensitivity in Cancer, a project which has generated a large quantity of viability
data.

The data are freely available under the license agreement described on their website:
https://www.cancerrxgene.org/downloads

The required files can be downloaded from here:
ftp://ftp.sanger.ac.uk/pub/project/cancerrxgene/releases/release-6.0/

You’ll need to download two files to convert to Thunor format:
- The list of drugs, “Screened_Compounds.xlsx”
- Sensitivity data, “v17a_public_raw_data.xlsx”

Please note that the layout of wells in each plate after conversion is arbitrary, since this information is not in the
original files.

Please make sure you have the “tables” and “xlrd” python packages installed, in addition to the standard Thunor
Core requirements.

You can run this function at the command line to convert the files; assuming the two files are in the current
directory, simply run:

```python
python -c "from thunor.converters import convert_gdsc; convert_gdsc()
```

This script will take several minutes to run, please be patient. It is also resource-intensive, due to the size of the
dataset. We recommend you utilize the highest-spec machine that you have available.

This will output a file called (by default) gdsc-v17a.h5, which can be opened with thunor.io.
read_hdf(), or used with Thunor Web.

Parameters

- **drug_list_file** *(str)* – Filename of GDSC list of drugs, to convert drug IDs to names
- **screen_data_file** *(str)* – Filename of GDSC sensitivity data
- **output_file** *(str)* – Filename of output file (Thunor HDF5 format)

```python
thunor.converters.convert_gdsc(drug_list_file='Screened_Compounds.xlsx',
screen_data_file='v17a_public_raw_data.xlsx',
output_file='gdsc-v17a.h5')
```

Convert GDSC cell line tissue descriptors to Thunor tags

```python
thunor.converters.convert_gdsc_tags(cell_line_file='Cell_Lines_Details.xlsx',
output_file='gdsc_cell_line_primary_site_tags.txt')
```

3.7. Conversion tools for external formats and databases *(thunor.converters)*
GDSC is the Genomics of Drug Sensitivity in Cancer, a project which has generated a large quantity of viability data.

The data are freely available under the license agreement described on their website: https://www.cancerrxgene.org/downloads

The required files can be downloaded from here: ftp://ftp.sanger.ac.uk/pub/project/cancerrxgene/releases/release-6.0/

You’ll need to download one file:

- Cell line details, “Cell_Lines_Details.xlsx”

You can run this function at the command line to convert the files; assuming the downloaded file is in the current directory, simply run:

```
python -c "from thunor.converters import convert_gdsc_tags; convert_gdsc_tags()"
```

This will output a file called (by default) `gdsc_cell_line_primary_site_tags.txt`, which can be loaded into Thunor Web using the “Upload cell line tags” function.

**Parameters**

- `cell_line_file` (str) – Filename of GDSC cell line details (Excel .xlsx format)
- `output_file` (str) – Filename of output file (tab separated values format)

```
thunor.converters.convert_gdsc_tags(directory='.', output_file='gdsc_cell_line_primary_site_tags.txt')
```

Convert Teicher data to Thunor format

The “Teicher” data is a dataset of dose-response data on a panel of small cell lung cancer (SCLC) cell lines. The data can be downloaded from the following link (select the Compound Concentration/Response Data link):

https://sclccelllines.cancer.gov/sclc/downloads.xhtml

Unzip the downloaded file. The dataset can then be converted on the command line:

```
python -c "from thunor.converters import convert_teicher; convert_teicher()"
```

Please note that the layout of wells in each plate after conversion is arbitrary, since this information is not in the original files.

This will output a file called (by default) `teicher.h5`, which can be opened with `thunor.io.read_hdf()`, or used with Thunor Web.

**Parameters**

- `directory` (str) – Directory containing the Teicher dataset
- `output_file` (str) – Filename of output file (Thunor HDF5 format)
• genindex
• modindex
PYTHON MODULE INDEX

t
  thunor.converters, 24
  thunor.curve_fit, 14
  thunor.dip, 12
  thunor.helpers, 24
  thunor.io, 9
  thunor.plots, 21
  thunor.viability, 13
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>aa() (thunor.curve_fit.HillCurveLL4 method)</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>aa_obs() (in module thunor.curve_fit)</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>AAFitWarning</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>adjusted_r_squared() (in module thunor.dip)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>assay_names (thunor.io.HtsPandas attribute)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>auc() (thunor.curve_fit.HillCurveLL4 method)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>AUCFitWarning</td>
<td>14</td>
</tr>
<tr>
<td>C</td>
<td>CannotPlotError</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>cell_lines (thunor.io.HtsPandas attribute)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>col_iterator() (thunor.io.PlateMap method)</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>convert_ctrp() (in module thunor.converters)</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>convert_gdsc() (in module thunor.converters)</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>convert_gdsc_tags() (in module thunor.converters)</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>ctrl_dip_rates() (in module thunor.dip)</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>dip_assay_name (thunor.io.HtsPandas attribute)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>dip_rates() (in module thunor.dip)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>doses_unstacked() (thunor.io.HtsPandas method)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>DrugCombosNotImplementedError</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>drugs (thunor.io.HtsPandas attribute)</td>
<td>9</td>
</tr>
<tr>
<td>E</td>
<td>ec() (thunor.curve_fit.HillCurveLL4 method)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>expt_dip_rates() (in module thunor.dip)</td>
<td>12</td>
</tr>
<tr>
<td>F</td>
<td>filter() (thunor.io.HtsPandas method)</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>fit_drc() (in module thunor.curve_fit)</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>fit_fn() (thunor.curve_fit.HillCurveLL2 class method)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>fit_fn() (thunor.curve_fit.HillCurveLL3u class method)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>fit_fn() (thunor.curve_fit.HillCurveLL4 class method)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>fit_params() (in module thunor.curve_fit)</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>fit_params_from_base() (in module thunor.curve_fit)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>fit_params_minimal() (in module thunor.curve_fit)</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>format_dose() (in module thunor.helpers)</td>
<td>24</td>
</tr>
<tr>
<td>H</td>
<td>HillCurve (class in thunor.curve_fit)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>HillCurveLL2 (class in thunor.curve_fit)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>HillCurveLL3u (class in thunor.curve_fit)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>HillCurveLL4 (class in thunor.curve_fit)</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>HillCurveNull (class in thunor.curve_fit)</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>HtsPandas (class in thunor.io)</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>ic() (thunor.curve_fit.HillCurveLL4 method)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>initial_guess() (thunor.curve_fit.HillCurveLL2 class method)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>initial_guess() (thunor.curve_fit.HillCurveLL3u class method)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>initial_guess() (thunor.curve_fit.HillCurveLL4 class method)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>is_param_truncated() (in module thunor.curve_fit)</td>
<td>20</td>
</tr>
<tr>
<td>M</td>
<td>module thunor.converters</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>thunor.curve_fit</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>thunor.dip</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>thunor.helpers</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>thunor.io</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>thunor.plots</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>thunor.viability</td>
<td>13</td>
</tr>
<tr>
<td>N</td>
<td>null_response_fn() (thunor.curve_fit.HillCurve method)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>null_response_fn() (thunor.curve_fit.HillCurveLL3u static method)</td>
<td>16</td>
</tr>
</tbody>
</table>
num_wells() (thunor.io.PlateMap property), 10

P
plate() (thunor.io.HtsPandas method), 10
plate_size_from_num_wells() (thunor.io.PlateMap class method), 10
PlateData (class in thunor.io), 10
PlateFileParseException, 10
PlateMap (class in thunor.io), 10
plot_ctrl_cell_counts_by_plate() (in module thunor.plots), 21
plot_ctrl_dip_by_plate() (in module thunor.plots), 21
plot_drc() (in module thunor.plots), 21
plot_drc_params() (in module thunor.plots), 21
plot_drug_combination_heatmap() (in module thunor.plots), 22
plot_plate_map() (in module thunor.plots), 23
plot_time_course() (in module thunor.plots), 23
plot_two_dataset_param_scatter() (in module thunor.plots), 23
plotly_to_dataframe() (in module thunor.helpers), 24

R
read_hdf() (in module thunor.io), 11
read_vanderbilt_hts() (in module thunor.io), 11
row_iterator() (thunor.io.PlateMap method), 10

T
thunor.converters module, 24
thunor.curve_fit module, 14
thunor.dip module, 12
thunor.helpers module, 24
thunor.io module, 9
thunor.plots module, 21
thunor.viability module, 13
tyson1() (in module thunor.dip), 13

V
ValueWarning, 19
viability() (in module thunor.viability), 13

W
well_id_to_name() (thunor.io.PlateMap method), 10
well_iterator() (thunor.io.PlateMap method), 10
well_list() (thunor.io.PlateMap method), 11
well_name_to_id() (thunor.io.PlateMap method), 11
write_hdf() (in module thunor.io), 11
write_vanderbilt_hts() (in module thunor.io), 12