welltestpy Documentation

Release 1.0.0

Sebastian Mueller

Sep 02, 2020

CONTENTS

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 . . . 14
  - Point triangulation . . . 20

3 welltestpy API **23**
- 3.1 Purpose . . . 23
  - Subpackages . . . 23
  - Classes . . . 23
  - Loading routines . . . 23
- 3.2 welltestpy.data . . . 25
  - Subpackages . . . 25
  - Classes . . . 25
  - Routines . . . 25
  - welltestpy.data.data_io . . . 27
  - welltestpy.data.varlib . . . 30
  - welltestpy.data.testslib . . . 40
  - welltestpy.data.campaignlib . . . 44
- 3.3 welltestpy.estimate . . . 48
  - Estimators . . . 48
  - Base Classes . . . 48
- 3.4 welltestpy.process . . . 63
  - Included functions . . . 63
- 3.5 welltestpy.tools . . . 64
  - Included functions . . . 64

Python Module Index **67**

Index **69**

CHAPTER 1

WELLTESTPY QUICKSTART

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 triagulation
```

1.3 Requirements

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

- AnaFlow >= 1.0.0
- SpotPy >= 1.5.0
- Matplotlib >= 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

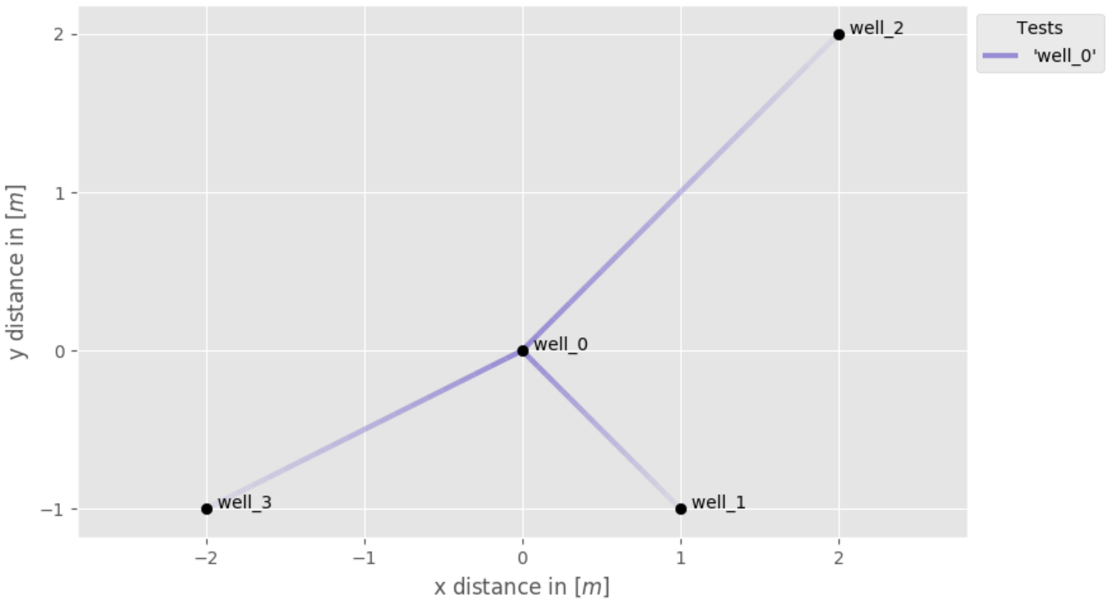
```
pumptest.add_transient_obs("well_0", time, drawdown[:, 0])
pumptest.add_transient_obs("well_1", time, drawdown[:, 1])
pumptest.add_transient_obs("well_2", time, drawdown[:, 2])
pumptest.add_transient_obs("well_3", time, drawdown[:, 3])
```

Add the pumping test to the campaign

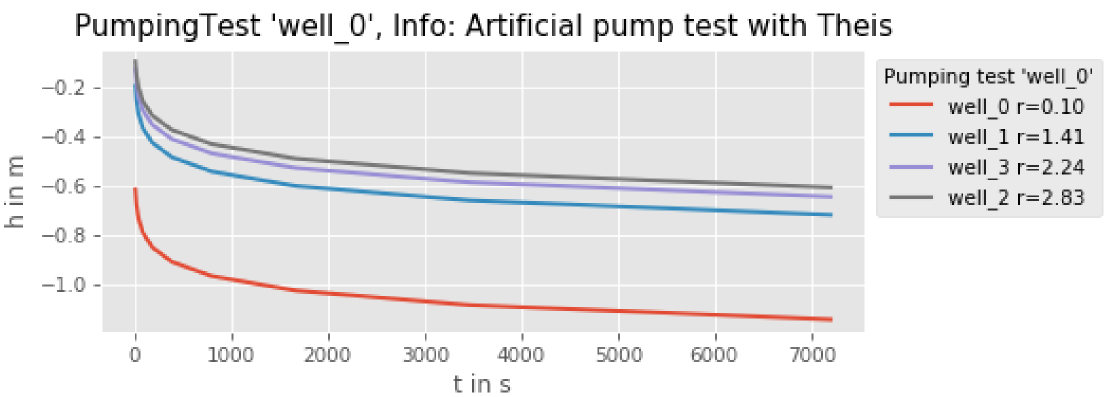
```
campaign.addtests(pumptest)
# optionally make the test (quasi)steady
# campaign.tests["well_0"].make_steady()
```

Plot the well constellation and a test overview

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



•

- 

Save the whole campaign to a file

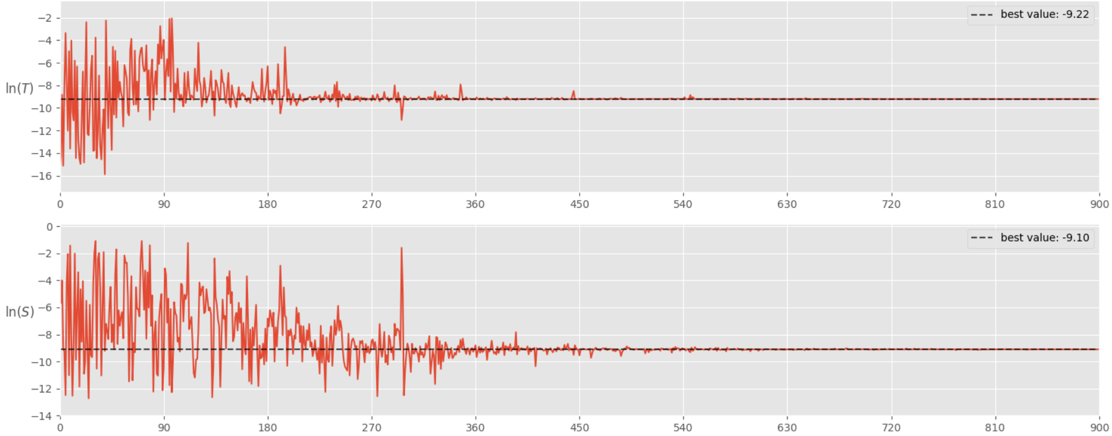
```
campaign.save()
```

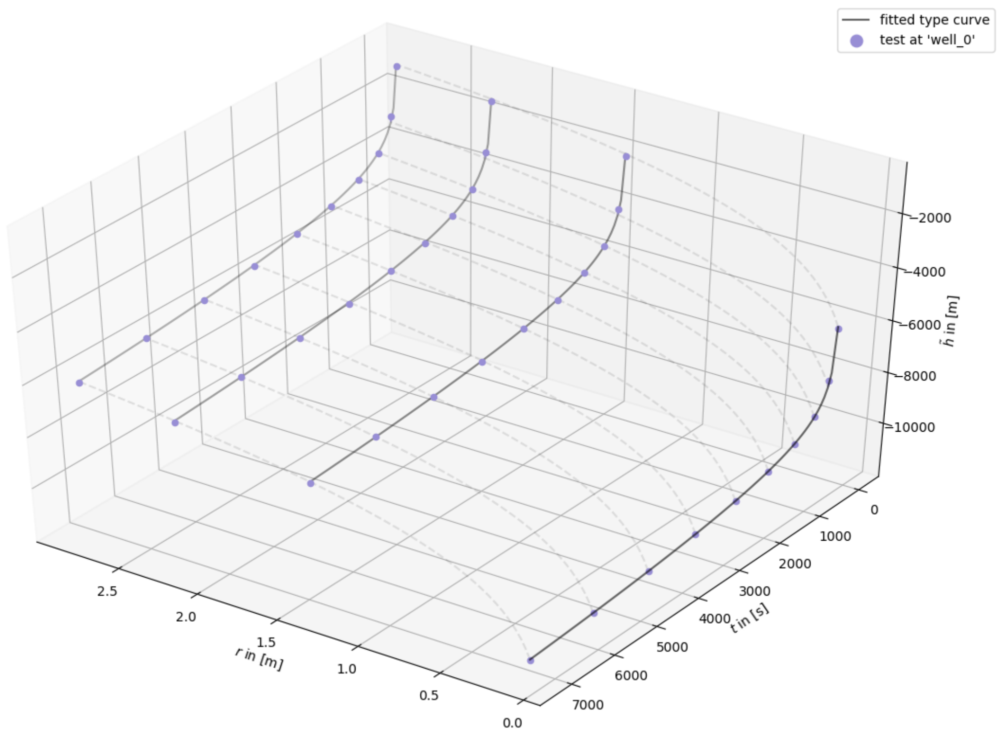
Total running time of the script: (0 minutes 1.559 seconds)

Estimate homogeneous parameters

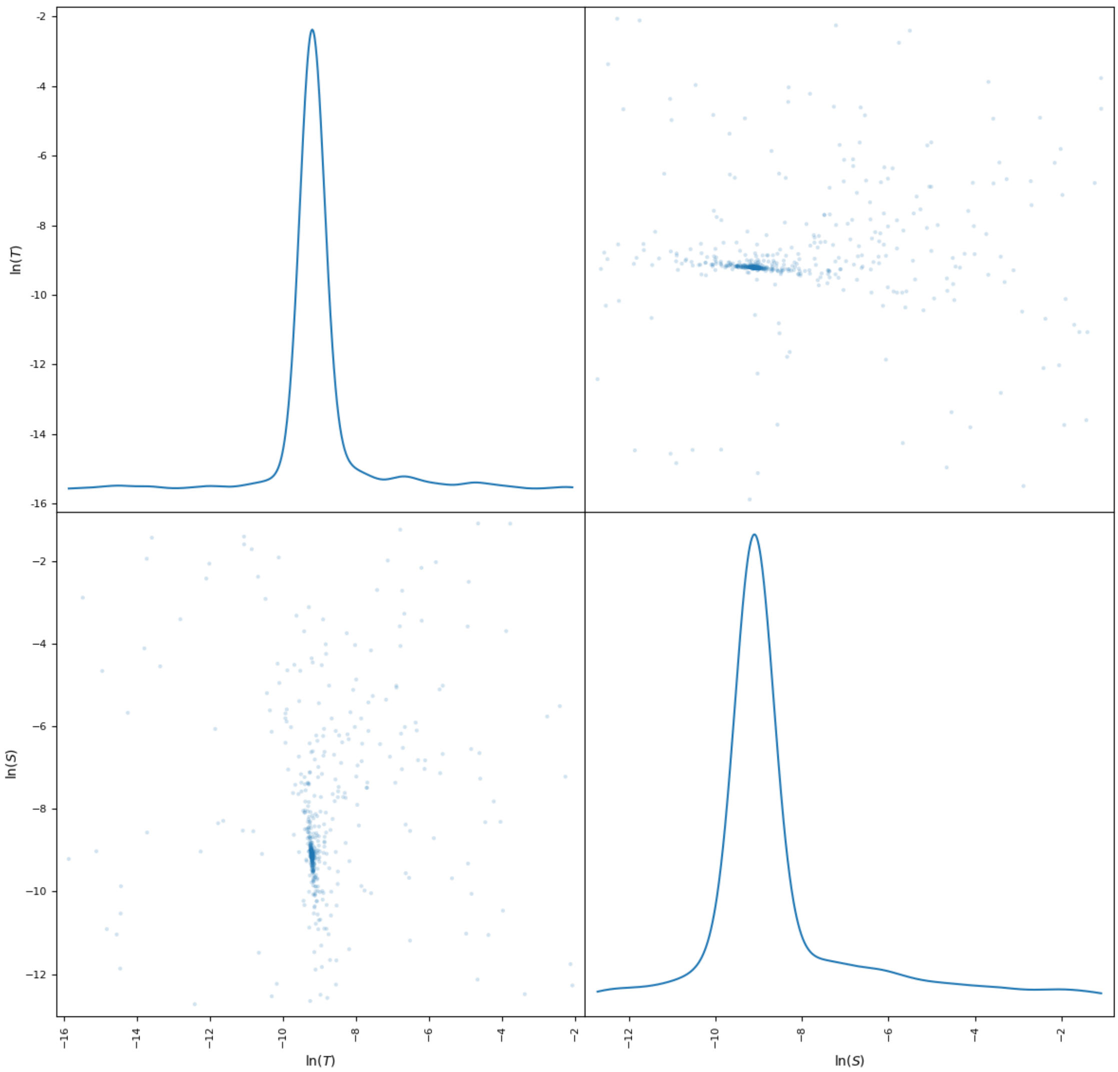
Here we estimate transmissivity and storage from a pumping test campaign with the classical theis solution.

```
import welltestpy as wtp

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

- 


fitted type curve
test at 'well_0'
r in [m]
t in [s]
h̃ in [m]
2.5
2.0
1.5
1.0
0.5
0.0
0
1000
2000
3000
4000
5000
6000
7000
−2000
−4000
−6000
−8000
−10000

•



•

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/v1.0.0/examples/Estimate_theis/2020-09-02_14-47-48_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...
```

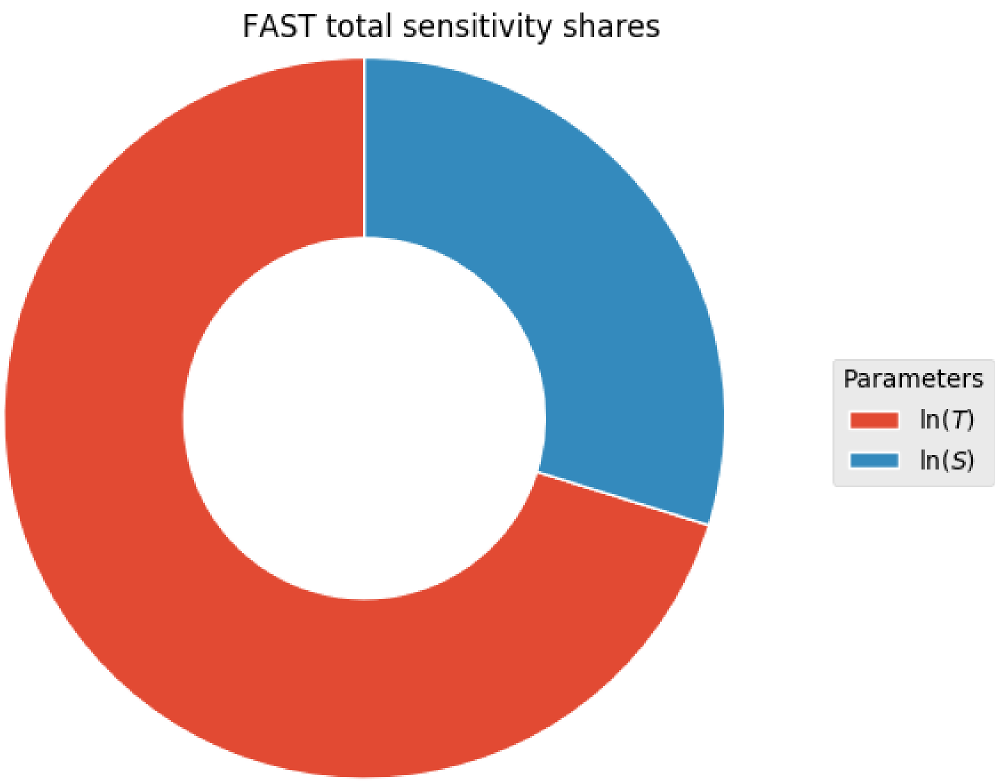
(continues on next page)

(continued from previous page)

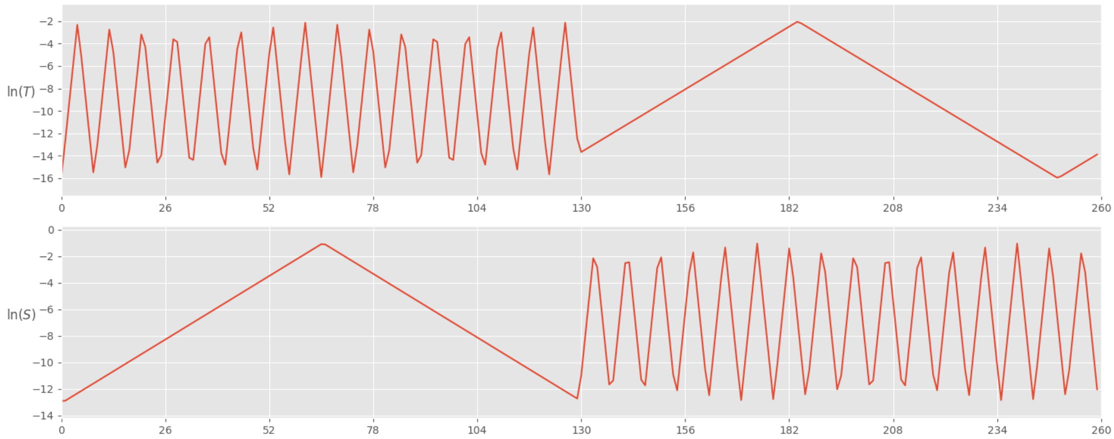
```
ComplexEvo loop #12 in progress...
ComplexEvo loop #13 in progress...
ComplexEvo loop #14 in progress...
ComplexEvo loop #15 in progress...
ComplexEvo loop #16 in progress...
ComplexEvo loop #17 in progress...
THE POPULATION HAS CONVERGED TO A PRESPECIFIED SMALL PARAMETER SPACE
SEARCH WAS STOPPED AT TRIAL NUMBER: 2321
NUMBER OF DISCARDED TRIALS: 0
NORMALIZED GEOMETRIC RANGE = 0.000631
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 1.34 seconds
Total Repetitions: 2321
Minimal objective value: 77.2517
Corresponding parameter setting:
mu: -9.21583
lnS: -9.10172
******************************

Best parameter set:
mu=-9.215825246032006, lnS=-9.10171769507417
```

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

```
estimation.sensitivity()
```



•

- 

Out:

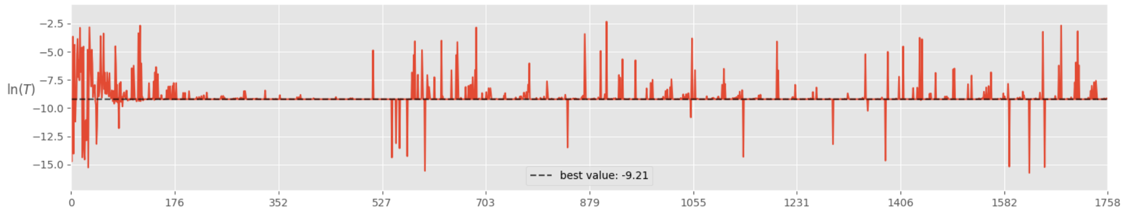
```
Initializing the  Fourier Amplitude Sensitivity Test (FAST)  with  260  repetitions
Starting the FAST algotrithm with 260 repetitions...
Creating FAST Matrix
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/v1.0.0/examples/Estimate_theis/2020-09-02_14-47-56_sensitivity_db.csv'␣
↪created.

*** Final SPOTPY summary ***
Total Duration: 0.2 seconds
Total Repetitions: 260
Minimal objective value: 1088.64
Corresponding parameter setting:
mu: -8.77605
lnS: -11.8128
Maximal objective value: 3.74776e+06
Corresponding parameter setting:
mu: -15.9061
lnS: -12.9195
******************************

260
Parameter First Total
mu 0.427928 0.912579
lnS 0.061301 0.387304
260
```

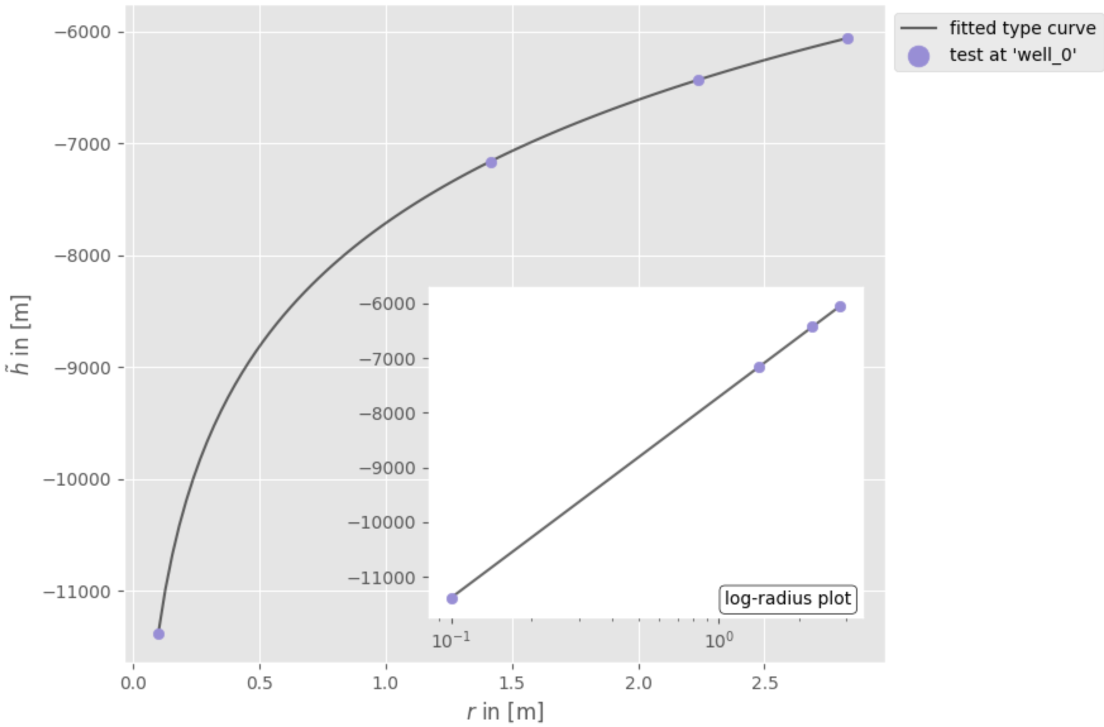
Total running time of the script: (0 minutes 8.543 seconds)

Estimate steady homogeneous parameters

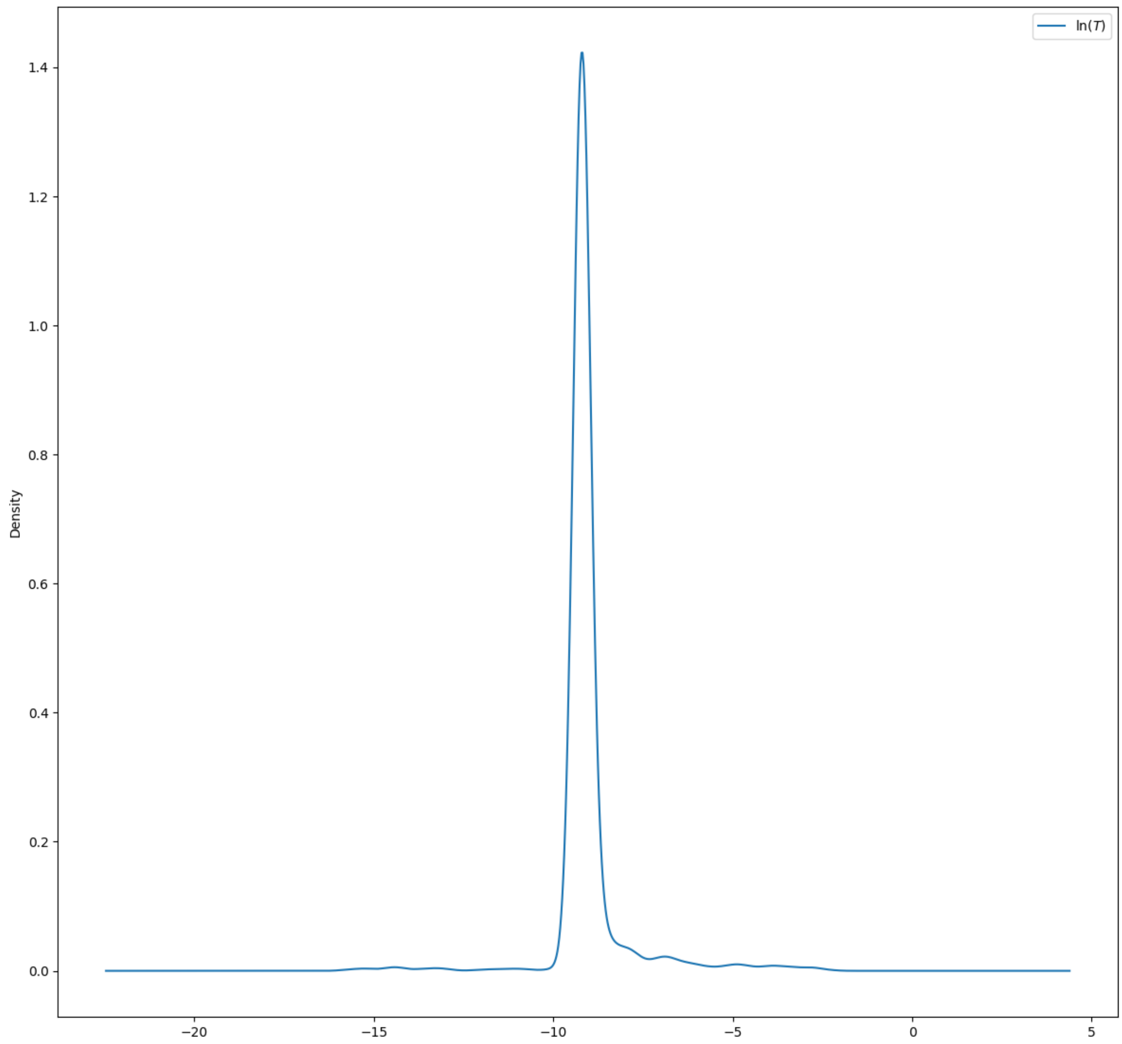
Here we estimate transmissivity from the quasi steady state of a pumping test campaign with the classical thiem solution.

```
import welltestpy as wtp

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

- 
ln(T)
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
best value: -9.21
0
176
352
527
703
879
1055
1231
1406
1582
1758

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

-



•

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/v1.0.0/examples/Estimate_thiem/2020-09-02_14-47-57_db.csv' created.
Burn-in sampling completed...
Starting Complex Evolution...
ComplexEvo loop #1 in progress...
ComplexEvo loop #2 in progress...
ComplexEvo loop #3 in progress...
ComplexEvo loop #4 in progress...
ComplexEvo loop #5 in progress...
ComplexEvo loop #6 in progress...
ComplexEvo loop #7 in progress...
ComplexEvo loop #8 in progress...
ComplexEvo loop #9 in progress...
ComplexEvo loop #10 in progress...
ComplexEvo loop #11 in progress...
```

(continues on next page)

(continued from previous page)

```
ComplexEvo loop #12 in progress...
ComplexEvo loop #13 in progress...
ComplexEvo loop #14 in progress...
ComplexEvo loop #15 in progress...
ComplexEvo loop #16 in progress...
ComplexEvo loop #17 in progress...
ComplexEvo loop #18 in progress...
ComplexEvo loop #19 in progress...
ComplexEvo loop #20 in progress...
ComplexEvo loop #21 in progress...
ComplexEvo loop #22 in progress...
ComplexEvo loop #23 in progress...
ComplexEvo loop #24 in progress...
ComplexEvo loop #25 in progress...
ComplexEvo loop #26 in progress...
ComplexEvo loop #27 in progress...
ComplexEvo loop #28 in progress...
ComplexEvo loop #29 in progress...
ComplexEvo loop #30 in progress...
ComplexEvo loop #31 in progress...
ComplexEvo loop #32 in progress...
ComplexEvo loop #33 in progress...
ComplexEvo loop #34 in progress...
ComplexEvo loop #35 in progress...
ComplexEvo loop #36 in progress...
ComplexEvo loop #37 in progress...
ComplexEvo loop #38 in progress...
ComplexEvo loop #39 in progress...
ComplexEvo loop #40 in progress...
ComplexEvo loop #41 in progress...
ComplexEvo loop #42 in progress...
ComplexEvo loop #43 in progress...
ComplexEvo loop #44 in progress...
ComplexEvo loop #45 in progress...
ComplexEvo loop #46 in progress...
ComplexEvo loop #47 in progress...
ComplexEvo loop #48 in progress...
4102 of 5000, minimal objective function=0.0672645, time remaining: 00:00:00
ComplexEvo loop #49 in progress...
ComplexEvo loop #50 in progress...
ComplexEvo loop #51 in progress...
ComplexEvo loop #52 in progress...
ComplexEvo loop #53 in progress...
ComplexEvo loop #54 in progress...
ComplexEvo loop #55 in progress...
ComplexEvo loop #56 in progress...
ComplexEvo loop #57 in progress...
ComplexEvo loop #58 in progress...
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
Skipping saving
*** OPTIMIZATION SEARCH TERMINATED BECAUSE THE LIMIT
```

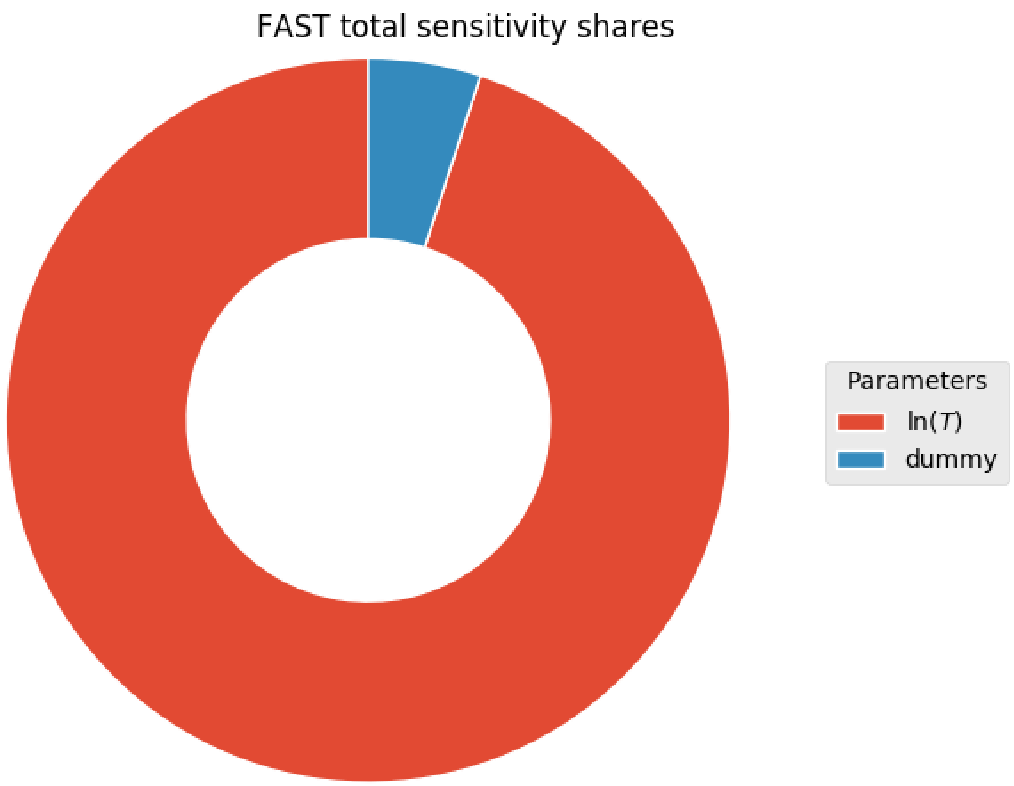
(continues on next page)

(continued from previous page)

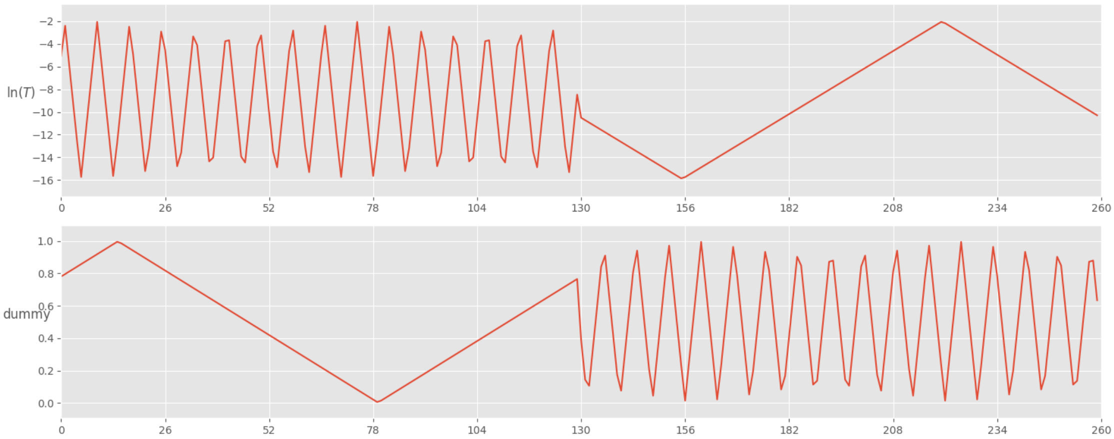
```
ON THE MAXIMUM NUMBER OF TRIALS
5000
HAS BEEN EXCEEDED.
SEARCH WAS STOPPED AT TRIAL NUMBER: 5034
NUMBER OF DISCARDED TRIALS: 12
NORMALIZED GEOMETRIC RANGE = 0.107884
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 2.43 seconds
Total Repetitions: 5034
Minimal objective value: 0.0672645
Corresponding parameter setting:
mu: -9.21029
******************************

Best parameter set:
mu=-9.210293557336158
```

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

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



•

- 

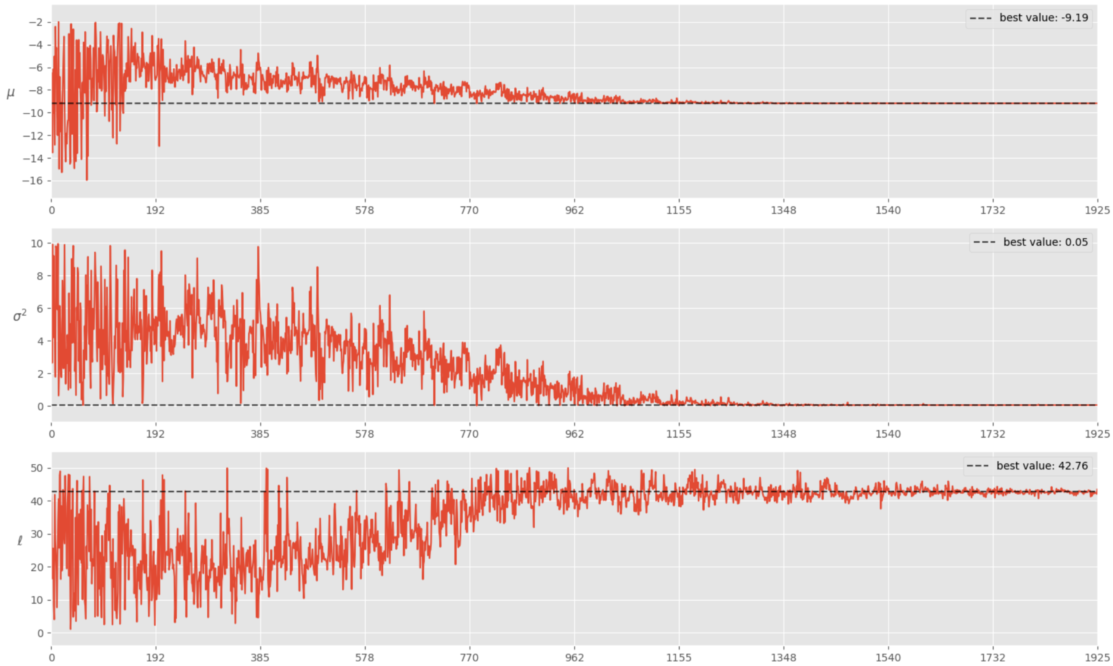
Out:

```
Initializing the  Fourier Amplitude Sensitivity Test (FAST)  with  260  repetitions
Starting the FAST algotrithm with 260 repetitions...
Creating FAST Matrix
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/v1.0.0/examples/Estimate_thiem/2020-09-02_14-48-02_sensitivity_db.csv'␣
↪created.

*** Final SPOTPY summary ***
Total Duration: 0.17 seconds
Total Repetitions: 260
Minimal objective value: 50.8679
Corresponding parameter setting:
mu: -9.2288
dummy: 0.136662
Maximal objective value: 2.0766e+06
Corresponding parameter setting:
mu: -15.8485
dummy: 0.23594
******************************

260
Parameter First Total
mu 0.804354 0.980549
dummy 0.001831 0.051718
260
```

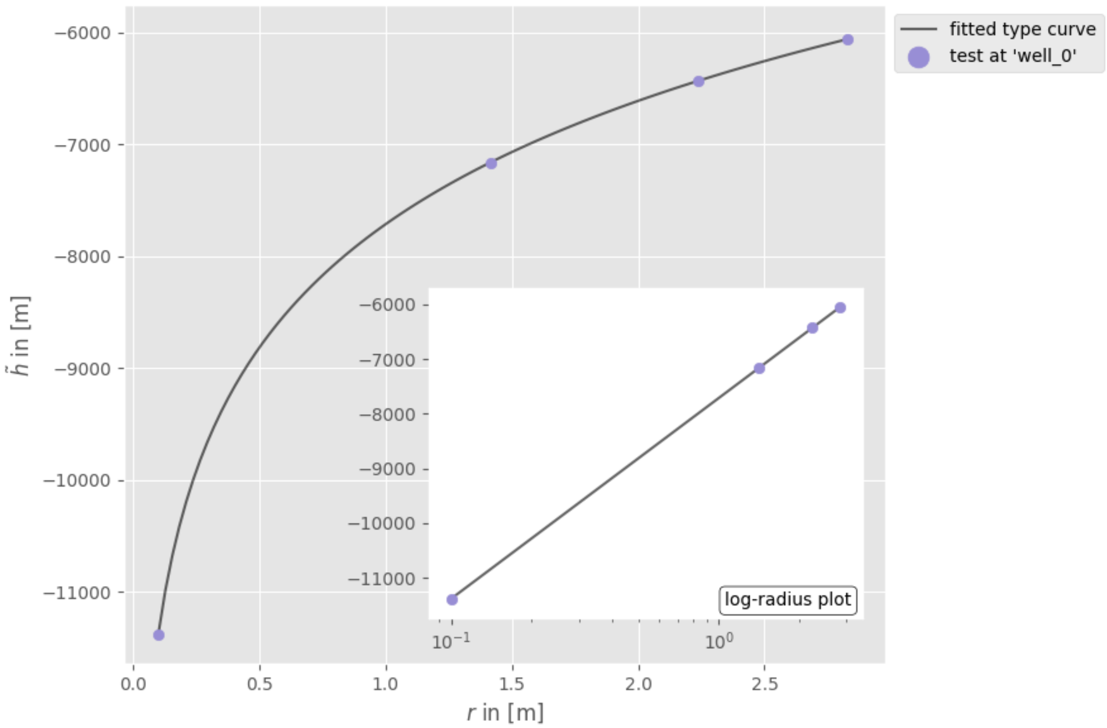
Total running time of the script: (0 minutes 6.289 seconds)

Estimate steady heterogeneous parameters

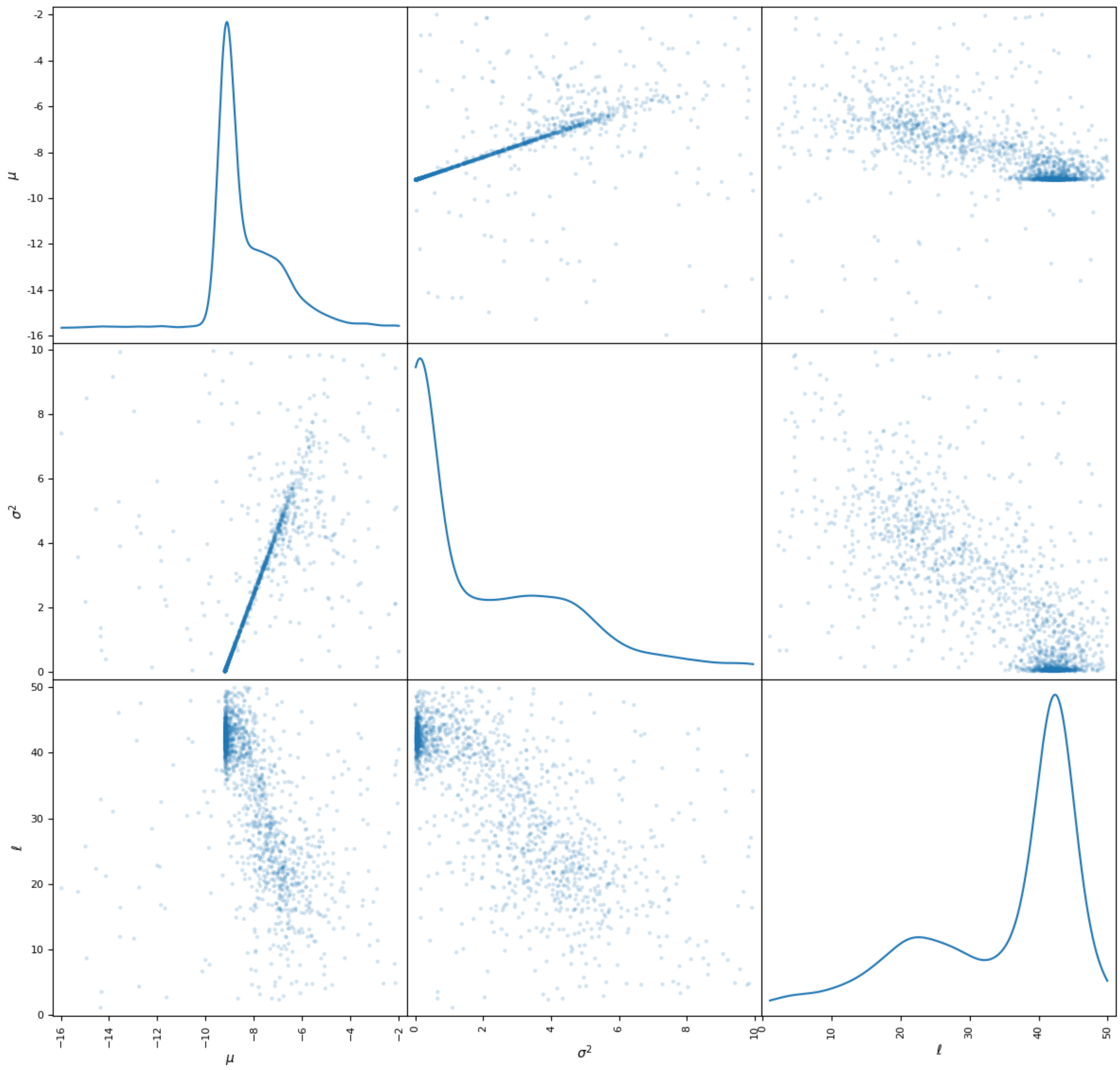
Here we demonstrate how to estimate parameters of heterogeneity, namely mean, variance and correlation length of log-transmissivity, with the aid the the extended Thiem solution in 2D.

- 

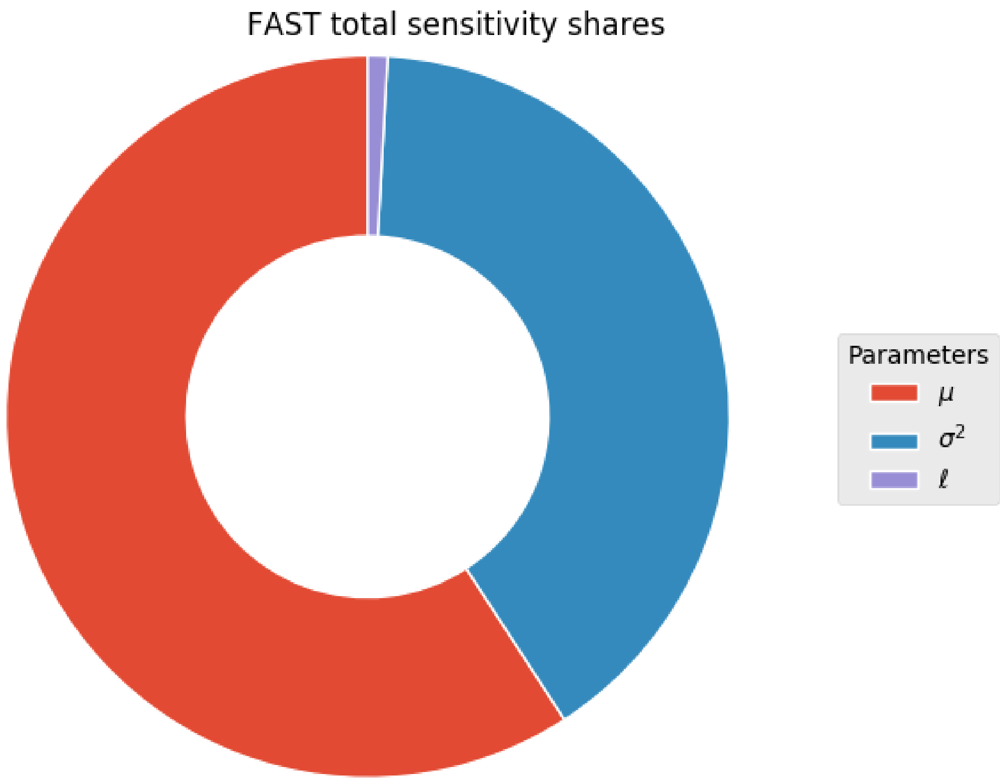
best value: -9.19
best value: 0.05
best value: 42.76
μ
σ²
ℓ

- 

fitted type curve
test at 'well_0'
log-radius plot
h̃ in [m]
r in [m]


μ
σ²
ℓ
-2
-4
-6
-8
-10
-12
-14
-16
10
8
6
4
2
0
50
40
30
20
10
0
−16
−14
−12
−10
−8
−6
−4
−2
0
2
4
6
8
10
0
10
20
30
40
50

•



•

•

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/v1.0.0/examples/Est_steady_het/2020-09-02_14-48-03_db.csv'
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
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: 5085
NUMBER OF DISCARDED TRIALS: 35
NORMALIZED GEOMETRIC RANGE = 0.002279
THE BEST POINT HAS IMPROVED IN LAST 100 LOOPS BY 100000.000000 PERCENT

*** Final SPOTPY summary ***
Total Duration: 1.95 seconds
Total Repetitions: 5085
Minimal objective value: 9.53537e-05
Corresponding parameter setting:
mu: -9.18534
var: 0.0500107
len_scale: 42.7641
******************************

Best parameter set:
mu=-9.185335170518913, var=0.0500106603181542, len_scale=42.76412084375356
Initializing the  Fourier Amplitude Sensitivity Test (FAST)  with  1158 
↪repetitions
Starting the FAST algotrithm with 1158 repetitions...
Creating FAST Matrix
Initialize database...
['csv', 'hdf5', 'ram', 'sql', 'custom', 'noData']
* Database file '/home/docs/checkouts/readthedocs.org/user_builds/welltestpy/
↪checkouts/v1.0.0/examples/Est_steady_het/2020-09-02_14-48-08_sensitivity_db.csv'
↪created.

*** Final SPOTPY summary ***
Total Duration: 0.56 seconds
Total Repetitions: 1158
Minimal objective value: 30.3117
Corresponding parameter setting:
mu: -4.8882
var: 8.6911
len_scale: 26.5871
Maximal objective value: 1.94511e+08
Corresponding parameter setting:
mu: -15.9023
var: 8.99875
len_scale: 32.0772
******************************

1158
Parameter First Total
mu 0.378846 0.888522
var 0.083174 0.600185
len_scale 0.000252 0.013226
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 7.745 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.

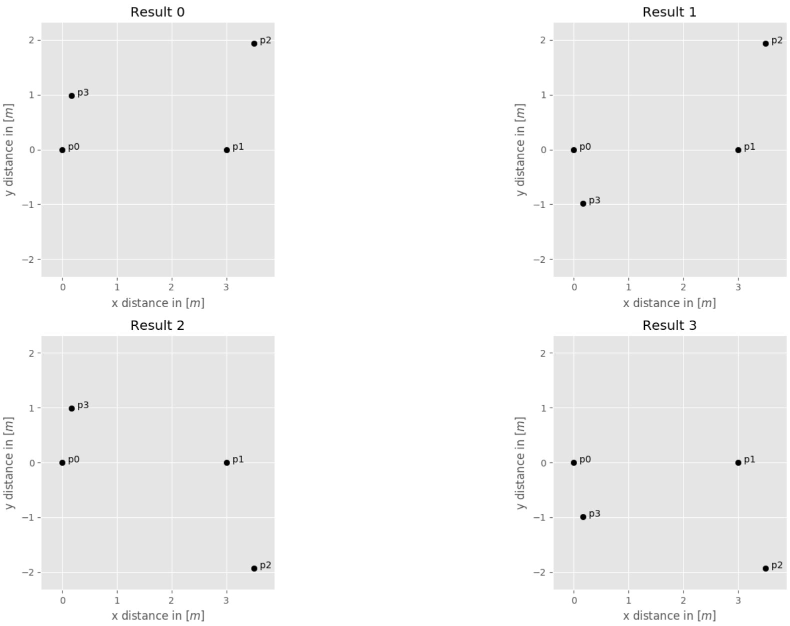
```
import numpy as np
from welltestpy.tools import triangulate, sym, plot_well_pos

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:

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

Now we can plot all possible well constellations

```
plot_well_pos(well_const)
```



Total running time of the script: (0 minutes 0.700 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

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

Classes

Campaign classes

The following classes can be used to handle field campaigns.

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

Field Test classes

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

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

Loading routines

Campaign related loading routines

load_campaign(cmpfile)	Load a campaign from file.

3.2 welltestpy.data

welltestpy subpackage providing datastructures.

Subpackages

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

Classes

Campaign classes

The following classes can be used to handle field campaigns.

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

Field Test classes

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

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

Variable classes

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

Routines

Loading routines

Campaign related loading routines

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

Field test related loading routines

load_test(tstfile)	Load a test from file.

Variable related loading routines

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

welltestpy.data.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

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

Parameters

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

Attributes

info `str`: Info about the Variable.

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

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

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

Methods

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

__call__ (*value=None*)

Call a variable.

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

Parameters

- **value** (`int` or `float` or `numpy.ndarray`,) –

- **optional** – Value of the Variable. Default: `None`

Returns **value** – Value of the Variable.

Return type `int` or `float` or `numpy.ndarray`

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

Save a variable to file.

This writes the variable to a csv file.

Parameters

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

Notes

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

---

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.
- **symbole** (`str`, optional) – Name of the Variable. Default: `"t"`
- **units** (`str`, optional) – Units of the Variable. Default: `"s"`
- **description** (`str`, optional) – Description of the Variable. Default: `"time given in seconds"`

Notes

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

Attributes

info `str`: Info about the Variable.

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

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

value int or float or numpy.ndarray: Value.

Methods

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

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.
- **symbole** (str, optional) – Name of the Variable. Default: "h"
- **units** (str, optional) – Units of the Variable. Default: "m"
- **description** (str, optional) – Description of the Variable. Default: "head given in meters"

Notes

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

Attributes

info str: Info about the Variable.

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

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

value int or float or numpy.ndarray: Value.

Methods

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

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

__call__([value])	Call a variable.
save([path, name])	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) – Longitutional values of the coordinates.
- **symbole** (str, optional) – Name of the Variable. Default: "[Lat,Lon]"
- **units** (str, optional) – Units of the Variable. Default: "[deg,deg]"
- **description** (str, optional) – Description of the Variable. Default: "Coordinates given in degree-North and degree-East"

Notes

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

lat and lon should have the same shape.

Attributes

info str: Info about the Variable.

label str: String containing: symbol in units.

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

value int or float or numpy.ndarray: Value.

Methods

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

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 descrition.

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

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

__call__(*observation=None*, *time=None*)
Call a variable.

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

Parameters

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

Returns

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

reshape()
Reshape 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

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

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

class StdyHeadObs (*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

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

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

Bases: *welltestpy.data.varlib.Observation*

Time series obeservation.

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

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

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

distance(*well*)

Calculate distance to the well.

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

save(*path=”*, *name=None*)

Save a well to file.

This writes the variable to a csv file.

Parameters

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

Notes

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

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

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

Parameters

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

Attributes

testtype `str`: String containing the test type.

Methods

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

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

Generate a plot of the pumping test.

This will plot the test on the given figure axes.

Parameters

- **ax** (`Axes`) – Axes where the plot should be done.
- **wells** (`dict`) – Dictonary 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*, *aquiferradius=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 descrition, 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 ot the field site. If a *float* is given, it is assumed to be given in m. Default: inf
- **description** (str, optional) – Description of the test. Default: "Pumpingtest"
- **timeframe** (str, optional) – Timeframe of the test. Default: None

Attributes

aquiferdepth 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

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

add_observations (*obs*)

Add some specified observations.

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

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

Add steady drawdown observations.

Parameters

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

add_transient_obs (*well*, *time*, *observation*, *description='Transient Drawdown observation'*)
Add transient drawdown observations.

Parameters

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

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) – Dictonary 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

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

Parameters

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

Attributes

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

info `str`: Info about the field site.

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

Methods

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

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

Save a field site to file.

This writes the field site to a csv file.

Parameters

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

Notes

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

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

Attributes

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

tests `dict`: Tests within the campaign.

wells `dict`: Wells within the campaign.

Methods

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

add_well(*name, radius, coordinates, welldepth=1.0, aquiferdepth=None*)
Add a single well to the campaign.

Parameters

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

addtests(*tests*)
Add some specified tests.

This will add tests to the campaign.

Parameters **tests** (dict) – Tests to be added.

addwells (*wells*)

Add some specified wells.

This will add wells to the campaign.

Parameters **wells** (dict) – Wells to be added.

deltests (*tests*)

Delete some specified tests.

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

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

delwells (*wells*)

Delete some specified wells.

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

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

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

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

Base Classes

Transient

All transient estimators are derived from the following class

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

Steady Pumping

All steady estimators are derived from the following class

SteadyPumping(name, campaign, type_curve, ...)	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 <= 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) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

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) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

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) – dictonary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

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

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) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

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 <= 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 signiture or replaced in val_kw_names. Default: None

- **testinclude** (dict, optional) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

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 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) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

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) – dictonary of which tests should be included. If `None` is given, all available tests are included. Default: `None`
- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

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

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 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) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

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 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
- **fit_type** (dict or None) – Dictionary containing fitting type for each value in the type-curve. Names should be as in the type-curve signiture 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) – dictonary of which tests should be included. If `None` is given, all available tests are included. Default: `None`

- **generate** (`bool`, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: `False`

Methods

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

gen_data()

Generate the observed drawdown at given time points.

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

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

Generate the Spotpy Setup.

Parameters

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

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

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

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 determind by the observation data. Default: None
- **tmin** (float, optional) – Minimal time value. It will set a minimal value of 10s. Default: 10
- **tmax** (float, optional) – Maximal time value. Default: inf
- **typ** (str or float, optional) – Typ of the time selection. You can select from:
  - "exp": for exponential behavior
  - "log": for logarithmic behavior
  - "geo": for geometric behavior
  - "lin": for linear behavior
  - "quad": for quadratic behavior
  - "cub": for cubic behavior
  - float: here you can specifi any exponent ("quad" would be equivalent to 2)

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

campaign = 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
dictonary 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 signiture 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 signiture 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 signiture 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) – dictonary of which tests should be included. If None is given, all available tests are included. Default: None
- **generate** (bool, optional) – State if time stepping, processed observation data and estimation setup should be generated with default values. Default: False

Methods

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

gen_data()

Generate the observed drawdown.

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

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

Generate the Spotpy Setup.

Parameters

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

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

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
dictonary 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

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

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

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

The following plotting routines are provided

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

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

Triangulate points by given distances.

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

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

Parameters

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

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

  If a distance is unknown, you can set it to `-1`.
- **prec** (`float`) – Given Precision to be used within the algorithm. This can be used to smooth away messure 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 stlye. 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 stlye. 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 stlye. 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.

PYTHON MODULE INDEX

W

welltestpy, 23
welltestpy.data, 25
welltestpy.data.campaignlib, 44
welltestpy.data.data_io, 27
welltestpy.data.testslib, 40
welltestpy.data.varlib, 30
welltestpy.estimate, 48
welltestpy.process, 63
welltestpy.tools, 64

INDEX

Symbols

__call__() (*Observation method*), 34
__call__() (*Variable method*), 30

A

add_observations() (*PumpingTest method*), 41
add_steady_obs() (*PumpingTest method*), 41
add_transient_obs() (*PumpingTest method*), 42
add_well() (*Campaign method*), 45
addtests() (*Campaign method*), 45
addwells() (*Campaign method*), 46
aquiferdepth (*PumpingTest attribute*), 43
aquiferdepth (*Well attribute*), 39
aquiferradius (*PumpingTest attribute*), 43

C

Campaign (*class in welltestpy.data.campaignlib*), 45
campaign (*SteadyPumping attribute*), 60
campaign (*TransientPumping attribute*), 57
campaign_plot() (*in module welltestpy.tools*), 64
campaign_raw (*SteadyPumping attribute*), 60
campaign_raw (*TransientPumping attribute*), 57
campaign_well_plot() (*in module welltestpy.tools*), 65
combinepumptest() (*in module welltestpy.process*), 63
constant_rate (*PumpingTest attribute*), 43
coordinates (*FieldSite attribute*), 44
coordinates (*Well attribute*), 39
CoordinatesVar (*class in welltestpy.data.varlib*), 33

D

data (*SteadyPumping attribute*), 61
data (*TransientPumping attribute*), 57
del_observations() (*PumpingTest method*), 42
deltests() (*Campaign method*), 46
delwells() (*Campaign method*), 46
depth (*PumpingTest attribute*), 43
depth (*Well attribute*), 39
distance() (*Well method*), 38
DrawdownObs (*class in welltestpy.data.varlib*), 36

E

estimated_para (*SteadyPumping attribute*), 61
estimated_para (*TransientPumping attribute*), 57
ExtTheis2D (*class in welltestpy.estimate*), 49
ExtTheis3D (*class in welltestpy.estimate*), 48
ExtThiem2D (*class in welltestpy.estimate*), 52
ExtThiem3D (*class in welltestpy.estimate*), 51

F

fadeline() (*in module welltestpy.tools*), 65
fieldsite (*Campaign attribute*), 46
FieldSite (*class in welltestpy.data.campaignlib*), 44
filterdrawdown() (*in module welltestpy.process*), 63

G

gen_data() (*SteadyPumping method*), 59
gen_data() (*TransientPumping method*), 55
gen_setup() (*SteadyPumping method*), 59
gen_setup() (*TransientPumping method*), 55

H

h_ref (*SteadyPumping attribute*), 61
HeadVar (*class in welltestpy.data.varlib*), 32

I

info (*FieldSite attribute*), 44
info (*Observation attribute*), 34
info (*Variable attribute*), 31
info (*Well attribute*), 39

K

kind (*Observation attribute*), 35

L

label (*Observation attribute*), 35
label (*Variable attribute*), 31
labels (*Observation attribute*), 35
load_campaign() (*in module welltestpy.data.data_io*), 27
load_fieldsite() (*in module welltestpy.data.data_io*), 27
load_obs() (*in module welltestpy.data.data_io*), 27

load_test() (*in module welltestpy.data.data_io*), 27
load_var() (*in module welltestpy.data.data_io*), 27
load_well() (*in module welltestpy.data.data_io*), 27

M

make_steady() (*PumpingTest method*), 42

N

name (*SteadyPumping attribute*), 61
name (*TransientPumping attribute*), 57
Neuman2004 (*class in welltestpy.estimate*), 50
Neuman2004Steady (*class in welltestpy.estimate*), 53
normpumptest() (*in module welltestpy.process*), 63

O

Observation (*class in welltestpy.data.varlib*), 33
observation (*Observation attribute*), 35
observations (*PumpingTest attribute*), 43
observationwells (*PumpingTest attribute*), 43

P

plot() (*Campaign method*), 46
plot() (*PumpingTest method*), 42
plot() (*Test method*), 40
plot_well_pos() (*in module welltestpy.tools*), 65
plot_wells() (*Campaign method*), 46
plotfit_steady() (*in module welltestpy.tools*), 66
plotfit_transient() (*in module welltestpy.tools*), 66
plotparainteract() (*in module welltestpy.tools*), 66
plotparatrace() (*in module welltestpy.tools*), 66
plotsensitivity() (*in module welltestpy.tools*), 66
pos (*FieldSite attribute*), 44
pos (*Well attribute*), 39
prate (*SteadyPumping attribute*), 61
prate (*TransientPumping attribute*), 57
pumpingrate (*PumpingTest attribute*), 43
PumpingTest (*class in welltestpy.data.testslib*), 40

R

r_ref (*SteadyPumping attribute*), 61
rad (*SteadyPumping attribute*), 61
rad (*TransientPumping attribute*), 57
radius (*PumpingTest attribute*), 43
radius (*Well attribute*), 39
radnames (*SteadyPumping attribute*), 61
radnames (*TransientPumping attribute*), 57
rate (*PumpingTest attribute*), 43
reshape() (*Observation method*), 34
result (*SteadyPumping attribute*), 61
result (*TransientPumping attribute*), 57
rinf (*SteadyPumping attribute*), 61
rinf (*TransientPumping attribute*), 57
run() (*SteadyPumping method*), 59
run() (*TransientPumping method*), 55
rwell (*SteadyPumping attribute*), 61
rwell (*TransientPumping attribute*), 58

S

save() (*Campaign method*), 46
save() (*FieldSite method*), 44
save() (*Observation method*), 34
save() (*PumpingTest method*), 42
save() (*Variable method*), 31
save() (*Well method*), 38
save_campaign() (*in module welltestpy.data.data_io*), 27
save_fieldsite() (*in module welltestpy.data.data_io*), 27
save_obs() (*in module welltestpy.data.data_io*), 28
save_pumping_test() (*in module welltestpy.data.data_io*), 28
save_var() (*in module welltestpy.data.data_io*), 28
save_well() (*in module welltestpy.data.data_io*), 28
scalar (*Variable attribute*), 31
sens (*SteadyPumping attribute*), 61
sens (*TransientPumping attribute*), 58
sensitivity() (*SteadyPumping method*), 60
sensitivity() (*TransientPumping method*), 56
setpumprate() (*SteadyPumping method*), 60
setpumprate() (*TransientPumping method*), 56
settime() (*TransientPumping method*), 56
setup_kw (*SteadyPumping attribute*), 61
setup_kw (*TransientPumping attribute*), 58
state (*Observation attribute*), 35
state() (*PumpingTest method*), 43
StdyHeadObs (*class in welltestpy.data.varlib*), 36
StdyObs (*class in welltestpy.data.varlib*), 35
SteadyPumping (*class in welltestpy.estimate*), 58
sym() (*in module welltestpy.tools*), 64

T

TemporalVar (*class in welltestpy.data.varlib*), 32
Test (*class in welltestpy.data.testslib*), 40
testinclude (*SteadyPumping attribute*), 61
testinclude (*TransientPumping attribute*), 58
tests (*Campaign attribute*), 46
testtype (*Test attribute*), 40
Theis (*class in welltestpy.estimate*), 50
Thiem (*class in welltestpy.estimate*), 53
time (*Observation attribute*), 35
time (*TransientPumping attribute*), 58
TimeSeries (*class in welltestpy.data.varlib*), 37
TimeVar (*class in welltestpy.data.varlib*), 31
TransientPumping (*class in welltestpy.estimate*), 54
triangulate() (*in module welltestpy.tools*), 64

U

units (*Observation attribute*), 35

V

value (*Observation attribute*), 35
value (*Variable attribute*), 31
Variable (*class in welltestpy.data.varlib*), 30

W

Well (*class in welltestpy.data.varlib*), 38
welldepth (*Well attribute*), 39
wellradius (*Well attribute*), 39
wells (*Campaign attribute*), 46
wells (*PumpingTest attribute*), 43
welltestpy (*module*), 23
welltestpy.data (*module*), 25
welltestpy.data.campaignlib (*module*), 44
welltestpy.data.data_io (*module*), 27
welltestpy.data.testslib (*module*), 40
welltestpy.data.varlib (*module*), 30
welltestpy.estimate (*module*), 48
welltestpy.process (*module*), 63
welltestpy.tools (*module*), 64