



welltestpy Documentation

Release 1.0.1

Sebastian Mueller

Sep 02, 2020

1	welltestpy Quickstart	1
1.1	Installation	1
1.2	Provided Subpackages	1
1.3	Requirements	1
1.4	License	2
2	welltestpy Tutorial	3
2.1	Gallery	3
	Creating a pumping test campaign	3
	Estimate homogeneous parameters	5
	Estimate steady homogeneous parameters	9
	Estimate steady heterogeneous parameters	13
	Point triangulation	19
3	welltestpy API	21
3.1	Purpose	21
	Subpackages	21
	Classes	21
	Loading routines	21
3.2	welltestpy.data	23
	Subpackages	23
	Classes	23
	Routines	23
	welltestpy.data.data_io	25
	welltestpy.data.varlib	28
	welltestpy.data.testslib	38
	welltestpy.data.campaignlib	42
3.3	welltestpy.estimate	46
	Estimators	46
	Base Classes	46
3.4	welltestpy.process	61
	Included functions	61
3.5	welltestpy.tools	62
	Included functions	62
	Python Module Index	65
	Index	67



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

1.1 Installation

The package can be installed via [pip](#). On Windows you can install [WinPython](#) to get Python and pip running.

```
pip install welltestpy
```

1.2 Provided Subpackages

The following functions are provided directly

```
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 triangulation
```

1.3 Requirements

- NumPy >= 1.14.5
- SciPy >= 1.1.0
- Pandas >= 0.23.2

- AnaFlow \geq 1.0.0
- SpotPy \geq 1.5.0
- Matplotlib \geq 3.0.0

1.4 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 numpy as np
import welltestpy as wtp
import anaflow as ana
```

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
]
drawdown = ana.theis(
    time=time,
    rad=rad,
```

(continues on next page)

(continued from previous page)

```

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()

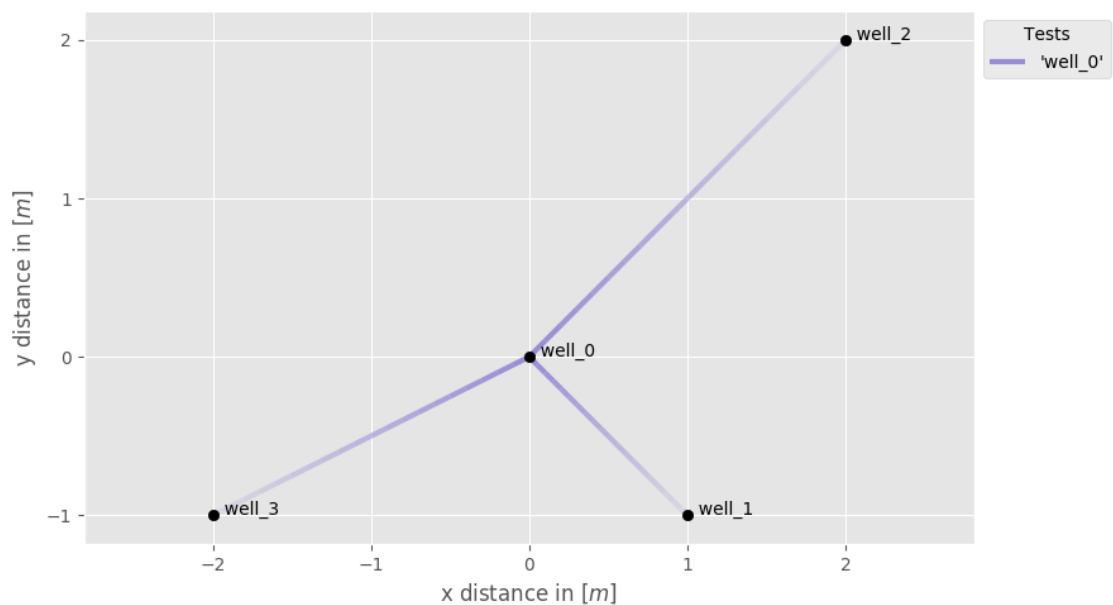
```

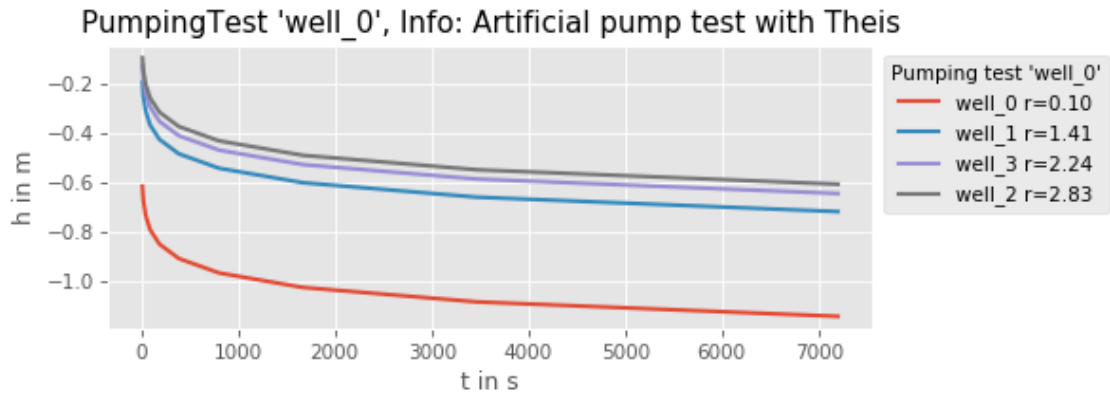
Plot the well constellation and a test overview

```

campaign.plot_wells()
campaign.plot()

```





Save the whole campaign to a file

```
campaign.save()
```

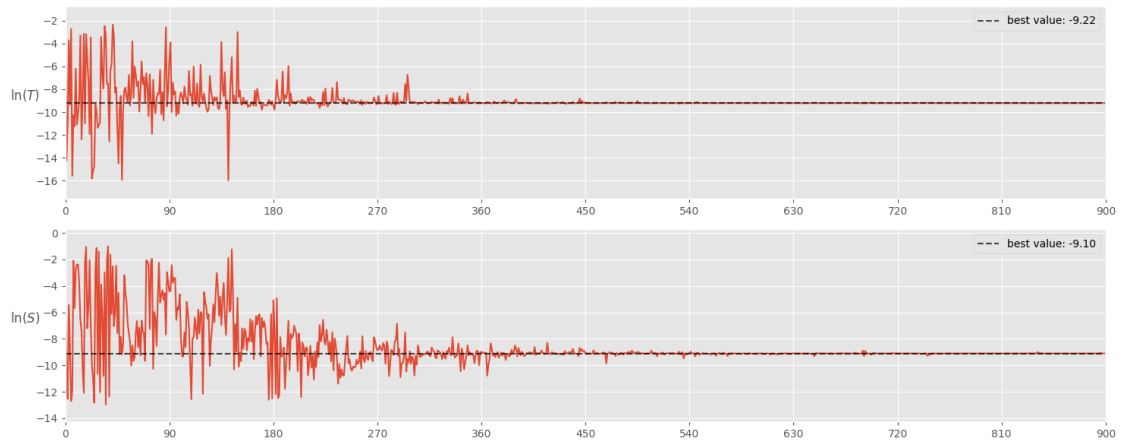
Total running time of the script: (0 minutes 1.761 seconds)

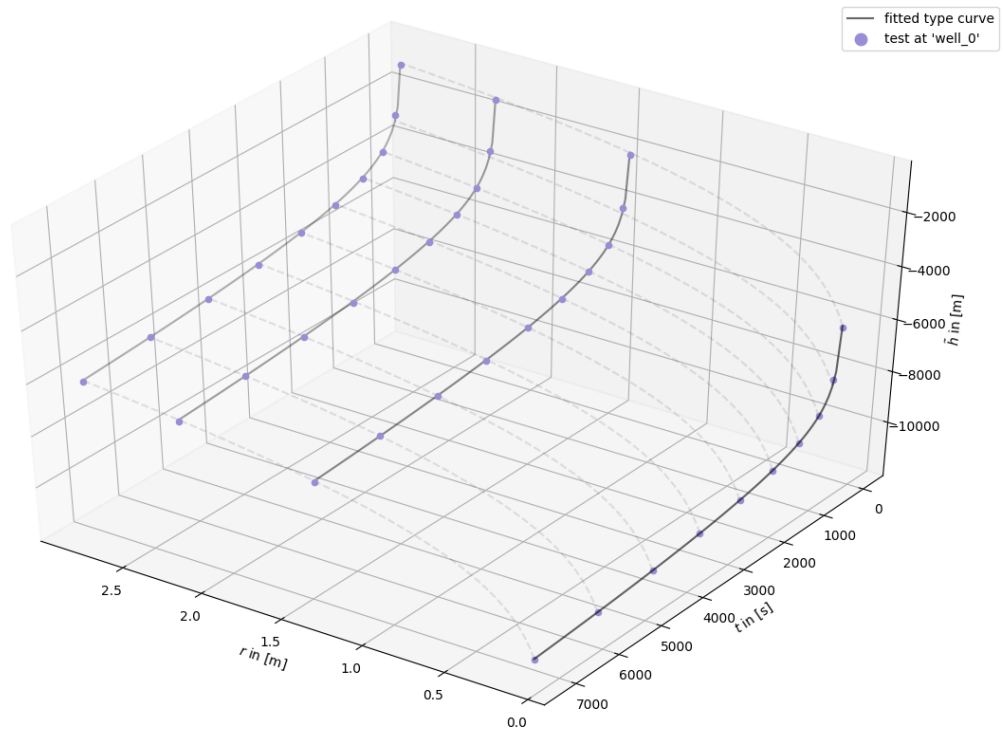
Estimate homogeneous parameters

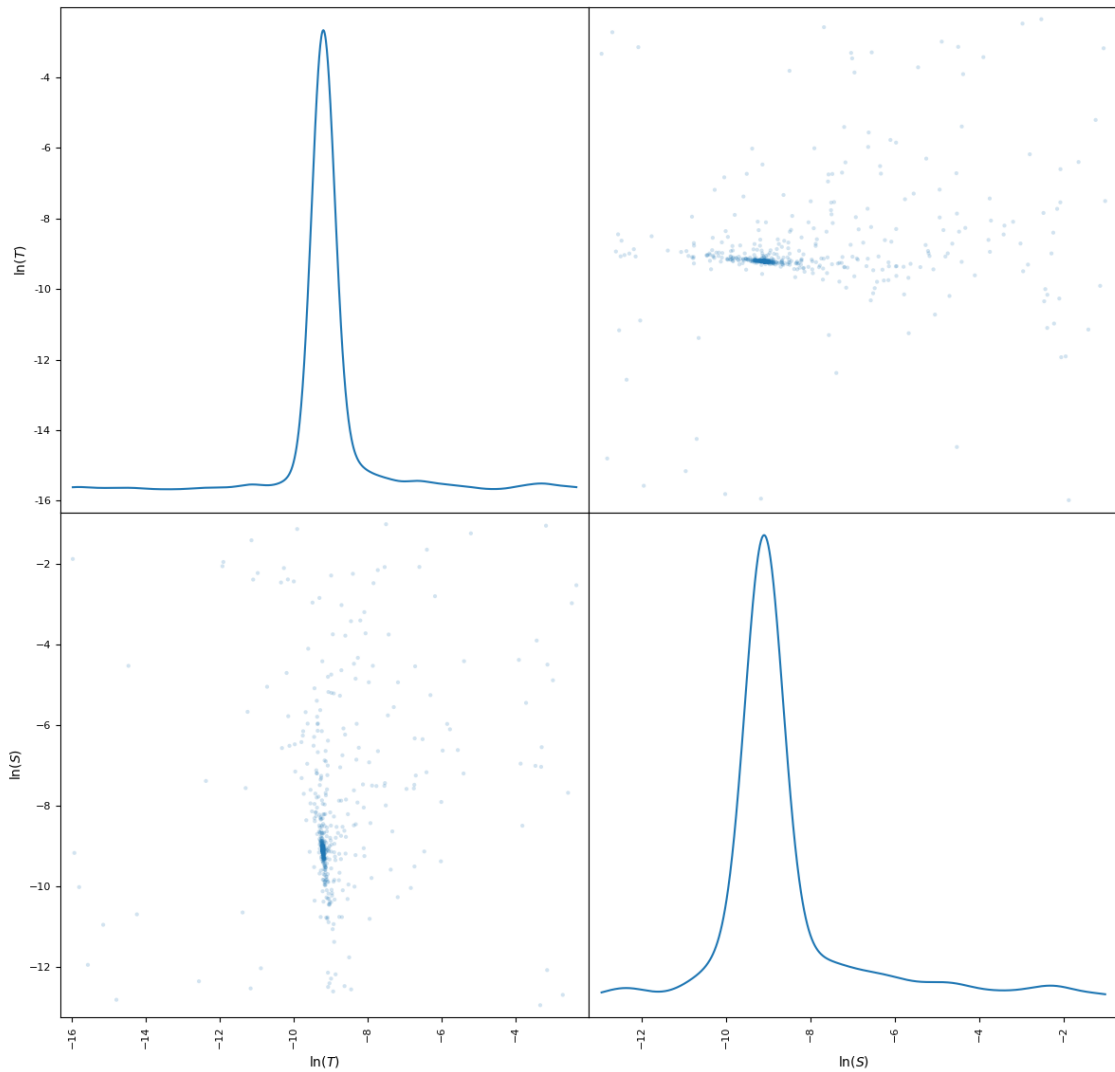
Here we estimate transmissivity and storage from a pumping test campaign with the classical theis solution.

```
import welltestpy as wtp

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







Out:

```

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/v1.0.1/examples/Estimate_theis/2020-09-02_14-47-45_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...
THE POPULATION HAS CONVERGED TO A PRESPECIFIED SMALL PARAMETER SPACE
SEARCH WAS STOPPED AT TRIAL NUMBER: 2314
NUMBER OF DISCARDED TRIALS: 0
NORMALIZED GEOMETRIC RANGE = 0.000849
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 1.19 seconds
Total Repetitions: 2314
Minimal objective value: 77.2517
Corresponding parameter setting:
mu: -9.21583
lnS: -9.10167
*****

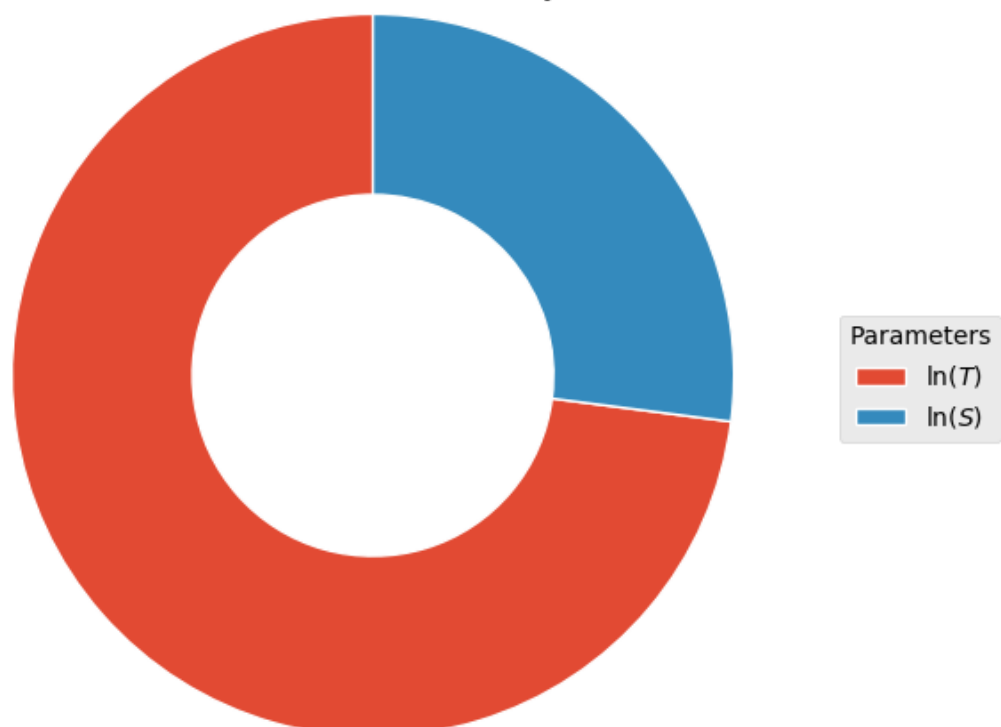
Best parameter set:
mu=-9.215826791505409, lnS=-9.101673728278687

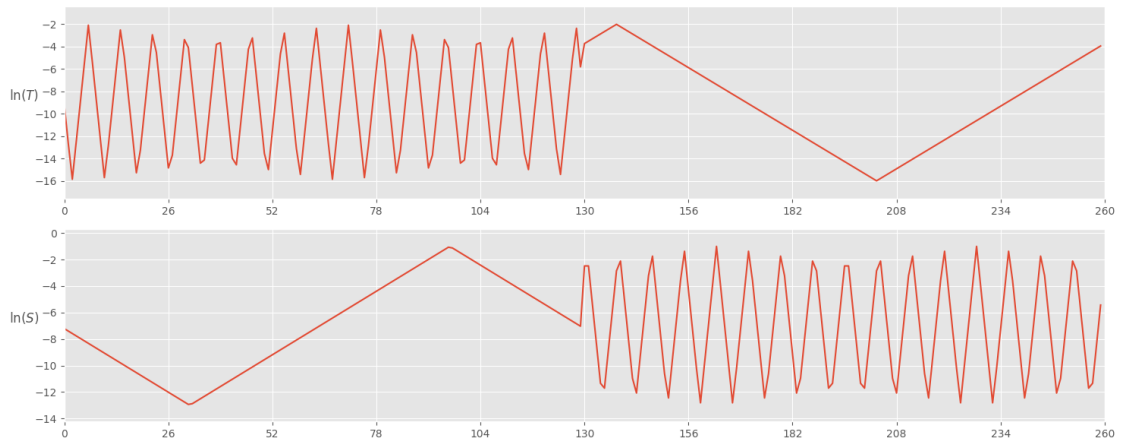
```

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

```
estimation.sensitivity()
```

FAST total sensitivity shares





Out:

```

Initializing the Fourier Amplitude Sensitivity Test (FAST) with 260 repetitions
Starting the FAST algorithm 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/v1.0.1/examples/Estimate_theis/2020-09-02_14-47-53_sensitivity_db.csv'
↳created.

*** Final SPOTPY summary ***
Total Duration: 0.19 seconds
Total Repetitions: 260
Minimal objective value: 957.337
Corresponding parameter setting:
mu: -9.33675
lnS: -7.28114
Maximal objective value: 1.95621e+06
Corresponding parameter setting:
mu: -15.3559
lnS: -11.7082
*****

260
Parameter First Total
mu 0.670819 0.941559
lnS 0.063200 0.349012
260

```

Total running time of the script: (0 minutes 8.576 seconds)

Estimate steady homogeneous parameters

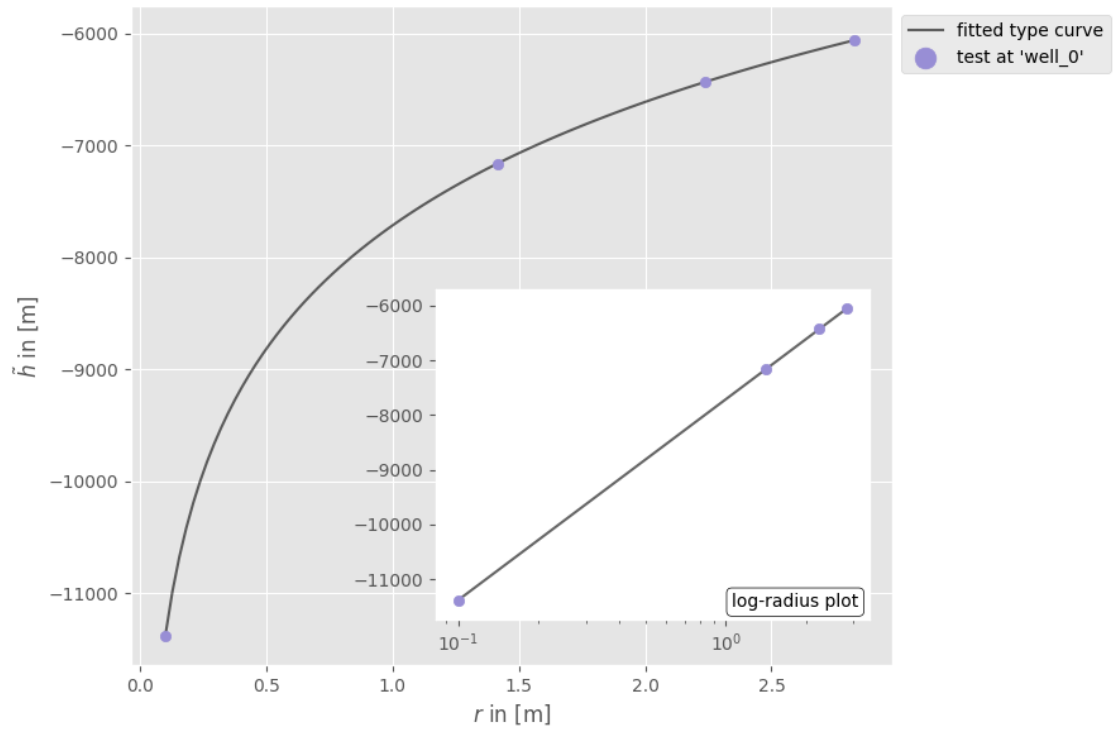
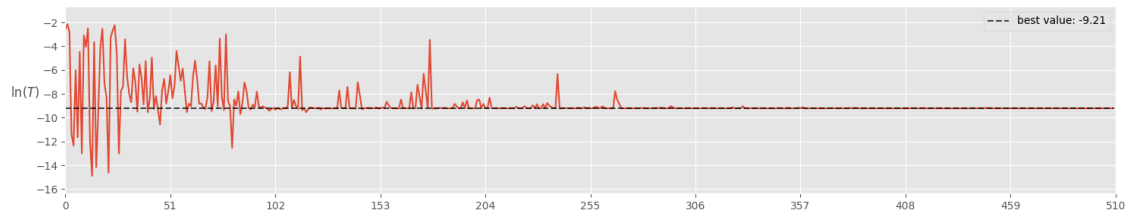
Here we estimate transmissivity from the quasi steady state of a pumping test campaign with the classical thiem solution.

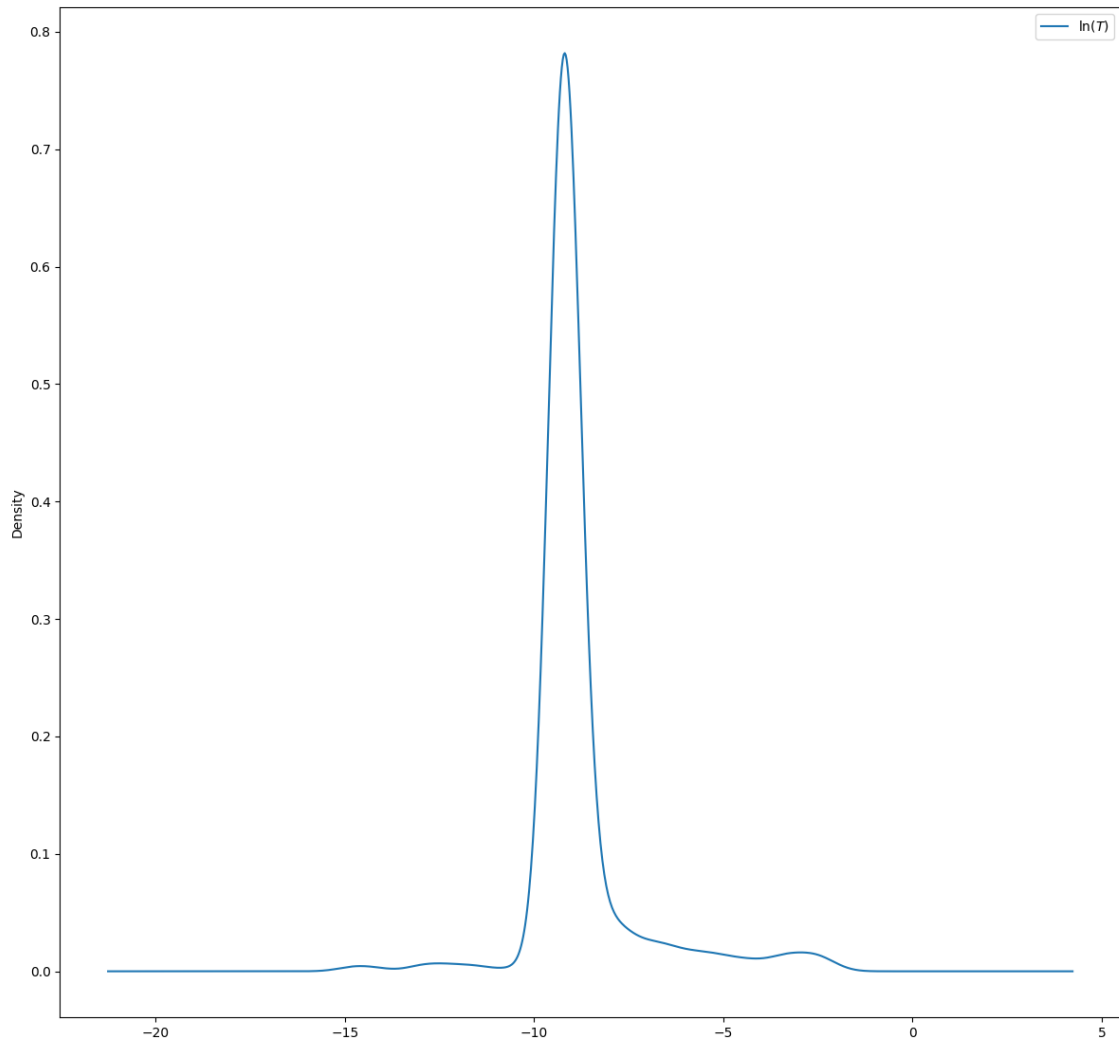
```

import welltestpy as wtp

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

```





•
Out:

```

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/v1.0.1/examples/Estimate_thiem/2020-09-02_14-47-54_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...
THE POPULATION HAS CONVERGED TO A PRESPECIFIED SMALL PARAMETER SPACE
SEARCH WAS STOPPED AT TRIAL NUMBER: 1366
NUMBER OF DISCARDED TRIALS: 0
NORMALIZED GEOMETRIC RANGE = 0.000991
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 0.62 seconds
Total Repetitions: 1366
Minimal objective value: 0.0672645
Corresponding parameter setting:
mu: -9.21029
*****

Best parameter set:
mu=-9.210293556543638

```

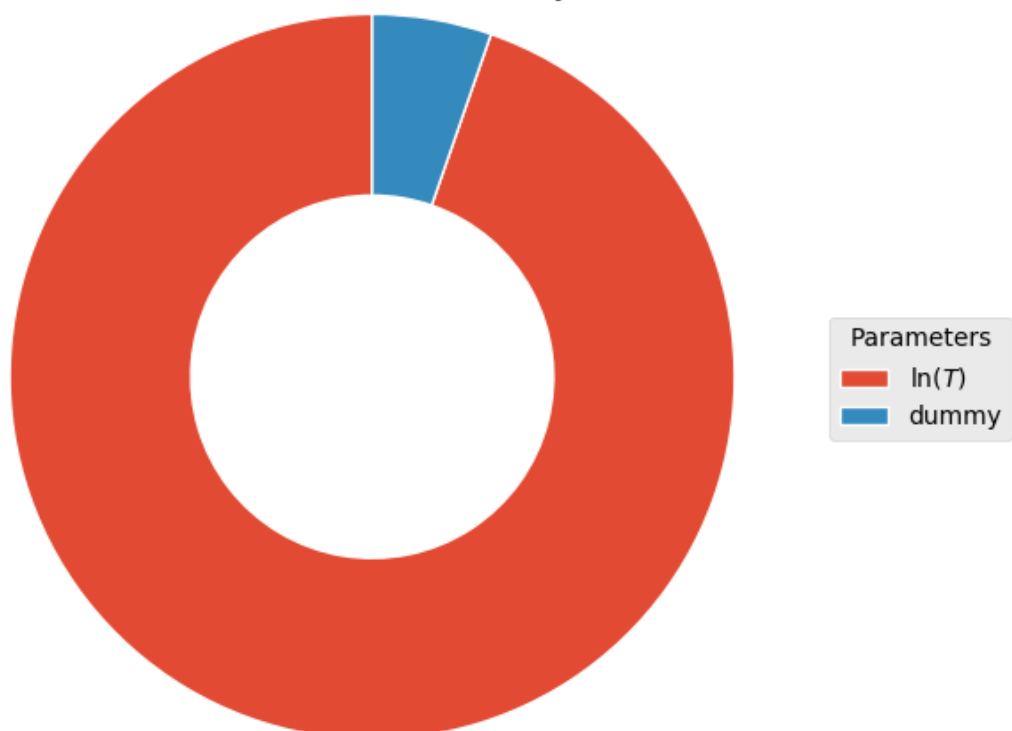
since we only have one parameter, we need a dummy parameter to estimate sensitivity

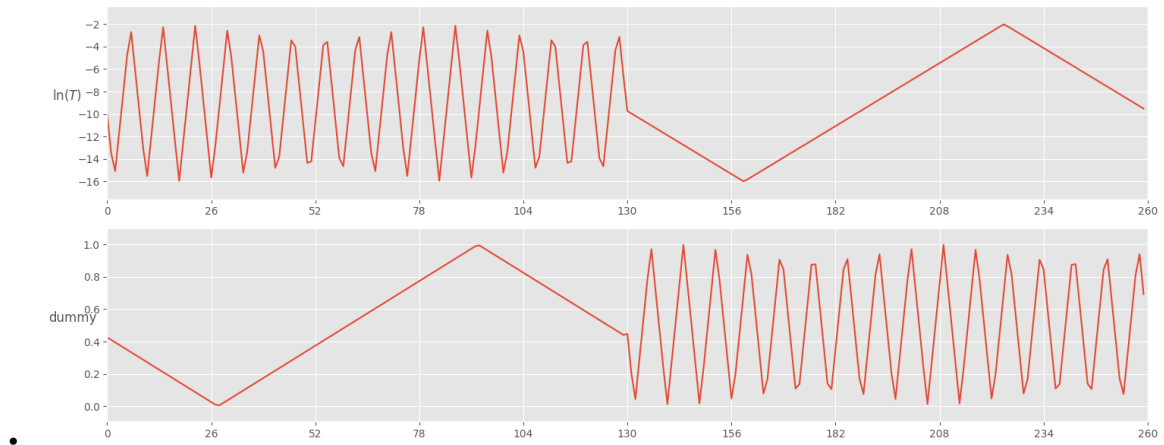
```

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

```

FAST total sensitivity shares





Out:

```

Initializing the Fourier Amplitude Sensitivity Test (FAST) with 260 repetitions
Starting the FAST algorithm 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/v1.0.1/examples/Estimate_thiem/2020-09-02_14-47-56_sensitivity_db.csv'
↳created.

*** Final SPOTPY summary ***
Total Duration: 0.16 seconds
Total Repetitions: 260
Minimal objective value: 74.8884
Corresponding parameter setting:
mu: -9.1824
dummy: 0.179636
Maximal objective value: 2.35929e+06
Corresponding parameter setting:
mu: -15.976
dummy: 0.690593
*****

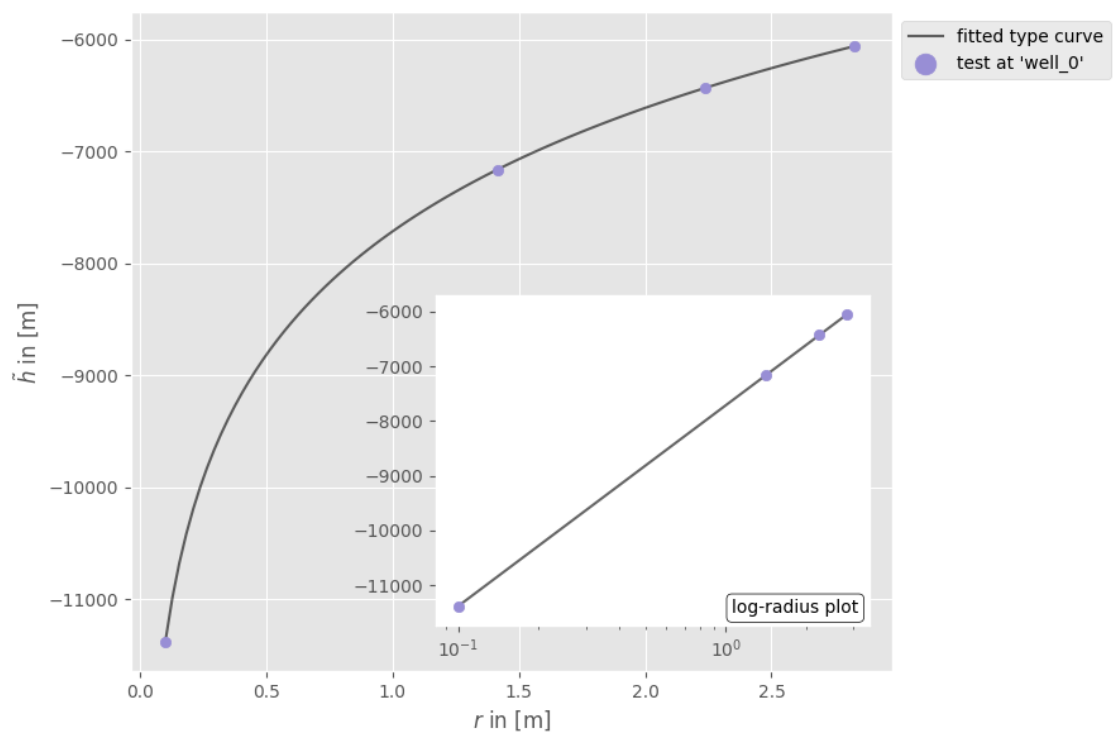
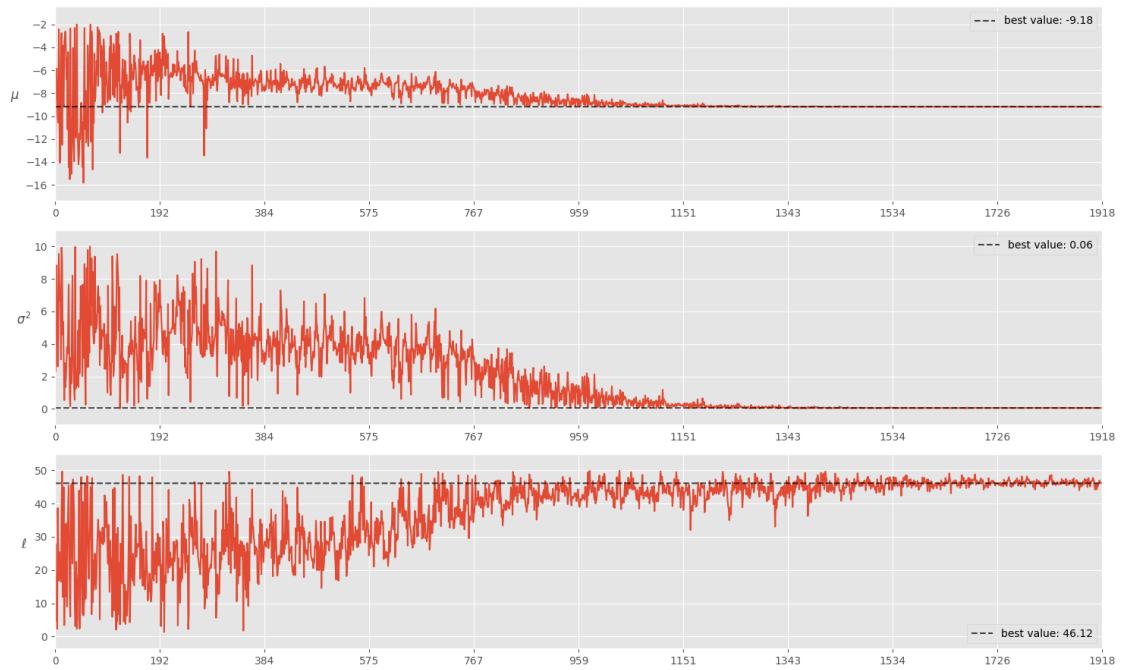
260
Parameter First Total
mu 0.788866 0.978326
dummy 0.002011 0.054925
260

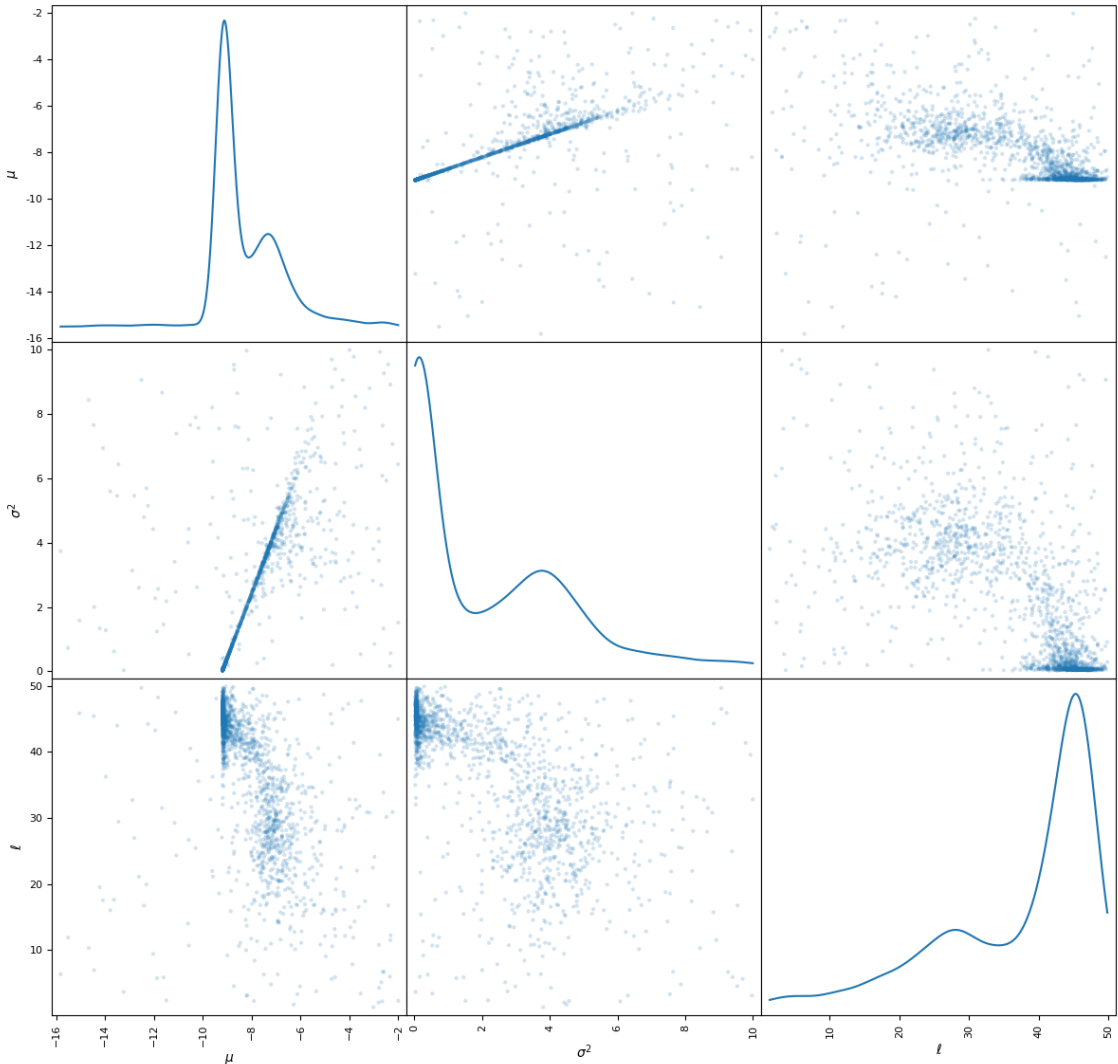
```

Total running time of the script: (0 minutes 3.227 seconds)

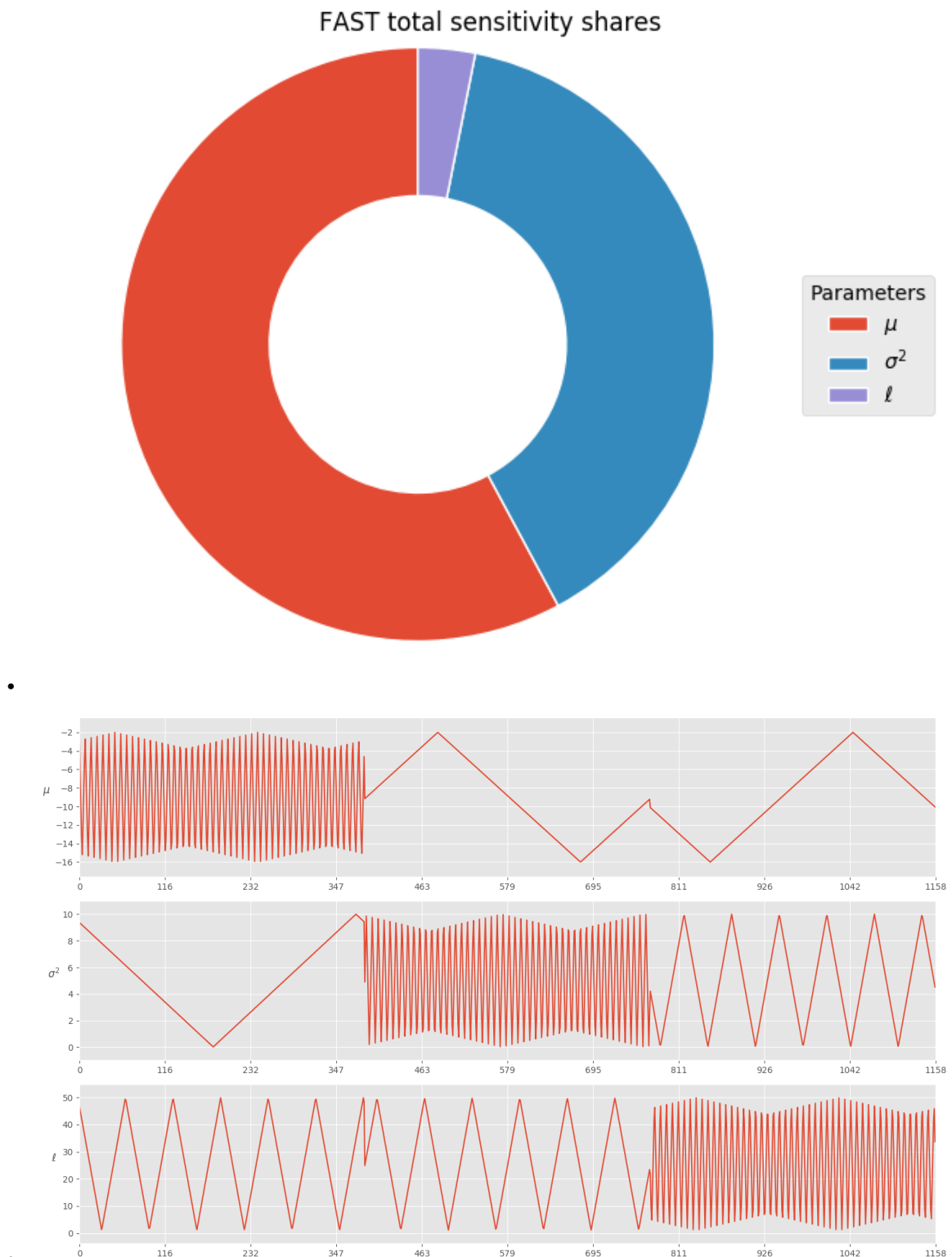
Estimate steady heterogeneous parameters

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.





•



Out:

```

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/v1.0.1/examples/Est_steady_het/2020-09-02_14-47-57_db.csv (continues on next page)

```

(continued from previous page)

[illegible]

(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
*** OPTIMIZATION SEARCH TERMINATED BECAUSE THE LIMIT
ON THE MAXIMUM NUMBER OF TRIALS
5000
HAS BEEN EXCEEDED.
SEARCH WAS STOPPED AT TRIAL NUMBER: 5115
NUMBER OF DISCARDED TRIALS: 42
NORMALIZED GEOMETRIC RANGE = 0.002994
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 1.89 seconds
Total Repetitions: 5115
Minimal objective value: 8.00673e-05
Corresponding parameter setting:
mu: -9.1813
var: 0.058086
len_scale: 46.1165
*****

Best parameter set:
mu=-9.181297499737951, var=0.05808595138355535, len_scale=46.11645961299604
Initializing the Fourier Amplitude Sensitivity Test (FAST) with 1158
↳repetitions
Starting the FAST algorithm 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/v1.0.1/examples/Est_steady_het/2020-09-02_14-48-02_sensitivity_db.csv'
↳created.

*** Final SPOTPY summary ***
Total Duration: 0.54 seconds
Total Repetitions: 1158
Minimal objective value: 46.1202
Corresponding parameter setting:
mu: -7.65959
var: 3.0731
len_scale: 42.8213
Maximal objective value: 1.95459e+08
Corresponding parameter setting:
mu: -15.9734
var: 8.89563
len_scale: 22.2295
*****

1158
Parameter First Total
mu 0.377589 0.887750
var 0.081528 0.598985
len_scale 0.003824 0.047765

```

(continues on next page)

(continued from previous page)

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 7.561 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 triangulate, sym, plot_well_pos

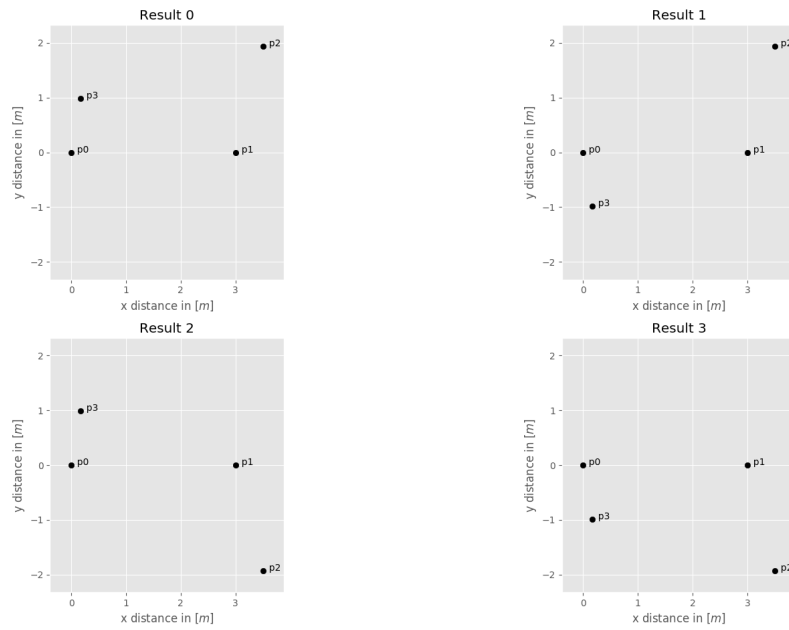
dist_mat = np.zeros((4, 4), dtype=float)
dist_mat[0, 1] = 3 # distance between well 0 and 1
dist_mat[0, 2] = 4 # distance between well 0 and 2
dist_mat[1, 2] = 2 # distance between well 1 and 2
dist_mat[0, 3] = 1 # distance between well 0 and 3
dist_mat[1, 3] = 3 # distance between well 1 and 3
dist_mat[2, 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)
```

Out:

```
Starting constellation 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 0.658 seconds)

CHAPTER 3

WELLTESTPY API

3.1 Purpose

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

Subpackages

<i>data</i>	welltestpy subpackage providing datastructures.
<i>estimate</i>	welltestpy subpackage providing routines to estimate pump test parameters.
<i>process</i>	welltestpy subpackage providing routines to pre process test data.
<i>tools</i>	welltestpy subpackage providing miscellaneous tools.

Classes

Campaign classes

The following classes can be used to handle field campaigns.

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

Field Test classes

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

<i>PumpingTest</i> (name, pumpingwell, pumpingrate)	Class for a pumping test.
---	---------------------------

Loading routines

Campaign related loading routines

`load_campaign(cmpfile)`

Load a campaign from file.

3.2 welltestpy.data

welltestpy subpackage providing datastructures.

Subpackages

<i>data_io</i>	welltestpy subpackage providing input-output routines.
<i>varlib</i>	welltestpy subpackage providing flow datastructures for variables.
<i>testslib</i>	welltestpy subpackage providing flow datastructures for tests on a fieldsite.
<i>campaignlib</i>	Welltestpy subpackage providing flow datastructures for field-campaigns.

Classes

Campaign classes

The following classes can be used to handle field campaigns.

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

Field Test classes

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

<i>PumpingTest</i> (name, pumpingwell, pumpingrate)	Class for a pumping test.
---	---------------------------

Variable classes

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

Routines

Loading routines

Campaign related loading routines

<code>load_campaign(cmpfile)</code>	Load a campaign from file.
<code>load_fieldsite(fdsfile)</code>	Load a field site from file.

Field test related loading routines

<code>load_test(tstfile)</code>	Load a test from file.
---------------------------------	------------------------

Variable related loading routines

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

welltestpy.data.data_io

welltestpy subpackage providing input-output routines.

The following functions are provided

load_campaign (*cmpfile*)

Load a campaign from file.

This reads a campaign from a csv file.

Parameters **cmpfile** (*str*) – Path to the file

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

load_obs (*obsfile*)

Load an observation from file.

This reads a observation from a csv file.

Parameters **obsfile** (*str*) – Path to the file

load_test (*tstfile*)

Load a test from file.

This reads a test from a csv file.

Parameters **tstfile** (*str*) – Path to the file

load_var (*varfile*)

Load a variable from file.

This reads a variable from a csv file.

Parameters **varfile** (*str*) – Path to the file

load_well (*welfile*)

Load a well from file.

This reads a well from a csv file.

Parameters **welfile** (*str*) – Path to the file

save_campaign (*campaign*, *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".

save_fieldsite (*fieldsite*, *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".

save_obs (*obs*, *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".

save_pumping_test (*pump_test*, *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".

save_var (*var*, *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".

save_well (*well*, *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".

welltestpy.data.varlib

welltestpy subpackage providing flow datastructures for variables.

The following classes and functions are provided

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

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 description string.

Parameters

- **name** (`str`) – Name of the Variable.
- **value** (`int` or `float` or `numpy.ndarray`) – Value of the Variable.
- **symbol** (`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

<code>__call__([value])</code>	Call a variable.
<code>save([path, name])</code>	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".

info

Info about the Variable.

Type `str`

label

symbol in units.

Type `str`

Type String containing

scalar

State if the variable is of scalar type.

Type `bool`

value

Value.

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

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

Bases: `welltestpy.data.varlib.Variable`

Variable class special for time series.

Parameters

- **value** (`int` or `float` or `numpy.ndarray`) – Value of the Variable.
- **symbol** (`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

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

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

Bases: `welltestpy.data.varlib.Variable`

Variable class special for groundwater head.

Parameters

- **value** (`int` or `float` or `numpy.ndarray`) – Value of the Variable.
- **symbol** (`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

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

class TemporalVar (*value=0.0*)

Bases: `welltestpy.data.varlib.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

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

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

Bases: `welltestpy.data.varlib.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`) – Longitudinal values of the coordinates.
- **symbol** (`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

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

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 informations 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

<code>__call__</code> ([<i>observation</i> , <i>time</i>])	Call a variable.
<code>reshape</code> ()	Reshape obeservations to flat array.
<code>save</code> ([<i>path</i> , <i>name</i>])	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 obeservations 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".

info

Get informations about the observation.

Here you can display informations about the observation.

kind

name of the observation variable.

Type `str`

label

symbol in units.

Type `[tuple of] str`

labels

symbol in units.

Type `[tuple of] str`

observation

Observed values of the observation.

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

state

String containing state of the observation.

Either "steady" or "transient".

Type `str`

time

Time values of the observation.

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

units

units of the observation.

Type `[tuple of] str`

value

Value of the Observation.

`[tuple of] int` or `float` or `numpy.ndarray`

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

Bases: `welltestpy.data.varlib.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 informations 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

<code>__call__([observation, time])</code>	Call a variable.
<code>reshape()</code>	Reshape obeservations to flat array.
<code>save([path, name])</code>	Save an observation to file.

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

Bases: `welltestpy.data.varlib.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 informations 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

<code>__call__([observation, time])</code>	Call a variable.
<code>reshape()</code>	Reshape obeservations to flat array.
<code>save([path, name])</code>	Save an observation to file.

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

Bases: `welltestpy.data.varlib.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 informations 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

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

class TimeSeries (*name, values, time, description='Timeseries.'*)

Bases: *welltestpy.data.varlib.Observation*

Time series observation.

Parameters

- **name** (*str*) – Name of the Variable.
- **values** (*Variable*) – Values of the time-series.
- **time** (*Variable*) – Time points of the time-series.
- **description** (*str*, optional) – Description of the Variable. Default: "Timeseries."

Attributes

info Get informations 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

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

class Well (*name, radius, coordinates, welldepth=1.0, aquiferdepth=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. Default: 1.0
- **aquiferdepth** (`Variable` or `float`, optional) – Depth of the aquifer at the well. Default: "None"

Notes

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

Attributes

aquiferdepth `float`: Aquifer depth at the well.

coordinates `numpy.ndarray`: Coordinates variable of the well.

depth `float`: Depth of the well.

info Get informations about the variable.

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

radius `float`: Radius of the well.

welldepth `float`: Depth variable of the well.

wellradius `float`: Radius variable of the well.

Methods

<code>distance(well)</code>	Calculate distance to the well.
<code>save([path, name])</code>	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".

aquiferdepth

Aquifer depth at the well.

Type `float`

coordinates

Coordinates variable of the well.

Type `numpy.ndarray`

depth

Depth of the well.

Type `float`

info

Get informations about the variable.

Here you can display informations about the variable.

pos

Position of the well.

Type `numpy.ndarray`

radius

Radius of the well.

Type `float`

welldepth

Depth variable of the well.

Type `float`

wellradius

Radius variable of the well.

Type `float`

welltestpy.data.testslib

welltestpy subpackage providing flow datastructures for tests on a fieldsite.

The following classes and functions are provided

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

<code>plot(wells[, exclude, fig, ax])</code>	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.

testtype

String containing the test type.

Type `str`

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

Bases: `welltestpy.data.testslib.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. If a *float* is given, it is assumed to be given in m. Default: 1.0
- **aquiferradius** (*float* or *Variable*, optional) – Aquifer radius of 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 *float*: aquifer depth at the field site.
aquiferradius *float*: aquifer radius at the field site.
constant_rate *bool*: state if this is a constant rate test.
depth *float*: aquifer depth at the field site.
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

<i>add_observations</i> (<i>obs</i>)	Add some specified observations.
<i>add_steady_obs</i> (<i>well</i> , <i>observation</i> [, <i>description</i>])	Add steady drawdown observations.
<i>add_transient_obs</i> (<i>well</i> , <i>time</i> , <i>observation</i> [, ...])	Add transient drawdown observations.
<i>del_observations</i> (<i>obs</i>)	Delete some specified observations.
<i>make_steady</i> ([<i>time</i>])	Convert the pumping test to a steady state test.
<i>plot</i> (<i>wells</i> [, <i>exclude</i> , <i>fig</i> , <i>ax</i>])	Generate a plot of the pumping test.
<i>save</i> ([<i>path</i> , <i>name</i>])	Save a pumping test to file.
<i>state</i> ([<i>wells</i>])	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"

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.

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

aquiferdepth

aquifer depth at the field site.

Type *float*

aquiferradius

aquifer radius at the field site.

Type *float*

constant_rate

state if this is a constant rate test.

Type *bool*

depth

aquifer depth at the field site.

Type *float*

observations

observations made at the field site.

Type *dict*

observationwells

all well names.

Type *tuple of str*

pumpingrate

pumping rate variable at the pumping well.

Type *float*

radius

aquifer radius at the field site.

Type *float*

rate

pumping rate at the pumping well.

Type *float*

wells

all well names.

Type *tuple of str*

welltestpy.data.campaignlib

Welltestpy subpackage providing flow datastructures for field-campaigns.

The following classes and functions are provided

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

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

<code>save([path, name])</code>	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".

coordinates

Coordinates of the field site.

Type `numpy.ndarray`

info

Info about the field site.

Type `str`

pos

Position of the field site.

Type `numpy.ndarray`

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
- **tests** – 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**

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

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".

fieldsite

Field site where the campaign was realised.

Type *FieldSite*

tests

Tests within the campaign.

Type *dict*

wells

Wells within the campaign.

Type `dict`

3.3 welltestpy.estimate

welltestpy subpackage providing routines to estimate pump test parameters.

Estimators

The following estimators are provided

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

Base Classes

Transient

All transient estimators are derived from the following class

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

Steady Pumping

All steady estimators are derived from the following class

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

```
class ExtTheis3D (name, campaign, val_ranges=None, val_fix=None, testinclude=None, generate=False)
```

Bases: `welltestpy.estimate.transient_lib.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 transmissivity field with a gaussian correlation function and an anisotropy ratio $0 < e \leq 1$.

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`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signiture or replaced in `val_kw_names`. 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 signiture or replaced in `val_kw_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

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

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

Bases: `welltestpy.estimate.transient_lib.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.

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`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signiture or replaced in `val_kw_names`. 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 signiture or replaced in `val_kw_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

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

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

Bases: `welltestpy.estimate.transient_lib.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.

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`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signiture or replaced in `val_kw_names`. 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 signiture or replaced in `val_kw_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

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

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

Bases: `welltestpy.estimate.transient_lib.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.

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`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signiture or replaced in `val_kw_names`. 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 signiture or replaced in `val_kw_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

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

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

Bases: `welltestpy.estimate.steady_lib.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 transmissivity field with a gaussian correlation function and an anisotropy ratio $0 < e \leq 1$.

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the paramters
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: `PumpingTest.make_steady`. Default: `True`
- **val_ranges** (`dict`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signiture or replaced in `val_kw_names`. 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 or replaced in `val_kw_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

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

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

Bases: `welltestpy.estimate.steady_lib.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.

Parameters

- **name** (`str`) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the paramters
- **make_steady** (`bool`, optional) – State if the tests should be converted to steady observations. See: `PumpingTest.make_steady`. Default: `True`
- **val_ranges** (`dict`) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature or replaced in `val_kw_names`. 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 or replaced in `val_kw_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

<code>gen_data()</code>	Generate the observed drawdown.
<code>gen_setup([prate_kw, rad_kw, r_ref_kw, ...])</code>	Generate the Spotpy Setup.
<code>run([rep, parallel, run, folder, dbname, ...])</code>	Run the estimation.
<code>sensitivity([rep, parallel, folder, dbname, ...])</code>	Run the sensitivity analysis.

Continued on next page

Table 39 – continued from previous page

setpumprate([prate])	Set a uniform pumping rate at all pumpingwells wells.
----------------------	---

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

Bases: `welltestpy.estimate.steady_lib.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.

Parameters

- **name** (*str*) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the paramters
- **make_steady** (*bool*, optional) – State if the tests should be converted to steady observations. See: `PumpingTest.make_steady`. Default: `True`
- **val_ranges** (*dict*) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signiture or replaced in `val_kw_names`. 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 signiture or replaced in `val_kw_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

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

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

Bases: `welltestpy.estimate.steady_lib.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.

Parameters

- **name** (*str*) – Name of the Estimation.
- **campaign** (`welltestpy.data.Campaign`) – The pumping test campaign which should be used to estimate the paramters

- **make_steady** (*bool*, optional) – State if the tests should be converted to steady observations. See: `PumpingTest.make_steady`. Default: `True`
- **val_ranges** (*dict*) – Dictionary containing the fit-ranges for each value in the type-curve. Names should be as in the type-curve signature or replaced in `val_kw_names`. 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 or replaced in `val_kw_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

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

class TransientPumping (*name, campaign, type_curve, val_ranges, val_fix=None, fit_type=None, val_kw_names=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 paramters
- **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 or replaced in `val_kw_names`. 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 or replaced in `val_kw_names`. Default: `None`
- **fit_type** (*dict* or *None*) – Dictionary containing fitting type for each value in the type-curve. Names should be as in the type-curve signature or replaced in `val_kw_names`. `fit_type` can be “lin”, “log” (`np.exp(val)` will be used) or a callable function. By default, values will be fit linearly. Default: `None`
- **val_kw_names** (*dict* or *None*) – Dictionary containing keyword names in the type-curve for each value.

{value-name: kwargs-name in type_curve}

This is usefull if fitting is not done by linear values. By default, parameter names will be value names. 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 usefull 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

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

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

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 packageDefault: "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

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 specify any exponent ("quad" would be equivalent to 2)Default: "quad"
- **steps** (`int`, optional) – Number of generated time steps. Default: 10

campaign = None

Copy of the input campaign to be modified

Type `welltestpy.data.Campaign`

campaign_raw = None

Copy of the original input campaign

Type `welltestpy.data.Campaign`

data = None

observation data

Type `numpy.ndarray`

estimated_para = None

estimated parameters by name

Type `dict`

name = None

Name of the Estimation

Type `str`

prate = None

Pumpingrate at the pumping well

Type `float`

rad = None

array of the radii from the wells

Type `numpy.ndarray`

radnames = None

names of the radii well combination

Type `numpy.ndarray`

result = None

result of the spotpy estimation

Type `list`

rinf = None
radius of the furthest wells
Type `float`

rwell = None
radius of the pumping wells
Type `float`

sens = None
result of the spotpy sensitivity analysis
Type `dict`

setup_kw = None
TypeCurve Spotpy Setup definition
Type `dict`

testinclude = None
dictionary of which tests should be included
Type `dict`

time = None
time points of the observation
Type `numpy.ndarray`

class SteadyPumping (*name, campaign, type_curve, val_ranges, make_steady=True, val_fix=None, fit_type=None, val_kw_names=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 paramters
- **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 or replaced in `val_kw_names`. 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 or replaced in `val_kw_names`. Default: None
- **fit_type** (`dict` or `None`) – Dictionary containing fitting type for each value in the type-curve. Names should be as in the type-curve signature or replaced in `val_kw_names`. `fit_type` can be “lin”, “log” (`np.exp(val)` will be used) or a callable function. By default, values will be fit linearly. Default: None
- **val_kw_names** (`dict` or `None`) – Dictionary containing keyword names in the type-curve for each value.

{value-name: kwargs-name in type_curve}

This is usefull if fitting is not done by linear values. By default, parameter names will be value names. 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 usefull 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

<code>gen_data()</code>	Generate the observed drawdown.
<code>gen_setup([prate_kw, rad_kw, r_ref_kw, ...])</code>	Generate the Spotpy Setup.
<code>run([rep, parallel, run, folder, dbname, ...])</code>	Run the estimation.
<code>sensitivity([rep, parallel, folder, dbname, ...])</code>	Run the sensitivity analysis.
<code>setpumprate([prate])</code>	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*)
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`

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

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`

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 = `None`

Copy of the input campaign to be modified

Type `welltestpy.data.Campaign`

campaign_raw = None
Copy of the original input campaign
Type `welltestpy.data.Campaign`

data = None
observation data
Type `numpy.ndarray`

estimated_para = None
estimated parameters by name
Type `dict`

h_ref = None
reference head at the biggest distance
Type `float`

name = None
Name of the Estimation
Type `str`

prate = None
Pumpingrate at the pumping well
Type `float`

r_ref = None
reference radius of the biggest distance
Type `float`

rad = None
array of the radii from the wells
Type `numpy.ndarray`

radnames = None
names of the radii well combination
Type `numpy.ndarray`

result = None
result of the spotpy estimation
Type `list`

rinf = None
radius of the furthest wells
Type `float`

rwell = None
radius of the pumping wells
Type `float`

sens = None
result of the spotpy sensitivity analysis
Type `dict`

setup_kw = None
TypeCurve Spotpy Setup definition
Type `dict`

testinclude = None
dictionary of which tests should be included

Type `dict`

3.4 welltestpy.process

welltestpy subpackage providing routines to pre process test data.

Included functions

The following classes and functions are provided

<code>normpumptest(pumptest[, pumpingrate, factor])</code>	Normalize the pumping rate of a pumping test.
<code>combinepumptest(campaign, test1, test2[, ...])</code>	Combine two pumping tests to one.
<code>filterdrawdown(observation[, tout, dxscale])</code>	Smooth the drawdown data of an observation well.

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

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*

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

3.5 welltestpy.tools

welltestpy subpackage providing miscellaneous tools.

Included functions

The following functions are provided for point triangulation

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

The following plotting routines are provided

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

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 has 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*

sym (*A*)

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

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

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*

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

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

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

Plot of transient estimation fitting.

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

Plot of steady estimation fitting.

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

Plot of parameter interaction.

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

Plot of parameter trace.

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

Plot of sensitivity results.

W

- `welltestpy`, [21](#)
- `welltestpy.data`, [23](#)
- `welltestpy.data.campaignlib`, [42](#)
- `welltestpy.data.data_io`, [25](#)
- `welltestpy.data.testslib`, [38](#)
- `welltestpy.data.varlib`, [28](#)
- `welltestpy.estimate`, [46](#)
- `welltestpy.process`, [61](#)
- `welltestpy.tools`, [62](#)

Symbols

`__call__()` (*Observation method*), 32

`__call__()` (*Variable method*), 28

A

`add_observations()` (*PumpingTest method*), 39

`add_steady_obs()` (*PumpingTest method*), 39

`add_transient_obs()` (*PumpingTest method*), 40

`add_well()` (*Campaign method*), 43

`addtests()` (*Campaign method*), 43

`addwells()` (*Campaign method*), 44

`aquiferdepth` (*PumpingTest attribute*), 41

`aquiferdepth` (*Well attribute*), 37

`aquiferradius` (*PumpingTest attribute*), 41

C

`Campaign` (*class in welltestpy.data.campaignlib*), 43

`campaign` (*SteadyPumping attribute*), 58

`campaign` (*TransientPumping attribute*), 55

`campaign_plot()` (*in module welltestpy.tools*), 62

`campaign_raw` (*SteadyPumping attribute*), 58

`campaign_raw` (*TransientPumping attribute*), 55

`campaign_well_plot()` (*in module welltestpy.tools*), 63

`combinepumptest()` (*in module welltestpy.process*), 61

`constant_rate` (*PumpingTest attribute*), 41

`coordinates` (*FieldSite attribute*), 42

`coordinates` (*Well attribute*), 37

`CoordinatesVar` (*class in welltestpy.data.varlib*), 31

D

`data` (*SteadyPumping attribute*), 59

`data` (*TransientPumping attribute*), 55

`del_observations()` (*PumpingTest method*), 40

`deltests()` (*Campaign method*), 44

`delwells()` (*Campaign method*), 44

`depth` (*PumpingTest attribute*), 41

`depth` (*Well attribute*), 37

`distance()` (*Well method*), 36

`DrawdownObs` (*class in welltestpy.data.varlib*), 34

E

`estimated_para` (*SteadyPumping attribute*), 59

`estimated_para` (*TransientPumping attribute*), 55

`ExtTheis2D` (*class in welltestpy.estimate*), 47

`ExtTheis3D` (*class in welltestpy.estimate*), 46

`ExtThiem2D` (*class in welltestpy.estimate*), 50

`ExtThiem3D` (*class in welltestpy.estimate*), 49

F

`fadeline()` (*in module welltestpy.tools*), 63

`fieldsite` (*Campaign attribute*), 44

`FieldSite` (*class in welltestpy.data.campaignlib*), 42

`filterdrawdown()` (*in module welltestpy.process*), 61

G

`gen_data()` (*SteadyPumping method*), 57

`gen_data()` (*TransientPumping method*), 53

`gen_setup()` (*SteadyPumping method*), 57

`gen_setup()` (*TransientPumping method*), 53

H

`h_ref` (*SteadyPumping attribute*), 59

`HeadVar` (*class in welltestpy.data.varlib*), 30

I

`info` (*FieldSite attribute*), 42

`info` (*Observation attribute*), 32

`info` (*Variable attribute*), 29

`info` (*Well attribute*), 37

K

`kind` (*Observation attribute*), 33

L

`label` (*Observation attribute*), 33

`label` (*Variable attribute*), 29

`labels` (*Observation attribute*), 33

`load_campaign()` (*in module welltestpy.data.data_io*), 25

`load_fieldsite()` (*in module welltestpy.data.data_io*), 25

`load_obs()` (*in module welltestpy.data.data_io*), 25

load_test() (in module welltestpy.data.data_io), 25
load_var() (in module welltestpy.data.data_io), 25
load_well() (in module welltestpy.data.data_io), 25

M

make_steady() (PumpingTest method), 40

N

name (SteadyPumping attribute), 59
name (TransientPumping attribute), 55
Neuman2004 (class in welltestpy.estimate), 48
Neuman2004Steady (class in welltestpy.estimate), 51
normpumptest() (in module welltestpy.process), 61

O

Observation (class in welltestpy.data.varlib), 31
observation (Observation attribute), 33
observations (PumpingTest attribute), 41
observationwells (PumpingTest attribute), 41

P

plot() (Campaign method), 44
plot() (PumpingTest method), 40
plot() (Test method), 38
plot_well_pos() (in module welltestpy.tools), 63
plot_wells() (Campaign method), 44
plotfit_steady() (in module welltestpy.tools), 64
plotfit_transient() (in module welltestpy.tools), 64
plotparainteract() (in module welltestpy.tools), 64
plotparatrace() (in module welltestpy.tools), 64
plotsensitivity() (in module welltestpy.tools), 64
pos (FieldSite attribute), 42
pos (Well attribute), 37
prate (SteadyPumping attribute), 59
prate (TransientPumping attribute), 55
pumpingrate (PumpingTest attribute), 41
PumpingTest (class in welltestpy.data.testslib), 38

R

r_ref (SteadyPumping attribute), 59
rad (SteadyPumping attribute), 59
rad (TransientPumping attribute), 55
radius (PumpingTest attribute), 41
radius (Well attribute), 37
radnames (SteadyPumping attribute), 59
radnames (TransientPumping attribute), 55
rate (PumpingTest attribute), 41
reshape() (Observation method), 32
result (SteadyPumping attribute), 59
result (TransientPumping attribute), 55
rinf (SteadyPumping attribute), 55

rinf (TransientPumping attribute), 55
run() (SteadyPumping method), 57
run() (TransientPumping method), 53
rwell (SteadyPumping attribute), 59
rwell (TransientPumping attribute), 56

S

save() (Campaign method), 44
save() (FieldSite method), 42
save() (Observation method), 32
save() (PumpingTest method), 40
save() (Variable method), 29
save() (Well method), 36
save_campaign() (in module welltestpy.data.data_io), 25
save_fieldsite() (in module welltestpy.data.data_io), 25
save_obs() (in module welltestpy.data.data_io), 26
save_pumping_test() (in module welltestpy.data.data_io), 26
save_var() (in module welltestpy.data.data_io), 26
save_well() (in module welltestpy.data.data_io), 26
scalar (Variable attribute), 29
sens (SteadyPumping attribute), 59
sens (TransientPumping attribute), 56
sensitivity() (SteadyPumping method), 58
sensitivity() (TransientPumping method), 54
setpumprate() (SteadyPumping method), 58
setpumprate() (TransientPumping method), 54
settime() (TransientPumping method), 54
setup_kw (SteadyPumping attribute), 59
setup_kw (TransientPumping attribute), 56
state (Observation attribute), 33
state() (PumpingTest method), 41
StdyHeadObs (class in welltestpy.data.varlib), 34
StdyObs (class in welltestpy.data.varlib), 33
SteadyPumping (class in welltestpy.estimate), 56
sym() (in module welltestpy.tools), 62

T

TemporalVar (class in welltestpy.data.varlib), 30
Test (class in welltestpy.data.testslib), 38
testinclude (SteadyPumping attribute), 59
testinclude (TransientPumping attribute), 56
tests (Campaign attribute), 44
testtype (Test attribute), 38
Theis (class in welltestpy.estimate), 48
Thiem (class in welltestpy.estimate), 51
time (Observation attribute), 33
time (TransientPumping attribute), 56
TimeSeries (class in welltestpy.data.varlib), 35
TimeVar (class in welltestpy.data.varlib), 29
TransientPumping (class in welltestpy.estimate), 52
triangulate() (in module welltestpy.tools), 62

U

units (*Observation attribute*), 33

V

value (*Observation attribute*), 33

value (*Variable attribute*), 29

Variable (*class in welltestpy.data.varlib*), 28

W

Well (*class in welltestpy.data.varlib*), 36

welldepth (*Well attribute*), 37

wellradius (*Well attribute*), 37

wells (*Campaign attribute*), 44

wells (*PumpingTest attribute*), 41

welltestpy (*module*), 21

welltestpy.data (*module*), 23

welltestpy.data.campaignlib (*module*), 42

welltestpy.data.data_io (*module*), 25

welltestpy.data.testslib (*module*), 38

welltestpy.data.varlib (*module*), 28

welltestpy.estimate (*module*), 46

welltestpy.process (*module*), 61

welltestpy.tools (*module*), 62