welltestpy Documentation

Release 1.1.1.dev25

Sebastian Müller, Jarno Herrmann

Apr 18, 2023

CONTENTS

CHAPTER 1

WELCOME TO WELLTESTPY

1.1 Purpose

welltestpy provides a framework to handle, process, plot and analyse data from well based field campaigns.

1.2 Installation

You can install the latest version with the following command:

```
pip install welltestpy
```

Or from conda

```
conda install -c conda-forge welltestpy
```

1.3 Documentation for welltestpy

You can find the documentation including tutorials and examples under https://welltestpy.readthedocs.io.

1.4 Citing welltestpy

If you are using this package you can cite our Groundwater publication by:

> Müller, S., Leven, C., Dietrich, P., Attinger, S. and Zech, A. (2021): How to Find Aquifer Statistics Utilizing Pumping Tests? Two Field Studies Using welltestpy. Groundwater, https://doi.org/10.1111/gwat.13121

To cite the code, please visit the Zenodo page.

1.5 Provided Subpackages

```
welltestpy.data      # Subpackage to handle data from field campaigns
welltestpy.estimate  # Subpackage to estimate field parameters
welltestpy.process   # Subpackage to pre- and post-process data
welltestpy.tools     # Subpackage with tools for plotting and triagulation
```

1.6 Requirements

- NumPy >= 1.14.5
- SciPy >= 1.1.0
- AnaFlow >= 1.0.0
- SpotPy >= 1.5.0
- Matplotlib >= 3.0.0

1.7 Contact

You can contact us via info@geostat-framework.org.

1.8 License

MIT

CHAPTER 2

WELLTESTPY TUTORIAL

In the following you will find several Tutorials on how to use welltestpy to explore its whole beauty and power.

2.1 Gallery

Creating a pumping test campaign

In the following we are going to create an artificial pumping test campaign on a field site.

```
import anaflow as ana
import numpy as np

import welltestpy as wtp
```

Create the field-site and the campaign

```
field = wtp.FieldSite(name="UFZ", coordinates=[51.353839, 12.431385])
campaign = wtp.Campaign(name="UFZ-campaign", fieldsite=field)
```

Add 4 wells to the campaign

```
campaign.add_well(name="well_0", radius=0.1, coordinates=(0.0, 0.0))
campaign.add_well(name="well_1", radius=0.1, coordinates=(1.0, -1.0))
campaign.add_well(name="well_2", radius=0.1, coordinates=(2.0, 2.0))
campaign.add_well(name="well_3", radius=0.1, coordinates=(-2.0, -1.0))
```

Generate artificial drawdown data with the Theis solution

```
rate = -1e-4
time = np.geomspace(10, 7200, 10)
transmissivity = 1e-4
storage = 1e-4
rad = [
    campaign.wells["well_0"].radius,  # well radius of well_0
    campaign.wells["well_0"] - campaign.wells["well_1"],  # distance 0-1
    campaign.wells["well_0"] - campaign.wells["well_2"],  # distance 0-2
    campaign.wells["well_0"] - campaign.wells["well_3"],  # distance 0-3
]
```

(continues on next page)

(continued from previous page)

```
drawdown = ana.theis(
    time=time,
    rad=rad,
    storage=storage,
    transmissivity=transmissivity,
    rate=rate,
)
```

Create a pumping test at well_0

```
pumptest = wtp.PumpingTest(
    name="well_0",
    pumpingwell="well_0",
    pumpingrate=rate,
    description="Artificial pump test with Theis",
)
```

Add the drawdown observation at the 4 wells

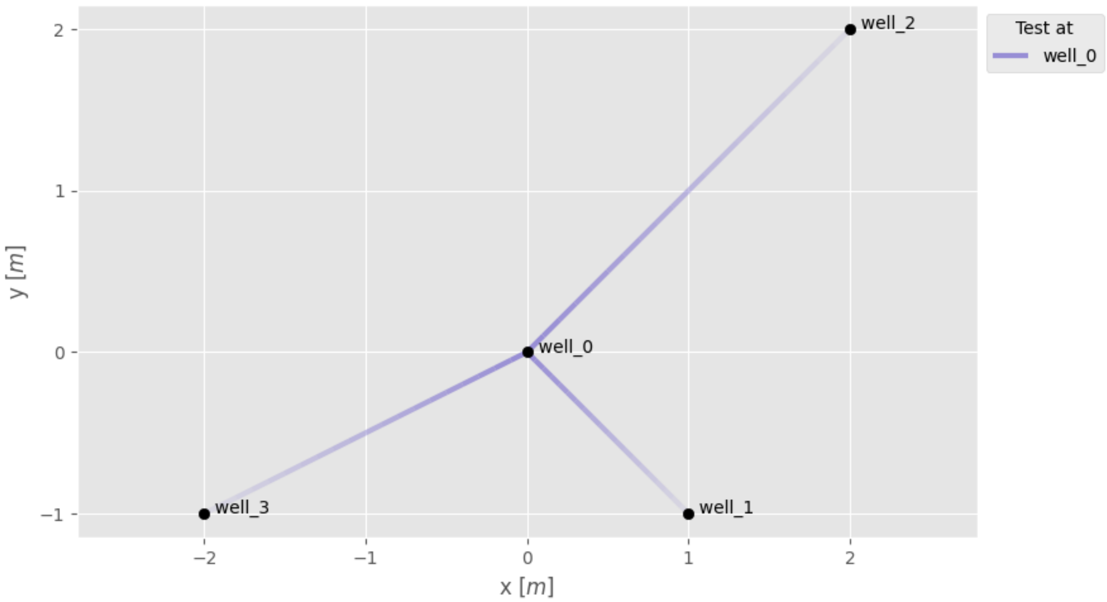
```
pumptest.add_transient_obs("well_0", time, drawdown[:, 0])
pumptest.add_transient_obs("well_1", time, drawdown[:, 1])
pumptest.add_transient_obs("well_2", time, drawdown[:, 2])
pumptest.add_transient_obs("well_3", time, drawdown[:, 3])
```

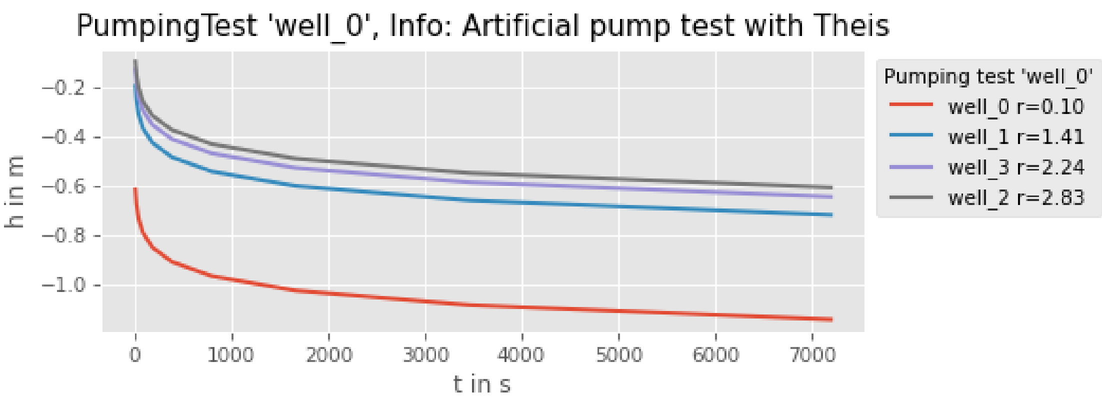
Add the pumping test to the campaign

```
campaign.addtests(pumptest)
# optionally make the test (quasi)steady
# campaign.tests["well_0"].make_steady()
```

Plot the well constellation and a test overview

```
campaign.plot_wells()
campaign.plot()
```

- 

- 

Save the whole campaign to a file

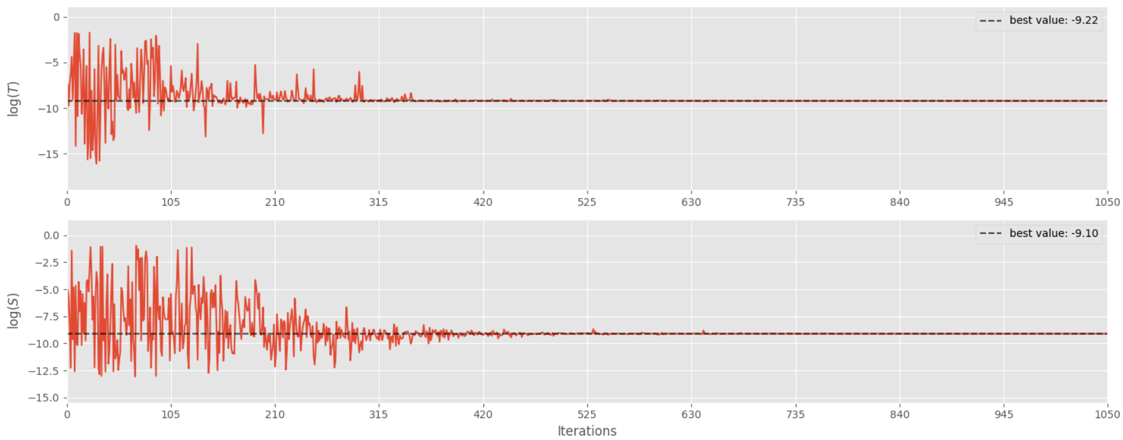
```
campaign.save()
```

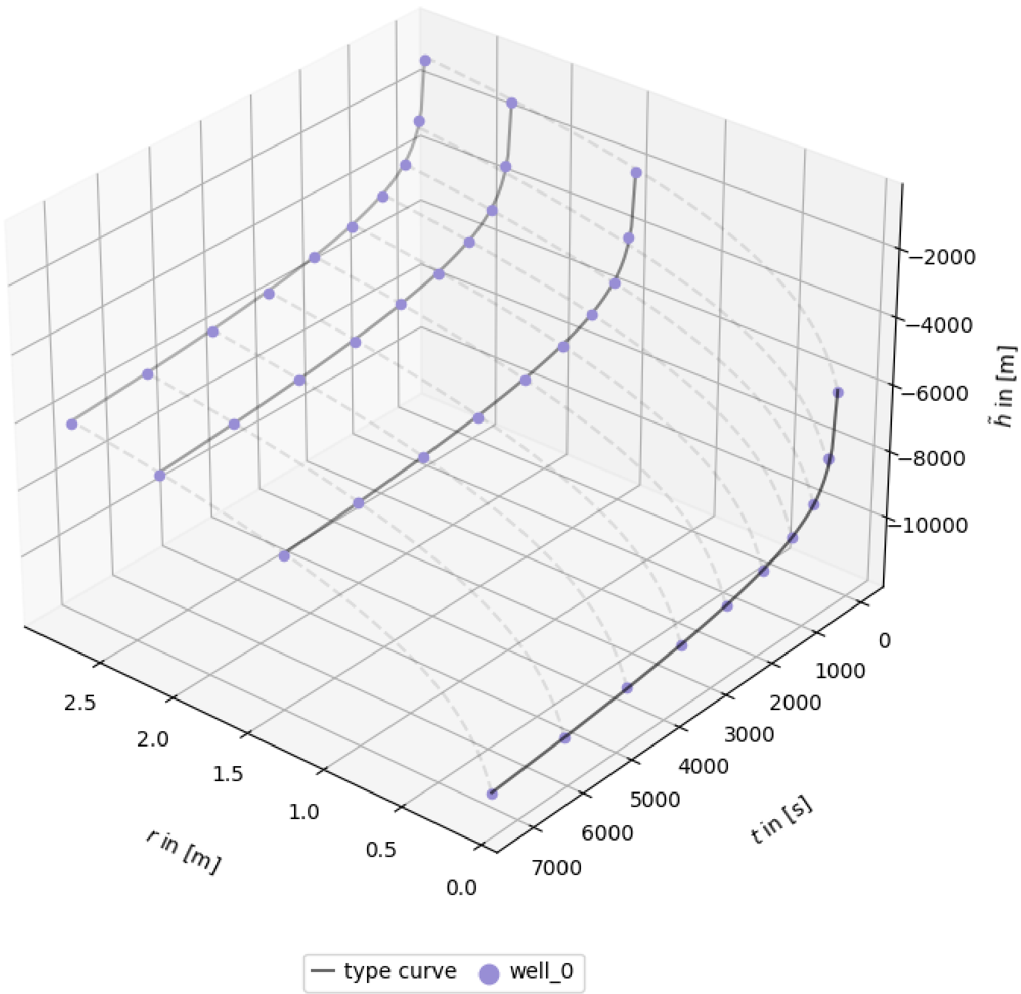
Total running time of the script: (0 minutes 0.692 seconds)

Estimate homogeneous parameters

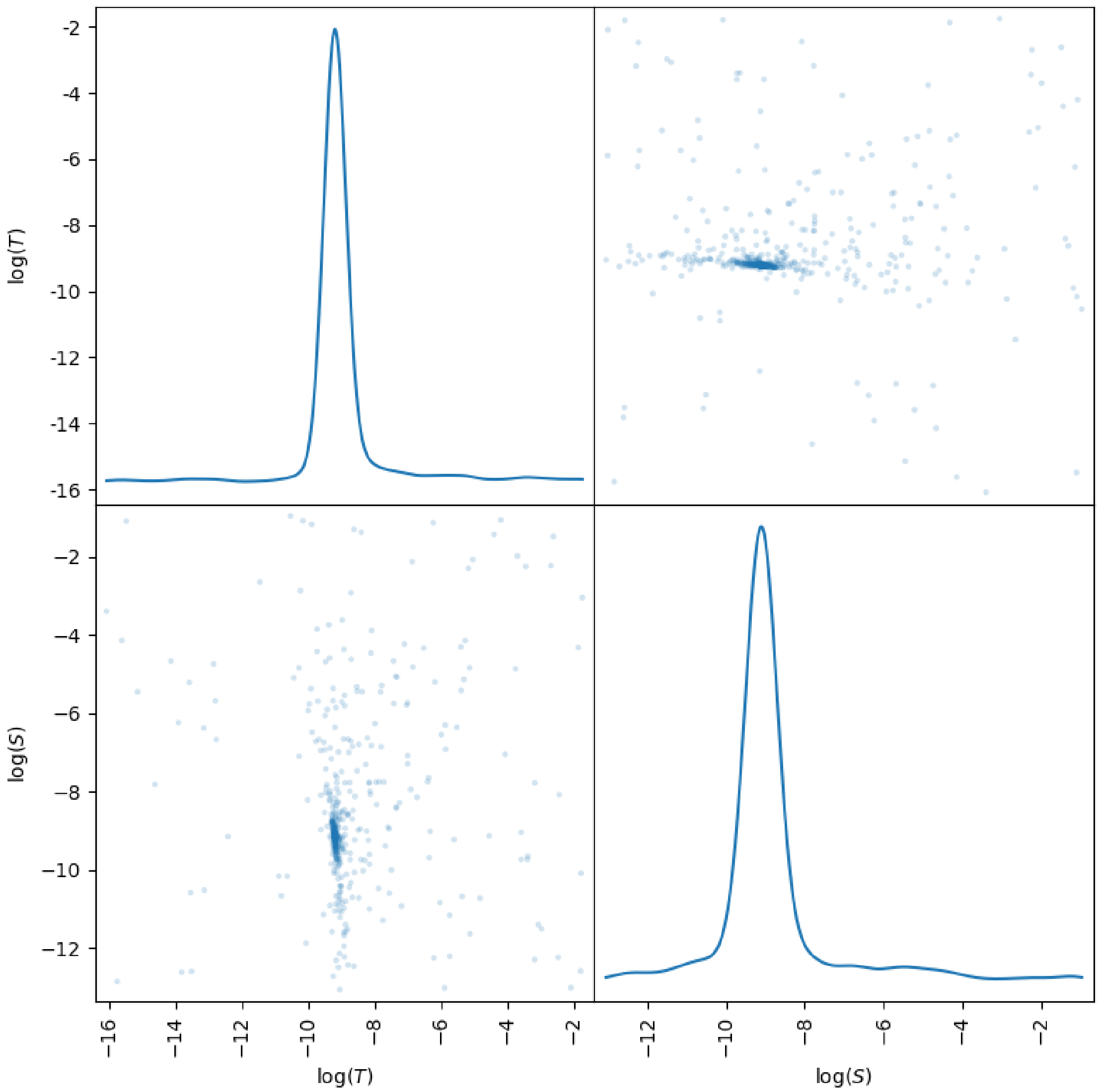
Here we estimate transmissivity and storage from a pumping test campaign with the classical theis solution.

```
import welltestpy as wtp

campaign = wtp.load_campaign("Cmp_UFZ-campaign.cmp")
estimation = wtp.estimate.Theis("Estimate_theis", campaign, generate=True)
estimation.run()
```

- 


−2000
−4000
−6000
−8000
−10000
h̃ in [m]
0
1000
2000
3000
4000
5000
6000
7000
t in [s]
2.5
2.0
1.5
1.0
0.5
0.0
r in [m]
type curve
well_0

-



•

```
Initializing the  Shuffled Complex Evolution (SCE-UA) algorithm  with  5000 ␣
↪repetitions
The objective function will be minimized
Starting burn-in sampling...
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/latest/examples/Estimate_theis/2023-04-18_09-58-22_db.csv' created.
Burn-in sampling completed...
Starting Complex Evolution...
ComplexEvo loop #1 in progress...
ComplexEvo loop #2 in progress...
ComplexEvo loop #3 in progress...
ComplexEvo loop #4 in progress...
ComplexEvo loop #5 in progress...
ComplexEvo loop #6 in progress...
ComplexEvo loop #7 in progress...
ComplexEvo loop #8 in progress...
ComplexEvo loop #9 in progress...
ComplexEvo loop #10 in progress...
ComplexEvo loop #11 in progress...
```

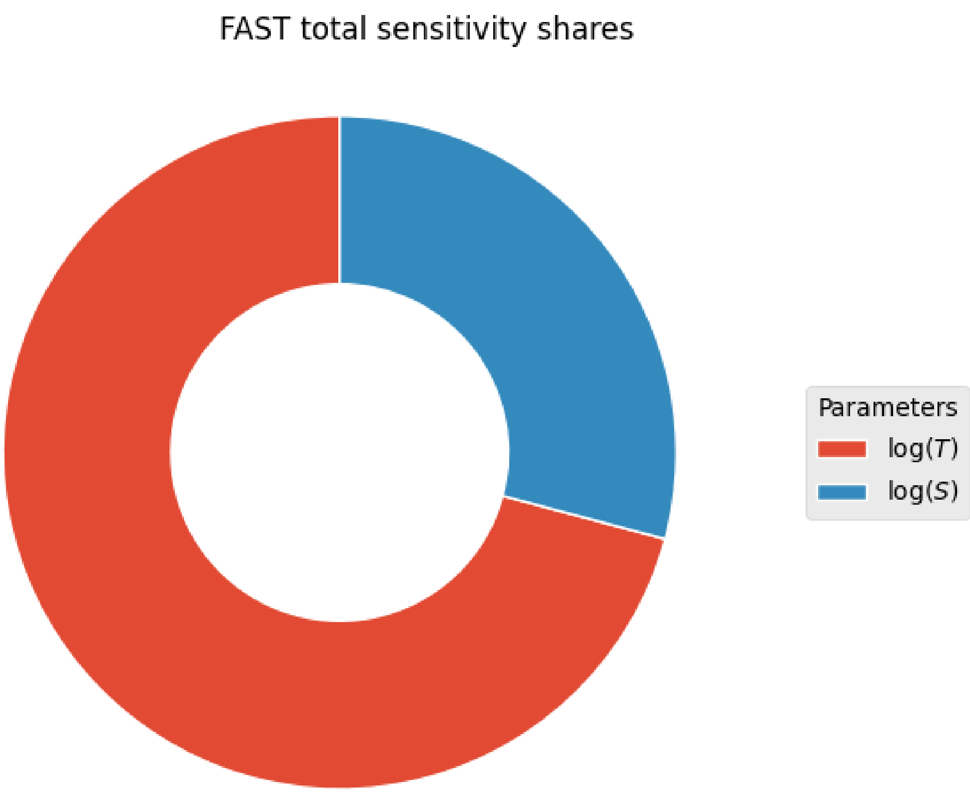
(continues on next page)

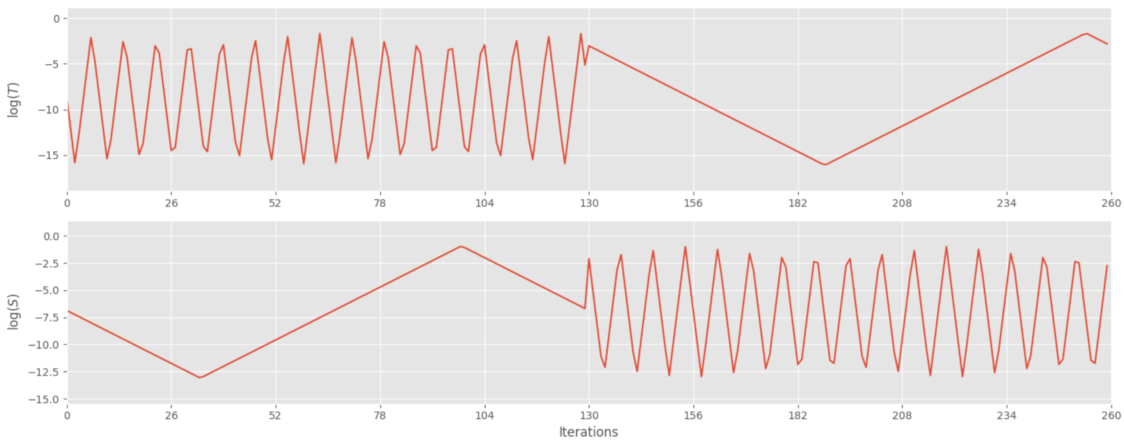
(continued from previous page)

```
ComplexEvo loop #12 in progress...
ComplexEvo loop #13 in progress...
ComplexEvo loop #14 in progress...
ComplexEvo loop #15 in progress...
ComplexEvo loop #16 in progress...
ComplexEvo loop #17 in progress...
ComplexEvo loop #18 in progress...
ComplexEvo loop #19 in progress...
ComplexEvo loop #20 in progress...
THE POPULATION HAS CONVERGED TO A PRESPECIFIED SMALL PARAMETER SPACE
SEARCH WAS STOPPED AT TRIAL NUMBER: 2706
NUMBER OF DISCARDED TRIALS: 0
NORMALIZED GEOMETRIC RANGE = 0.000706
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 0.64 seconds
Total Repetitions: 2706
Minimal objective value: 77.2517
Corresponding parameter setting:
transmissivity: -9.21583
storage: -9.10167
******************************

Best parameter set:
transmissivity=-9.215828396011176, storage=-9.101673860745393
```

In addition, we run a sensitivity analysis, to get an impression of the impact of each parameter

```
estimation.sensitivity()
```



- 

-

```
Initializing the  Fourier Amplitude Sensitivity Test (FAST)  with  260  repetitions
Starting the FAST algotrithm with 260 repetitions...
Creating FAST Matrix
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/latest/examples/Estimate_theis/2023-04-18_09-58-25_sensitivity_db.csv'␣
↪created.

*** Final SPOTPY summary ***
Total Duration: 0.08 seconds
Total Repetitions: 260
Minimal objective value: 516.425
Corresponding parameter setting:
transmissivity: -9.03998
```

(continues on next page)

(continued from previous page)

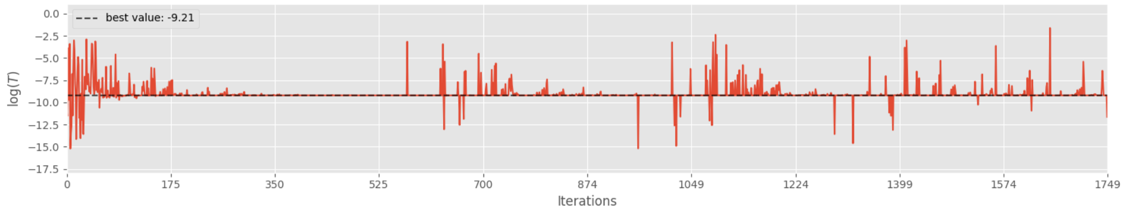
```
storage: -9.9678
Maximal objective value: 2.77063e+06
Corresponding parameter setting:
transmissivity: -15.8137
storage: -11.467
******************************

260
```

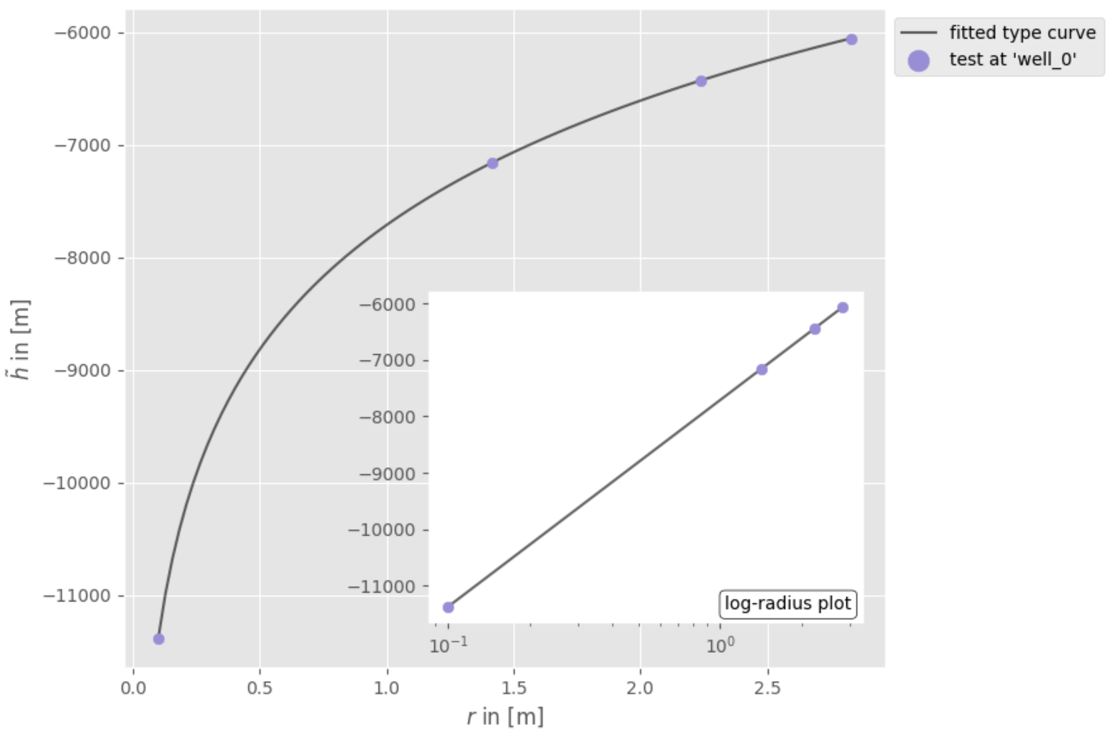
Total running time of the script: (0 minutes 3.639 seconds)

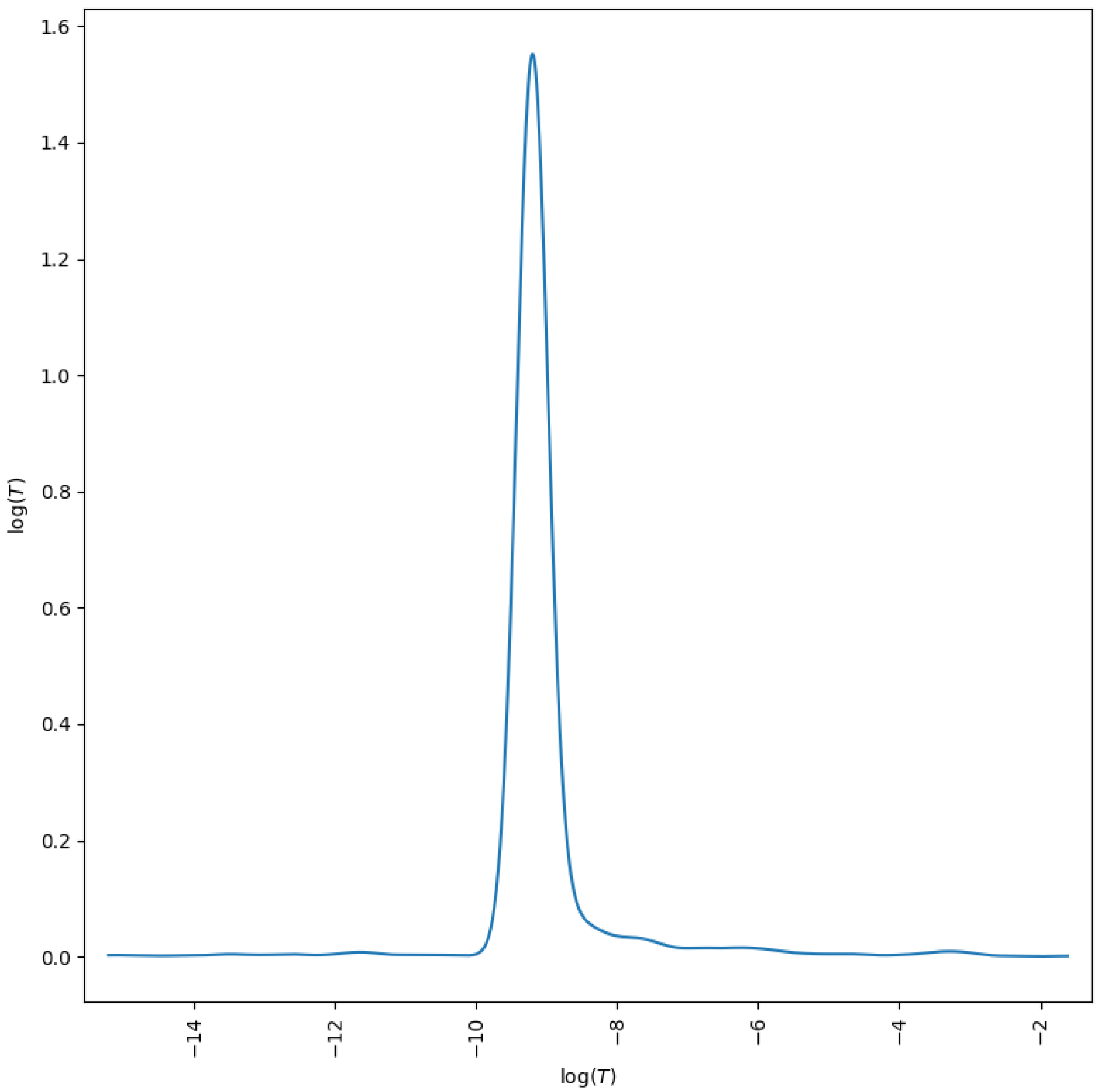
Estimate steady homogeneous parameters

Here we estimate transmissivity from the quasi steady state of a pumping test campaign with the classical thiem solution.

```
import welltestpy as wtp

campaign = wtp.load_campaign("Cmp_UFZ-campaign.cmp")
estimation = wtp.estimate.Thiem("Estimate_thiem", campaign, generate=True)
estimation.run()
```

- 

- 



•

```
Initializing the  Shuffled Complex Evolution (SCE-UA) algorithm  with  5000 ␣
↪repetitions
The objective function will be minimized
Starting burn-in sampling...
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/latest/examples/Estimate_thiem/2023-04-18_09-58-26_db.csv' created.
Burn-in sampling completed...
Starting Complex Evolution...
ComplexEvo loop #1 in progress...
ComplexEvo loop #2 in progress...
ComplexEvo loop #3 in progress...
ComplexEvo loop #4 in progress...
ComplexEvo loop #5 in progress...
ComplexEvo loop #6 in progress...
ComplexEvo loop #7 in progress...
ComplexEvo loop #8 in progress...
ComplexEvo loop #9 in progress...
ComplexEvo loop #10 in progress...
ComplexEvo loop #11 in progress...
```

(continues on next page)

(continued from previous page)

```
ComplexEvo loop #12 in progress...
ComplexEvo loop #13 in progress...
ComplexEvo loop #14 in progress...
ComplexEvo loop #15 in progress...
ComplexEvo loop #16 in progress...
ComplexEvo loop #17 in progress...
ComplexEvo loop #18 in progress...
ComplexEvo loop #19 in progress...
ComplexEvo loop #20 in progress...
ComplexEvo loop #21 in progress...
ComplexEvo loop #22 in progress...
ComplexEvo loop #23 in progress...
ComplexEvo loop #24 in progress...
ComplexEvo loop #25 in progress...
ComplexEvo loop #26 in progress...
ComplexEvo loop #27 in progress...
ComplexEvo loop #28 in progress...
ComplexEvo loop #29 in progress...
ComplexEvo loop #30 in progress...
ComplexEvo loop #31 in progress...
ComplexEvo loop #32 in progress...
ComplexEvo loop #33 in progress...
ComplexEvo loop #34 in progress...
ComplexEvo loop #35 in progress...
ComplexEvo loop #36 in progress...
ComplexEvo loop #37 in progress...
ComplexEvo loop #38 in progress...
ComplexEvo loop #39 in progress...
ComplexEvo loop #40 in progress...
ComplexEvo loop #41 in progress...
ComplexEvo loop #42 in progress...
ComplexEvo loop #43 in progress...
ComplexEvo loop #44 in progress...
ComplexEvo loop #45 in progress...
ComplexEvo loop #46 in progress...
ComplexEvo loop #47 in progress...
ComplexEvo loop #48 in progress...
ComplexEvo loop #49 in progress...
ComplexEvo loop #50 in progress...
ComplexEvo loop #51 in progress...
ComplexEvo loop #52 in progress...
ComplexEvo loop #53 in progress...
ComplexEvo loop #54 in progress...
ComplexEvo loop #55 in progress...
ComplexEvo loop #56 in progress...
ComplexEvo loop #57 in progress...
ComplexEvo loop #58 in progress...
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
```

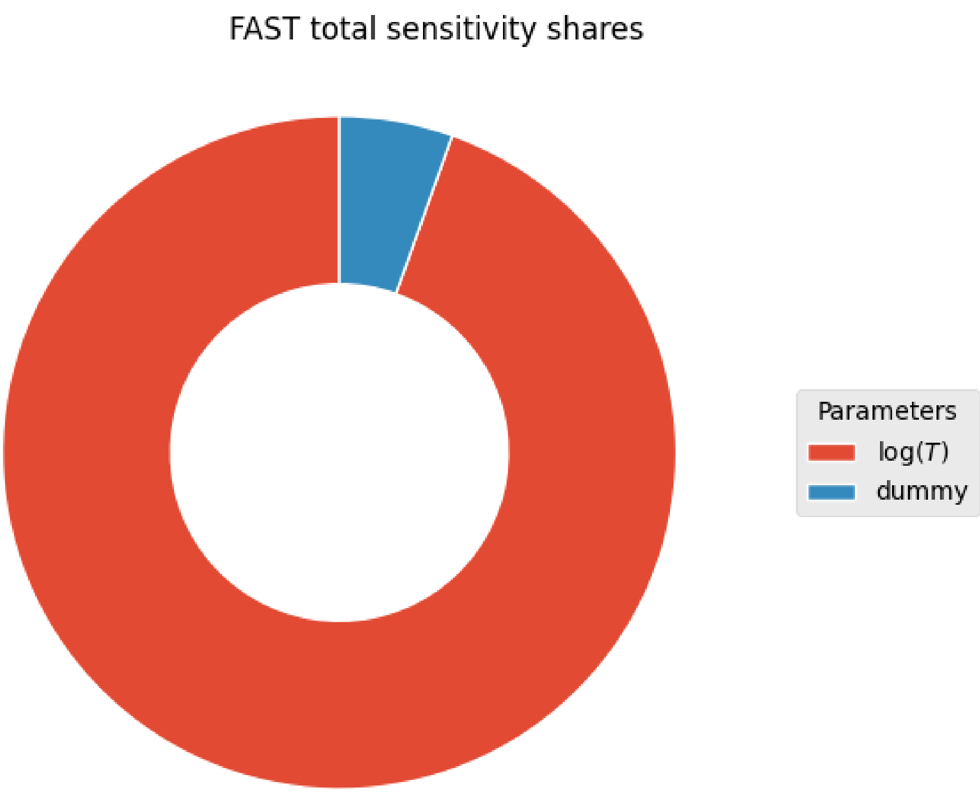
(continues on next page)

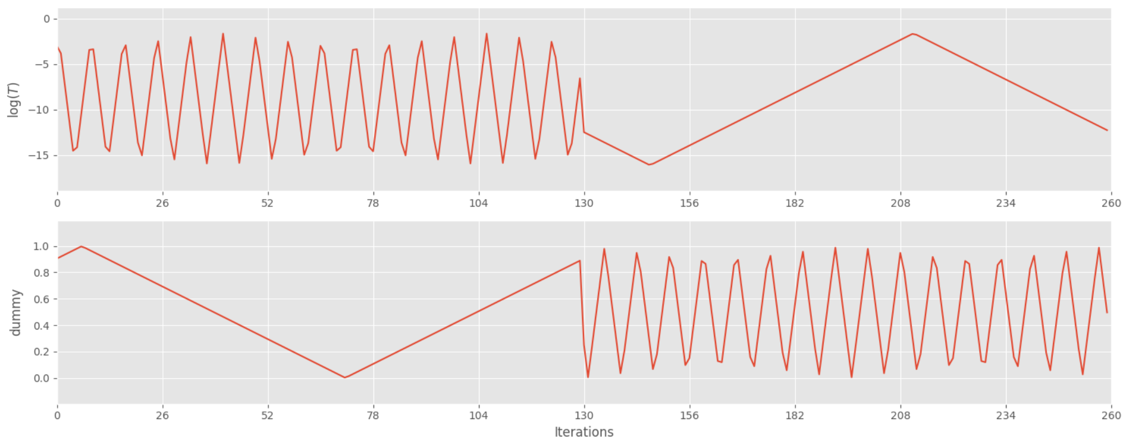
(continued from previous page)

```
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
*** OPTIMIZATION SEARCH TERMINATED BECAUSE THE LIMIT
ON THE MAXIMUM NUMBER OF TRIALS
5000
HAS BEEN EXCEEDED.
SEARCH WAS STOPPED AT TRIAL NUMBER: 5057
NUMBER OF DISCARDED TRIALS: 21
NORMALIZED GEOMETRIC RANGE = 0.264139
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 1.01 seconds
Total Repetitions: 5057
Minimal objective value: 0.0672645
Corresponding parameter setting:
transmissivity: -9.21029
******************************

Best parameter set:
transmissivity=-9.21029355734211
```

since we only have one parameter, we need a dummy parameter to estimate sensitivity

```
estimation.gen_setup(dummy=True)
estimation.sensitivity()
```



- 

-

```
Initializing the  Fourier Amplitude Sensitivity Test (FAST)  with  260  repetitions
Starting the FAST algotrithm with 260 repetitions...
Creating FAST Matrix
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/latest/examples/Estimate_thiem/2023-04-18_09-58-28_sensitivity_db.csv'␣
↪created.

*** Final SPOTPY summary ***
Total Duration: 0.06 seconds
Total Repetitions: 260
Minimal objective value: 42.4221
Corresponding parameter setting:
transmissivity: -9.22575
```

(continues on next page)

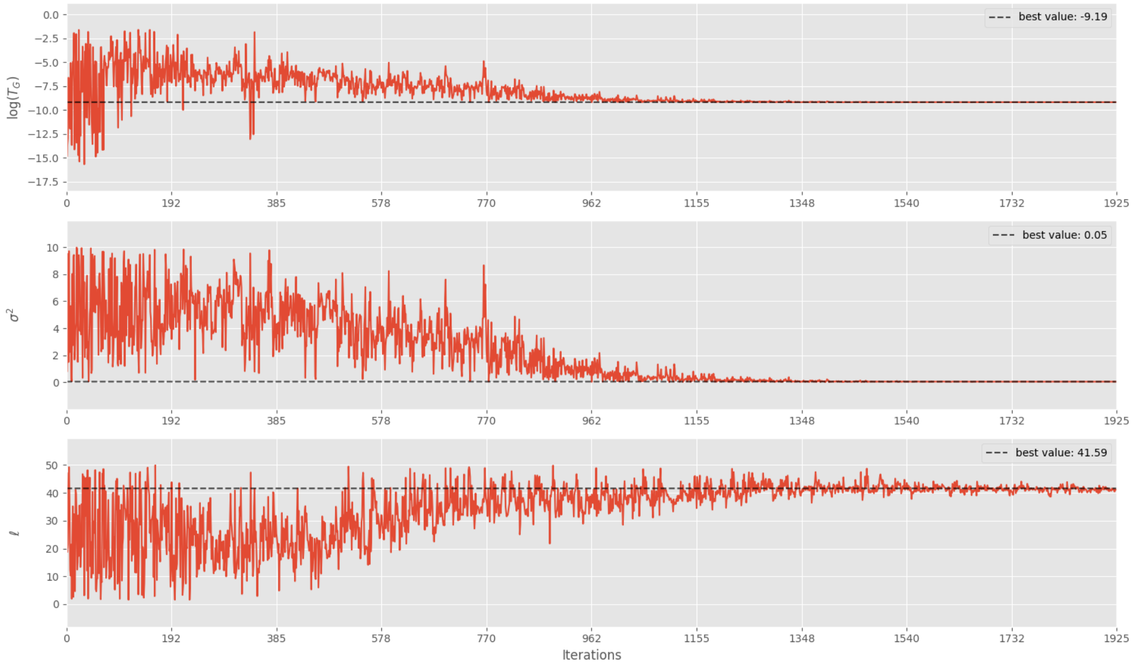
(continued from previous page)

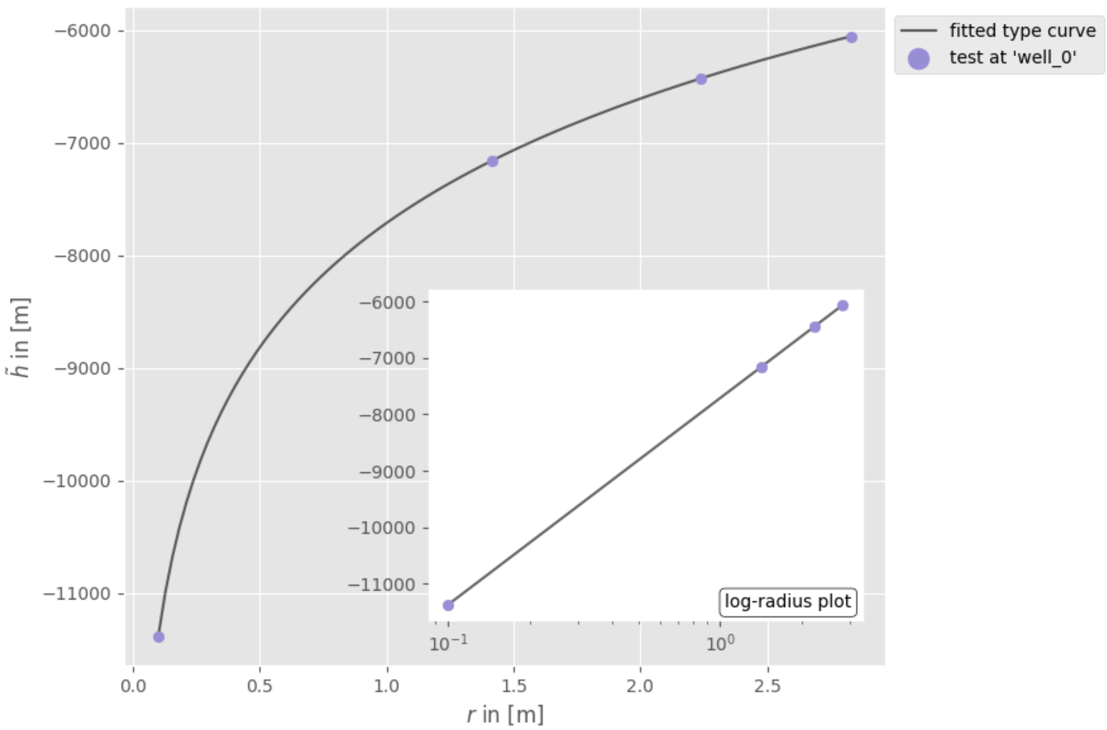
```
dummy: 0.37107
Maximal objective value: 2.50666e+06
Corresponding parameter setting:
transmissivity: -16.0365
dummy: 0.311892
******************************

260
```

Total running time of the script: (0 minutes 3.409 seconds)

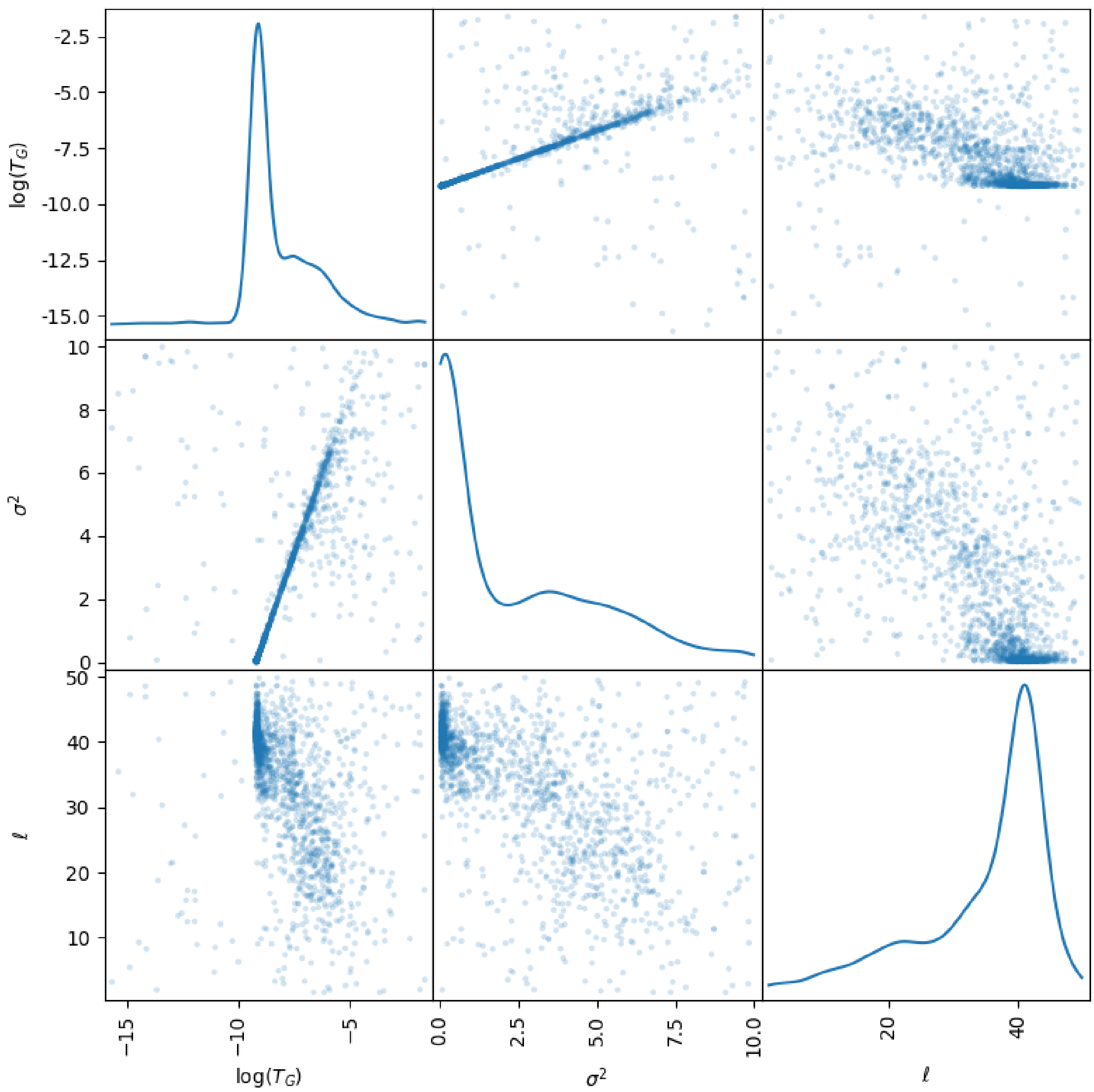
Estimate steady heterogeneous parameters

Here we demonstrate how to estimate parameters of heterogeneity, namely mean, variance and correlation length of log-transmissivity, with the aid the the extended Thiem solution in 2D.

- 

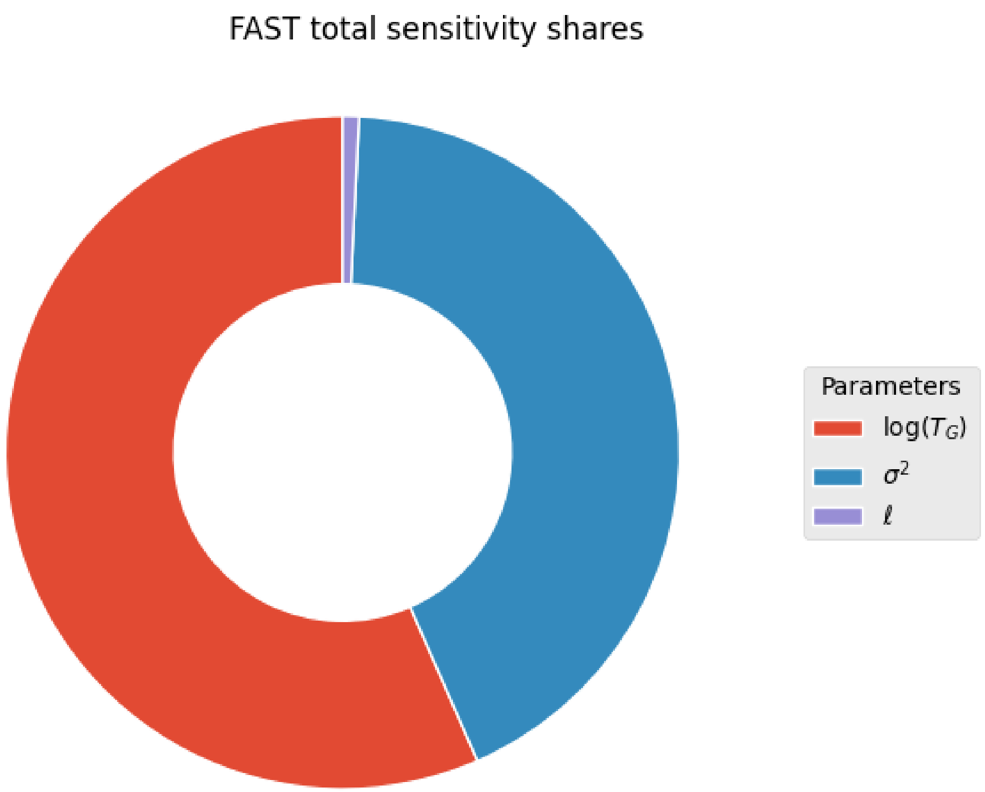

fitted type curve
test at 'well_0'
$\tilde{h}$ in [m]
r in [m]
log-radius plot
−6000
−7000
−8000
−9000
−10000
−11000
0.0
0.5
1.0
1.5
2.0
2.5
10^{-1}
10^{0}

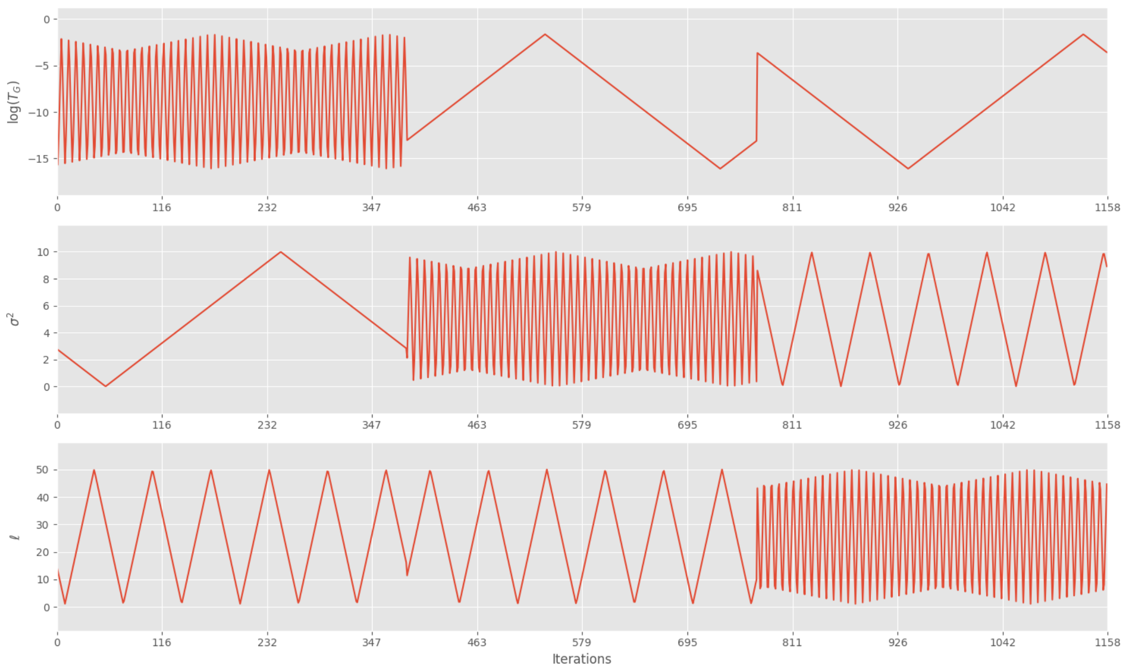
•



log(T_G)
σ^2
ℓ

•



- 

-

```
Initializing the  Shuffled Complex Evolution (SCE-UA) algorithm  with  5000 ␣
↪repetitions
The objective function will be minimized
Starting burn-in sampling...
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/latest/examples/Est_steady_het/2023-04-18_09-58-29_db.csv' created.
```

(continues on next page)

(continued from previous page)

```
Burn-in sampling completed...
Starting Complex Evolution...
ComplexEvo loop #1 in progress...
ComplexEvo loop #2 in progress...
ComplexEvo loop #3 in progress...
ComplexEvo loop #4 in progress...
ComplexEvo loop #5 in progress...
ComplexEvo loop #6 in progress...
ComplexEvo loop #7 in progress...
ComplexEvo loop #8 in progress...
ComplexEvo loop #9 in progress...
ComplexEvo loop #10 in progress...
ComplexEvo loop #11 in progress...
ComplexEvo loop #12 in progress...
ComplexEvo loop #13 in progress...
ComplexEvo loop #14 in progress...
ComplexEvo loop #15 in progress...
ComplexEvo loop #16 in progress...
ComplexEvo loop #17 in progress...
ComplexEvo loop #18 in progress...
ComplexEvo loop #19 in progress...
ComplexEvo loop #20 in progress...
ComplexEvo loop #21 in progress...
ComplexEvo loop #22 in progress...
ComplexEvo loop #23 in progress...
ComplexEvo loop #24 in progress...
ComplexEvo loop #25 in progress...
ComplexEvo loop #26 in progress...
ComplexEvo loop #27 in progress...
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
```

(continues on next page)

(continued from previous page)

```
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
*** OPTIMIZATION SEARCH TERMINATED BECAUSE THE LIMIT
ON THE MAXIMUM NUMBER OF TRIALS
5000
HAS BEEN EXCEEDED.
SEARCH WAS STOPPED AT TRIAL NUMBER: 5084
NUMBER OF DISCARDED TRIALS: 35
NORMALIZED GEOMETRIC RANGE = 0.002615
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 0.81 seconds
Total Repetitions: 5084
Minimal objective value: 0.000102931
Corresponding parameter setting:
trans_gmean: -9.18665
var: 0.0473718
len_scale: 41.5893
******************************

Best parameter set:
trans_gmean=-9.186654615190932, var=0.04737184562132764, len_scale=41.58933403842806
Initializing the  Fourier Amplitude Sensitivity Test (FAST)  with  1158  repetitions
Starting the FAST algotrithm with 1158 repetitions...
Creating FAST Matrix
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/latest/examples/Est_steady_het/2023-04-18_09-58-32_sensitivity_db.csv'␣
↪created.

*** Final SPOTPY summary ***
Total Duration: 0.16 seconds
Total Repetitions: 1158
Minimal objective value: 77.4336
Corresponding parameter setting:
trans_gmean: -7.55702
var: 3.3744
len_scale: 30.7267
Maximal objective value: 2.76442e+08
Corresponding parameter setting:
trans_gmean: -15.8022
var: 9.88788
len_scale: 46.9621
******************************

1158
```

```
import welltestpy as wtp

campaign = wtp.load_campaign("Cmp_UFZ-campaign.cmp")
estimation = wtp.estimate.ExtThiem2D("Est_steady_het", campaign, generate=True)
estimation.run()
estimation.sensitivity()
```

Total running time of the script: (0 minutes 5.340 seconds)

Point triangulation

Often, we only know the distances between wells within a well base field campaign. To retrieve their spatial positions, we provide a routine, that triangulates their positions from a given distance matrix.

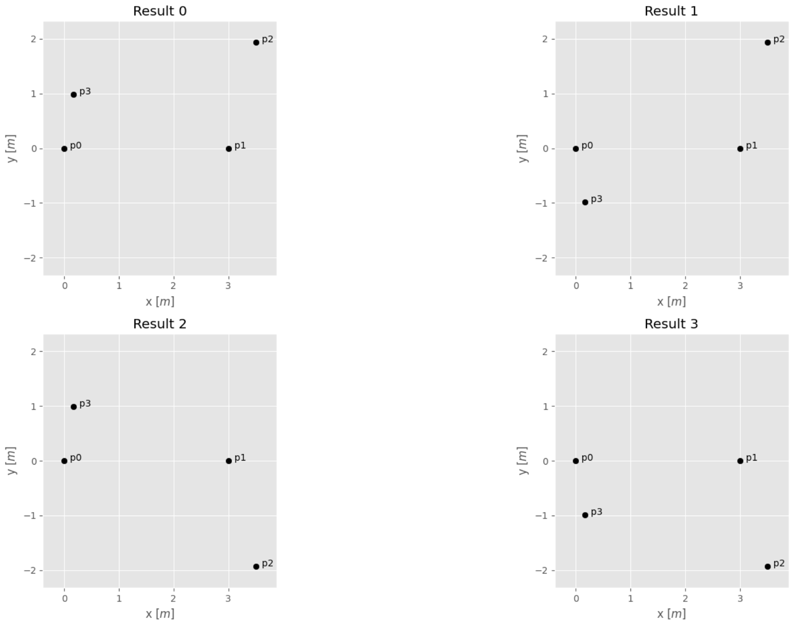
If the solution is not unique, all possible constellations will be returned.

```
import numpy as np

from welltestpy.tools import plot_well_pos, sym, triangulate

dist_mat = np.zeros((4, 4), dtype=float)
dist_mat[0, 1] = 3  # distance between well 0 and 1
dist_mat[0, 2] = 4  # distance between well 0 and 2
dist_mat[1, 2] = 2  # distance between well 1 and 2
dist_mat[0, 3] = 1  # distance between well 0 and 3
dist_mat[1, 3] = 3  # distance between well 1 and 3
dist_mat[2, 3] = -1  # unknown distance between well 2 and 3
dist_mat = sym(dist_mat)  # make the distance matrix symmetric
well_const = triangulate(dist_mat, prec=0.1)
```

```
Startingconstelation 0 1
add point 0
add point 1
number of temporal results: 8
number of overall results: 8
```

Now we can plot all possible well constellations

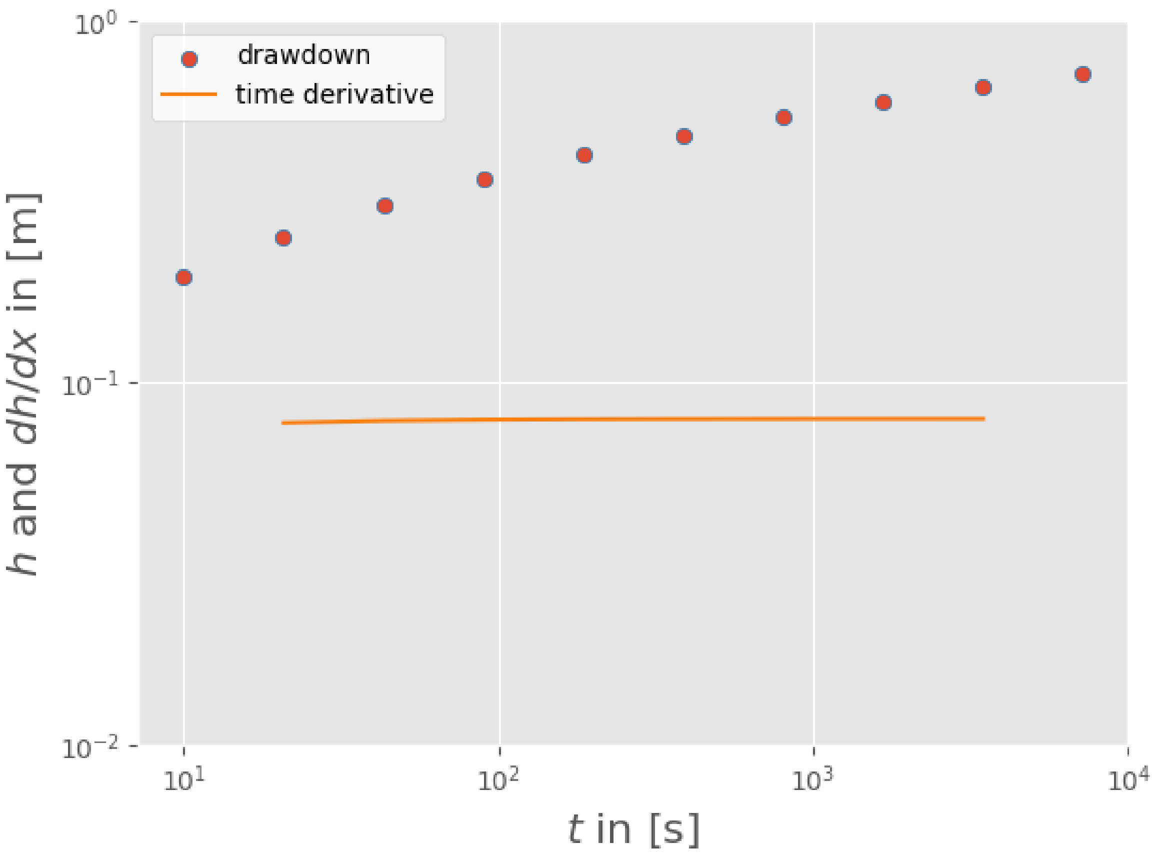
```
plot_well_pos(well_const)
```



Total running time of the script: (0 minutes 0.608 seconds)

Diagnostic plot

A diagnostic plot is a simultaneous plot of the drawdown and the logarithmic derivative of the drawdown in a log-log plot. Often, this plot is used to identify the right approach for the aquifer estimations.

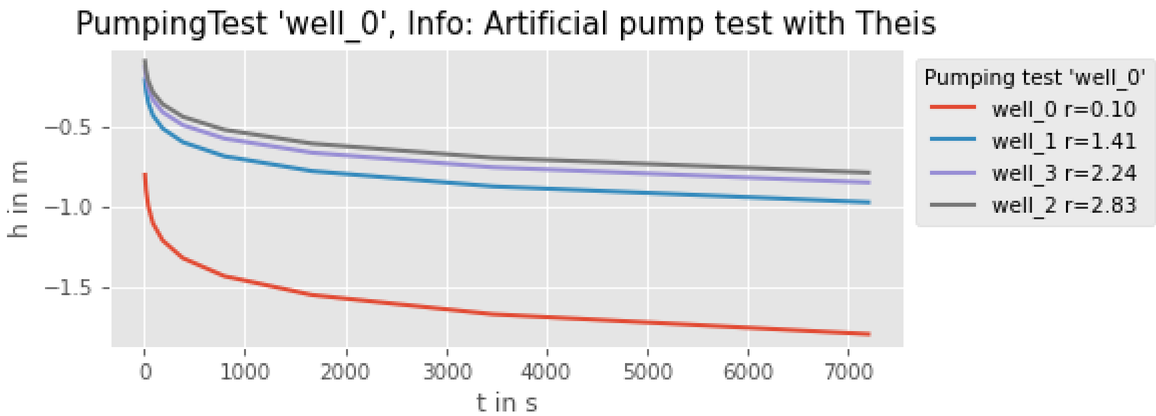


```
import welltestpy as wtp

campaign = wtp.load_campaign("Cmp_UFZ-campaign.cmp")
campaign.diagnostic_plot("well_0", "well_1")
```

Total running time of the script: (0 minutes 0.198 seconds)

Correcting drawdown: The Cooper-Jacob method

Here we demonstrate the correction established by Cooper and Jacob in 1946. This method corrects drawdown data for the reduction in saturated thickness resulting from groundwater withdrawal by a pumping well and thereby enables pumping tests in an unconfined aquifer to be interpreted by methods for confined aquifers.




```
import welltestpy as wtp

campaign = wtp.load_campaign("Cmp_UFZ-campaign.cmp")
campaign.tests["well_0"].correct_observations()
campaign.plot()
```

Total running time of the script: (0 minutes 0.207 seconds)

CHAPTER 3

WELLTESTPY API

welltestpy - a Python package to handle well-based Field-campaigns.

welltestpy provides a framework to handle and plot data from well based field campaigns as well as a parameter estimation module.

3.1 Subpackages

`data`	welltestpy subpackage providing datastructures.
`estimate`	welltestpy subpackage providing routines to estimate pump test parameters.
`process`	welltestpy subpackage providing routines to pre process test data.
`tools`	welltestpy subpackage providing miscellaneous tools.

welltestpy.data

welltestpy subpackage providing datastructures.

Campaign classes

The following classes can be used to handle field campaigns.

`Campaign`(name[, fieldsite, wells, tests, ...])	Class for a well based campaign.
`FieldSite`(name[, description, coordinates])	Class for a field site.

welltestpy.data.Campaign

class Campaign(*name*, *fieldsite='Fieldsite'*, *wells=None*, *tests=None*, *timeframe=None*, *description='Welltest campaign'*)

Bases: `object`

Class for a well based campaign.

This is a class for a well based test campaign on a field site. It has a name, a description and a timeframe.

Parameters

- **name** (`str`) – Name of the campaign.
- **fieldsite** (`str` or *`Variable`*, optional) – The field site. Default: `"Fieldsite"`
- **wells** (`dict`, optional) – The wells within the field site. Keys are the well names and values are an instance of *`Well`*. Default: `None`
- **wells** (`dict`, optional) – The tests within the campaign. Keys are the test names and values are an instance of *`Test`*. Default: `None`
- **timeframe** (`str`, optional) – Timeframe of the campaign. Default: `None`
- **description** (`str`, optional) – Description of the field site. Default: `"Welltest campaign"`

Attributes

fieldsite
`FieldSite`: Field site where the campaign was realised.

tests
`dict`: Tests within the campaign.

wells
`dict`: Wells within the campaign.

Methods

add_well(name, radius, coordinates[, ...])	Add a single well to the campaign.
addtests(tests)	Add some specified tests.
addwells(wells)	Add some specified wells.
deltests(tests)	Delete some specified tests.
delwells(wells)	Delete some specified wells.
diagnostic_plot(pumping_test, ...)	Generate a diagnostic plot.
plot([select_tests])	Generate a plot of the tests within the campaign.
plot_wells(**kwargs)	Generate a plot of the wells within the campaign.
save([path, name])	Save the campaign to file.

add_well(*name*, *radius*, *coordinates*, *welldepth=1.0*, *aquiferdepth=None*)

Add a single well to the campaign.

Parameters

- **name** (`str`) – Name of the Variable.
- **radius** (*`Variable`* or `float`) – Value of the Variable.
- **coordinates** (*`Variable`* or `numpy.ndarray`) – Value of the Variable.
- **welldepth** (*`Variable`* or `float`, optional) – Depth of the well. Default: 1.0
- **aquiferdepth** (*`Variable`* or `float`, optional) – Depth of the aquifer at the well. Default: `"None"`

addtests(*tests*)

Add some specified tests.

This will add tests to the campaign.

Parameters
tests (dict) – Tests to be added.

addwells(*wells*)

Add some specified wells.

This will add wells to the campaign.

Parameters
wells (dict) – Wells to be added.

deltests(*tests*)

Delete some specified tests.

This will delete tests from the campaign. You can give a list of tests or a single test by name.

Parameters
tests (list of str or str) – Tests to be deleted.

delwells(*wells*)

Delete some specified wells.

This will delete wells from the campaign. You can give a list of wells or a single well by name.

Parameters
wells (list of str or str) – Wells to be deleted.

diagnostic_plot(*pumping_test*, *observation_well*, ***kwargs*)

Generate a diagnostic plot.

Parameters

- **pumping_test** (str) – The pumping well that is saved in the campaign.
- **observation_well** (str) – Observation point to make the diagnostic plot.
- ****kwargs** – Keyword-arguments forwarded to *campaign_well_plot()*.

plot(*select_tests=None*, ***kwargs*)

Generate a plot of the tests within the campaign.

This will plot an overview of the tests within the campaign.

Parameters

- **select_tests** (list, optional) – Tests that should be plotted. If None, all will be displayed. Default: None
- ****kwargs** – Keyword-arguments forwarded to *campaign_plot()*

plot_wells(***kwargs*)

Generate a plot of the wells within the campaign.

This will plot an overview of the wells within the campaign.

Parameters
****kwargs** – Keyword-arguments forwarded to *campaign_well_plot()*.

save(*path=''*, *name=None*)

Save the campaign to file.

This writes the campaign to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Cmp_"+name`. Default: `None`

Notes

The file will get the suffix `".cmp"`.

property fieldsite

Field site where the campaign was realised.

Type
FieldSite

property tests

Tests within the campaign.

Type
`dict`

property wells

Wells within the campaign.

Type
`dict`

welltestpy.data.FieldSite

class FieldSite(*name*, *description='Field site'*, *coordinates=None*)

Bases: `object`

Class for a field site.

This is a class for a field site. It has a name and a description.

Parameters

- **name** (`str`) – Name of the field site.
- **description** (`str`, optional) – Description of the field site. Default: `"no description"`
- **coordinates** (*`Variable`*, optional) – Coordinates of the field site (lat, lon). Default: `None`

Attributes

`coordinates`
`numpy.ndarray`: Coordinates of the field site.

`info`
`str`: Info about the field site.

`pos`
`numpy.ndarray`: Position of the field site.

Methods

save([path, name])	Save a field site to file.

save(*path=''*, *name=None*)

Save a field site to file.

This writes the field site to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Field_"+name`. Default: `None`

Notes

The file will get the suffix `".fds"`.

property coordinates

Coordinates of the field site.

Type
`numpy.ndarray`

property info

Info about the field site.

Type
`str`

`property pos`

Position of the field site.

Type

`numpy.ndarray`

Field Test classes

The following classes can be used to handle field test within a campaign.

PumpingTest(name, pumpingwell, pumpingrate)	Class for a pumping test.
Test(name[, description, timeframe])	General class for a well based test.

welltestpy.data.PumpingTest

`class PumpingTest`(*name*, *pumpingwell*, *pumpingrate*, *observations=None*, *aquiferdepth=1.0*, *aquiferradius=inf*, *description='Pumpingtest'*, *timeframe=None*)

Bases: *Test*

Class for a pumping test.

This is a class for a pumping test on a field site. It has a name, a description, a timeframe and a pumpingwell string.

Parameters

- **name** (`str`) – Name of the test.
- **pumpingwell** (`str`) – Pumping well of the test.
- **pumpingrate** (`float` or *Variable*) – Pumping rate of at the pumping well. If a *float* is given, it is assumed to be given in `m^3/s`.
- **observations** (`dict`, optional) – Observations made within the pumping test. The dict-keys are the well names of the observation wells or the pumpingwell. Values need to be an instance of *Observation* Default: `None`
- **aquiferdepth** (`float` or *Variable*, optional) – Aquifer depth at the field site. Can also be used to store the saturated thickness of the aquifer. If a *float* is given, it is assumed to be given in `m`. Default: `1.0`
- **aquiferradius** (`float` or *Variable*, optional) – Aquifer radius ot the field site. If a *float* is given, it is assumed to be given in `m`. Default: `inf`
- **description** (`str`, optional) – Description of the test. Default: `"Pumpingtest"`
- **timeframe** (`str`, optional) – Timeframe of the test. Default: `None`

Attributes

aquiferdepth
`Variable`: aquifer depth or saturated thickness.

aquiferradius
`float`: aquifer radius at the field site.

constant_rate
`bool`: state if this is a constant rate test.

depth
`float`: aquifer depth or saturated thickness.

observations
`dict`: observations made at the field site.

observationwells
`tuple` of `str`: all well names.

pumpingrate
`float`: pumping rate variable at the pumping well.

radius
float: aquifer radius at the field site.

rate
float: pumping rate at the pumping well.

testtype
str: String containing the test type.

wells
tuple of str: all well names.

Methods

add_observations(obs)	Add some specified observations.
add_steady_obs(well, observation[, description])	Add steady drawdown observations.
add_transient_obs(well, time, observation[, ...])	Add transient drawdown observations.
correct_observations([aquiferdepth, wells, ...])	Correct observations with the selected method.
del_observations(obs)	Delete some specified observations.
diagnostic_plot(observation_well, **kwargs)	Make a diagnostic plot.
make_steady([time])	Convert the pumping test to a steady state test.
plot(wells[, exclude, fig, ax])	Generate a plot of the pumping test.
save([path, name])	Save a pumping test to file.
state([wells])	Get the state of observation.

add_observations(*obs*)

Add some specified observations.

Parameters

obs (dict, list, *Observation*) – Observations to be added.

add_steady_obs(*well*, *observation*, *description='Steady State Drawdown observation'*)

Add steady drawdown observations.

Parameters

- **well** (str) – well where the observation is made.
- **observation** (*Variable*) – Observation.
- **description** (str, optional) – Description of the Variable. Default: "Steady observation"

add_transient_obs(*well*, *time*, *observation*, *description='Transient Drawdown observation'*)

Add transient drawdown observations.

Parameters

- **well** (str) – well where the observation is made.
- **time** (*Variable*) – Time points of observation.
- **observation** (*Variable*) – Observation.
- **description** (str, optional) – Description of the Variable. Default: "Drawdown observation"

correct_observations(*aquiferdepth=None*, *wells=None*, *method='cooper_jacob'*)

Correct observations with the selected method.

Parameters

- **aquiferdepth** (float, optional) – Aquifer depth at the field site. Default: PumpingTest.depth
- **wells** (list, optional) – List of wells, to check the observation state at. Default: all
- **method** (*:class: 'str', optional*) – Method to correct the drawdown data. Default: ''cooper_jacob''

Notes

This will be used by the Campaign class.

`del_observations`(*obs*)

Delete some specified observations.

This will delete observations from the pumping test. You can give a list of observations or a single observation by name.

Parameters
obs (list of str or str) – Observations to be deleted.

`diagnostic_plot`(*observation_well*, ***kwargs*)

Make a diagnostic plot.

Parameters
observation_well (str) – The observation well for the data to make the diagnostic plot.

Notes

This will be used by the Campaign class.

`make_steady`(*time='latest'*)

Convert the pumping test to a steady state test.

Parameters
time (str or float, optional) – Selected time point for steady state. If "latest", the latest common time point is used. If None, it takes the last observation per well. If float, it will be interpolated. Default: "latest"

`plot`(*wells*, *exclude=None*, *fig=None*, *ax=None*, ***kwargs*)

Generate a plot of the pumping test.

This will plot the pumping test on the given figure axes.

Parameters

- **ax** (Axes) – Axes where the plot should be done.
- **wells** (dict) – Dictionary containing the well classes sorted by name.
- **exclude** (list, optional) – List of wells that should be excluded from the plot. Default: None

Notes

This will be used by the Campaign class.

`save`(*path=''*, *name=None*)

Save a pumping test to file.

This writes the variable to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Test_"+name`. Default: `None`

Notes

The file will get the suffix `".tst"`.

`state`(*wells=None*)

Get the state of observation.

Either None, "steady", "transient" or "mixed".

Parameters
wells (`list`, optional) – List of wells, to check the observation state at. Default: all

property aquiferdepth

aquifer depth or saturated thickness.

Type
Variable

property aquiferradius

aquifer radius at the field site.

Type
`float`

property constant_rate

state if this is a constant rate test.

Type
`bool`

property depth

aquifer depth or saturated thickness.

Type
`float`

property observations

observations made at the field site.

Type
`dict`

property observationwells

all well names.

Type
`tuple` of `str`

property pumpingrate

pumping rate variable at the pumping well.

Type
`float`

property radius

aquifer radius at the field site.

Type
`float`

property rate

pumping rate at the pumping well.

Type
`float`

property testtype

String containing the test type.

Type
`str`

property wells

all well names.

Type
`tuple` of `str`

welltestpy.data.Test

`class Test(name, description='no description', timeframe=None)`

Bases: `object`

General class for a well based test.

This is a class for a well based test on a field site. It has a name, a description and a timeframe string.

Parameters

- **name** (`str`) – Name of the test.
- **description** (`str`, optional) – Description of the test. Default: `"no description"`
- **timeframe** (`str`, optional) – Timeframe of the test. Default: `None`

Attributes

testtype
`str`: String containing the test type.

Methods

plot(wells[, exclude, fig, ax])	Generate a plot of the pumping test.

`plot(wells, exclude=None, fig=None, ax=None, **kwargs)`

Generate a plot of the pumping test.

This will plot the test on the given figure axes.

Parameters

- **ax** (`Axes`) – Axes where the plot should be done.
- **wells** (`dict`) – Dictionary containing the well classes sorted by name.
- **exclude** (`list`, optional) – List of wells that should be excluded from the plot. Default: `None`

Notes

This will be used by the Campaign class.

`property testtype`

String containing the test type.

Type
`str`

Variable classes

`Variable`(name, value[, symbol, units, ...])	Class for a variable.
`TimeVar`(value[, symbol, units, description])	Variable class special for time series.
`HeadVar`(value[, symbol, units, description])	Variable class special for groundwater head.
`TemporalVar`([value])	Variable class for a temporal variable.
`CoordinatesVar`(lat, lon[, symbol, units, ...])	Variable class special for coordinates.
`Observation`(name, observation[, time, ...])	Class for a observation.
`StdyObs`(name, observation[, description])	Observation class special for steady observations.
`DrawdownObs`(name, observation, time[, ...])	Observation class special for drawdown observations.
`StdyHeadObs`(name, observation[, description])	Observation class special for steady drawdown observations.
`Well`(name, radius, coordinates[, welldepth, ...])	Class for a pumping-/observation-well.

welltestpy.data.Variable

class Variable(*name*, *value*, *symbol='x'*, *units='-'*, *description='no description'*)

Bases: `object`

Class for a variable.

This is a class for a physical variable which is either a scalar or an array.

It has a name, a value, a symbol, a unit and a descrition string.

Parameters

- **name** (`str`) – Name of the Variable.
- **value** (`int` or `float` or `numpy.ndarray`) – Value of the Variable.
- **symbole** (`str`, optional) – Name of the Variable. Default: `"x"`
- **units** (`str`, optional) – Units of the Variable. Default: `"-"`
- **description** (`str`, optional) – Description of the Variable. Default: `"no description"`

Attributes

info
`str`: Info about the Variable.

label
`str`: String containing: `symbol in units`.

scalar
`bool`: State if the variable is of scalar type.

value
`int` or `float` or `numpy.ndarray`: Value.

Methods

`__call__`([value])	Call a variable.
`save`([path, name])	Save a variable to file.

`__call__`(*value=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **value** (`int` or `float` or `numpy.ndarray`,)
- **optional** – Value of the Variable. Default: `None`

Returns

value – Value of the Variable.

Return type

`int` or `float` or `numpy.ndarray`

`save`(*path=''*, *name=None*)

Save a variable to file.

This writes the variable to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Var_"+name`. Default: `None`

Notes

The file will get the suffix `".var"`.

`property info`

Info about the Variable.

Type

`str`

`property label`

`symbol in units`.

Type

`str`

Type

String containing

`property scalar`

State if the variable is of scalar type.

Type

`bool`

`property value`

Value.

Type

`int` or `float` or `numpy.ndarray`

welltestpy.data.TimeVar

class TimeVar(*value*, *symbol='t'*, *units='s'*, *description='time given in seconds'*)

Bases: *Variable*

Variable class special for time series.

Parameters

- **value** (int or float or numpy.ndarray) – Value of the Variable.
- **symbole** (str, optional) – Name of the Variable. Default: "t"
- **units** (str, optional) – Units of the Variable. Default: "s"
- **description** (str, optional) – Description of the Variable. Default: "time given in seconds"

Notes

Here the variable should be at most 1 dimensional and the name is fix set to "time".

Attributes

info
str: Info about the Variable.

label
str: String containing: symbol in units.

scalar
bool: State if the variable is of scalar type.

value
int or float or numpy.ndarray: Value.

Methods

__call__([value])	Call a variable.
save([path, name])	Save a variable to file.

__call__(*value=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **value** (int or float or numpy.ndarray,)
- **optional** – Value of the Variable. Default: None

Returns

value – Value of the Variable.

Return type

int or float or numpy.ndarray

save(*path=''*, *name=None*)

Save a variable to file.

This writes the variable to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Var_"+name`. Default: `None`

Notes

The file will get the suffix `".var"`.

property info

Info about the Variable.

Type
`str`

property label

`symbol in units`.

Type
`str`

Type
String containing

property scalar

State if the variable is of scalar type.

Type
`bool`

property value

Value.

Type
`int` or `float` or `numpy.ndarray`

welltestpy.data.HeadVar

class HeadVar(*value*, *symbol='h'*, *units='m'*, *description='head given in meters'*)

Bases: *Variable*

Variable class special for groundwater head.

Parameters

- **value** (int or float or numpy.ndarray) – Value of the Variable.
- **symbole** (str, optional) – Name of the Variable. Default: "h"
- **units** (str, optional) – Units of the Variable. Default: "m"
- **description** (str, optional) – Description of the Variable. Default: "head given in meters"

Notes

Here the variable name is fix set to "head".

Attributes

info
str: Info about the Variable.

label
str: String containing: `symbol in units`.

scalar
bool: State if the variable is of scalar type.

value
int or float or numpy.ndarray: Value.

Methods

__call__([value])	Call a variable.
save([path, name])	Save a variable to file.

__call__(*value=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **value** (int or float or numpy.ndarray,)
- **optional** – Value of the Variable. Default: None

Returns
value – Value of the Variable.

Return type
int or float or numpy.ndarray

save(*path=''*, *name=None*)

Save a variable to file.

This writes the variable to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Var_"+name`. Default: `None`

Notes

The file will get the suffix `".var"`.

property info

Info about the Variable.

Type
`str`

property label

`symbol in units`.

Type
`str`

Type
String containing

property scalar

State if the variable is of scalar type.

Type
`bool`

property value

Value.

Type
`int` or `float` or `numpy.ndarray`

welltestpy.data.TemporalVar

class TemporalVar(*value=0.0*)

Bases: *Variable*

Variable class for a temporal variable.

Parameters

- **value** (int or float or numpy.ndarray,)
- **optional** – Value of the Variable. Default: 0.0

Attributes

info
str: Info about the Variable.

label
str: String containing: symbol in units.

scalar
bool: State if the variable is of scalar type.

value
int or float or numpy.ndarray: Value.

Methods

__call__([value])	Call a variable.
save([path, name])	Save a variable to file.

__call__(*value=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **value** (int or float or numpy.ndarray,)
- **optional** – Value of the Variable. Default: None

Returns
value – Value of the Variable.

Return type
int or float or numpy.ndarray

save(*path='', name=None*)

Save a variable to file.

This writes the variable to a csv file.

Parameters

- **path** (str, optional) – Path where the variable should be saved. Default: ""
- **name** (str, optional) – Name of the file. If None, the name will be generated by "Var_"+name. Default: None

Notes

The file will get the suffix ".var".

property info

Info about the Variable.

Type
`str`

property label

`symbol in units`.

Type
`str`

Type
String containing

property scalar

State if the variable is of scalar type.

Type
`bool`

property value

Value.

Type
`int` or `float` or `numpy.ndarray`

welltestpy.data.CoordinatesVar

class CoordinatesVar(*lat*, *lon*, *symbol='[Lat,Lon]'*, *units='[deg,deg]'*, *description='Coordinates given in degree-North and degree-East'*)

Bases: *Variable*

Variable class special for coordinates.

Parameters

- **lat** (int or float or numpy.ndarray) – Lateral values of the coordinates.
- **lon** (int or float or numpy.ndarray) – Longitutional values of the coordinates.
- **symbole** (str, optional) – Name of the Variable. Default: "[Lat,Lon]"
- **units** (str, optional) – Units of the Variable. Default: "[deg,deg]"
- **description** (str, optional) – Description of the Variable. Default: "Coordinates given in degree-North and degree-East"

Notes

Here the variable name is fix set to "coordinates".

lat and lon should have the same shape.

Attributes

info
str: Info about the Variable.

label
str: String containing: symbol in units.

scalar
bool: State if the variable is of scalar type.

value
int or float or numpy.ndarray: Value.

Methods

__call__([value])	Call a variable.
save([path, name])	Save a variable to file.

__call__(*value=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **value** (int or float or numpy.ndarray,)
- **optional** – Value of the Variable. Default: None

Returns
value – Value of the Variable.

Return type
int or float or numpy.ndarray

`save`(*path=''*, *name=None*)

Save a variable to file.

This writes the variable to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Var_"+name`. Default: `None`

Notes

The file will get the suffix `".var"`.

`property info`

Info about the Variable.

Type
`str`

`property label`

`symbol in units`.

Type
`str`

Type
String containing

`property scalar`

State if the variable is of scalar type.

Type
`bool`

`property value`

Value.

Type
`int` or `float` or `numpy.ndarray`

welltestpy.data.Observation

class Observation(*name*, *observation*, *time=None*, *description='Observation'*)

Bases: `object`

Class for a observation.

This is a class for time-dependent observations. It has a name and a description.

Parameters

- **name** (`str`) – Name of the Variable.
- **observation** (*Variable*) – Name of the Variable. Default: `"x"`
- **time** (*Variable*) – Value of the Variable.
- **description** (`str`, optional) – Description of the Variable. Default: `"Observation"`

Attributes

info
Get information about the observation.

kind
`str`: name of the observation variable.

label
[`tuple` of] `str`: `symbol in units`.

labels
[`tuple` of] `str`: `symbol in units`.

observation
Observed values of the observation.

state
`str`: String containing state of the observation.

time
Time values of the observation.

units
[`tuple` of] `str`: units of the observation.

value
Value of the Observation.

Methods

__call__([observation, time])	Call a variable.
reshape()	Reshape observations to flat array.
save([path, name])	Save an observation to file.

__call__(*observation=None*, *time=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **observation** (scalar, `numpy.ndarray`, *Variable*, optional) – New Value for observation. Default: `"None"`
- **time** (scalar, `numpy.ndarray`, *Variable*, optional) – New Value for time. Default: `"None"`

Returns

- [`tuple` of] `int` or `float`
- or `numpy.ndarray` – `(time, observation)` or `observation`.

reshape()

Reshape observations to flat array.

save(*path=''*, *name=None*)

Save an observation to file.

This writes the observation to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Obs_"+name`. Default: `None`

Notes

The file will get the suffix `".obs"`.

property info

Get information about the observation.

Here you can display information about the observation.

property kind

name of the observation variable.

Type
`str`

property label

`symbol in units`.

Type
[`tuple` of] `str`

property labels

`symbol in units`.

Type
[`tuple` of] `str`

property observation

Observed values of the observation.

`int` or `float` or `numpy.ndarray`

property state

String containing state of the observation.

Either `"steady"` or `"transient"`.

Type
`str`

property time

Time values of the observation.

`int` or `float` or `numpy.ndarray`

property units

units of the observation.

Type

[tuple of] str

property value

Value of the Observation.

[tuple of] int or float or numpy.ndarray

welltestpy.data.StdyObs

class StdyObs(*name*, *observation*, *description='Steady observation'*)

Bases: *Observation*

Observation class special for steady observations.

Parameters

- **name** (str) – Name of the Variable.
- **observation** (*Variable*) – Name of the Variable. Default: "x"
- **description** (str, optional) – Description of the Variable. Default: "Steady observation"

Attributes

info
Get information about the observation.

kind
str: name of the observation variable.

label
[tuple of] str: symbol in units.

labels
[tuple of] str: symbol in units.

observation
Observed values of the observation.

state
str: String containing state of the observation.

time
Time values of the observation.

units
[tuple of] str: units of the observation.

value
Value of the Observation.

Methods

__call__([observation, time])	Call a variable.
reshape()	Reshape observations to flat array.
save([path, name])	Save an observation to file.

__call__(*observation=None*, *time=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **observation** (scalar, numpy.ndarray, *Variable*, optional) – New Value for observation. Default: "None"
- **time** (scalar, numpy.ndarray, *Variable*, optional) – New Value for time. Default: "None"

Returns

- [`tuple` of] `int` or `float`
- or `numpy.ndarray` – `(time, observation)` or `observation`.

reshape()

Reshape observations to flat array.

save(*path=''*, *name=None*)

Save an observation to file.

This writes the observation to a csv file.

Parameters

- **path** (`str`, optional) – Path where the variable should be saved. Default: `""`
- **name** (`str`, optional) – Name of the file. If `None`, the name will be generated by `"Obs_"+name`. Default: `None`

Notes

The file will get the suffix `".obs"`.

---

property info

Get information about the observation.

Here you can display information about the observation.

property kind

name of the observation variable.

Type
`str`

property label

`symbol in units`.

Type
[`tuple` of] `str`

property labels

`symbol in units`.

Type
[`tuple` of] `str`

property observation

Observed values of the observation.

`int` or `float` or `numpy.ndarray`

property state

String containing state of the observation.

Either `"steady"` or `"transient"`.

Type
`str`

property time

Time values of the observation.

`int` or `float` or `numpy.ndarray`

property units

units of the observation.

Type

[tuple of] str

property value

Value of the Observation.

[tuple of] int or float or numpy.ndarray

welltestpy.data.DrawdownObs

class DrawdownObs(*name*, *observation*, *time*, *description='Drawdown observation'*)

Bases: `Observation`

Observation class special for drawdown observations.

Parameters

- **name** (`str`) – Name of the Variable.
- **observation** (`Variable`) – Observation.
- **time** (`Variable`) – Time points of observation.
- **description** (`str`, optional) – Description of the Variable. Default: `"Drawdown observation"`

Attributes

`info`
: Get information about the observation.

`kind`
: `str`: name of the observation variable.

`label`
: [`tuple` of] `str`: `symbol in units`.

`labels`
: [`tuple` of] `str`: `symbol in units`.

`observation`
: Observed values of the observation.

`state`
: `str`: String containing state of the observation.

`time`
: Time values of the observation.

`units`
: [`tuple` of] `str`: units of the observation.

`value`
: Value of the Observation.

Methods

`__call__`([observation, time])	Call a variable.
`reshape`()	Reshape observations to flat array.
`save`([path, name])	Save an observation to file.

__call__(*observation=None*, *time=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **observation** (scalar, `numpy.ndarray`, `Variable`, optional) – New Value for observation. Default: `"None"`
- **time** (scalar, `numpy.ndarray`, `Variable`, optional) – New Value for time. Default: `"None"`

Returns

- [tuple of] int or float
- or numpy.ndarray – (time, observation) or observation.

reshape()

Reshape observations to flat array.

save(*path=''*, *name=None*)

Save an observation to file.

This writes the observation to a csv file.

Parameters

- **path** (str, optional) – Path where the variable should be saved. Default: ""
- **name** (str, optional) – Name of the file. If None, the name will be generated by "Obs_"+name. Default: None

---

Notes

The file will get the suffix ".obs".

---

property info

Get information about the observation.

Here you can display information about the observation.

property kind

name of the observation variable.

Type
str

property label

symbol in units.

Type
[tuple of] str

property labels

symbol in units.

Type
[tuple of] str

property observation

Observed values of the observation.

int or float or numpy.ndarray

property state

String containing state of the observation.

Either "steady" or "transient".

Type
str

property time

Time values of the observation.

int or float or numpy.ndarray

property units

units of the observation.

Type

[tuple of] str

property value

Value of the Observation.

[tuple of] int or float or numpy.ndarray

welltestpy.data.StdyHeadObs

class StdyHeadObs(*name*, *observation*, *description='Steady State Drawdown observation'*)

Bases: *Observation*

Observation class special for steady drawdown observations.

Parameters

- **name** (str) – Name of the Variable.
- **observation** (*Variable*) – Observation.
- **description** (str, optional) – Description of the Variable. Default: "Steady observation"

Attributes

info
Get information about the observation.

kind
str: name of the observation variable.

label
[tuple of] str: symbol in units.

labels
[tuple of] str: symbol in units.

observation
Observed values of the observation.

state
str: String containing state of the observation.

time
Time values of the observation.

units
[tuple of] str: units of the observation.

value
Value of the Observation.

Methods

__call__([observation, time])	Call a variable.
reshape()	Reshape observations to flat array.
save([path, name])	Save an observation to file.

__call__(*observation=None*, *time=None*)

Call a variable.

Here you can set a new value or you can get the value of the variable.

Parameters

- **observation** (scalar, numpy.ndarray, *Variable*, optional) – New Value for observation. Default: "None"
- **time** (scalar, numpy.ndarray, *Variable*, optional) – New Value for time. Default: "None"

Returns

- [tuple of] int or float
- or numpy.ndarray – (time, observation) or observation.

reshape()

Reshape observations to flat array.

save(*path=''*, *name=None*)

Save an observation to file.

This writes the observation to a csv file.

Parameters

- **path** (str, optional) – Path where the variable should be saved. Default: ""
- **name** (str, optional) – Name of the file. If None, the name will be generated by "Obs_"+name. Default: None

---

Notes

The file will get the suffix ".obs".

---

property info

Get information about the observation.

Here you can display information about the observation.

property kind

name of the observation variable.

Type

str

property label

symbol in units.

Type

[tuple of] str

property labels

symbol in units.

Type

[tuple of] str

property observation

Observed values of the observation.

int or float or numpy.ndarray

property state

String containing state of the observation.

Either "steady" or "transient".

Type

str

property time

Time values of the observation.

int or float or numpy.ndarray

property units

units of the observation.

Type

[tuple of] str

property value

Value of the Observation.

[tuple of] int or float or numpy.ndarray

welltestpy.data.Well

class Well(*name*, *radius*, *coordinates*, *welldepth=1.0*, *aquiferdepth=None*, *screensize=None*)

Bases: `object`

Class for a pumping-/observation-well.

This is a class for a well within a aquifer-testing campaign.

It has a name, a radius, coordinates and a depth.

Parameters

- **name** (`str`) – Name of the Variable.
- **radius** (`Variable` or `float`) – Value of the Variable.
- **coordinates** (`Variable` or `numpy.ndarray`) – Value of the Variable.
- **welldepth** (`Variable` or `float`, optional) – Depth of the well (in saturated zone). Default: 1.0
- **aquiferdepth** (`Variable` or `float`, optional) – Aquifer depth at the well (saturated zone). Defaults to welldepth. Default: `"None"`
- **screensize** (`Variable` or `float`, optional) – Size of the screen at the well. Defaults to 0.0. Default: `"None"`

Notes

You can calculate the distance between two wells `w1` and `w2` by simply calculating the difference `w1 - w2`.

Attributes

`aquifer`
`float`: Aquifer depth at the well.

`aquiferdepth`
`Variable`: Aquifer depth at the well.

`coordinates`
`Variable`: Coordinates variable of the well.

`depth`
`float`: Depth of the well.

`info`
Get information about the variable.

`is_piezometer`
`bool`: Whether the well is only a standpipe piezometer.

`pos`
`numpy.ndarray`: Position of the well.

`radius`
`float`: Radius of the well.

`screen`
`float`: Screen size at the well.

`screensize`
`Variable`: Screen size at the well.

`welldepth`
`Variable`: Depth variable of the well.

wellradius
Variable: Radius variable of the well.

Methods

distance(well)	Calculate distance to the well.
save([path, name])	Save a well to file.

distance(*well*)

Calculate distance to the well.

Parameters
well (*Well* or tuple of float) – Coordinates to calculate the distance to or another well.

save(*path=''*, *name=None*)

Save a well to file.

This writes the variable to a csv file.

Parameters

- **path** (str, optional) – Path where the variable should be saved. Default: `""`
- **name** (str, optional) – Name of the file. If `None`, the name will be generated by `"Well_"+name`. Default: `None`

Notes

The file will get the suffix `".wel"`.

property aquifer

Aquifer depth at the well.

Type
float

property aquiferdepth

Aquifer depth at the well.

Type
Variable

property coordinates

Coordinates variable of the well.

Type
Variable

property depth

Depth of the well.

Type
float

property info

Get information about the variable.

Here you can display information about the variable.

property is_piezometer

Whether the well is only a standpipe piezometer.

Type
`bool`

property pos

Position of the well.

Type
`numpy.ndarray`

property radius

Radius of the well.

Type
`float`

property screen

Screen size at the well.

Type
`float`

property screensize

Screen size at the well.

Type
Variable

property welldepth

Depth variable of the well.

Type
Variable

property wellradius

Radius variable of the well.

Type
Variable

Routines

Loading routines

Campaign related loading routines

load_campaign(cmpfile)	Load a campaign from file.
load_fieldsite(fdsfile)	Load a field site from file.

welltestpy.data.load_campaign

load_campaign(*cmpfile*)

Load a campaign from file.

This reads a campaign from a csv file.

Parameters
cmpfile (str) – Path to the file

welltestpy.data.load_fieldsite

load_fieldsite(*fdsfile*)

Load a field site from file.

This reads a field site from a csv file.

Parameters
fdsfile (str) – Path to the file

Field test related loading routines

load_test(tstfile)	Load a test from file.

welltestpy.data.load_test

load_test(*tstfile*)

Load a test from file.

This reads a test from a csv file.

Parameters
tstfile (str) – Path to the file

Variable related loading routines

load_var(varfile)	Load a variable from file.
load_obs(obsfile)	Load an observation from file.
load_well(welfile)	Load a well from file.

welltestpy.data.load_var

load_var(*varfile*)

Load a variable from file.

This reads a variable from a csv file.

Parameters
varfile (str) – Path to the file

welltestpy.data.load_obs

load_obs(*obsfile*)

Load an observation from file.

This reads a observation from a csv file.

Parameters
obsfile (str) – Path to the file

welltestpy.data.load_well

load_well(*welfile*)

Load a well from file.

This reads a well from a csv file.

Parameters
welfile (str) – Path to the file

welltestpy.estimate

welltestpy subpackage providing routines to estimate pump test parameters.

Estimators

The following estimators are provided

ExtTheis3D(name, campaign[, val_ranges, ...])	Class for an estimation of stochastic subsurface parameters.
ExtTheis2D(name, campaign[, val_ranges, ...])	Class for an estimation of stochastic subsurface parameters.
Neuman2004(name, campaign[, val_ranges, ...])	Class for an estimation of stochastic subsurface parameters.
Theis(name, campaign[, val_ranges, val_fix, ...])	Class for an estimation of homogeneous subsurface parameters.
ExtThiem3D(name, campaign[, make_steady, ...])	Class for an estimation of stochastic subsurface parameters.
ExtThiem2D(name, campaign[, make_steady, ...])	Class for an estimation of stochastic subsurface parameters.
Neuman2004Steady(name, campaign[, ...])	Class for an estimation of stochastic subsurface parameters.
Thiem(name, campaign[, make_steady, ...])	Class for an estimation of homogeneous subsurface parameters.

welltestpy.estimate.ExtTheis3D

class ExtTheis3D(*name*, *campaign*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: *TransientPumping*

Class for an estimation of stochastic subsurface parameters.

With this class you can run an estimation of statistical subsurface parameters. It utilizes the extended Theis solution in 3D which assumes a log-normal distributed conductivity field with a gaussian correlation function and an anisotropy ratio 0 < e <= 1.

Available values for fitting:

- `cond_gmean`: geometric mean conductivity
- `var`: variance of log-conductivity
- `len_scale`: correlation length scale of log-conductivity
- `anis`: anisotropy between horizontal and vertical correlation length
- `storage`: storage

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*

- **val_fix** (dict, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (dict, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, conductivity and storage will be fitted logarithmically and other values linearly. Default: None
- **val_fit_name** (dict, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (dict, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

gen_data()	Generate the observed drawdown at given time points.
gen_setup([prate_kw, rad_kw, time_kw, dummy])	Generate the Spotpy Setup.
run([rep, parallel, run, folder, dbname, ...])	Run the estimation.
sensitivity([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
setpumprate([prate])	Set a uniform pumping rate at all pumpingwells wells.
settime([time, tmin, tmax, typ, steps])	Set uniform time points for the observations.

gen_data()

Generate the observed drawdown at given time points.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *time_kw='time'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (str, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (str, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **time_kw** (str, optional) – Keyword name for the time in the used type curve. Default: "time"
- **dummy** (bool, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (int, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`

- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`

- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`
- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`sensitivity`(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`

- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`setpumprate`(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters
prate (`float`, optional) – Pumping rate. Default: `-1.0`

`settime`(*time=None*, *tmin=10.0*, *tmax=inf*, *typ='quad'*, *steps=10*)

Set uniform time points for the observations.

Parameters

- **time** (`numpy.ndarray`, optional) – Array of specified time points. If `None` is given, they will be determined by the observation data. Default: `None`
- **tmin** (`float`, optional) – Minimal time value. It will set a minimal value of 10s. Default: `10`
- **tmax** (`float`, optional) – Maximal time value. Default: `inf`
- **typ** (`str` or `float`, optional) –

  Typ of the time selection. You can select from:

  – `"exp"`: for exponential behavior

  – `"log"`: for logarithmic behavior

  – `"geo"`: for geometric behavior

  – `"lin"`: for linear behavior

  – `"quad"`: for quadratic behavior

  – `"cub"`: for cubic behavior

  – `float`: here you can specifi any exponent ("quad" would be equivalent to 2)

  Default: "quad"
- **steps** (`int`, optional) – Number of generated time steps. Default: 10

`campaign`

Copy of the input campaign to be modified

Type
welltestpy.data.Campaign

`campaign_raw`

Copy of the original input campaign

Type
welltestpy.data.Campaign

`data`

observation data

Type
`numpy.ndarray`

`default_ranges = {'anis': (0, 1), 'cond_gmean': (1e-07, 0.2), 'len_scale': (1, 50), 'storage': (2e-06, 0.4), 'var': (0, 10)}`

Default value ranges for the estimator.

Type
`dict`

estimated_para

estimated parameters by name

Type
`dict`

name

Name of the Estimation

Type
`str`

prate

Pumpingrate at the pumping well

Type
`float`

rad

array of the radii from the wells

Type
`numpy.ndarray`

radnames

names of the radii well combination

Type
`numpy.ndarray`

result

result of the spotpy estimation

Type
`list`

rinf

radius of the furthest wells

Type
`float`

rwell

radius of the pumping wells

Type
`float`

sens

result of the spotpy sensitivity analysis

Type
`dict`

setup_kw

TypeCurve Spotpy Setup definition

Type
`dict`

testinclude

dictionary of which tests should be included

Type
`dict`

`time`

time points of the observation

Type

`numpy.ndarray`

welltestpy.estimate.ExtTheis2D

class ExtTheis2D(*name*, *campaign*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: `TransientPumping`

Class for an estimation of stochastic subsurface parameters.

With this class you can run an estimation of statistical subsurface parameters. It utilizes the extended Theis solution in 2D which assumes a log-normal distributed transmissivity field with a gaussian correlation function.

Available values for fitting:

- `trans_gmean`: geometric mean transmissivity
- `var`: variance of log-transmissivity
- `len_scale`: correlation length scale of log-transmissivity
- `storage`: storage

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the paramters
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be “lin”, “log”, “exp”, “sqrt”, “quad”, “inv” or a tuple of two callable functions where the first is the transformation and the second is its inverse. “log” is for example equivalent to `(np.log, np.exp)`. By default, transmissivity and storage will be fitted logarithmically and other values linearly. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

`gen_data`()	Generate the observed drawdown at given time points.
`gen_setup`([prate_kw, rad_kw, time_kw, dummy])	Generate the Spotpy Setup.
`run`([rep, parallel, run, folder, dbname, ...])	Run the estimation.
`sensitivity`([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
`setpumprate`([prate])	Set a uniform pumping rate at all pumpingwells wells.
`settime`([time, tmin, tmax, typ, steps])	Set uniform time points for the observations.

`gen_data()`

Generate the observed drawdown at given time points.

It will also generate an array containing all radii of all well combinations.

`gen_setup`(*prate_kw='rate'*, *rad_kw='rad'*, *time_kw='time'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **time_kw** (`str`, optional) – Keyword name for the time in the used type curve. Default: "time"
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

`run`(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`
- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`sensitivity`(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`setpumprate`(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (`float`, optional) – Pumping rate. Default: `-1.0`

`settime`(*time=None*, *tmin=10.0*, *tmax=inf*, *typ='quad'*, *steps=10*)

Set uniform time points for the observations.

Parameters

- **time** (`numpy.ndarray`, optional) – Array of specified time points. If `None` is given, they will be determined by the observation data. Default: `None`
- **tmin** (`float`, optional) – Minimal time value. It will set a minimal value of 10s. Default: `10`
- **tmax** (`float`, optional) – Maximal time value. Default: `inf`
- **typ** (`str` or `float`, optional) –

  Typ of the time selection. You can select from:

  – `"exp"`: for exponential behavior

  – `"log"`: for logarithmic behavior

  – `"geo"`: for geometric behavior

  – `"lin"`: for linear behavior

  – `"quad"`: for quadratic behavior

  – `"cub"`: for cubic behavior

  – `float`: here you can specifi any exponent ("quad" would be equivalent to 2)

  Default: "quad"

- **steps** (`int`, optional) – Number of generated time steps. Default: 10

campaign

Copy of the input campaign to be modified

Type
welltestpy.data.Campaign

campaign_raw

Copy of the original input campaign

Type
welltestpy.data.Campaign

data

observation data

Type
`numpy.ndarray`

default_ranges = {'len_scale': (1, 50), 'storage': (2e-06, 0.4), 'trans_gmean': (1e-07, 0.2), 'var': (0, 10)}

Default value ranges for the estimator.

Type
`dict`

estimated_para

estimated parameters by name

Type
`dict`

name

Name of the Estimation

Type
`str`

prate

Pumpingrate at the pumping well

Type
`float`

rad

array of the radii from the wells

Type
`numpy.ndarray`

radnames

names of the radii well combination

Type
`numpy.ndarray`

result

result of the spotpy estimation

Type
`list`

rinf

radius of the furthest wells

Type
float

rwell

radius of the pumping wells

Type
float

sens

result of the spotpy sensitivity analysis

Type
dict

setup_kw

TypeCurve Spotpy Setup definition

Type
dict

testinclude

dictionary of which tests should be included

Type
dict

time

time points of the observation

Type
numpy.ndarray

welltestpy.estimate.Neuman2004

`class Neuman2004`(*name*, *campaign*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: `TransientPumping`

Class for an estimation of stochastic subsurface parameters.

With this class you can run an estimation of statistical subsurface parameters. It utilizes the apparent Transmissivity from Neuman 2004 which assumes a log-normal distributed transmissivity field with an exponential correlation function.

Available values for fitting:

- `trans_gmean`: geometric mean transmissivity
- `var`: variance of log-transmissivity
- `len_scale`: correlation length scale of log-transmissivity
- `storage`: storage

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the parameters
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be “lin”, “log”, “exp”, “sqrt”, “quad”, “inv” or a tuple of two callable functions where the first is the transformation and the second is its inverse. “log” is for example equivalent to `(np.log, np.exp)`. By default, transmissivity and storage will be fitted logarithmically and other values linearly. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

`gen_data`()	Generate the observed drawdown at given time points.
`gen_setup`([prate_kw, rad_kw, time_kw, dummy])	Generate the Spotpy Setup.
`run`([rep, parallel, run, folder, dbname, ...])	Run the estimation.
`sensitivity`([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
`setpumprate`([prate])	Set a uniform pumping rate at all pumpingwells wells.
`settime`([time, tmin, tmax, typ, steps])	Set uniform time points for the observations.

`gen_data()`

Generate the observed drawdown at given time points.

It will also generate an array containing all radii of all well combinations.

`gen_setup`(*prate_kw='rate'*, *rad_kw='rad'*, *time_kw='time'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **time_kw** (`str`, optional) – Keyword name for the time in the used type curve. Default: "time"
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

`run`(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`
- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`sensitivity`(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`setpumprate`(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (`float`, optional) – Pumping rate. Default: `-1.0`

`settime`(*time=None*, *tmin=10.0*, *tmax=inf*, *typ='quad'*, *steps=10*)

Set uniform time points for the observations.

Parameters

- **time** (`numpy.ndarray`, optional) – Array of specified time points. If `None` is given, they will be determined by the observation data. Default: `None`
- **tmin** (`float`, optional) – Minimal time value. It will set a minimal value of 10s. Default: `10`
- **tmax** (`float`, optional) – Maximal time value. Default: `inf`
- **typ** (`str` or `float`, optional) –

  Typ of the time selection. You can select from:

  – `"exp"`: for exponential behavior

  – `"log"`: for logarithmic behavior

  – `"geo"`: for geometric behavior

  – `"lin"`: for linear behavior

  – `"quad"`: for quadratic behavior

  – `"cub"`: for cubic behavior

  – `float`: here you can specifi any exponent ("quad" would be equivalent to 2)

  Default: "quad"

- **steps** (int, optional) – Number of generated time steps. Default: 10

`campaign`

Copy of the input campaign to be modified

Type
welltestpy.data.Campaign

`campaign_raw`

Copy of the original input campaign

Type
welltestpy.data.Campaign

`data`

observation data

Type
numpy.ndarray

`default_ranges = {'len_scale': (1, 50), 'storage': (2e-06, 0.4), 'trans_gmean': (1e-07, 0.2), 'var': (0, 10)}`

Default value ranges for the estimator.

Type
dict

`estimated_para`

estimated parameters by name

Type
dict

`name`

Name of the Estimation

Type
str

`prate`

Pumpingrate at the pumping well

Type
float

`rad`

array of the radii from the wells

Type
numpy.ndarray

`radnames`

names of the radii well combination

Type
numpy.ndarray

`result`

result of the spotpy estimation

Type
list

`rinf`

radius of the furthest wells

Type
`float`

`rwell`

radius of the pumping wells

Type
`float`

`sens`

result of the spotpy sensitivity analysis

Type
`dict`

`setup_kw`

TypeCurve Spotpy Setup definition

Type
`dict`

`testinclude`

dictionary of which tests should be included

Type
`dict`

`time`

time points of the observation

Type
`numpy.ndarray`

welltestpy.estimate.Theis

class Theis(*name*, *campaign*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: *TransientPumping*

Class for an estimation of homogeneous subsurface parameters.

With this class you can run an estimation of homogeneous subsurface parameters. It utilizes the Theis solution.

Available values for fitting:

- `transmissivity`: transmissivity
- `storage`: storage

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, transmissivity and storage will be fitted logarithmically. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

gen_data()	Generate the observed drawdown at given time points.
gen_setup([prate_kw, rad_kw, time_kw, dummy])	Generate the Spotpy Setup.
run([rep, parallel, run, folder, dbname, ...])	Run the estimation.
sensitivity([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
setpumprate([prate])	Set a uniform pumping rate at all pumpingwells wells.
settime([time, tmin, tmax, typ, steps])	Set uniform time points for the observations.

`gen_data()`

Generate the observed drawdown at given time points.

It will also generate an array containing all radii of all well combinations.

`gen_setup`(*prate_kw='rate'*, *rad_kw='rad'*, *time_kw='time'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **time_kw** (`str`, optional) – Keyword name for the time in the used type curve. Default: "time"
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

`run`(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`
- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`sensitivity`(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`setpumprate`(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (`float`, optional) – Pumping rate. Default: `-1.0`

`settime`(*time=None*, *tmin=10.0*, *tmax=inf*, *typ='quad'*, *steps=10*)

Set uniform time points for the observations.

Parameters

- **time** (`numpy.ndarray`, optional) – Array of specified time points. If `None` is given, they will be determined by the observation data. Default: `None`
- **tmin** (`float`, optional) – Minimal time value. It will set a minimal value of 10s. Default: `10`
- **tmax** (`float`, optional) – Maximal time value. Default: `inf`
- **typ** (`str` or `float`, optional) –

  Typ of the time selection. You can select from:

  – `"exp"`: for exponential behavior

  – `"log"`: for logarithmic behavior

  – `"geo"`: for geometric behavior

  – `"lin"`: for linear behavior

  – `"quad"`: for quadratic behavior

  – `"cub"`: for cubic behavior

  – `float`: here you can specifi any exponent ("quad" would be equivalent to 2)

  Default: "quad"

- **steps** (int, optional) – Number of generated time steps. Default: 10

`campaign`
: Copy of the input campaign to be modified

    Type
    welltestpy.data.Campaign

`campaign_raw`
: Copy of the original input campaign

    Type
    welltestpy.data.Campaign

`data`
: observation data

    Type
    numpy.ndarray

`default_ranges = {'storage': (2e-06, 0.4), 'transmissivity': (1e-07, 0.2)}`
: Default value ranges for the estimator.

    Type
    dict

`estimated_para`
: estimated parameters by name

    Type
    dict

`name`
: Name of the Estimation

    Type
    str

`prate`
: Pumpingrate at the pumping well

    Type
    float

`rad`
: array of the radii from the wells

    Type
    numpy.ndarray

`radnames`
: names of the radii well combination

    Type
    numpy.ndarray

`result`
: result of the spotpy estimation

    Type
    list

`rinf`
: radius of the furthest wells

    Type
    float

`rwell`
radius of the pumping wells

Type
`float`

`sens`
result of the spotpy sensitivity analysis

Type
`dict`

`setup_kw`
TypeCurve Spotpy Setup definition

Type
`dict`

`testinclude`
dictionary of which tests should be included

Type
`dict`

`time`
time points of the observation

Type
`numpy.ndarray`

welltestpy.estimate.ExtThiem3D

class ExtThiem3D(*name*, *campaign*, *make_steady=True*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: *SteadyPumping*

Class for an estimation of stochastic subsurface parameters.

With this class you can run an estimation of statistical subsurface parameters. It utilizes the extended Thiem solution in 3D which assumes a log-normal distributed conductivity field with a gaussian correlation function and an anisotropy ratio 0 < e <= 1.

Available values for fitting:

- `cond_gmean`: geometric mean conductivity
- `var`: variance of log-conductivity
- `len_scale`: correlation length scale of log-conductivity
- `anis`: anisotropy between horizontal and vertical correlation length

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: *PumpingTest.make_steady*. Default: True
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, conductivity will be fitted logarithmically and other values linearly. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

gen_data()	Generate the observed drawdown.
gen_setup([prate_kw, rad_kw, r_ref_kw, ...])	Generate the Spotpy Setup.
run([rep, parallel, run, folder, dbname, ...])	Run the estimation.
sensitivity([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
setpumprate([prate])	Set a uniform pumping rate at all pumpingwells wells.

gen_data()

Generate the observed drawdown.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *r_ref_kw='r_ref'*, *h_ref_kw='h_ref'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (str, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (str, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **r_ref_kw** (str, optional) – Keyword name for the reference radius in the used type curve. Default: "r_ref"
- **h_ref_kw** (str, optional) – Keyword name for the reference head in the used type curve. Default: "h_ref"
- **dummy** (bool, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (int, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (str, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (bool, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (str, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (str, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (str, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (str, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`

- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

sensitivity(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

setpumprate(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (`float`, optional) – Pumping rate. Default: `-1.0`

campaign

Copy of the input campaign to be modified

Type

welltestpy.data.Campaign

campaign_raw

Copy of the original input campaign

Type

welltestpy.data.Campaign

data
observation data

Type
numpy.ndarray

default_ranges = {'anis': (0, 1), 'cond_gmean': (1e-07, 0.2), 'len_scale': (1, 50), 'var': (0, 10)}
Default value ranges for the estimator.

Type
dict

estimated_para
estimated parameters by name

Type
dict

h_ref
reference head at the biggest distance

Type
float

name
Name of the Estimation

Type
str

prate
Pumpingrate at the pumping well

Type
float

r_ref
reference radius of the biggest distance

Type
float

rad
array of the radii from the wells

Type
numpy.ndarray

radnames
names of the radii well combination

Type
numpy.ndarray

result
result of the spotpy estimation

Type
list

rinf
radius of the furthest wells

Type
float

`rwell`
radius of the pumping wells

Type
`float`

`sens`
result of the spotpy sensitivity analysis

Type
`dict`

`setup_kw`
TypeCurve Spotpy Setup definition

Type
`dict`

`testinclude`
dictionary of which tests should be included

Type
`dict`

welltestpy.estimate.ExtThiem2D

class `ExtThiem2D`(*name*, *campaign*, *make_steady=True*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: `SteadyPumping`

Class for an estimation of stochastic subsurface parameters.

With this class you can run an estimation of statistical subsurface parameters. It utilizes the extended Thiem solution in 2D which assumes a log-normal distributed transmissivity field with a gaussian correlation function.

Available values for fitting:

- `trans_gmean`: geometric mean transmissivity
- `var`: variance of log-transmissivity
- `len_scale`: correlation length scale of log-transmissivity

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the parameters
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: `PumpingTest.make_steady`. Default: True
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, transmissivity will be fitted logarithmically and other values linearly. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

`gen_data`()	Generate the observed drawdown.
`gen_setup`([prate_kw, rad_kw, r_ref_kw, ...])	Generate the Spotpy Setup.
`run`([rep, parallel, run, folder, dbname, ...])	Run the estimation.
`sensitivity`([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
`setpumprate`([prate])	Set a uniform pumping rate at all pumpingwells wells.

gen_data()

Generate the observed drawdown.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *r_ref_kw='r_ref'*, *h_ref_kw='h_ref'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **r_ref_kw** (`str`, optional) – Keyword name for the reference radius in the used type curve. Default: "r_ref"
- **h_ref_kw** (`str`, optional) – Keyword name for the reference head in the used type curve. Default: "h_ref"
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`

- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`sensitivity`(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  - `"seq"`: sequential on one CPU
  - `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`setpumprate`(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters
prate (`float`, optional) – Pumping rate. Default: `-1.0`

`campaign`

Copy of the input campaign to be modified

Type
welltestpy.data.Campaign

`campaign_raw`

Copy of the original input campaign

Type
welltestpy.data.Campaign

data

observation data

Type
numpy.ndarray

default_ranges = {'len_scale': (1, 50), 'trans_gmean': (1e-07, 0.2), 'var': (0, 10)}

Default value ranges for the estimator.

Type
dict

estimated_para

estimated parameters by name

Type
dict

h_ref

reference head at the biggest distance

Type
float

name

Name of the Estimation

Type
str

prate

Pumpingrate at the pumping well

Type
float

r_ref

reference radius of the biggest distance

Type
float

rad

array of the radii from the wells

Type
numpy.ndarray

radnames

names of the radii well combination

Type
numpy.ndarray

result

result of the spotpy estimation

Type
list

rinf

radius of the furthest wells

Type
float

`rwell`

radius of the pumping wells

Type
`float`

`sens`

result of the spotpy sensitivity analysis

Type
`dict`

`setup_kw`

TypeCurve Spotpy Setup definition

Type
`dict`

`testinclude`

dictionary of which tests should be included

Type
`dict`

welltestpy.estimate.Neuman2004Steady

`class Neuman2004Steady(`*name*, *campaign*, *make_steady=True*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*`)`

Bases: *SteadyPumping*

Class for an estimation of stochastic subsurface parameters.

With this class you can run an estimation of statistical subsurface parameters from steady drawdown. It utilizes the apparent Transmissivity from Neuman 2004 which assumes a log-normal distributed transmissivity field with an exponential correlation function.

Available values for fitting:

- `trans_gmean`: geometric mean transmissivity
- `var`: variance of log-transmissivity
- `len_scale`: correlation length scale of log-transmissivity

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: *PumpingTest.make_steady*. Default: True
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be “lin”, “log”, “exp”, “sqrt”, “quad”, “inv” or a tuple of two callable functions where the first is the transformation and the second is its inverse. “log” is for example equivalent to `(np.log, np.exp)`. By default, transmissivity will be fitted logarithmically and other values linearly. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

`gen_data`()	Generate the observed drawdown.
`gen_setup`([prate_kw, rad_kw, r_ref_kw, ...])	Generate the Spotpy Setup.
`run`([rep, parallel, run, folder, dbname, ...])	Run the estimation.
`sensitivity`([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
`setpumprate`([prate])	Set a uniform pumping rate at all pumpingwells wells.

gen_data()

Generate the observed drawdown.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *r_ref_kw='r_ref'*, *h_ref_kw='h_ref'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: “rate”
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: “rad”
- **r_ref_kw** (`str`, optional) – Keyword name for the reference radius in the used type curve. Default: “r_ref”
- **h_ref_kw** (`str`, optional) – Keyword name for the reference head in the used type curve. Default: “h_ref”
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`

- **interactplotname** (str, optional) – File-name of the parameter interaction plot of the spotpy estimation. If None, it will be the current time + "_parainteract.pdf". Default: None
- **estname** (str, optional) – File-name of the results of the spotpy estimation. If None, it will be the current time + "_estimate". Default: None
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

sensitivity(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (int, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (str, optional) –

  State if the estimation should be run in parallel or not. Options:

  – "seq": sequential on one CPU

  – "mpi": use the mpi4py package

  Default: "seq"
- **folder** (str, optional) – Path to the output folder. If None the CWD is used. Default: None
- **dbname** (str, optional) – File-name of the database of the spotpy estimation. If None, it will be the current time + "_sensitivity_db". Default: None
- **plotname** (str, optional) – File-name of the result plot of the sensitivity analysis. If None, it will be the current time + "_sensitivity.pdf". Default: None
- **traceplotname** (str, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If None, it will be the current time + "_senstrace.pdf". Default: None
- **sensname** (str, optional) – File-name of the results of the FAST estimation. If None, it will be the current time + "_estimate". Default: None
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

setpumprate(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (float, optional) – Pumping rate. Default: -1.0

campaign

Copy of the input campaign to be modified

Type

welltestpy.data.Campaign

campaign_raw

Copy of the original input campaign

Type

welltestpy.data.Campaign

data
observation data

Type
numpy.ndarray

default_ranges = {'len_scale': (1, 50), 'trans_gmean': (1e-07, 0.2), 'var': (0, 10)}
Default value ranges for the estimator.

Type
dict

estimated_para
estimated parameters by name

Type
dict

h_ref
reference head at the biggest distance

Type
float

name
Name of the Estimation

Type
str

prate
Pumpingrate at the pumping well

Type
float

r_ref
reference radius of the biggest distance

Type
float

rad
array of the radii from the wells

Type
numpy.ndarray

radnames
names of the radii well combination

Type
numpy.ndarray

result
result of the spotpy estimation

Type
list

rinf
radius of the furthest wells

Type
float

`rwell`
radius of the pumping wells

Type
float

`sens`
result of the spotpy sensitivity analysis

Type
dict

`setup_kw`
TypeCurve Spotpy Setup definition

Type
dict

`testinclude`
dictionary of which tests should be included

Type
dict

welltestpy.estimate.Thiem

class Thiem(*name*, *campaign*, *make_steady=True*, *val_ranges=None*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *testinclude=None*, *generate=False*)

Bases: *SteadyPumping*

Class for an estimation of homogeneous subsurface parameters.

With this class you can run an estimation of homogeneous subsurface parameters. It utilizes the Thiem solution.

Available values for fitting:

- `transmissivity`: transmissivity

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: *PumpingTest.make_steady*. Default: True
- **val_ranges** (`dict`, optional) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value. Will default to *default_ranges*
- **val_fix** (`dict`, optional) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict`, optional) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, transmissivity will be fitted logarithmically. Default: None
- **val_fit_name** (`dict`, optional) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

gen_data()	Generate the observed drawdown.
gen_setup([prate_kw, rad_kw, r_ref_kw, ...])	Generate the Spotpy Setup.
run([rep, parallel, run, folder, dbname, ...])	Run the estimation.
sensitivity([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
setpumprate([prate])	Set a uniform pumping rate at all pumpingwells wells.

gen_data()

Generate the observed drawdown.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *r_ref_kw='r_ref'*, *h_ref_kw='h_ref'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (str, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (str, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **r_ref_kw** (str, optional) – Keyword name for the reference radius in the used type curve. Default: "r_ref"
- **h_ref_kw** (str, optional) – Keyword name for the reference head in the used type curve. Default: "h_ref"
- **dummy** (bool, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (int, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (str, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (bool, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (str, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (str, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (str, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (str, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`
- **interactplotname** (str, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (str, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

sensitivity(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (int, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (str, optional) –

  State if the estimation should be run in parallel or not. Options:

  – "seq": sequential on one CPU

  – "mpi": use the mpi4py package

  Default: "seq"
- **folder** (str, optional) – Path to the output folder. If None the CWD is used. Default: None
- **dbname** (str, optional) – File-name of the database of the spotpy estimation. If None, it will be the current time + "_sensitivity_db". Default: None
- **plotname** (str, optional) – File-name of the result plot of the sensitivity analysis. If None, it will be the current time + "_sensitivity.pdf". Default: None
- **traceplotname** (str, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If None, it will be the current time + "_senstrace.pdf". Default: None
- **sensname** (str, optional) – File-name of the results of the FAST estimation. If None, it will be the current time + "_estimate". Default: None
- **plot_style** (*str, optional*) – Plot style. The default is “WTP”.

setpumprate(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters
prate (float, optional) – Pumping rate. Default: -1.0

campaign

Copy of the input campaign to be modified

Type
welltestpy.data.Campaign

campaign_raw

Copy of the original input campaign

Type
welltestpy.data.Campaign

data

observation data

Type
numpy.ndarray

default_ranges = {'transmissivity': (1e-07, 0.2)}

Default value ranges for the estimator.

Type
dict

estimated_para

estimated parameters by name

Type
dict

h_ref
reference head at the biggest distance

Type
float

name
Name of the Estimation

Type
str

prate
Pumpingrate at the pumping well

Type
float

r_ref
reference radius of the biggest distance

Type
float

rad
array of the radii from the wells

Type
numpy.ndarray

radnames
names of the radii well combination

Type
numpy.ndarray

result
result of the spotpy estimation

Type
list

rinf
radius of the furthest wells

Type
float

rwell
radius of the pumping wells

Type
float

sens
result of the spotpy sensitivity analysis

Type
dict

setup_kw
TypeCurve Spotpy Setup definition

Type
dict

`testinclude`

dictionary of which tests should be included

Type

`dict`

Base Classes

Transient

All transient estimators are derived from the following class

TransientPumping(name, campaign, type_curve, ...)	Class to estimate transient Type-Curve parameters.

welltestpy.estimate.TransientPumping

class TransientPumping(*name*, *campaign*, *type_curve*, *val_ranges*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *val_plot_names=None*, *testinclude=None*, *generate=False*)

Bases: `object`

Class to estimate transient Type-Curve parameters.

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **type_curve** (`callable`) – The given type-curve. Output will be reshaped to flat array.
- **val_ranges** (`dict`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value.
- **val_fix** (`dict` or `None`) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict` or `None`) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, values will be fitted linear. Default: None
- **val_fit_name** (`dict` or `None`) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **val_plot_names** (`dict` or `None`) –

  Dictionary containing keyword names in the type-curve for each value.

  {value-name: string for plot legend}

  This is useful to get better plots. By default, parameter names will be value names. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

gen_data()	Generate the observed drawdown at given time points.
gen_setup([prate_kw, rad_kw, time_kw, dummy])	Generate the Spotpy Setup.
run([rep, parallel, run, folder, dbname, ...])	Run the estimation.
sensitivity([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
setpumprate([prate])	Set a uniform pumping rate at all pumpingwells wells.
settime([time, tmin, tmax, typ, steps])	Set uniform time points for the observations.

gen_data()

Generate the observed drawdown at given time points.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *time_kw='time'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: "rate"
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: "rad"
- **time_kw** (`str`, optional) – Keyword name for the time in the used type curve. Default: "time"
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`

- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`
- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`sensitivity`(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

`setpumprate`(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (`float`, optional) – Pumping rate. Default: `-1.0`

`settime`(*time=None*, *tmin=10.0*, *tmax=inf*, *typ='quad'*, *steps=10*)

Set uniform time points for the observations.

Parameters

- **time** (`numpy.ndarray`, optional) – Array of specified time points. If `None` is given, they will be determined by the observation data. Default: `None`
- **tmin** (`float`, optional) – Minimal time value. It will set a minimal value of 10s. Default: `10`
- **tmax** (`float`, optional) – Maximal time value. Default: `inf`

- **typ** (str or float, optional) –

  Typ of the time selection. You can select from:

  - "exp": for exponential behavior
  - "log": for logarithmic behavior
  - "geo": for geometric behavior
  - "lin": for linear behavior
  - "quad": for quadratic behavior
  - "cub": for cubic behavior
  - float: here you can specifi any exponent ("quad" would be equivalent to 2)

  Default: "quad"

- **steps** (int, optional) – Number of generated time steps. Default: 10

`campaign`

Copy of the input campaign to be modified

Type
welltestpy.data.Campaign

`campaign_raw`

Copy of the original input campaign

Type
welltestpy.data.Campaign

`data`

observation data

Type
numpy.ndarray

`estimated_para`

estimated parameters by name

Type
dict

`name`

Name of the Estimation

Type
str

`prate`

Pumpingrate at the pumping well

Type
float

`rad`

array of the radii from the wells

Type
numpy.ndarray

`radnames`

names of the radii well combination

Type
numpy.ndarray

result

result of the spotpy estimation

Type

list

rinf

radius of the furthest wells

Type

float

rwell

radius of the pumping wells

Type

float

sens

result of the spotpy sensitivity analysis

Type

dict

setup_kw

TypeCurve Spotpy Setup definition

Type

dict

testinclude

dictionary of which tests should be included

Type

dict

time

time points of the observation

Type

numpy.ndarray

Steady Pumping

All steady estimators are derived from the following class

SteadyPumping(name, campaign, type_curve, ...)	Class to estimate steady Type-Curve parameters.

welltestpy.estimate.SteadyPumping

class SteadyPumping(*name*, *campaign*, *type_curve*, *val_ranges*, *make_steady=True*, *val_fix=None*, *val_fit_type=None*, *val_fit_name=None*, *val_plot_names=None*, *testinclude=None*, *generate=False*)

Bases: `object`

Class to estimate steady Type-Curve parameters.

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used to estimate the parameters
- **type_curve** (`callable`) – The given type-curve. Output will be reshaped to flat array.
- **val_ranges** (`dict`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature. Ranges should be a tuple containing min and max value.
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: *PumpingTest.make_steady*. Default: True
- **val_fix** (`dict` or `None`) – Dictionary containing fixed values for the type-curve. Names should be as in the type-curve signature. Default: None
- **val_fit_type** (`dict` or `None`) – Dictionary containing fitting transformation type for each value. Names should be as in the type-curve signature. val_fit_type can be "lin", "log", "exp", "sqrt", "quad", "inv" or a tuple of two callable functions where the first is the transformation and the second is its inverse. "log" is for example equivalent to `(np.log, np.exp)`. By default, values will be fitted linear. Default: None
- **val_fit_name** (`dict` or `None`) – Display name of the fitting transformation. Will be the val_fit_type string if it is a predefined one, or `f` if it is a given callable as default for each value. Default: None
- **val_plot_names** (`dict` or `None`) –

  Dictionary containing keyword names in the type-curve for each value.

  {value-name: string for plot legend}

  This is useful to get better plots. By default, parameter names will be value names. Default: None
- **testinclude** (`dict`, optional) – Dictionary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

`gen_data`()	Generate the observed drawdown.
`gen_setup`([prate_kw, rad_kw, r_ref_kw, ...])	Generate the Spotpy Setup.
`run`([rep, parallel, run, folder, dbname, ...])	Run the estimation.
`sensitivity`([rep, parallel, folder, dbname, ...])	Run the sensitivity analysis.
`setpumprate`([prate])	Set a uniform pumping rate at all pumpingwells wells.

gen_data()

Generate the observed drawdown.

It will also generate an array containing all radii of all well combinations.

gen_setup(*prate_kw='rate'*, *rad_kw='rad'*, *r_ref_kw='r_ref'*, *h_ref_kw='h_ref'*, *dummy=False*)

Generate the Spotpy Setup.

Parameters

- **prate_kw** (`str`, optional) – Keyword name for the pumping rate in the used type curve. Default: “rate”
- **rad_kw** (`str`, optional) – Keyword name for the radius in the used type curve. Default: “rad”
- **r_ref_kw** (`str`, optional) – Keyword name for the reference radius in the used type curve. Default: “r_ref”
- **h_ref_kw** (`str`, optional) – Keyword name for the reference head in the used type curve. Default: “h_ref”
- **dummy** (`bool`, optional) – Add a dummy parameter to the model. This could be used to equalize sensitivity analysis. Default: False

run(*rep=5000*, *parallel='seq'*, *run=True*, *folder=None*, *dbname=None*, *traceplotname=None*, *fittingplotname=None*, *interactplotname=None*, *estname=None*, *plot_style='WTP'*)

Run the estimation.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the SCEua algorithm in spotpy. Default: `5000`
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **run** (`bool`, optional) – State if the estimation should be executed. Otherwise all plots will be done with the previous results. Default: `True`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_db"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy estimation. If `None`, it will be the current time + `"_paratrace.pdf"`. Default: `None`
- **fittingplotname** (`str`, optional) – File-name of the fitting plot of the estimation. If `None`, it will be the current time + `"_fit.pdf"`. Default: `None`

- **interactplotname** (`str`, optional) – File-name of the parameter interaction plot of the spotpy estimation. If `None`, it will be the current time + `"_parainteract.pdf"`. Default: `None`
- **estname** (`str`, optional) – File-name of the results of the spotpy estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

sensitivity(*rep=None*, *parallel='seq'*, *folder=None*, *dbname=None*, *plotname=None*, *traceplotname=None*, *sensname=None*, *plot_style='WTP'*)

Run the sensitivity analysis.

Parameters

- **rep** (`int`, optional) – The number of repetitions within the FAST algorithm in spotpy. Default: estimated
- **parallel** (`str`, optional) –

  State if the estimation should be run in parallel or not. Options:

  – `"seq"`: sequential on one CPU

  – `"mpi"`: use the mpi4py package

  Default: `"seq"`
- **folder** (`str`, optional) – Path to the output folder. If `None` the CWD is used. Default: `None`
- **dbname** (`str`, optional) – File-name of the database of the spotpy estimation. If `None`, it will be the current time + `"_sensitivity_db"`. Default: `None`
- **plotname** (`str`, optional) – File-name of the result plot of the sensitivity analysis. If `None`, it will be the current time + `"_sensitivity.pdf"`. Default: `None`
- **traceplotname** (`str`, optional) – File-name of the parameter trace plot of the spotpy sensitivity analysis. If `None`, it will be the current time + `"_senstrace.pdf"`. Default: `None`
- **sensname** (`str`, optional) – File-name of the results of the FAST estimation. If `None`, it will be the current time + `"_estimate"`. Default: `None`
- **plot_style** (*str, optional*) – Plot style. The default is "WTP".

setpumprate(*prate=-1.0*)

Set a uniform pumping rate at all pumpingwells wells.

We assume linear scaling by the pumpingrate.

Parameters

prate (`float`, optional) – Pumping rate. Default: `-1.0`

campaign

Copy of the input campaign to be modified

Type

`welltestpy.data.Campaign`

campaign_raw

Copy of the original input campaign

Type

`welltestpy.data.Campaign`

data

observation data

Type

`numpy.ndarray`

estimated_para

estimated parameters by name

Type

`dict`

h_ref

reference head at the biggest distance

Type

`float`

name

Name of the Estimation

Type

`str`

prate

Pumpingrate at the pumping well

Type

`float`

r_ref

reference radius of the biggest distance

Type

`float`

rad

array of the radii from the wells

Type

`numpy.ndarray`

radnames

names of the radii well combination

Type

`numpy.ndarray`

result

result of the spotpy estimation

Type

`list`

rinf

radius of the furthest wells

Type

`float`

rwell

radius of the pumping wells

Type

`float`

sens

result of the spotpy sensitivity analysis

Type

dict

setup_kw

TypeCurve Spotpy Setup definition

Type

dict

testinclude

dictionary of which tests should be included

Type

dict

Helper

fast_rep(para_no[, infer_fac, freq_step])	Get number of iterations needed for the FAST algorithm.

welltestpy.estimate.fast_rep

`fast_rep`(*para_no*, *infer_fac=4*, *freq_step=2*)

Get number of iterations needed for the FAST algorithm.

Parameters

- **para_no** (`int`) – Number of parameters in the model.
- **infer_fac** (`int`, optional) – The inference factor. Default: 4
- **freq_step** (`int`, optional) – The frequency step. Default: 2

welltestpy.process

welltestpy subpackage providing routines to pre process test data.

Included functions

The following classes and functions are provided

normpumptest(pumptest[, pumpingrate, factor])	Normalize the pumping rate of a pumping test.
combinepumptest(campaign, test1, test2[, ...])	Combine two pumping tests to one.
filterdrawdown(observation[, tout, dxscale])	Smooth the drawdown data of an observation well.
cooper_jacob_correction(observation, ...)	Correction method for observed drawdown for unconfined aquifers.
smoothing_derivative(head, time[, method])	Calculate the derivative of the drawdown curve.

welltestpy.process.normpumptest

normpumptest(*pumptest*, *pumpingrate=-1.0*, *factor=1.0*)

Normalize the pumping rate of a pumping test.

Parameters

- **pumpingrate** (`float`, optional) – Pumping rate. Default: `-1.0`
- **factor** (`float`, optional) – Scaling factor that can be used for unit conversion. Default: `1.0`

welltestpy.process.combinepumptest

combinepumptest(*campaign*, *test1*, *test2*, *pumpingrate=None*, *finalname=None*, *factor1=1.0*, *factor2=1.0*, *infooftest1=True*, *replace=True*)

Combine two pumping tests to one.

They need to have the same pumping well.

Parameters

- **campaign** (*welltestpy.data.Campaign*) – The pumping test campaign which should be used.
- **test1** (`str`) – Name of test 1.
- **test2** (`str`) – Name of test 2.
- **pumpingrate** (`float`, optional) – Pumping rate. Default: `-1.0`
- **finalname** (`str`, optional) – Name of the final test. If *replace* is *True* and *finalname* is *None*, it will get the name of test 1. Else it will get a combined name of test 1 and test 2. Default: `None`
- **factor1** (`float`, optional) – Scaling factor for test 1 that can be used for unit conversion. Default: `1.0`
- **factor2** (`float`, optional) – Scaling factor for test 2 that can be used for unit conversion. Default: `1.0`
- **infooftest1** (`bool`, optional) – State if the final test should take the information from test 1. Default: `True`
- **replace** (`bool`, optional) – State if the original tests should be erased. Default: `True`

welltestpy.process.filterdrawdown

`filterdrawdown`(*observation*, *tout=None*, *dxscale=2*)

Smooth the drawdown data of an observation well.

Parameters

- **observation** (*welltestpy.data.Observation*) – The observation to be smoothed.
- **tout** (`numpy.ndarray`, optional) – Time points to evaluate the smoothed observation at. If `None`, the original time points of the observation are taken. Default: `None`
- **dxscale** (`int`, optional) – Scale of time-steps used for smoothing. Default: `2`

welltestpy.process.cooper_jacob_correction

`cooper_jacob_correction`(*observation*, *sat_thickness*)

Correction method for observed drawdown for unconfined aquifers.

Parameters

- **observation** (*welltestpy.data.Observation*) – The observation to be corrected.
- **sat_thickness** (`float`) – Vertical length of the aquifer in which its pores are filled with water.

Return type

The corrected drawdown

welltestpy.process.smoothing_derivative

`smoothing_derivative`(*head*, *time*, *method='bourdet'*)

Calculate the derivative of the drawdown curve.

Parameters

- **head** (*:class: 'array'*) – An array with the observed head values.
- **time** (*:class: 'array'*) – An array with the time values for the observed head values.
- **method** (`str`, optional) – Method to calculate the time derivative. Default: "bourdet"

Return type

The derivative of the observed heads.

welltestpy.tools

welltestpy subpackage providing miscellaneous tools.

Included functions

The following functions are provided for point triangulation

triangulate(distances, prec[, all_pos])	Triangulate points by given distances.
sym(A)	Get the symmetrized version of a lower or upper triangle-matrix A.

welltestpy.tools.triangulate

triangulate(*distances*, *prec*, *all_pos=False*)

Triangulate points by given distances.

try to triangulate points by given distances within a symmetric matrix 'distances' with `distances[i,j] = |pi-pj|`

thereby `p0` will be set to the origin `(0,0)` and `p1` to `(|p0-p1|,0)`

Parameters

- **distances** (`numpy.ndarray`) – Given distances among the point to be triangulated. It hat to be a symmetric matrix with a vanishing diagonal and

  ```
  distances[i,j] = |pi-pj|
  ```

  If a distance is unknown, you can set it to `-1`.

- **prec** (`float`) – Given Precision to be used within the algorithm. This can be used to smooth away measure errors
- **all_pos** (`bool`, optional) – If *True* all possible constellations will be calculated. Otherwise, the first possibility will be returned. Default: False

welltestpy.tools.sym

sym(*A*)

Get the symmetrized version of a lower or upper triangle-matrix A.

The following plotting routines are provided

campaign_plot(campaign[, select_test, fig, ...])	Plot an overview of the tests within the campaign.
fadeline(ax, x, y[, label, color, steps])	Fading line for matplotlib.
plot_well_pos(well_const[, names, title, ...])	Plot all well constellations and label the points with the names.
campaign_well_plot(campaign[, plot_tests, ...])	Plot of the well constellation within the campaign.
plotfit_transient(setup, data, para, rad, ...)	Plot of transient estimation fitting.
plotfit_steady(setup, data, para, rad, ...)	Plot of steady estimation fitting.
plotparainteract(result, paranames[, ...])	Plot of parameter interaction.
plotparatrace(result[, parameternames, ...])	Plot of parameter trace.
plotsensitivity(paralabels, sensitivities[, ...])	Plot of sensitivity results.
diagnostic_plot_pump_test(observation, rate)	Plot the derivative with the original data.

welltestpy.tools.campaign_plot

`campaign_plot`(*campaign*, *select_test=None*, *fig=None*, *style='WTP'*, ***kwargs*)

Plot an overview of the tests within the campaign.

Parameters

- **campaign** (`Campaign`) – The campaign to be plotted.
- **select_test** (*dict, optional*) – The selected tests to be added to the plot. The default is None.
- **fig** (*Figure, optional*) – Matplotlib figure to plot on. The default is None.
- **style** (*str, optional*) – Plot style. The default is "WTP".
- ****kwargs** (*TYPE*) – Keyword arguments forwarded to the tests plotting routines.

Returns

fig – The created matplotlib figure.

Return type

Figure

welltestpy.tools.fadeline

`fadeline`(*ax*, *x*, *y*, *label=None*, *color=None*, *steps=20*, ***kwargs*)

Fading line for matplotlib.

This is a workaround to produce a fading line.

Parameters

- **ax** (*axis*) – Axis to plot on.
- **x** (`list`) – start and end value of x components of the line
- **y** (`list`) – start and end value of y components of the line
- **label** (`str`, optional) – label for the legend. Default: `None`
- **color** (*MPL color, optional*) – color of the line Default: `None`
- **steps** (`int`, optional) – steps of fading Default: `20`
- ****kwargs** – keyword arguments that are forwarded to *plt.plot*

welltestpy.tools.plot_well_pos

`plot_well_pos`(*well_const*, *names=None*, *title=''*, *filename=None*, *plot_well_names=True*, *ticks_set='auto'*, *fig=None*, *style='WTP'*)

Plot all well constellations and label the points with the names.

Parameters

- **well_const** (*list*) – List of well constellations.
- **names** (*list of str, optional*) – Names for the wells. The default is None.
- **title** (*str, optional*) – Plot title. The default is "".
- **filename** (*str, optional*) – Filename if the result should be saved. The default is None.
- **plot_well_names** (*bool, optional*) – Whether to plot the well-names. The default is True.
- **ticks_set** (*int or str, optional*) – Tick spacing in the plot. The default is "auto".

- **fig** (*Figure, optional*) – Matplotlib figure to plot on. The default is None.
- **style** (*str, optional*) – Plot style. The default is "WTP".

Returns
fig – The created matplotlib figure.

Return type
Figure

welltestpy.tools.campaign_well_plot

`campaign_well_plot`(*campaign*, *plot_tests=True*, *plot_well_names=True*, *fig=None*, *style='WTP'*)

Plot of the well constellation within the campaign.

Parameters

- **campaign** (`Campaign`) – The campaign to be plotted.
- **plot_tests** (*bool, optional*) – DESCRIPTION. The default is True.
- **plot_well_names** (*TYPE, optional*) – DESCRIPTION. The default is True.
- **fig** (*Figure, optional*) – Matplotlib figure to plot on. The default is None.
- **style** (*str, optional*) – Plot style. The default is "WTP".

Returns
ax – The created matplotlib axes.

Return type
Axes

welltestpy.tools.plotfit_transient

`plotfit_transient`(*setup*, *data*, *para*, *rad*, *time*, *radnames*, *extra*, *plotname=None*, *fig=None*, *ax=None*, *style='WTP'*)

Plot of transient estimation fitting.

welltestpy.tools.plotfit_steady

`plotfit_steady`(*setup*, *data*, *para*, *rad*, *radnames*, *extra*, *plotname=None*, *ax_ins=True*, *fig=None*, *ax=None*, *style='WTP'*)

Plot of steady estimation fitting.

welltestpy.tools.plotparainteract

`plotparainteract`(*result*, *paranames*, *plotname=None*, *fig=None*, *style='WTP'*)

Plot of parameter interaction.

welltestpy.tools.plotparatrace

`plotparatrace`(*result*, *parameternames=None*, *parameterlabels=None*, *xticks=None*, *stdvalues=None*, *plotname=None*, *fig=None*, *style='WTP'*)

Plot of parameter trace.

welltestpy.tools.plotsensitivity

`plotsensitivity`(*paralabels*, *sensitivities*, *plotname=None*, *fig=None*, *ax=None*, *style='WTP'*)

Plot of sensitivity results.

welltestpy.tools.diagnostic_plot_pump_test

`diagnostic_plot_pump_test`(*observation*, *rate*, *method='bourdet'*, *linthresh_time=1.0*, *linthresh_head=1e-05*, *fig=None*, *ax=None*, *plotname=None*, *style='WTP'*)

Plot the derivative with the original data.

Parameters

- **observation** (`welltestpy.data.Observation`) – The observation to calculate the derivative.
- **rate** (`float`) – Pumping rate.
- **method** (`str`, optional) – Method to calculate the time derivative. Default: "bourdet"
- **linthresh_time** (*:class: 'float'*) – Range of time around 0 that behaves linear. Default: 1
- **linthresh_head** (*:class: 'float'*) – Range of head values around 0 that behaves linear. Default: 1e-5
- **fig** (*Figure, optional*) – Matplotlib figure to plot on. Default: None.
- **ax** (`Axes`) – Matplotlib axes to plot on. Default: None.
- **plotname** (*str, optional*) – Plot name if the result should be saved. Default: None.
- **style** (*str, optional*) – Plot style. Default: "WTP".

Return type

Diagnostic plot

3.2 Classes

Campaign classes

The following classes can be used to handle field campaigns.

Campaign(name[, fieldsite, wells, tests, ...])	Class for a well based campaign.
FieldSite(name[, description, coordinates])	Class for a field site.

Field Test classes

The following classes can be used to handle field test within a campaign.

PumpingTest(name, pumpingwell, pumpingrate)	Class for a pumping test.

3.3 Loading routines

Campaign related loading routines

load_campaign(cmpfile)	Load a campaign from file.

CHAPTER 4

CHANGELOG

All notable changes to **welltestpy** will be documented in this file.

4.1 1.2.0 - 2023-04

See #28, #31 and #32

Enhancements

- added archive support
- simplify documentation
- new arguments `val_fit_type` and `val_fit_name` for all estimators to select fitting transformation
- `val_fit_name` will be incorporated into the generated plots and the header of the estimation result file

Changes

- move to `src/` based package structure
- use hatchling as build backend
- drop py36 support
- value names for all arguments in the estimators now need to match the call signatures of the used type-curves

Bugfixes

- minor fixes for the plotting routines and the estimators

4.2 1.1.0 - 2021-07

Enhancements

- added `cooper_jacob_correction` to `process` (thanks to Jarno Herrmann)
- added `diagnostic_plots` module (thanks to Jarno Herrmann)
- added `screensize`, `screen`, `aquifer` and `is_piezometer` attribute to `Well` class
- added version information to output files
- added `__repr__` to `Campaign`

Changes

- modernized packaging workflow using `pyproject.toml`
- removed `setup.py` (use `pip>21.1` for editable installs)
- removed `dev` as extra install dependencies
- better exceptions in loading routines
- removed pandas dependency
- simplified readme

Bugfixes

- loading steady pumping tests was not possible due to a bug

4.3 1.0.3 - 2021-02

Enhancements

- Estimations: run method now provides `plot_style` keyword to control plotting

Changes

- Fit plot style for transient pumping tests was updated

Bugfixes

- Estimations: run method was throwing an Error when setting `run=False`
- Plotter: all plotting routines now respect setted font-type from matplotlib

4.4 1.0.2 - 2020-09-03

Bugfixes

- `StdyHeadObs` and `StdyObs` weren't usable due to an unnecessary `time` check

4.5 1.0.1 - 2020-04-09

Bugfixes

- Wrong URL in setup

4.6 1.0.0 - 2020-04-09

Enhancements

- new estimators
  - ExtTheis3D
  - ExtTheis2D
  - Neuman2004
  - Theis
  - ExtThiem3D
  - ExtThiem2D
  - Neuman2004Steady
  - Thiem
- better plotting
- unit-tests run with py35-py38 on Linux/Win/Mac
- coverage calculation
- sphinx gallery for examples
- allow style setting in plotting routines

Bugfixes

- estimation results stored as dict (order could alter before)

Changes

- py2 support dropped
- `Fieldsite.coordinates` now returns a `Variable`; `Fieldsite.pos` as shortcut
- `Fieldsite.pumpingrate` now returns a `Variable`; `Fieldsite.rate` as shortcut
- `Fieldsite.auqiferradius` now returns a `Variable`; `Fieldsite.radius` as shortcut
- `Fieldsite.auqiferdepth` now returns a `Variable`; `Fieldsite.depth` as shortcut
- `Well.coordinates` now returns a `Variable`; `Well.pos` as shortcut
- `Well.welldepth` now returns a `Variable`; `Well.depth` as shortcut
- `Well.wellradius` added and returns the radius `Variable`
- `Well.aquiferdepth` now returns a `Variable`
- `Fieldsite.addobservations` renamed to `Fieldsite.add_observations`
- `Fieldsite.delobservations` renamed to `Fieldsite.del_observations`
- `Observation` has changed order of inputs/outputs. Now: `observation`, `time`

4.7 0.3.2 - 2019-03-08

Bugfixes

- adopt AnaFlow API

4.8 0.3.1 - 2019-03-08

Bugfixes

- update travis workflow

4.9 0.3.0 - 2019-02-28

Enhancements

- added documentation

4.10 0.2.0 - 2018-04-25

Enhancements

- added license

4.11 0.1.0 - 2018-04-25

First alpha release of welltespy.

PYTHON MODULE INDEX

W

INDEX

Symbols

A

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W