AnaFlow Documentation

Release 1.1.0

Sebastian Mueller

Apr 16, 2023

CONTENTS

CHAPTER 1

ANAFLOW QUICKSTART

AnaFlow provides several analytical and semi-analytical solutions for the groundwater-flow equation.

1.1 Installation

The package can be installed via pip. On Windows you can install WinPython to get Python and pip running.

```
pip install anaflow
```

1.2 Provided Functions

The following functions are provided directly

- *thiem* Thiem solution for steady state pumping
- *theis* Theis solution for transient pumping
- *ext_thiem_2d* extended Thiem solution in 2D from *Zech 2013*
- *ext_theis_2d* extended Theis solution in 2D from *Mueller 2015*
- *ext_thiem_3d* extended Thiem solution in 3D from *Zech 2013*
- *ext_theis_3d* extended Theis solution in 3D from *Mueller 2015*
- *neuman2004* transient solution from *Neuman 2004*
- *neuman2004_steady* steady solution from *Neuman 2004*
- *grf* "General Radial Flow" Model from *Barker 1988*

- *ext_grf* the transient extended GRF model
- *ext_grf_steady* the steady extended GRF model
- *ext_thiem_tpl* extended Thiem solution for truncated power laws
- *ext_theis_tpl* extended Theis solution for truncated power laws
- *ext_thiem_tpl_3d* extended Thiem solution in 3D for truncated power laws
- *ext_theis_tpl_3d* extended Theis solution in 3D for truncated power laws

1.3 Laplace Transformation

We provide routines to calculate the laplace-transformation as well as the inverse laplace-transformation of a given function

- *get_lap* Get the laplace transformation of a function
- *get_lap_inv* Get the inverse laplace transformation of a function

1.4 Requirements

- NumPy >= 1.14.5
- SciPy >= 1.1.0
- pentapy

1.5 License

MIT

CHAPTER 2

ANAFLOW TUTORIAL

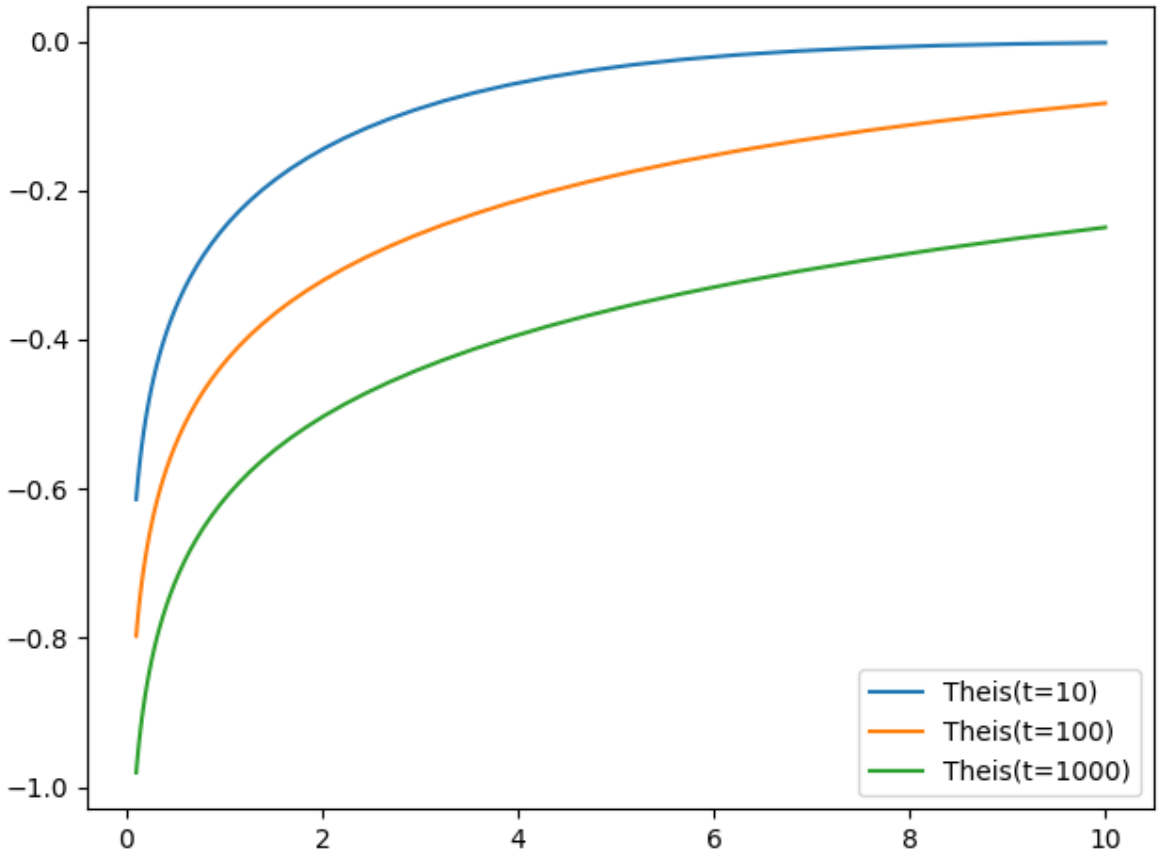
In the following you will find several Tutorials on how to use AnaFlow to explore its whole beauty and power.

2.1 Gallery

The Theis solution

In the following the well known Theis function is called an plotted for three different time-steps.

Reference: Theis 1935



```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import theis

time = [10, 100, 1000]
rad = np.geomspace(0.1, 10)

head = theis(time=time, rad=rad, storage=1e-4, transmissivity=1e-4, rate=-1e-4)

for i, step in enumerate(time):
    plt.plot(rad, head[i], label="Theis(t={})".format(step))

plt.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.214 seconds)

The extended Theis solution in 2D

We provide an extended theis solution, that incorporates the effectes of a heterogeneous transmissivity field on a pumping test.

In the following this extended solution is compared to the standard theis solution for well flow. You can nicely see, that the extended solution represents a transition between the theis solutions for the geometric- and harmonic-mean transmissivity.

Reference: Zech et. al. 2016

```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_theis_2d, theis
```

We use three time steps: 10s, 10min, 10h

```
time_labels = ["10 s", "10 min", "10 h"]
time = [10, 600, 36000]  # 10s, 10min, 10h
```

Radius from the pumping well should be in [0, 4].

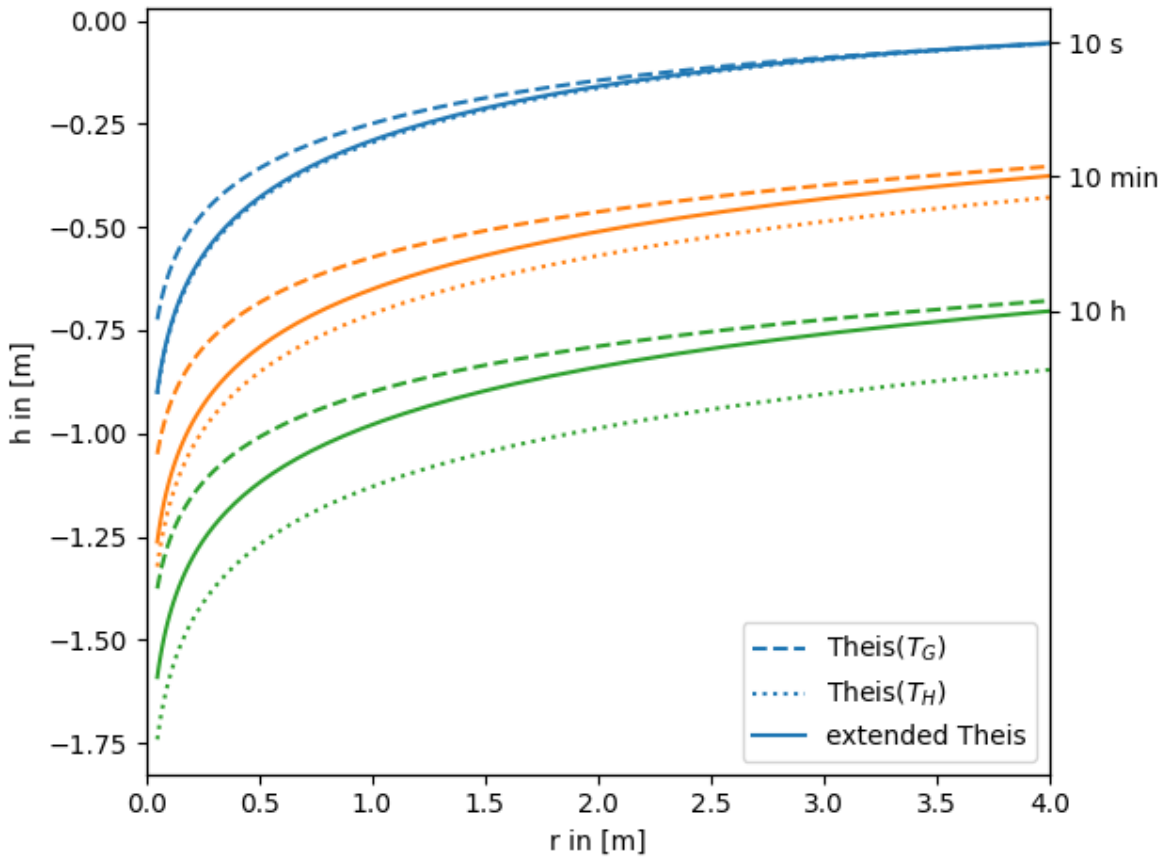
```
rad = np.geomspace(0.05, 4)
```

Parameters of heterogeneity, storage and pumping rate.

```
var = 0.5  # variance of the log-transmissivity
len_scale = 10.0  # correlation length of the log-transmissivity
TG = 1e-4  # the geometric mean of the transmissivity
TH = TG * np.exp(-var / 2.0)  # the harmonic mean of the transmissivity

S = 1e-4  # storativity
rate = -1e-4  # pumping rate
```

Now let's compare the extended Theis solution to the classical solutions for the near and far field values of transmissivity.

```
head_TG = theis(time, rad, S, TG, rate)
head_TH = theis(time, rad, S, TH, rate)
head_ef = ext_theis_2d(time, rad, S, TG, var, len_scale, rate)
time_ticks = []
for i, step in enumerate(time):
    label_TG = "Theis($T_G$)" if i == 0 else None
    label_TH = "Theis($T_H$)" if i == 0 else None
    label_ef = "extended Theis" if i == 0 else None
    plt.plot(
        rad, head_TG[i], label=label_TG, color="C" + str(i), linestyle="--"
    )
    plt.plot(
        rad, head_TH[i], label=label_TH, color="C" + str(i), linestyle=":"
    )
    plt.plot(rad, head_ef[i], label=label_ef, color="C" + str(i))
    time_ticks.append(head_ef[i][-1])

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
ylim = plt.gca().get_ylim()
plt.gca().set_xlim([0, rad[-1]])
ax2 = plt.gca().twinx()
ax2.set_yticks(time_ticks)
ax2.set_yticklabels(time_labels)
ax2.set_ylim(ylim)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.544 seconds)

The extended Theis solution in 3D

We provide an extended theis solution, that incorporates the effectes of a heterogeneous conductivity field on a pumping test. It also includes an anisotropy ratio of the horizontal and vertical length scales.

In the following this extended solution is compared to the standard theis solution for well flow. You can nicely see, that the extended solution represents a transition between the theis solutions for the effective and harmonic-mean conductivity.

Reference: Müller 2015

```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_theis_3d, theis
from anaflow.tools.special import aniso
```

We use three time steps: 10s, 10min, 10h

```
time_labels = ["10 s", "10 min", "10 h"]
time = [10, 600, 36000]  # 10s, 10min, 10h
```

Radius from the pumping well should be in [0, 4].

```
rad = np.geomspace(0.05, 4)
```

Parameters of heterogeneity, storage, extend and pumping rate.

```
var = 0.5  # variance of the log-conductivity
len_scale = 10.0  # correlation length of the log-conductivity
anis = 0.75  # anisotropy ratio of the log-conductivity
KG = 1e-4  # the geometric mean of the conductivity
Kefu = KG * np.exp(
    var * (0.5 - aniso(anis))
)  # the effective conductivity for uniform flow
KH = KG * np.exp(-var / 2.0)  # the harmonic mean of the conductivity

S = 1e-4  # storage
L = 1.0  # vertical extend of the aquifer
rate = -1e-4  # pumping rate
```

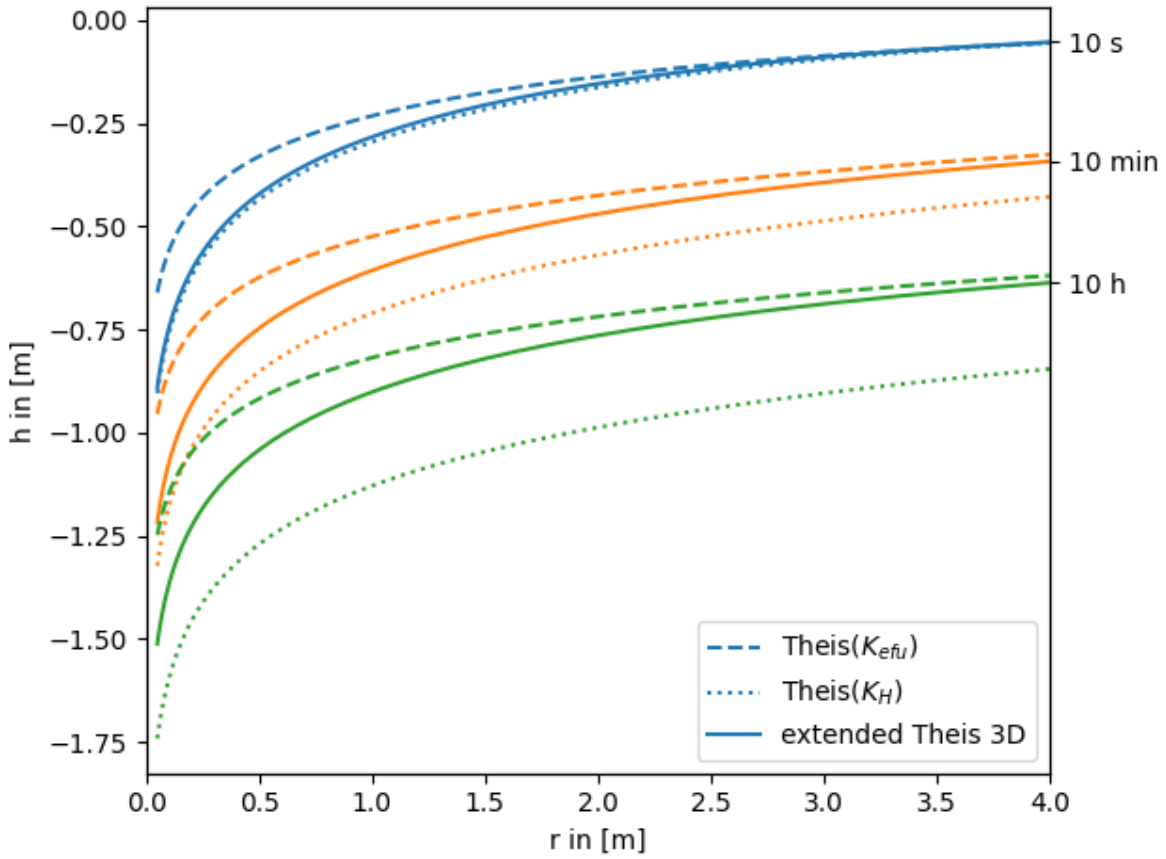
Now let's compare the extended Theis solution to the classical solutions for the near and far field values of transmissivity.

```
head_Kefu = theis(time, rad, S, Kefu * L, rate)
head_KH = theis(time, rad, S, KH * L, rate)
head_ef = ext_theis_3d(time, rad, S, KG, var, len_scale, anis, L, rate)
time_ticks = []
for i, step in enumerate(time):
    label_TG = "Theis($K_{efu}$)" if i == 0 else None
    label_TH = "Theis($K_H$)" if i == 0 else None
    label_ef = "extended Theis 3D" if i == 0 else None
    plt.plot(
        rad, head_Kefu[i], label=label_TG, color="C" + str(i), linestyle="--"
    )
    plt.plot(
```

(continues on next page)

(continued from previous page)

```
        rad, head_KH[i], label=label_TH, color="C" + str(i), linestyle=":"
    )
    plt.plot(rad, head_ef[i], label=label_ef, color="C" + str(i))
    time_ticks.append(head_ef[i][-1])

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
ylim = plt.gca().get_ylim()
plt.gca().set_xlim([0, rad[-1]])
ax2 = plt.gca().twinx()
ax2.set_yticks(time_ticks)
ax2.set_yticklabels(time_labels)
ax2.set_ylim(ylim)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.407 seconds)

The extended Theis solution for truncated power laws

We provide an extended theis solution, that incorporates the effectes of a heterogeneous conductivity field following a truncated power law. In addition, it incorporates the assumptions of the general radial flow model and provides an arbitrary flow dimension.

In the following this extended solution is compared to the standard theis solution for well flow. You can nicely see, that the extended solution represents a transition between the theis solutions for the well- and farfield-conductivity.

Reference: (not yet published)

```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_theis_tpl, theis
```

We use three time steps: 10s, 10min, 10h

```
time_labels = ["10 s", "10 min", "10 h"]
time = [10, 600, 36000]  # 10s, 10min, 10h
```

Radius from the pumping well should be in [0, 4].

```
rad = np.geomspace(0.05, 4)
```

Parameters of heterogeneity, storage and pumping rate.

```
var = 0.5  # variance of the log-conductivity
len_scale = 20.0  # upper bound for the length scale
hurst = 0.5  # hurst coefficient
KG = 1e-4  # the geometric mean of the conductivity
KH = KG * np.exp(-var / 2)  # the harmonic mean of the conductivity

S = 1e-4  # storage
rate = -1e-4  # pumping rate
```

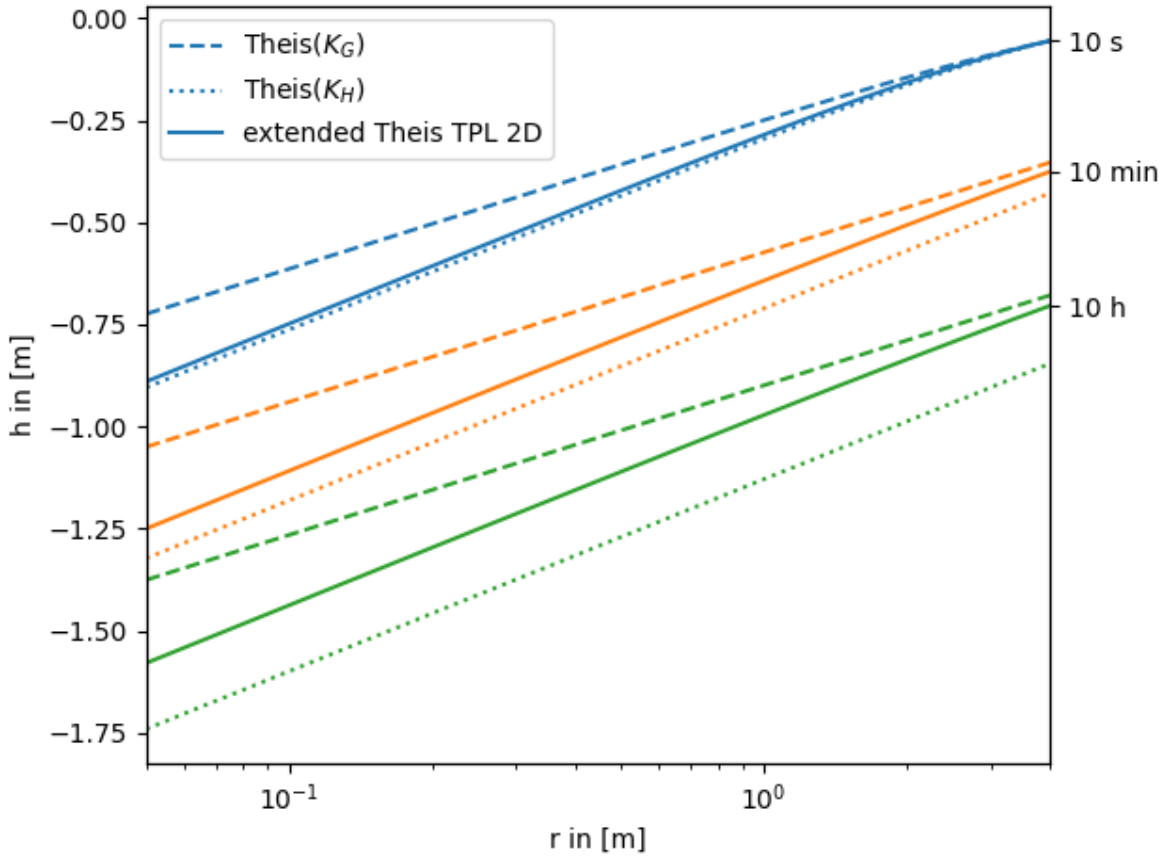
Now let's compare the extended Theis TPL solution to the classical solutions for the near and far field values of transmissivity.

```
head_KG = theis(time, rad, S, KG, rate)
head_KH = theis(time, rad, S, KH, rate)
head_ef = ext_theis_tpl(
    time=time,
    rad=rad,
    storage=S,
    cond_gmean=KG,
    len_scale=len_scale,
    hurst=hurst,
    var=var,
    rate=rate,
)
time_ticks = []
for i, step in enumerate(time):
    label_TG = "Theis($K_G$)" if i == 0 else None
    label_TH = "Theis($K_H$)" if i == 0 else None
    label_ef = "extended Theis TPL 2D" if i == 0 else None
    plt.plot(
        rad, head_KG[i], label=label_TG, color="C" + str(i), linestyle="--"
    )
```

(continues on next page)

(continued from previous page)

```
    plt.plot(
        rad, head_KH[i], label=label_TH, color="C" + str(i), linestyle=":"
    )
    plt.plot(rad, head_ef[i], label=label_ef, color="C" + str(i))
    time_ticks.append(head_ef[i][-1])

plt.xscale("log")
plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
ylim = plt.gca().get_ylim()
plt.gca().set_xlim([rad[0], rad[-1]])
ax2 = plt.gca().twinx()
ax2.set_yticks(time_ticks)
ax2.set_yticklabels(time_labels)
ax2.set_ylim(ylim)
plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.455 seconds)

The transient heterogeneous Neuman solution from 2004

We provide the transient pumping solution for the apparent transmissivity from Neuman 2004. This solution is build on the apparent transmissivity from Neuman 2004, which represents a mean drawdown in an ensemble of pumping tests in heterogeneous transmissivity fields following an exponential covariance.

In the following this solution is compared to the standard theis solution for well flow. You can nicely see, that the extended solution represents a transition between the theis solutions for the well- and farfield-conductivity.

Reference: Neuman 2004

```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import neuman2004, theis
```

We use three time steps: 10s, 10min, 10h

```
time_labels = ["10 s", "10 min", "10 h"]
time = [10, 600, 36000]  # 10s, 10min, 10h
```

Radius from the pumping well should be in [0, 4].

```
rad = np.geomspace(0.05, 4)
```

Parameters of heterogeneity, storage and pumping rate.

```
var = 0.5  # variance of the log-transmissivity
len_scale = 10.0  # correlation length of the log-transmissivity
TG = 1e-4  # the geometric mean of the transmissivity
TH = TG * np.exp(-var / 2.0)  # the harmonic mean of the transmissivity

S = 1e-4  # storativity
rate = -1e-4  # pumping rate
```

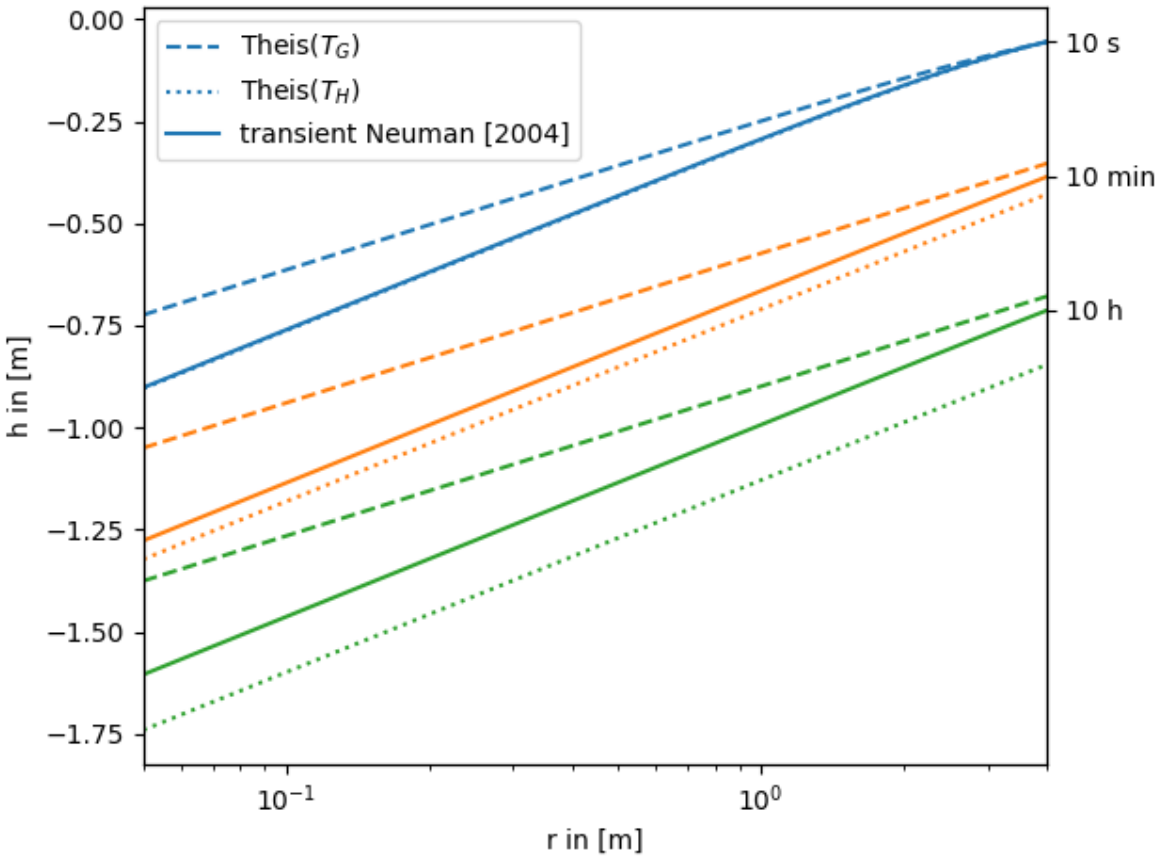
Now let's compare the apparent Neuman solution to the classical solutions for the near and far field values of transmissivity.

```
head_TG = theis(time, rad, S, TG, rate)
head_TH = theis(time, rad, S, TH, rate)
head_ef = neuman2004(
    time=time,
    rad=rad,
    trans_gmean=TG,
    var=var,
    len_scale=len_scale,
    storage=S,
    rate=rate,
)
time_ticks = []
for i, step in enumerate(time):
    label_TG = "Theis($T_G$)" if i == 0 else None
    label_TH = "Theis($T_H$)" if i == 0 else None
    label_ef = "transient Neuman [2004]" if i == 0 else None
    plt.plot(
        rad, head_TG[i], label=label_TG, color="C" + str(i), linestyle="--"
    )
    plt.plot(
        rad, head_TH[i], label=label_TH, color="C" + str(i), linestyle=":"
```

(continues on next page)

(continued from previous page)

```
    )
    plt.plot(rad, head_ef[i], label=label_ef, color="C" + str(i))
    time_ticks.append(head_ef[i][-1])

plt.xscale("log")
plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
ylim = plt.gca().get_ylim()
plt.gca().set_xlim([rad[0], rad[-1]])
ax2 = plt.gca().twinx()
ax2.set_yticks(time_ticks)
ax2.set_yticklabels(time_labels)
ax2.set_ylim(ylim)
plt.tight_layout()
plt.show()
```



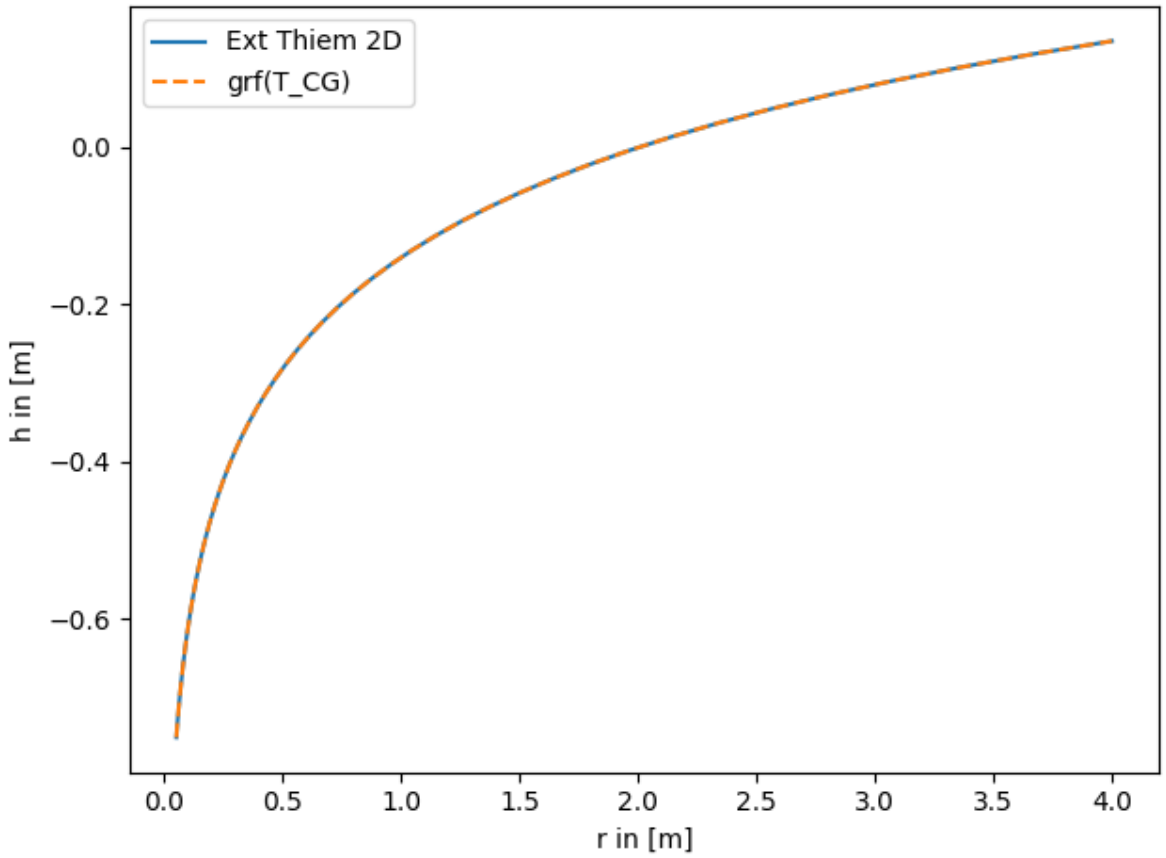
Total running time of the script: (0 minutes 0.392 seconds)

extended Thiem 2D vs. steady solution for coarse graining transmissivity

The extended Thiem 2D solutions is the analytical solution of the groundwater flow equation for the coarse graining transmissivity for pumping tests. Therefore the results should coincide.

References:

- Schneider & Attinger 2008
- Zech & Attinger 2016




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_grf_steady, ext_thiem_2d
from anaflow.tools.coarse_graining import T_CG

rad = np.geomspace(0.05, 4)  # radius from the pumping well in [0, 4]
r_ref = 2.0  # reference radius

var = 0.5  # variance of the log-transmissivity
len_scale = 10.0  # correlation length of the log-transmissivity
TG = 1e-4  # the geometric mean of the transmissivity

rate = -1e-4  # pumping rate

head1 = ext_thiem_2d(rad, r_ref, TG, var, len_scale, rate)
head2 = ext_grf_steady(
    rad, r_ref, T_CG, rate=rate, trans_gmean=TG, var=var, len_scale=len_scale
)
```

(continues on next page)

(continued from previous page)

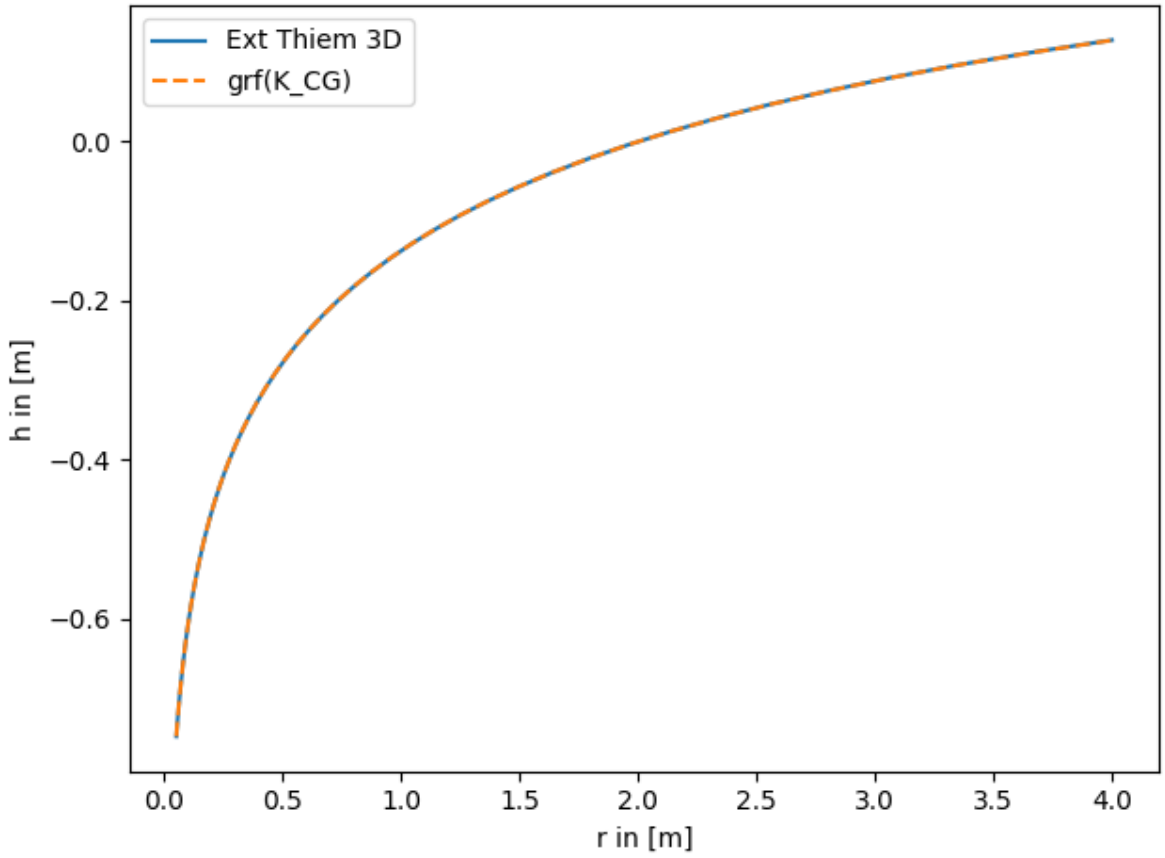
```
plt.plot(rad, head1, label="Ext Thiem 2D")
plt.plot(rad, head2, label="grf(T_CG)", linestyle="--")

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.180 seconds)

extended Thiem 3D vs. steady solution for coarse graining conductivity

The extended Thiem 3D solutions is the analytical solution of the groundwater flow equation for the coarse graining conductivity for pumping tests. Therefore the results should coincide.

Reference: Zech et. al. 2012




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_grf_steady, ext_thiem_3d
from anaflow.tools.coarse_graining import K_CG

rad = np.geomspace(0.05, 4)  # radius from the pumping well in [0, 4]
r_ref = 2.0  # reference radius
```

(continues on next page)

(continued from previous page)

```
var = 0.5  # variance of the log-transmissivity
len_scale = 10.0  # correlation length of the log-transmissivity
KG = 1e-4  # the geometric mean of the transmissivity
anis = 0.7  # aniso ratio

rate = -1e-4  # pumping rate

head1 = ext_thiem_3d(rad, r_ref, KG, var, len_scale, anis, 1, rate)
head2 = ext_grf_steady(
    rad,
    r_ref,
    K_CG,
    rate=rate,
    cond_gmean=KG,
    var=var,
    len_scale=len_scale,
    anis=anis,
)

plt.plot(rad, head1, label="Ext Thiem 3D")
plt.plot(rad, head2, label="grf(K_CG)", linestyle="--")

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
plt.tight_layout()
plt.show()
```

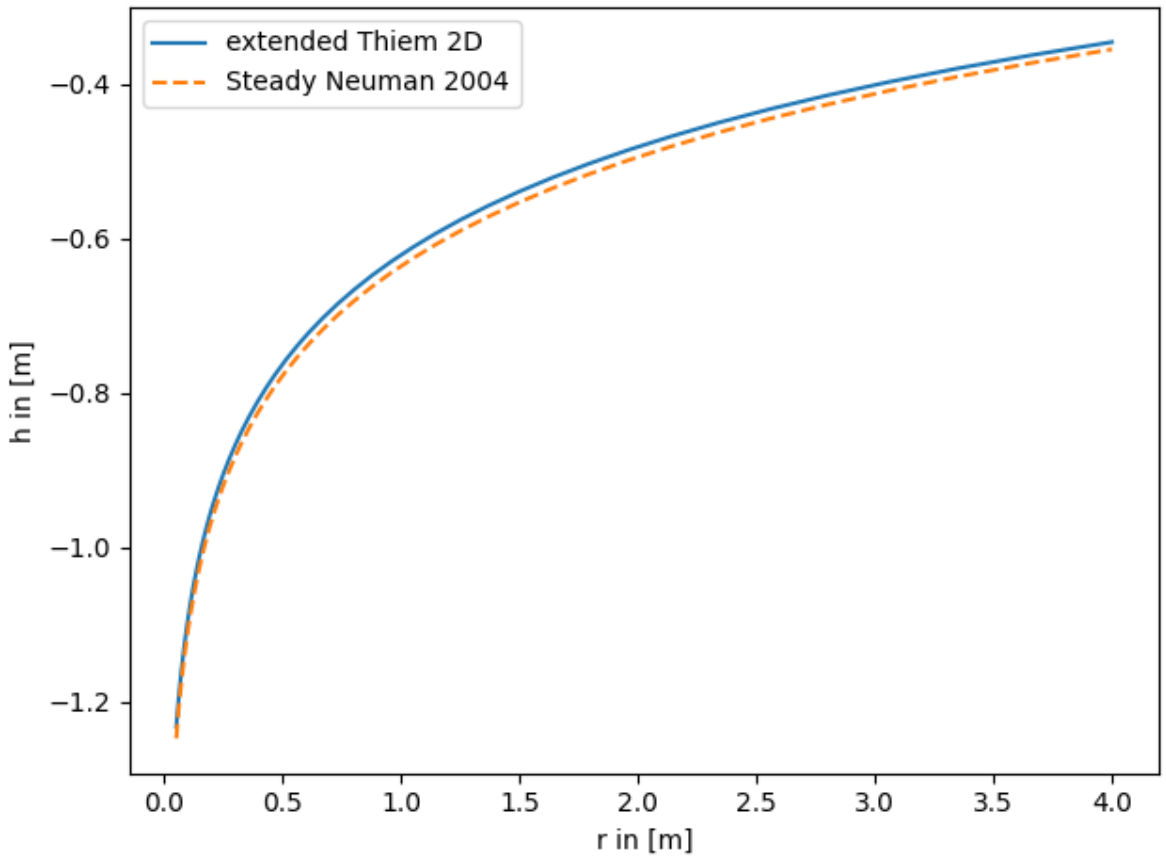
Total running time of the script: (0 minutes 0.702 seconds)

extended Thiem 2D vs. steady solution for apparent transmissivity from Neuman

Both, the extended Thiem and the Neuman solution, represent an effective steady drawdown in a heterogeneous aquifer. In both cases the heterogeneity is represented by two point statistics, characterized by mean, variance and length scale of the log transmissivity field. Therefore these approaches should lead to similar results.

References:

- Neuman 2004
- Zech & Attinger 2016




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_thiem_2d, neuman2004_steady

rad = np.geomspace(0.05, 4)  # radius from the pumping well in [0, 4]
r_ref = 30.0  # reference radius

var = 0.5  # variance of the log-transmissivity
len_scale = 10.0  # correlation length of the log-transmissivity
TG = 1e-4  # the geometric mean of the transmissivity

rate = -1e-4  # pumping rate

head1 = ext_thiem_2d(rad, r_ref, TG, var, len_scale, rate)
head2 = neuman2004_steady(rad, r_ref, TG, var, len_scale, rate)

plt.plot(rad, head1, label="extended Thiem 2D")
plt.plot(rad, head2, label="Steady Neuman 2004", linestyle="--")

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
plt.tight_layout()
plt.show()
```

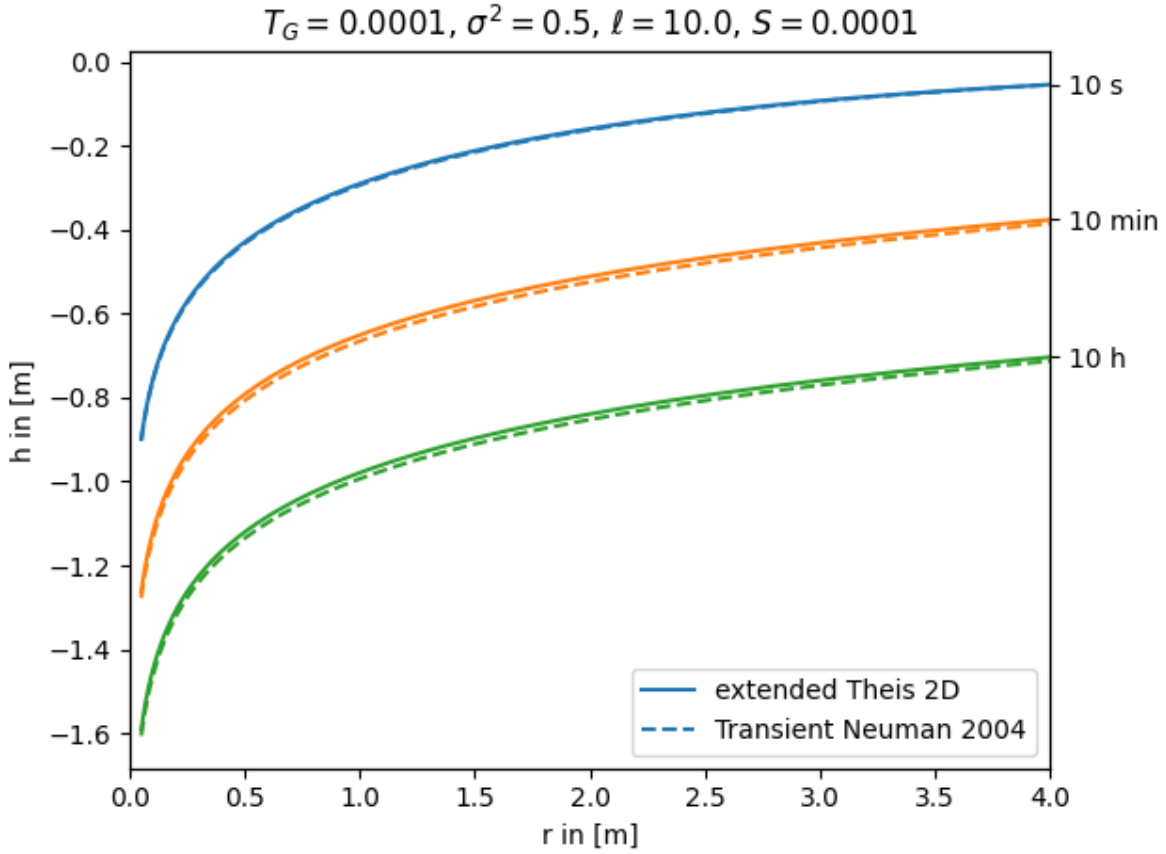
Total running time of the script: (0 minutes 0.546 seconds)

extended Theis 2D vs. transient solution for apparent transmissivity from Neuman

Both, the extended Theis and the Neuman solution, represent an effective transient drawdown in a heterogeneous aquifer. In both cases the heterogeneity is represented by two point statistics, characterized by mean, variance and length scale of the log transmissivity field. Therefore these approaches should lead to similar results.

References:

- Neuman 2004
- Zech et. al. 2016




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_theis_2d, neuman2004

time_labels = ["10 s", "10 min", "10 h"]
time = [10, 600, 36000]  # 10s, 10min, 10h

rad = np.geomspace(0.05, 4)  # radius from the pumping well in [0, 4]

TG = 1e-4  # the geometric mean of the transmissivity
var = 0.5  # correlation length of the log-transmissivity
len_scale = 10.0  # variance of the log-transmissivity

S = 1e-4  # storativity
rate = -1e-4  # pumping rate
```

(continues on next page)

(continued from previous page)

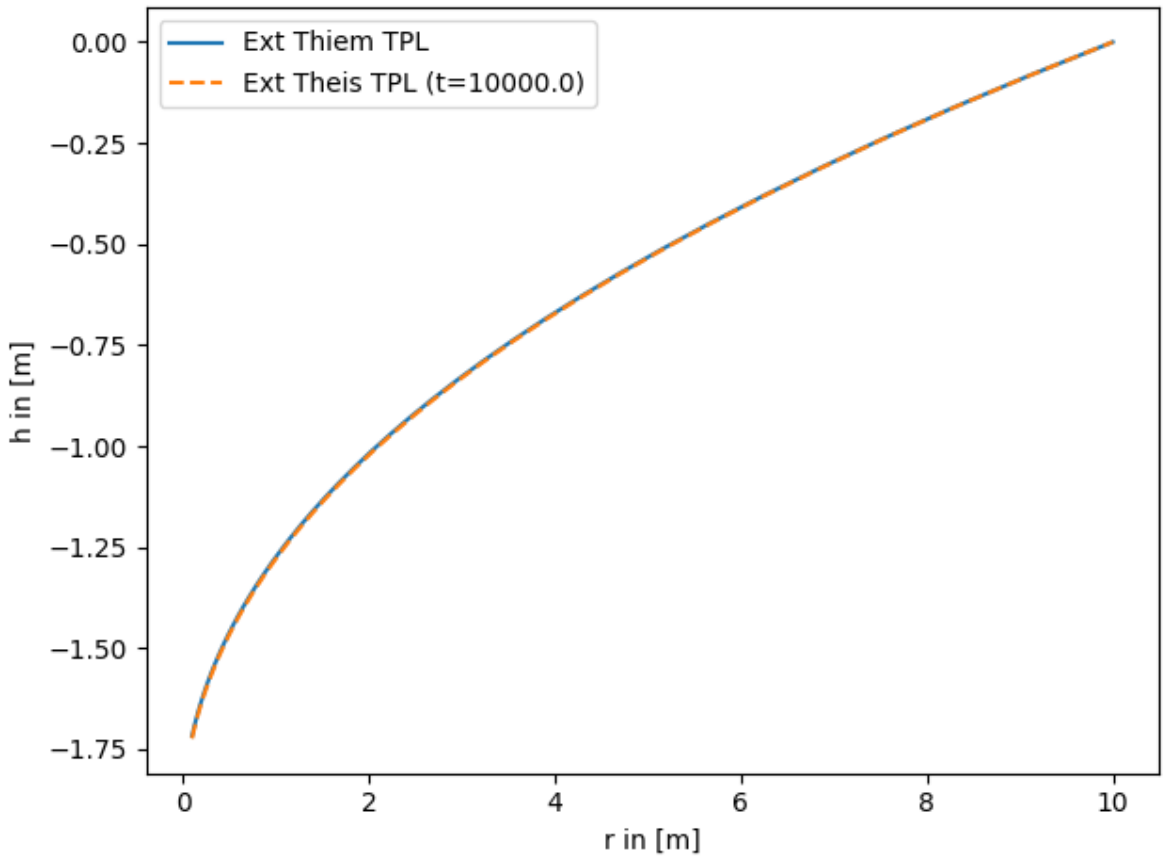
```
head1 = ext_theis_2d(time, rad, S, TG, var, len_scale, rate)
head2 = neuman2004(time, rad, S, TG, var, len_scale, rate)
time_ticks = []
for i, step in enumerate(time):
    label1 = "extended Theis 2D" if i == 0 else None
    label2 = "Transient Neuman 2004" if i == 0 else None
    plt.plot(rad, head1[i], label=label1, color="C" + str(i))
    plt.plot(rad, head2[i], label=label2, color="C" + str(i), linestyle="--")
    time_ticks.append(head1[i][-1])

plt.title(
    "$T_G={}$, $\sigma^2={}$, $\ell={}$, $S={}$".format(TG, var, len_scale, S)
)
plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
ylim = plt.gca().get_ylim()
plt.gca().set_xlim([0, rad[-1]])
ax2 = plt.gca().twinx()
ax2.set_yticks(time_ticks)
ax2.set_yticklabels(time_labels)
ax2.set_ylim(ylim)
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.373 seconds)

Convergence of the extended Theis solutions for truncated power laws

Here we set an outer boundary to the transient solution, so this condition coincides with the references head of the steady solution.

Reference: (not yet published)




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_theis_tpl, ext_thiem_tpl

time = 1e4  # time point for steady state
rad = np.geomspace(0.1, 10)  # radius from the pumping well in [0, 4]
r_ref = 10.0  # reference radius

KG = 1e-4  # the geometric mean of the transmissivity
len_scale = 5.0  # correlation length of the log-transmissivity
hurst = 0.5  # hurst coefficient
var = 0.5  # variance of the log-transmissivity
dim = 1.5  # using a fractional dimension

S = 1e-4  # storativity
rate = -1e-4  # pumping rate

head1 = ext_thiem_tpl(
    rad, r_ref, KG, len_scale, hurst, var, dim=dim, rate=rate
)
head2 = ext_theis_tpl(
    time, rad, S, KG, len_scale, hurst, var, dim=dim, rate=rate, r_bound=r_ref
)

plt.plot(rad, head1, label="Ext Thiem TPL")
plt.plot(rad, head2, label="Ext Theis TPL (t={})".format(time), linestyle="--")
```

(continues on next page)

(continued from previous page)

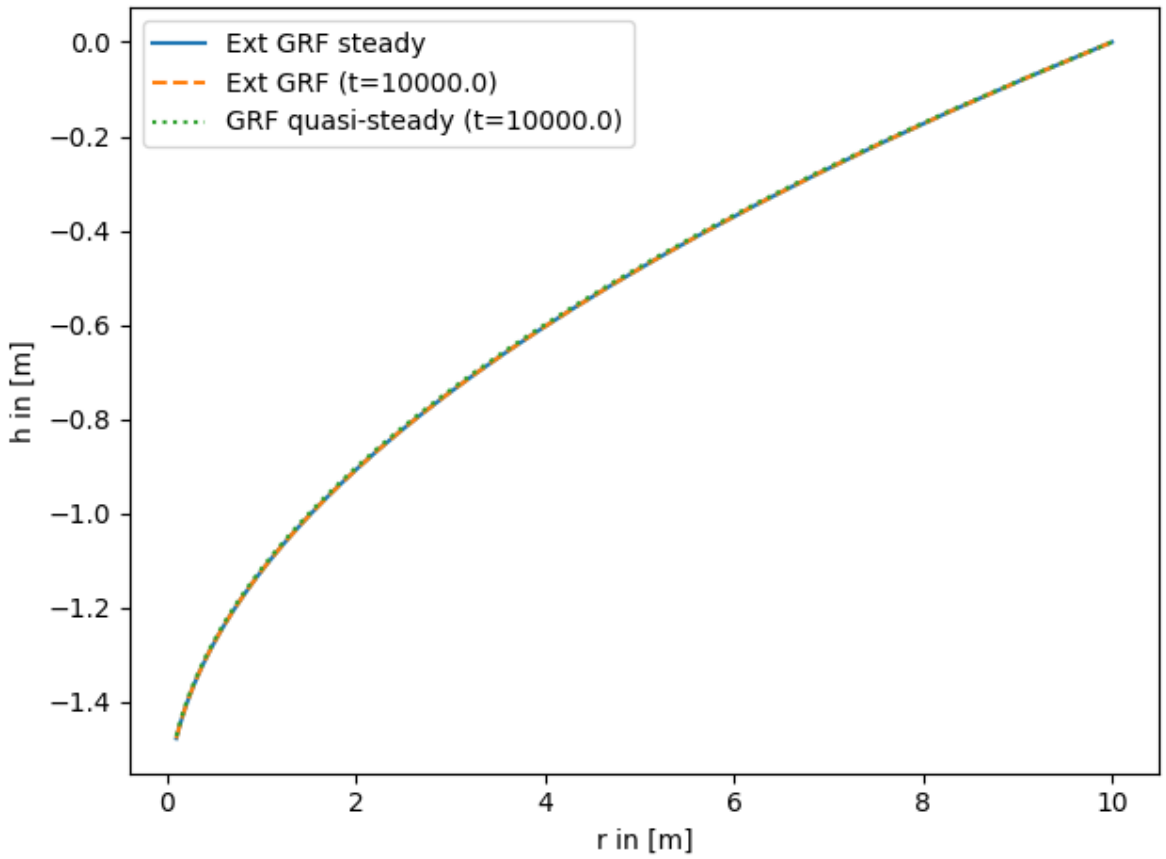
```
plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.632 seconds)

Convergence of the general radial flow model (GRF)

The GRF model introduces an arbitrary flow dimension and was presented to analyze groundwater flow in rock formations. In the following we compare the bounded transient solution for late times, the unbounded quasi steady solution and the steady state.

Reference: Barker 1988




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_grf, ext_grf_steady, grf

time = 1e4  # time point for steady state
rad = np.geomspace(0.1, 10)  # radius from the pumping well in [0, 4]
r_ref = 10.0  # reference radius

K = 1e-4  # the geometric mean of the transmissivity
dim = 1.5  # using a fractional dimension
```

(continues on next page)

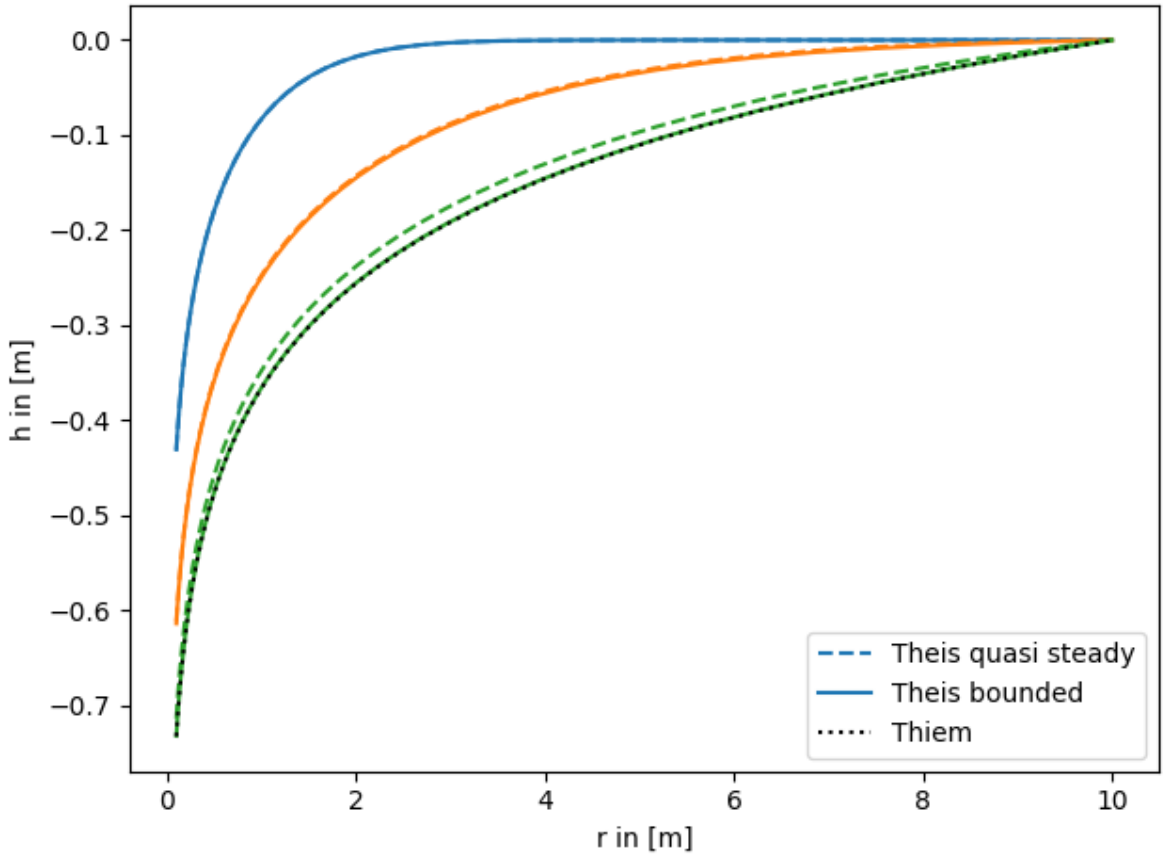
(continued from previous page)

```
rate = -1e-4  # pumping rate

head1 = ext_grf_steady(rad, r_ref, K, dim=dim, rate=rate)
head2 = ext_grf(time, rad, [1e-4], [K], [0, r_ref], dim=dim, rate=rate)
head3 = grf(time, rad, 1e-4, K, dim=dim, rate=rate)
head3 -= head3[-1]  # quasi-steady

plt.plot(rad, head1, label="Ext GRF steady")
plt.plot(rad, head2, label="Ext GRF (t={})".format(time), linestyle="--")
plt.plot(
    rad, head3, label="GRF quasi-steady (t={})".format(time), linestyle=":"
)

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.190 seconds)

Quasi steady convergence

The quasi steady is reached, when the radial shape of the drawdown in not changing anymore.



```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import theis, thiem

time = [10, 100, 1000]
rad = np.geomspace(0.1, 10)
r_ref = 10.0

head_ref = theis(
    time,
    np.full_like(rad, r_ref),
    storage=1e-3,
    transmissivity=1e-4,
    rate=-1e-4,
)
head1 = (
    theis(time, rad, storage=1e-3, transmissivity=1e-4, rate=-1e-4) - head_ref
)
head2 = theis(
    time, rad, storage=1e-3, transmissivity=1e-4, rate=-1e-4, r_bound=r_ref
)
head3 = thiem(rad, r_ref, transmissivity=1e-4, rate=-1e-4)

for i, step in enumerate(time):
    label_1 = "Theis quasi steady" if i == 0 else None
    label_2 = "Theis bounded" if i == 0 else None
    plt.plot(rad, head1[i], label=label_1, color="C" + str(i), linestyle="--")
    plt.plot(rad, head2[i], label=label_2, color="C" + str(i))

plt.plot(rad, head3, label="Thiem", color="k", linestyle=":")

plt.xlabel("r in [m]")
plt.ylabel("h in [m]")
plt.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.212 seconds)

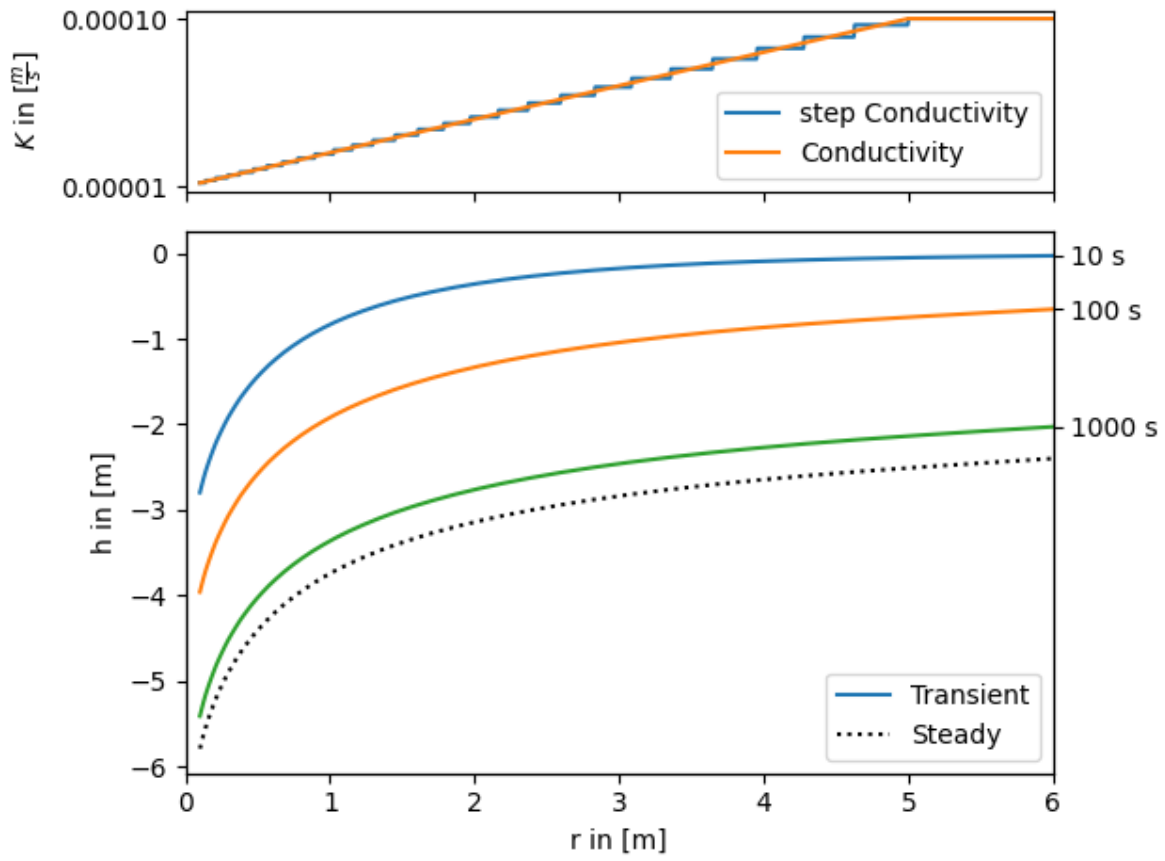
Self defined radial conductivity or transmissivity

All heterogeneous solutions of AnaFlow are derived by calculating an equivalent step function of a radial symmetric transmissivity resp. conductivity function.

The following code shows how to apply this workflow to a self defined transmissivity function. The function in use represents a linear transition from a local to a far field value of transmissivity within a given range.

The step function is calculated as the harmonic mean within given bounds, since the groundwater flow under a pumping condition is perpendicular to the different annular regions of transmissivity.

Reference: (not yet published)




```
import matplotlib.gridspec as gridspec
import numpy as np
from matplotlib import pyplot as plt

from anaflow import ext_grf, ext_grf_steady
from anaflow.tools import annular_hmean, specialrange_cut, step_f


def cond(rad, K_far, K_well, len_scale):
    """Conductivity with linear increase from K_well to K_far."""
    return np.minimum(np.abs(rad) / len_scale, 1.0) * (K_far - K_well) + K_well


time_labels = ["10 s", "100 s", "1000 s"]
time = [10, 100, 1000]
rad = np.geomspace(0.1, 6)

K_well = 1e-5
K_far = 1e-4
len_scale = 5.0
dim = 1.5

rate = -1e-4
S = 1e-4

cut_off = len_scale
parts = 30
r_well = 0.0
```

(continues on next page)

(continued from previous page)

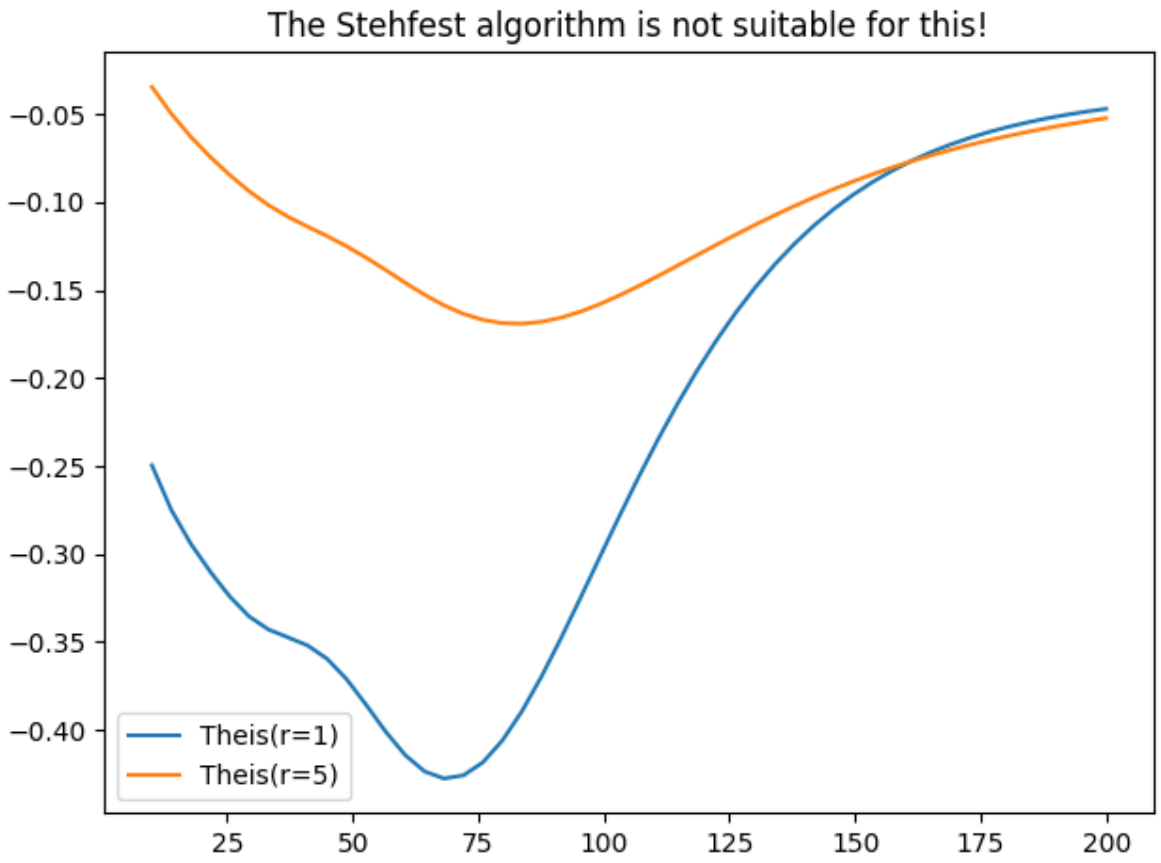
```
r_bound = 50.0

# calculate a disk-distribution of "trans" by calculating harmonic means
R_part = specialrange_cut(r_well, r_bound, parts, cut_off)
K_part = annular_hmean(
    cond, R_part, ann_dim=dim, K_far=K_far, K_well=K_well, len_scale=len_scale
)
S_part = np.full_like(K_part, S)
# calculate transient and steady heads
head1 = ext_grf(time, rad, S_part, K_part, R_part, dim=dim, rate=rate)
head2 = ext_grf_steady(
    rad,
    r_bound,
    cond,
    dim=dim,
    rate=-1e-4,
    K_far=K_far,
    K_well=K_well,
    len_scale=len_scale,
)

# plotting
gs = gridspec.GridSpec(2, 1, height_ratios=[1, 3])
ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1], sharex=ax1)
time_ticks = []
for i, step in enumerate(time):
    label = "Transient" if i == 0 else None
    ax2.plot(rad, head1[i], label=label, color="C" + str(i))
    time_ticks.append(head1[i][-1])

ax2.plot(rad, head2, label="Steady", color="k", linestyle=":")

rad_lin = np.linspace(rad[0], rad[-1], 1000)
ax1.plot(rad_lin, step_f(rad_lin, R_part, K_part), label="step Conductivity")
ax1.plot(
    rad_lin, cond(rad_lin, K_far, K_well, len_scale), label="Conductivity"
)
ax1.set_yticks([K_well, K_far])
ax1.set_ylabel(r"$K$ in $[\frac{m}{s}]$")
plt.setp(ax1.get_xticklabels(), visible=False)
ax1.legend()
ax2.set_xlabel("r in [m]")
ax2.set_ylabel("h in [m]")
ax2.legend()
ax2.set_xlim([0, rad[-1]])
ax3 = ax2.twinx()
ax3.set_yticks(time_ticks)
ax3.set_yticklabels(time_labels)
ax3.set_ylim(ax2.get_ylim())

plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.465 seconds)

Interval pumping

Another case often discussed in literatur is interval pumping, where the pumping is just done in a certain time frame.

Unfortunatly the Stehfest algorithm is not suitable for this kind of solution, which is demonstrated in the following script.




```
import numpy as np
from matplotlib import pyplot as plt

from anaflow import theis

time = np.linspace(10, 200)
rad = [1, 5]

# Q(t) = Q * characteristic([0, a])
lap_kwargs = {"cond": 3, "cond_kw": {"a": 100}}

head = theis(
    time=time,
    rad=rad,
    storage=1e-4,
    transmissivity=1e-4,
    rate=-1e-4,
    lap_kwargs=lap_kwargs,
)

for i, step in enumerate(rad):
```

(continues on next page)

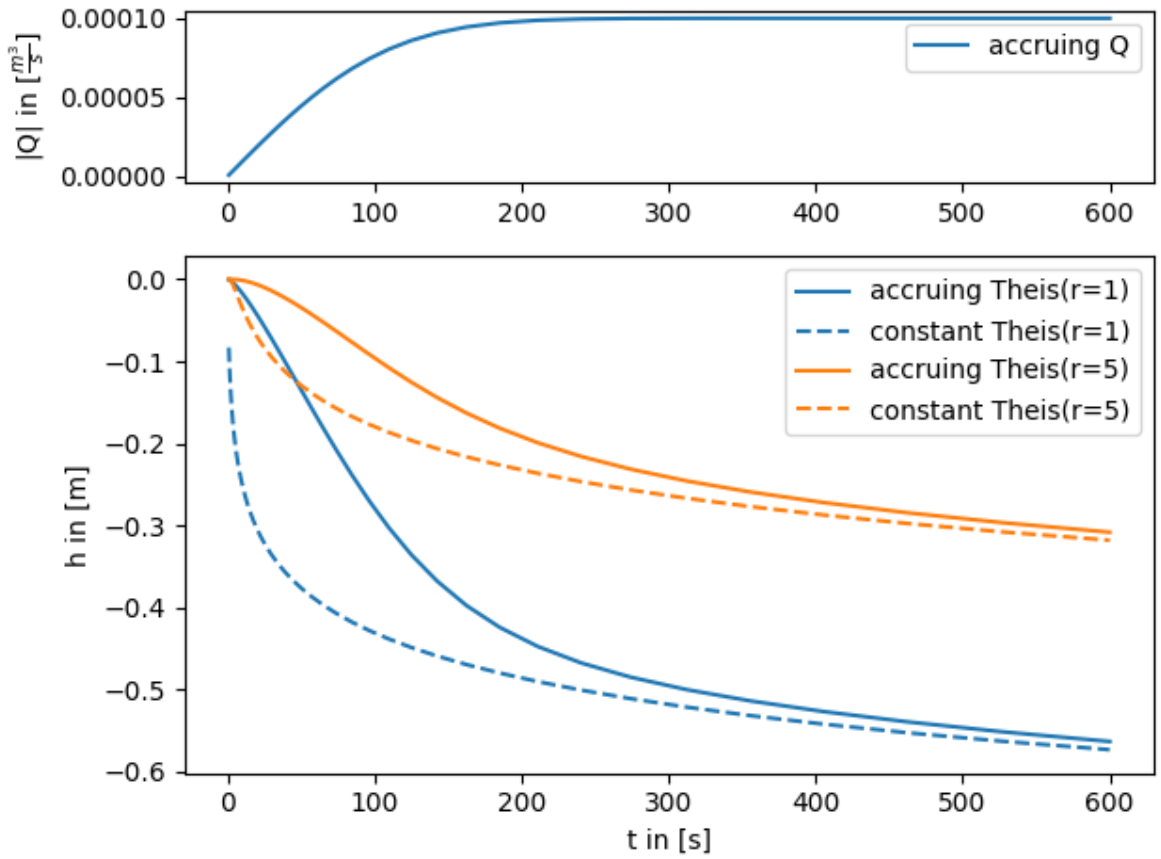
(continued from previous page)

```
    plt.plot(time, head[:, i], label="Theis(r={})".format(step))

plt.title("The Stehfest algorithm is not suitable for this!")
plt.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.215 seconds)

Accruing pumping rate

AnaFlow provides different representations for the pumping condition. One is an accruing pumping rate represented by the error function. This could be interpreted as that the water pump needs a certain time to reach its constant rate state.




```
import matplotlib.gridspec as gridspec
import numpy as np
from matplotlib import pyplot as plt
from scipy.special import erf

from anaflow import theis

time = np.geomspace(1, 600)
rad = [1, 5]

# Q(t) = Q * erf(t / a)
a = 120
```

(continues on next page)

(continued from previous page)

```
lap_kwargs = {"cond": 4, "cond_kw": {"a": a}}

head1 = theis(
    time=time,
    rad=rad,
    storage=1e-4,
    transmissivity=1e-4,
    rate=-1e-4,
    lap_kwargs=lap_kwargs,
)
head2 = theis(
    time=time,
    rad=rad,
    storage=1e-4,
    transmissivity=1e-4,
    rate=-1e-4,
)
gs = gridspec.GridSpec(2, 1, height_ratios=[1, 3])
ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1], sharex=ax1)

for i, step in enumerate(rad):
    ax2.plot(
        time,
        head1[:, i],
        color="C" + str(i),
        label="accruing Theis(r={})".format(step),
    )
    ax2.plot(
        time,
        head2[:, i],
        color="C" + str(i),
        label="constant Theis(r={})".format(step),
        linestyle="--",
    )
ax1.plot(time, 1e-4 * erf(time / a), label="accruing Q")
ax2.set_xlabel("t in [s]")
ax2.set_ylabel("h in [m]")
ax1.set_ylabel(r"|Q| in [$\frac{m^3}{s}$]")
ax1.legend()
ax2.legend()
plt.tight_layout()
plt.show()
```

Total running time of the script: (0 minutes 0.287 seconds)

CHAPTER 3

ANAFLOW API

3.1 Purpose

Anaflow provides several analytical and semi-analytical solutions for the groundwater-flow-equation.

3.2 Subpackages

flow	Anaflow subpackage providing flow-solutions for the groundwater flow equation.
tools	Anaflow subpackage providing miscellaneous tools.

anaflow.flow

Anaflow subpackage providing flow-solutions for the groundwater flow equation.

Subpackages

laplace	Anaflow subpackage providing flow solutions in laplace space.

anaflow.flow.laplace

Anaflow subpackage providing flow solutions in laplace space.

The following functions are provided

grf_laplace(s[, rad, S_part, K_part, ...])	The extended GRF-model for transient flow in Laplace-space.

anaflow.flow.laplace.grf_laplace

grf_laplace(*s*, *rad=None*, *S_part=None*, *K_part=None*, *R_part=None*, *dim=2*, *lat_ext=1.0*, *rate=None*, *K_well=None*, *cut_off_prec=1e-20*, *cond=0*, *cond_kw=None*)

The extended GRF-model for transient flow in Laplace-space.

The General Radial Flow (GRF) Model allowes fractured dimensions for transient flow under a pumping condition in a confined aquifer. The solutions assumes concentric annuli around the pumpingwell, where each annulus has its own conductivity and storativity value.

Parameters

- **s** (numpy.ndarray) – Array with all Laplace-space-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **S_part** (numpy.ndarray of length N) – Given storativity values for each disk
- **K_part** (numpy.ndarray of length N) – Given conductivity values for each disk
- **R_part** (numpy.ndarray of length N+1) – Given radii separating the disks as well as starting- and endpoints
- **dim** (float) – Flow dimension. Default: 3
- **lat_ext** (float) – The lateral extend of the flow-domain, used in *L^(3-dim)*. Default: 1
- **rate** (float) – Pumpingrate at the well
- **K_well** (float, optional) – Conductivity at the well. Default: `K_part[0]`
- **cut_off_prec** (float, optional) – Define a cut-off precision for the calculation to select the disks included in the calculation. Default `1e-20`
- **cond** (int, optional) –

  Type of the pumping condition:

  – 0 : constant

  – 1 : periodic (needs “w” as cond_kw)

  – 2 : slug (rate will be interpreted as slug-volume)

  – 3 : interval (needs “t” as cond_kw)

  – callable: laplace-transformation of the transient pumping-rate

  Default: 0
- **cond_kw** (dict optional) – Keyword args for the pumping condition. Default: None

Returns

grf_laplace – Array with all values in laplace-space

Return type

numpy.ndarray

Examples

```
>>> grf_laplace([5,10],[1,2,3],[1e-3,1e-3],[1e-3,2e-3],[0,2,10], 2, 1, -1)
array([[-2.71359196e+00, -1.66671965e-01, -2.82986917e-02],
       [-4.58447458e-01, -1.12056319e-02, -9.85673855e-04]])
```

Solutions

Homogeneous

Solutions for homogeneous aquifers

thiem(rad, r_ref, transmissivity[, rate, h_ref])	The Thiem solution.
theis(time, rad, storage, transmissivity[, ...])	The Theis solution.
grf(time, rad, storage, conductivity[, dim, ...])	The general radial flow (GRF) model for a pumping test.

anaflow.flow.thiem

thiem(*rad*, *r_ref*, *transmissivity*, *rate=-0.0001*, *h_ref=0.0*)

The Thiem solution.

The Thiem solution for steady-state flow under a pumping condition in a confined and homogeneous aquifer. This solution was presented in [Thiem1906].

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated.
- **r_ref** (float) – Reference radius with known head (see *h_ref*).
- **transmissivity** (float) – Transmissivity of the aquifer.
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_ref** (float, optional) – Reference head at the reference-radius *r_ref*. Default: `0.0`

Returns

head – Array with all heads at the given radii.

Return type

numpy.ndarray

References

Notes

The parameters `rad`, `r_ref` and `transmissivity` will be checked for positivity. If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

Examples

```
>>> thiem([1,2,3], 10, 0.001, -0.001)
array([-0.3664678 , -0.25615   , -0.19161822])
```

anaflow.flow.theis

theis(*time*, *rad*, *storage*, *transmissivity*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *struc_grid=True*, *lap_kwargs=None*)

The Theis solution.

The Theis solution for transient flow under a pumping condition in a confined and homogeneous aquifer. This solution was presented in [Theis35].

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **storage** (float) – Storage coefficient of the aquifer.
- **conductivity** (float) – Conductivity of the aquifer.
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (float, optional) – Inner radius of the pumping-well. Default: `0.0`
- **r_bound** (float, optional) – Radius of the outer boundariy of the aquifer. Default: `np.inf`
- **h_bound** (float, optional) – Reference head at the outer boundary, as well as initial condition. Default: `0.0`
- **struc_grid** (bool, optional) – If this is set to `False`, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`
- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: None

Returns
head – Array with all heads at the given radii and time-points.

Return type
numpy.ndarray

References

anaflow.flow.grf

grf(*time*, *rad*, *storage*, *conductivity*, *dim=2*, *lat_ext=1.0*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *struc_grid=True*, *lap_kwargs=None*)

The general radial flow (GRF) model for a pumping test.

This solution was presented in [Barker88].

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated.
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated.
- **storage** (float) – Storage coefficient of the aquifer.
- **conductivity** (float) – Conductivity of the aquifer.
- **dim** (float, optional) – Fractional dimension of the aquifer. Default: `2.0`

- **lat_ext** (float, optional) – Lateral extend of the aquifer. Default: `1.0`
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (float, optional) – Inner radius of the pumping-well. Default: `0.0`
- **r_bound** (float, optional) – Radius of the outer boundary of the aquifer. Default: `np.inf`
- **h_bound** (float, optional) – Reference head at the outer boundary, as well as initial condition. Default: `0.0`
- **struc_grid** (bool, optional) – If this is set to "False", the "rad" and "time" array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`
- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: None

Returns

head – Array with all heads at the given radii and time-points.

Return type

numpy.ndarray

References

Heterogeneous

Solutions for heterogeneous aquifers

ext_thiem_2d(rad, r_ref, trans_gmean, var, ...)	The extended Thiem solution in 2D.
ext_thiem_3d(rad, r_ref, cond_gmean, var, ...)	The extended Thiem solution in 3D.
ext_thiem_tpl(rad, r_ref, cond_gmean, ...[, ...])	The extended Thiem solution for truncated power-law fields.
ext_thiem_tpl_3d(rad, r_ref, cond_gmean, ...)	The extended Theis solution for truncated power-law fields in 3D.
ext_theis_2d(time, rad, storage, ...[, ...])	The extended Theis solution in 2D.
ext_theis_3d(time, rad, storage, cond_gmean, ...)	The extended Theis solution in 3D.
ext_theis_tpl(time, rad, storage, ...[, ...])	The extended Theis solution for truncated power-law fields.
ext_theis_tpl_3d(time, rad, storage, ...[, ...])	The extended Theis solution for truncated power-law fields in 3D.
neuman2004(time, rad, storage, trans_gmean, ...)	The transient solution for the apparent transmissivity from [Neuman2004].
neuman2004_steady(rad, r_ref, trans_gmean, ...)	The steady solution for the apparent transmissivity from [Neuman2004].

anaflow.flow.ext_thiem_2d

ext_thiem_2d(*rad*, *r_ref*, *trans_gmean*, *var*, *len_scale*, *rate=-0.0001*, *h_ref=0.0*, *T_well=None*, *prop=1.6*)

The extended Thiem solution in 2D.

The extended Thiem solution for steady-state flow under a pumping condition in a confined aquifer. The type curve is describing the effective drawdown in a 2D statistical framework, where the transmissivity distribution is following a log-normal distribution with a gaussian correlation function. Presented in [Zech2013].

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **r_ref** (float) – Radius of the reference head.
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_ref** (float, optional) – Reference head at the reference-radius *r_ref*. Default: `0.0`
- **T_well** (float, optional) – Explicit transmissivity value at the well. Default: `None`
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: `1.6`

Returns

head – Array with all heads at the given radii.

Return type

numpy.ndarray

References

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

Examples

```
>>> ext_thiem_2d([1,2,3], 10, 0.001, 1, 10, -0.001)
array([-0.53084596, -0.35363029, -0.25419375])
```

anaflow.flow.ext_thiem_3d

ext_thiem_3d(*rad*, *r_ref*, *cond_gmean*, *var*, *len_scale*, *anis=1.0*, *lat_ext=1.0*, *rate=-0.0001*, *h_ref=0.0*, *K_well='KH'*, *prop=1.6*)

The extended Thiem solution in 3D.

The extended Thiem solution for steady-state flow under a pumping condition in a confined aquifer. The type curve is describing the effective drawdown in a 3D statistical framework, where the conductivity distribution is following a log-normal distribution with a gaussian correlation function and taking vertical anisotropy into account. Presented in [Zech2013].

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated

- **r_ref** (float) – Reference radius with known head (see *h_ref*)
- **cond_gmean** (float) – Geometric-mean conductivity.
- **var** (float) – Variance of the log-conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **anis** (float, optional) – Anisotropy-ratio of the vertical and horizontal corralation-lengths. Default: 1.0
- **lat_ext** (float, optional) – Lateral extend of the aquifer (thickness). Default: `1.0`
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_ref** (float, optional) – Reference head at the reference-radius *r_ref*. Default: `0.0`
- **K_well** (*string/float, optional*) – Explicit conductivity value at the well. One can choose between the harmonic mean (`"KH"`), the arithmetic mean (`"KA"`) or an arbitrary float value. Default: `"KH"`
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: `1.6`

Returns
head – Array with all heads at the given radii.

Return type
numpy.ndarray

References

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

Examples

```
>>> ext_thiem_3d([1,2,3], 10, 0.001, 1, 10, 1, 1, -0.001)
array([-0.48828026, -0.31472059, -0.22043022])
```

anaflow.flow.ext_thiem_tpl

ext_thiem_tpl(*rad*, *r_ref*, *cond_gmean*, *len_scale*, *hurst*, *var=None*, *c=1.0*, *dim=2.0*, *lat_ext=1.0*, *rate=-0.0001*, *h_ref=0.0*, *K_well='KH'*, *prop=1.6*)

The extended Thiem solution for truncated power-law fields.

The extended Theis solution for steady flow under a pumping condition in a confined aquifer. The type curve is describing the effective drawdown in a d-dimensional statistical framework, where the conductivity distribution is following a log-normal distribution with a truncated power-law correlation function build on superposition of gaussian modes.

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **r_ref** (float) – Reference radius with known head (see *h_ref*)
- **cond_gmean** (float) – Geometric-mean conductivity. You can also treat this as transmissivity by leaving 'lat_ext=1'.
- **len_scale** (float) – Corralation-length of log-conductivity.

- **hurst** (float) – Hurst coefficient of the TPL model. Should be in (0, 1).
- **var** (float) – Variance of the log-conductivity. If var is given, c will be calculated accordingly. Default: None
- **c** (float, optional) – Intensity of variation in the TPL model. Is overwritten if var is given. Default: 1.0
- **dim** (float, optional) – Dimension of space. Default: 2.0
- **lat_ext** (float, optional) –

  Lateral extend of the aquifer:

  – sqare-root of cross-section in 1D

  – thickness in 2D

  – meaningless in 3D

  Default: 1.0
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_ref** (float, optional) – Reference head at the reference-radius *r_ref*. Default: 0.0
- **K_well** (float, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns
head – Array with all heads at the given radii and time-points.

Return type
numpy.ndarray

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

---

anaflow.flow.ext_thiem_tpl_3d

ext_thiem_tpl_3d(*rad*, *r_ref*, *cond_gmean*, *len_scale*, *hurst*, *var=None*, *c=1.0*, *anis=1*, *lat_ext=1.0*, *rate=-0.0001*, *h_ref=0.0*, *K_well='KH'*, *prop=1.6*)

The extended Theis solution for truncated power-law fields in 3D.

The extended Theis solution for transient flow under a pumping condition in a confined aquifer with anisotropy in 3D. The type curve is describing the effective drawdown in a 3-dimensional statistical framework, where the conductivity distribution is following a log-normal distribution with a truncated power-law correlation function build on superposition of gaussian modes.

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **storage** (float) – Storage of the aquifer.
- **cond_gmean** (float) – Geometric-mean conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.

- **hurst** (float) – Hurst coefficient of the TPL model. Should be in (0, 1).
- **var** (float) – Variance of the log-conductivity. If var is given, c will be calculated accordingly. Default: None
- **c** (float, optional) – Intensity of variation in the TPL model. Is overwritten if var is given. Default: 1.0
- **anis** (float, optional) – Anisotropy-ratio of the vertical and horizontal corralation-lengths. Default: 1.0
- **lat_ext** (float, optional) – Lateral extend of the aquifer (thickness). Default: 1.0
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (float, optional) – Radius of the pumping-well. Default: 0.0
- **r_bound** (float, optional) – Radius of the outer boundary of the aquifer. Default: np.inf
- **h_bound** (float, optional) – Reference head at the outer boundary as well as initial condition. Default: 0.0
- **K_well** (float, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6
- **far_err** (float, optional) – Relative error for the farfield transmissivity for calculating the cutoff-point of the solution. Default: 0.01
- **struc_grid** (bool, optional) – If this is set to False, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: True
- **parts** (int, optional) – Since the solution is calculated by setting the transmissivity to local constant values, one needs to specify the number of partitions of the transmissivity. Default: 30
- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: None

Returns

head – Array with all heads at the given radii and time-points.

Return type

numpy.ndarray

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

---

anaflow.flow.ext_theis_2d

ext_theis_2d(*time*, *rad*, *storage*, *trans_gmean*, *var*, *len_scale*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *T_well=None*, *prop=1.6*, *struc_grid=True*, *far_err=0.01*, *parts=30*, *lap_kwargs=None*)

The extended Theis solution in 2D.

The extended Theis solution for transient flow under a pumping condition in a confined aquifer. The type curve is describing the effective drawdown in a 2D statistical framework, where the transmissivity distribution is following a log-normal distribution with a gaussian correlation function.

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **storage** (float) – Storage of the aquifer.
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (float, optional) – Radius of the pumping-well. Default: `0.0`
- **r_bound** (float, optional) – Radius of the outer boundary of the aquifer. Default: `np.inf`
- **h_bound** (float, optional) – Reference head at the outer boundary as well as initial condition. Default: `0.0`
- **T_well** (float, optional) – Explicit transmissivity value at the well. Harmonic mean by default.
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: `1.6`
- **far_err** (float, optional) – Relative error for the farfield transmissivity for calculating the cutoff-point of the solution. Default: `0.01`
- **struc_grid** (bool, optional) – If this is set to `False`, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`
- **parts** (int, optional) – Since the solution is calculated by setting the transmissivity to local constant values, one needs to specify the number of partitions of the transmissivity. Default: `30`
- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: None

Returns

head – Array with all heads at the given radii and time-points.

Return type

numpy.ndarray

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

Examples

```
>>> ext_theis_2d([10,100], [1,2,3], 0.001, 0.001, 1, 10, -0.001)
array([[-0.33737576, -0.17400123, -0.09489812],
       [-0.58443489, -0.40847176, -0.31095166]])
```

anaflow.flow.ext_theis_3d

ext_theis_3d(*time*, *rad*, *storage*, *cond_gmean*, *var*, *len_scale*, *anis=1.0*, *lat_ext=1.0*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *K_well='KH'*, *prop=1.6*, *far_err=0.01*, *struc_grid=True*, *parts=30*, *lap_kwargs=None*)

The extended Theis solution in 3D.

The extended Theis solution for transient flow under a pumping condition in a confined aquifer. The type curve is describing the effective drawdown in a 3D statistical framework, where the transmissivity distribution is following a log-normal distribution with a gaussian correlation function and taking vertical anisotropy into account.

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **storage** (float) – Storage of the aquifer.
- **cond_gmean** (float) – Geometric-mean conductivity.
- **var** (float) – Variance of the log-conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **anis** (float, optional) – Anisotropy-ratio of the vertical and horizontal corralation-lengths. Default: 1.0
- **lat_ext** (float, optional) – Lateral extend of the aquifer (thickness). Default: `1.0`
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (float, optional) – Radius of the pumping-well. Default: `0.0`
- **r_bound** (float, optional) – Radius of the outer boundary of the aquifer. Default: `np.inf`
- **h_bound** (float, optional) – Reference head at the outer boundary as well as initial condition. Default: `0.0`
- **K_well** (float, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean (`"KH"`), the arithmetic mean (`"KA"`) or an arbitrary float value. Default: `"KH"`
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: `1.6`
- **far_err** (float, optional) – Relative error for the farfield transmissivity for calculating the cutoff-point of the solution. Default: `0.01`
- **struc_grid** (bool, optional) – If this is set to `False`, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`
- **parts** (int, optional) – Since the solution is calculated by setting the transmissivity to local constant values, one needs to specify the number of partitions of the transmissivity. Default: `30`

- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: None

Returns

head – Array with all heads at the given radii and time-points.

Return type

numpy.ndarray

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

Examples

```
>>> ext_theis_3d([10,100], [1,2,3], 0.001, 0.001, 1, 10, 1, 1, -0.001)
array([[-0.32756786, -0.16717569, -0.09141211],
       [-0.5416396 , -0.36982684, -0.27798614]])
```

anaflow.flow.ext_theis_tpl

ext_theis_tpl(*time*, *rad*, *storage*, *cond_gmean*, *len_scale*, *hurst*, *var=None*, *c=1.0*, *dim=2.0*, *lat_ext=1.0*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *K_well='KH'*, *prop=1.6*, *far_err=0.01*, *struc_grid=True*, *parts=30*, *lap_kwargs=None*)

The extended Theis solution for truncated power-law fields.

The extended Theis solution for transient flow under a pumping condition in a confined aquifer. The type curve is describing the effective drawdown in a d-dimensional statistical framework, where the conductivity distribution is following a log-normal distribution with a truncated power-law correlation function build on superposition of gaussian modes.

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **storage** (float) – Storage of the aquifer.
- **cond_gmean** (float) – Geometric-mean conductivity. You can also treat this as transmissivity by leaving 'lat_ext=1'.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **hurst** (float) – Hurst coefficient of the TPL model. Should be in (0, 1).
- **var** (float) – Variance of the log-conductivity. If var is given, c will be calculated accordingly. Default: None
- **c** (float, optional) – Intensity of variation in the TPL model. Is overwritten if var is given. Default: `1.0`
- **dim** (float, optional) – Dimension of space. Default: `2.0`
- **lat_ext** (float, optional) –

  Lateral extend of the aquifer:

  – sqare-root of cross-section in 1D

  – thickness in 2D

– meaningless in 3D

Default: `1.0`

- **rate** (`float`, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (`float`, optional) – Radius of the pumping-well. Default: `0.0`
- **r_bound** (`float`, optional) – Radius of the outer boundary of the aquifer. Default: `np.inf`
- **h_bound** (`float`, optional) – Reference head at the outer boundary as well as initial condition. Default: `0.0`
- **K_well** (`float`, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean (`"KH"`), the arithmetic mean (`"KA"`) or an arbitrary float value. Default: `"KH"`
- **prop** (`float`, optional) – Proportionality factor used within the upscaling procedure. Default: `1.6`
- **far_err** (`float`, optional) – Relative error for the farfield transmissivity for calculating the cutoff-point of the solution. Default: `0.01`
- **struc_grid** (`bool`, optional) – If this is set to `False`, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`
- **parts** (`int`, optional) – Since the solution is calculated by setting the transmissivity to local constant values, one needs to specify the number of partitions of the transmissivity. Default: `30`
- **lap_kwargs** (`dict` or `None` optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: `None`

Returns
head – Array with all heads at the given radii and time-points.

Return type
`numpy.ndarray`

Notes

If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

---

anaflow.flow.ext_theis_tpl_3d

ext_theis_tpl_3d(*time*, *rad*, *storage*, *cond_gmean*, *len_scale*, *hurst*, *var=None*, *c=1.0*, *anis=1*, *lat_ext=1.0*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *K_well='KH'*, *prop=1.6*, *far_err=0.01*, *struc_grid=True*, *parts=30*, *lap_kwargs=None*)

The extended Theis solution for truncated power-law fields in 3D.

The extended Theis solution for transient flow under a pumping condition in a confined aquifer with anisotropy in 3D. The type curve is describing the effective drawdown in a 3-dimensional statistical framework, where the conductivity distribution is following a log-normal distribution with a truncated power-law correlation function build on superposition of gaussian modes.

Parameters

- **time** (`numpy.ndarray`) – Array with all time-points where the function should be evaluated
- **rad** (`numpy.ndarray`) – Array with all radii where the function should be evaluated

- **storage** (float) – Storage of the aquifer.
- **cond_gmean** (float) – Geometric-mean conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **hurst** (float) – Hurst coefficient of the TPL model. Should be in (0, 1).
- **var** (float) – Variance of the log-conductivity. If var is given, c will be calculated accordingly. Default: None
- **c** (float, optional) – Intensity of variation in the TPL model. Is overwritten if var is given. Default: 1.0
- **anis** (float, optional) – Anisotropy-ratio of the vertical and horizontal corralation-lengths. Default: 1.0
- **lat_ext** (float, optional) – Lateral extend of the aquifer (thickness). Default: 1.0
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (float, optional) – Radius of the pumping-well. Default: 0.0
- **r_bound** (float, optional) – Radius of the outer boundary of the aquifer. Default: np.inf
- **h_bound** (float, optional) – Reference head at the outer boundary as well as initial condition. Default: 0.0
- **K_well** (float, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6
- **far_err** (float, optional) – Relative error for the farfield transmissivity for calculating the cutoff-point of the solution. Default: 0.01
- **struc_grid** (bool, optional) – If this is set to False, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: True
- **parts** (int, optional) – Since the solution is calculated by setting the transmissivity to local constant values, one needs to specify the number of partitions of the transmissivity. Default: 30
- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to lap_kwargs = {"method": "stehfest", "method_dict": None}. Default: None

Returns

head – Array with all heads at the given radii and time-points.

Return type

numpy.ndarray

Notes

If you want to use cartesian coordiantes, just use the formula r = sqrt(x**2 + y**2)

---

anaflow.flow.neuman2004

neuman2004(*time*, *rad*, *storage*, *trans_gmean*, *var*, *len_scale*, *rate=-0.0001*, *r_well=0.0*, *r_bound=inf*, *h_bound=0.0*, *struc_grid=True*, *parts=30*, *lap_kwargs=None*)

The transient solution for the apparent transmissivity from [Neuman2004].

This solution is build on the apparent transmissivity from Neuman 2004, which represents a mean drawdown in an ensemble of pumping tests in heterogeneous transmissivity fields following an exponential covariance. Presented in [Neuman2004].

Parameters

- **time** (`numpy.ndarray`) – Array with all time-points where the function should be evaluated.
- **rad** (`numpy.ndarray`) – Array with all radii where the function should be evaluated.
- **storage** (`float`) – Storage of the aquifer.
- **trans_gmean** (`float`) – Geometric-mean transmissivity.
- **var** (`float`) – Variance of log-transmissivity.
- **len_scale** (`float`) – Correlation-length of log-transmissivity.
- **rate** (`float`, optional) – Pumpingrate at the well. Default: -1e-4
- **r_well** (`float`, optional) – Radius of the pumping-well. Default: `0.0`
- **r_bound** (`float`, optional) – Radius of the outer boundary of the aquifer. Default: `np.inf`
- **h_bound** (`float`, optional) – Reference head at the outer boundary as well as initial condition. Default: `0.0`
- **struc_grid** (`bool`, optional) – If this is set to `False`, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`
- **parts** (`int`, optional) – Since the solution is calculated by setting the transmissivity to local constant values, one needs to specify the number of partitions of the transmissivity. Default: `30`
- **lap_kwargs** (`dict` or `None` optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to `lap_kwargs = {"method": "stehfest", "method_dict": None}`. Default: `None`

Returns

head – Array with all heads at the given radii and time-points.

Return type

`numpy.ndarray`

References

anaflow.flow.neuman2004_steady

neuman2004_steady(*rad*, *r_ref*, *trans_gmean*, *var*, *len_scale*, *rate=-0.0001*, *h_ref=0.0*)

The steady solution for the apparent transmissivity from [Neuman2004].

This solution is build on the apparent transmissivity from Neuman 1994, which represents a mean drawdown in an ensemble of pumping tests in heterogeneous transmissivity fields following an exponential covariance. Presented in [Neuman2004].

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **r_ref** (float) – Radius of the reference head.
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_ref** (float, optional) – Reference head at the reference-radius *r_ref*. Default: `0.0`

Returns
head – Array with all heads at the given radii.

Return type
numpy.ndarray

References

Extended GRF

The extended general radial flow model.

ext_grf(time, rad, S_part, K_part, R_part[, ...])	The extended "General radial flow" model for transient flow.
ext_grf_steady(rad, r_ref, conductivity[, ...])	The extended "General radial flow" model for steady flow.

anaflow.flow.ext_grf

ext_grf(*time*, *rad*, *S_part*, *K_part*, *R_part*, *dim=2*, *lat_ext=1.0*, *rate=-0.0001*, *h_bound=0.0*, *K_well=None*, *struc_grid=True*, *lap_kwargs=None*)

The extended “General radial flow” model for transient flow.

The general radial flow (GRF) model by Barker introduces an arbitrary dimension for radial groundwater flow. We introduced the possibility to define radial dependet conductivity and storage values.

This solution is based on the grf model presented in [Barker88].

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **S_part** (numpy.ndarray) – Given storativity values for each disk
- **K_part** (numpy.ndarray) – Given conductivity values for each disk
- **R_part** (numpy.ndarray) – Given radii separating the disks (including r_well and r_bound).
- **dim** (float, optional) – Fractional dimension of the aquifer. Default: `2.0`
- **lat_ext** (float, optional) – Lateral extend of the aquifer. Default: `1.0`
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_bound** (float, optional) – Reference head at the outer boundary *R_part[-1]*. Default: `0.0`
- **K_well** (float, optional) – Conductivity at the well. Default: `K_part[0]`

- **struc_grid** (bool, optional) – If this is set to False, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: True
- **lap_kwargs** (dict or None optional) – Dictionary for *get_lap_inv* containing *method* and *method_dict*. The default is equivalent to lap_kwargs = {"method": "stehfest", "method_dict": None}. Default: None

Returns
Array with all heads at the given radii and time-points.

Return type
numpy.ndarray

References

anaflow.flow.ext_grf_steady

ext_grf_steady(*rad*, *r_ref*, *conductivity*, *dim=2*, *lat_ext=1.0*, *rate=-0.0001*, *h_ref=0.0*, *arg_dict=None*, ***kwargs*)

The extended "General radial flow" model for steady flow.

The general radial flow (GRF) model by Barker introduces an arbitrary dimension for radial groundwater flow. We introduced the possibility to define radial dependet conductivity.

This solution is based on the grf model presented in [Barker88].

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **r_ref** (float) – Radius of the reference head.
- **conductivity** (float or callable) – Conductivity. Either callable function taking kwargs or float.
- **dim** (float, optional) – Fractional dimension of the aquifer. Default: 2.0
- **lat_ext** (float, optional) – Lateral extend of the aquifer. Default: 1.0
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_ref** (float, optional) – Reference head at the reference-radius *r_ref*. Default: 0.0
- **arg_dict** (dict or None, optional) – Keyword-arguments given as a dictionary that are forwarded to the conductivity function. Will be merged with **kwargs. This is designed for overlapping keywords in grf_steady and conductivity. Default: None
- ****kwargs** – Keyword-arguments that are forwarded to the conductivity function. Will be merged with arg_dict

Returns
Array with all heads at the given radii and time-points.

Return type
numpy.ndarray

References

anaflow.tools

Anaflow subpackage providing miscellaneous tools.

Subpackages

laplace	Anaflow subpackage providing functions concerning the laplace transformation.
mean	Anaflow subpackage providing several mean calculating routines.
special	Anaflow subpackage providing special functions.
coarse_graining	Anaflow subpackage providing helper functions related to coarse graining.

anaflow.tools.laplace

Anaflow subpackage providing functions concerning the laplace transformation.

The following functions are provided

get_lap(func[, arg_dict])	Callable Laplace transform.
get_lap_inv(func[, method, method_dict, ...])	Callable Laplace inversion.
lap_trans(func, phase[, arg_dict])	The laplace transform.
stehfest(func, time[, bound, arg_dict])	The stehfest-algorithm for numerical laplace inversion.

anaflow.tools.laplace.get_lap

get_lap(*func*, *arg_dict=None*, ***kwargs*)

Callable Laplace transform.

Get the Laplace transform of a given function as a callable function.

Parameters

- **func** (`callable`) – function that shall be transformed. The first argument needs to be the time-variable: `func(t, **kwargs)`

  func should be capable of taking numpy arrays as input for *s* and the first shape component of the output of *func* should match the shape of *s*.

- **arg_dict** (`dict` or `None`, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs` This is designed for overlapping keywords. Default: `None`

- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`.

Returns

The Laplace transformed of the given function.

Return type

`callable`

Raises

`ValueError` – If *func* is not callable.

anaflow.tools.laplace.get_lap_inv

get_lap_inv(*func*, *method='stehfest'*, *method_dict=None*, *arg_dict=None*, ***kwargs*)

Callable Laplace inversion.

Get the Laplace inversion of a given function as a callable function.

Parameters

- **func** (`callable`) – function in laplace-space that shall be inverted. The first argument needs to be the laplace-variable: `func(s, **kwargs)`

  func should be capable of taking numpy arrays as input for *s* and the first shape component of the output of *func* should match the shape of *s*.

- **method** (`str`) – Method that should be used to calculate the inverse. One can choose between

  – `"stehfest"`: for the stehfest algorithm

  Default: `"stehfest"`

- **method_dict** (`dict` or `None`, optional) – Keyword arguments for the used method.
- **arg_dict** (`dict` or `None`, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs` This is designed for overlapping keywords. Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`.

Returns
The Laplace inverse of the given function.

Return type
`callable`

Raises

- **ValueError** – If *func* is not callable.
- **ValueError** – If *method* is unknown.

anaflow.tools.laplace.lap_trans

lap_trans(*func*, *phase*, *arg_dict=None*, ***kwargs*)

The laplace transform.

Parameters

- **func** (`callable`) – function that shall be transformed. The first argument needs to be the time-variable: `func(s, **kwargs)`

  func should be capable of taking numpy arrays as input for *s* and the first shape component of the output of *func* should match the shape of *s*.

- **phase** (`float` or `numpy.ndarray`) – phase-points to evaluate the transformed function at
- **arg_dict** (`dict` or `None`, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs` This is designed for overlapping keywords in `stehfest` and `func`.Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns
Array with all evaluations in phase-space.

Return type
numpy.ndarray

Raises
ValueError – If *func* is not callable.

anaflow.tools.laplace.stehfest

stehfest(*func*, *time*, *bound=12*, *arg_dict=None*, ***kwargs*)

The stehfest-algorithm for numerical laplace inversion.

The Inversion was derivide in ''Stehfest 1970"[R1] and is given by the formula

$$f(t) = \frac{\ln 2}{t} \sum_{n=1}^{N} c_n \cdot \tilde{f}\left(n \cdot \frac{\ln 2}{t}\right)$$

$$c_n = (-1)^{n+\frac{N}{2}} \cdot \sum_{k=\lfloor \frac{n+1}{2} \rfloor}^{\min\left\{n, \frac{N}{2}\right\}} \frac{k^{\frac{N}{2}+1} \cdot \binom{2k}{k}}{\left(\frac{N}{2}-k\right)! \cdot (n-k)! \cdot (2k-n)!}$$

In the algorithm N corresponds to `bound`, $\tilde{f}$ to `func` and t to `time`.

Parameters

- **func** (callable) – function in laplace-space that shall be inverted. The first argument needs to be the laplace-variable: `func(s, **kwargs)`

  func should be capable of taking numpy arrays as input for *s* and the first shape component of the output of *func* should match the shape of *s*.
- **time** (float or numpy.ndarray) – time-points to evaluate the function at
- **bound** (int, optional) – Here you can specify the number of interations within this algorithm. Default: 12
- **arg_dict** (dict or None, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs` This is designed for overlapping keywords in `stehfest` and `func`.Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns
Array with all evaluations in Time-space.

Return type
numpy.ndarray

Raises
- **ValueError** – If *func* is not callable.
- **ValueError** – If *time* is not positive.
- **ValueError** – If *bound* is not positive.
- **ValueError** – If *bound* is not even.

References

Notes

The parameter `time` needs to be strictly positiv.

The algorithm gets unstable for `bound` values above 20.

Examples

```
>>> f = lambda x: x**-1
>>> stehfest(f, [1,10,100])
array([ 1.,  1.,  1.])
```

anaflow.tools.mean

Anaflow subpackage providing several mean calculating routines.

The following functions are provided

annular_fmean(func, val_arr, f_def, f_inv[, ...])	Calculating the annular generalized f-mean.
annular_amean(func, val_arr[, ann_dim, arg_dict])	Calculating the annular arithmetic mean.
annular_gmean(func, val_arr[, ann_dim, arg_dict])	Calculating the annular geometric mean.
annular_hmean(func, val_arr[, ann_dim, arg_dict])	Calculating the annular harmonic mean.
annular_pmean(func, val_arr[, p, ann_dim, ...])	Calculating the annular p-mean.

anaflow.tools.mean.annular_fmean

annular_fmean(*func*, *val_arr*, *f_def*, *f_inv*, *ann_dim=2*, *arg_dict=None*, ***kwargs*)

Calculating the annular generalized f-mean.

Calculating the annular generalized f-mean of a radial symmetric function for given consecutive radii defining annuli by the following formula

$$\varphi_i = f^{-1}\left(\frac{d}{r_{i+1}^d - r_i^d}\int_{r_i}^{r_{i+1}} r^{d-1}\cdot f(\varphi(r))\,dr\right)$$

Parameters

- **func** (`callable`) – Function that should be used (φ in the formula). The first argument needs to be the radial variable: `func(r, **kwargs)`
- **val_arr** (`numpy.ndarray`) – Radii defining the annuli.
- **ann_dim** (`float`, optional) – The dimension of the annuli. Default: `2.0`
- **f_def** (`callable`) – Function defining the f-mean.
- **f_inv** (`callable`) – Inverse of the function defining the f-mean.
- **arg_dict** (`dict` or `None`, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs`. This is designed for overlapping keywords in `annular_fmean` and `func`. Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns

Array with all calculated arithmetic means

Return type

`numpy.ndarray`

Raises

- `ValueError` – If *func* is not callable.
- `ValueError` – If *f_def* is not callable.
- `ValueError` – If *f_inv* is not callable.
- `ValueError` – If `val_arr` has less than 2 values.
- `ValueError` – If `val_arr` is not sorted in incresing order.

Notes

If the last value in val_arr is “inf”, the given function should provide a value for “inf” as input: `func(float("inf"))`

anaflow.tools.mean.annular_amean

annular_amean(*func*, *val_arr*, *ann_dim=2*, *arg_dict=None*, ***kwargs*)

Calculating the annular arithmetic mean.

Calculating the annular arithmetic mean of a radial symmetric function for given consecutive radii defining annuli by the following formula

$$\varphi_i = \frac{d}{r_{i+1}^d - r_i^d} \int_{r_i}^{r_{i+1}} r^{d-1} \cdot \varphi(r)\, dr$$

Parameters

- **func** (`callable`) – Function that should be used (φ in the formula). The first argument needs to be the radial variable: `func(r, **kwargs)`
- **val_arr** (`numpy.ndarray`) – Radii defining the annuli.
- **ann_dim** (`float`, optional) – The dimension of the annuli. Default: `2.0`
- **arg_dict** (`dict` or `None`, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs`. This is designed for overlapping keywords in `annular_amean` and `func`. Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns
Array with all calculated arithmetic means

Return type
`numpy.ndarray`

Raises

- `ValueError` – If *func* is not callable.
- `ValueError` – If `val_arr` has less than 2 values.
- `ValueError` – If `val_arr` is not sorted in incresing order.

Notes

If the last value in val_arr is “inf”, the given function should provide a value for “inf” as input: `func(float("inf"))`

anaflow.tools.mean.annular_gmean

annular_gmean(*func*, *val_arr*, *ann_dim=2*, *arg_dict=None*, ***kwargs*)

Calculating the annular geometric mean.

Calculating the annular geometric mean of a radial symmetric function for given consecutive radii defining annuli by the following formula

$$\varphi_i = \exp\left(\frac{d}{r_{i+1}^d - r_i^d} \int_{r_i}^{r_{i+1}} r^{d-1} \cdot \ln(\varphi(r))\, dr\right)$$

Parameters

- **func** (callable) – Function that should be used (φ in the formula). The first argument needs to be the radial variable: `func(r, **kwargs)`
- **val_arr** (numpy.ndarray) – Radii defining the annuli.
- **ann_dim** (float, optional) – The dimension of the annuli. Default: `2.0`
- **arg_dict** (dict or None, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs`. This is designed for overlapping keywords in `annular_gmean` and `func`. Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns

Array with all calculated geometric means

Return type

numpy.ndarray

Raises

- **ValueError** – If *func* is not callable.
- **ValueError** – If `val_arr` has less than 2 values.
- **ValueError** – If `val_arr` is not sorted in incresing order.

Notes

If the last value in val_arr is "inf", the given function should provide a value for "inf" as input: `func(float("inf"))`

Examples

```
>>> f = lambda x: x**2
>>> annular_gmean(f, [1,2,3])
array([ 2.33588885,  6.33423311])
```

anaflow.tools.mean.annular_hmean

annular_hmean(*func*, *val_arr*, *ann_dim=2*, *arg_dict=None*, ***kwargs*)

Calculating the annular harmonic mean.

Calculating the annular harmonic mean of a radial symmetric function for given consecutive radii defining annuli by the following formula

$$\varphi_i = \left(\frac{d}{r_{i+1}^d - r_i^d} \int_{r_i}^{r_{i+1}} r^{d-1} \cdot \varphi(r)^{-1} \, dr \right)^{-1}$$

Parameters

- **func** (callable) – Function that should be used (φ in the formula). The first argument needs to be the radial variable: `func(r, **kwargs)`
- **val_arr** (numpy.ndarray) – Radii defining the annuli.
- **ann_dim** (float, optional) – The dimension of the annuli. Default: `2.0`

- **arg_dict** (dict or None, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs`. This is designed for overlapping keywords in `annular_hmean` and `func`. Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns
Array with all calculated geometric means

Return type
numpy.ndarray

Raises

- **ValueError** – If *func* is not callable.
- **ValueError** – If `val_arr` has less than 2 values.
- **ValueError** – If `val_arr` is not sorted in incresing order.

Notes

If the last value in val_arr is "inf", the given function should provide a value for "inf" as input: `func(float("inf"))`

anaflow.tools.mean.annular_pmean

annular_pmean(*func*, *val_arr*, *p=2.0*, *ann_dim=2*, *arg_dict=None*, ***kwargs*)

Calculating the annular p-mean.

Calculating the annular p-mean of a radial symmetric function for given consecutive radii defining annuli by the following formula

$$\varphi_i = \left(\frac{d}{r_{i+1}^d - r_i^d} \int_{r_i}^{r_{i+1}} r^{d-1} \cdot \varphi(r)^p \, dr \right)^{\frac{1}{p}}$$

Parameters

- **func** (callable) – Function that should be used (φ in the formula). The first argument needs to be the radial variable: `func(r, **kwargs)`
- **val_arr** (numpy.ndarray) – Radii defining the annuli.
- **p** (float, optional) – The potency defining the p-mean. Default: `2.0`
- **ann_dim** (float, optional) – The dimension of the annuli. Default: `2.0`
- **arg_dict** (dict or None, optional) – Keyword-arguments given as a dictionary that are forwarded to the function given in `func`. Will be merged with `**kwargs`. This is designed for overlapping keywords in `annular_pmean` and `func`. Default: `None`
- ****kwargs** – Keyword-arguments that are forwarded to the function given in `func`. Will be merged with `arg_dict`

Returns
Array with all calculated p-means

Return type
numpy.ndarray

Raises

- **ValueError** – If *func* is not callable.

- **ValueError** – If `val_arr` has less than 2 values.
- **ValueError** – If `val_arr` is not sorted in incresing order.

Notes

If the last value in val_arr is "inf", the given function should provide a value for "inf" as input: `func(float("inf"))`

anaflow.tools.special

Anaflow subpackage providing special functions.

The following functions are provided

Shaper([time, rad, struc_grid])	A class to reshape radius-time input-output in a good way.
step_f(rad, r_part, f_part)	Callalbe step function.
sph_surf(dim)	Surface of the d-dimensional sphere.
specialrange(val_min, val_max, steps[, typ])	Calculation of special point ranges.
specialrange_cut(val_min, val_max, steps[, ...])	Calculation of special point ranges.
aniso(e)	The anisotropy function.
well_solution(time, rad, storage, transmissivity)	The classical Theis solution.
grf_solution(time, rad, storage, conductivity)	The general radial flow (GRF) model for a pumping test.
inc_gamma(s, x)	The (upper) incomplete gamma function.
tpl_hyp(rad, dim, hurst, corr, prop)	Hyp_2F1 for the TPL CG model.
neuman2004_trans(rad, trans_gmean, var, ...)	The apparent transmissivity from Neuman 2004.

anaflow.tools.special.Shaper

class Shaper(*time=0*, *rad=0*, *struc_grid=True*)

Bases: `object`

A class to reshape radius-time input-output in a good way.

Parameters

- **time** (`numpy.ndarray` or `float`, optional) – Array with all time-points where the function should be evaluated. Default: 0
- **rad** (`numpy.ndarray` or `float`, optional) – Array with all radii where the function should be evaluated. Default: 0
- **struc_grid** (`bool`, optional) – If this is set to `False`, the *rad* and *time* array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured t-r grid is created. Default: `True`

Methods

reshape(result)	Reshape a time-rad result according to the input shape.

reshape(*result*)

Reshape a time-rad result according to the input shape.

anaflow.tools.special.step_f

step_f(*rad*, *r_part*, *f_part*)

Callalbe step function.

anaflow.tools.special.sph_surf

sph_surf(*dim*)

Surface of the d-dimensional sphere.

anaflow.tools.special.specialrange

specialrange(*val_min*, *val_max*, *steps*, *typ='exp'*)

Calculation of special point ranges.

Parameters

- **val_min** (`float`) – Starting value.
- **val_max** (`float`) – Ending value
- **steps** (`int`) – Number of steps.
- **typ** (`str` or `float`, optional) – Setting the kind of range-distribution. One can choose between
  - `"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: `"exp"`

Returns

Array containing the special range

Return type

`numpy.ndarray`

Examples

```
>>> specialrange(1,10,4)
array([ 1.        ,  2.53034834,  5.23167968, 10.        ])
```

anaflow.tools.special.specialrange_cut

specialrange_cut(*val_min*, *val_max*, *steps*, *val_cut=None*, *typ='exp'*)

Calculation of special point ranges.

Calculation of special point ranges with a cut-off value.

Parameters

- **val_min** (float) – Starting value.
- **val_max** (float) – Ending value
- **steps** (int) – Number of steps.
- **val_cut** (float, optional) – Cutting value, if val_max is bigger than this value, the last interval will be between val_cut and val_max. Default: 100.0
- **typ** (str or float, optional) – Setting the kind of range-distribution. One can choose between
  - "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: "exp"

Returns
Array containing the special range

Return type
numpy.ndarray

Examples

```
>>> specialrange_cut(1,10,4)
array([ 1.        ,  2.53034834,  5.23167968, 10.        ])
```

anaflow.tools.special.aniso

aniso(*e*)

The anisotropy function.

Known from ''Dagan (1989)"[R2].

Parameters
e (float) – Anisotropy-ratio of the vertical and horizontal corralation-lengths

Returns
aniso – Value of the anisotropy function for the given value.

Return type
float

Raises
ValueError – If the Anisotropy-ratio e is not within 0 and 1.

References

Examples

```
>>> aniso(0.5)
0.2363998587187151
```

anaflow.tools.special.well_solution

well_solution(*time*, *rad*, *storage*, *transmissivity*, *rate=-0.0001*, *h_bound=0.0*, *struc_grid=True*)

The classical Theis solution.

The classical Theis solution for transient flow under a pumping condition in a confined and homogeneous aquifer.

This solution was presented in ''Theis 1935"[R9].

Parameters

- **time** (numpy.ndarray) – Array with all time-points where the function should be evaluated.
- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated.
- **storage** (float) – Storage of the aquifer.
- **transmissivity** (float) – Transmissivity of the aquifer.
- **rate** (float, optional) – Pumpingrate at the well. Default: -1e-4
- **h_bound** (float, optional) – Reference head at the outer boundary at infinity. Default: 0.0
- **struc_grid** (bool, optional) – If this is set to "False", the "rad" and "time" array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: True

Returns

head – Array with all heads at the given radii and time-points.

Return type

numpy.ndarray

Raises

- **ValueError** – If rad is not positiv.
- **ValueError** – If time is negative.
- **ValueError** – If shape of rad and time differ in case of struc_grid is True.
- **ValueError** – If transmissivity is not positiv.
- **ValueError** – If storage is not positiv.

References

Notes

The parameters `rad`, `T` and `S` will be checked for positivity. If you want to use cartesian coordiantes, just use the formula `r = sqrt(x**2 + y**2)`

Examples

```
>>> well_solution([10,100], [1,2,3], 0.001, 0.001, -0.001)
array([[-0.24959541, -0.14506368, -0.08971485],
       [-0.43105106, -0.32132823, -0.25778313]])
```

anaflow.tools.special.grf_solution

grf_solution(*time*, *rad*, *storage*, *conductivity*, *dim=2*, *lat_ext=1.0*, *rate=-0.0001*, *h_bound=0.0*, *struc_grid=True*)

The general radial flow (GRF) model for a pumping test.

Parameters

- **time** (`numpy.ndarray`) – Array with all time-points where the function should be evaluated.
- **rad** (`numpy.ndarray`) – Array with all radii where the function should be evaluated.
- **storage** (`float`) – Storage coefficient of the aquifer.
- **conductivity** (`float`) – Conductivity of the aquifer.
- **dim** (`float`, optional) – Fractional dimension of the aquifer. Default: `2.0`
- **lat_ext** (`float`, optional) – Lateral extend of the aquifer. Default: `1.0`
- **rate** (`float`, optional) – Pumpingrate at the well. Default: -1e-4
- **h_bound** (`float`, optional) – Reference head at the outer boundary at infinity. Default: `0.0`
- **struc_grid** (`bool`, optional) – If this is set to "False", the "rad" and "time" array will be merged and interpreted as single, r-t points. In this case they need to have the same shapes. Otherwise a structured r-t grid is created. Default: `True`

Returns

head – Array with all heads at the given radii and time-points.

Return type

`numpy.ndarray`

Raises

- `ValueError` – If `rad` is not positiv.
- `ValueError` – If `time` is negative.
- `ValueError` – If shape of `rad` and `time` differ in case of `struc_grid` is `True`.
- `ValueError` – If `conductivity` is not positiv.
- `ValueError` – If `storage` is not positiv.

anaflow.tools.special.inc_gamma

inc_gamma(*s*, *x*)

The (upper) incomplete gamma function.

Given by: $\Gamma(s, x) = \int_x^\infty t^{s-1}\, e^{-t}\, \mathrm{d}t$

Parameters

- **s** (float) – exponent in the integral
- **x** (numpy.ndarray) – input values

anaflow.tools.special.tpl_hyp

tpl_hyp(*rad*, *dim*, *hurst*, *corr*, *prop*)

Hyp_2F1 for the TPL CG model.

anaflow.tools.special.neuman2004_trans

neuman2004_trans(*rad*, *trans_gmean*, *var*, *len_scale*)

The apparent transmissivity from Neuman 2004.

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.

anaflow.tools.coarse_graining

Anaflow subpackage providing helper functions related to coarse graining.

The following functions are provided

T_CG(rad, trans_gmean, var, len_scale[, ...])	The coarse-graining Transmissivity.
T_CG_inverse(T, trans_gmean, var, len_scale)	The inverse coarse-graining Transmissivity.
T_CG_error(err, trans_gmean, var, len_scale)	Calculating the radial-point for given error.
K_CG(rad, cond_gmean, var, len_scale, anis)	The coarse-graining conductivity.
K_CG_inverse(K, cond_gmean, var, len_scale, anis)	The inverse coarse-graining conductivity.
K_CG_error(err, cond_gmean, var, len_scale, anis)	Calculating the radial-point for given error.
TPL_CG(rad, cond_gmean, len_scale, hurst[, ...])	The gaussian truncated power-law coarse-graining conductivity.
TPL_CG_error(err, cond_gmean, len_scale, hurst)	Calculating the radial-point for given error.

anaflow.tools.coarse_graining.T_CG

T_CG(*rad*, *trans_gmean*, *var*, *len_scale*, *T_well=None*, *prop=1.6*)

The coarse-graining Transmissivity.

This solution was presented in ''Schneider & Attinger 2008"[R3].

This functions gives an effective transmissivity for 2D pumpingtests in heterogenous aquifers, where the transmissivity is following a log-normal distribution and a gaussian correlation function.

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.
- **T_well** (float, optional) – Explicit transmissivity value at the well. Harmonic mean by default.
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns

T_CG – Array containing the effective transmissivity values.

Return type

numpy.ndarray

References

Examples

```
>>> T_CG([1,2,3], 0.001, 1, 10, 2)
array([0.00061831, 0.00064984, 0.00069236])
```

anaflow.tools.coarse_graining.T_CG_inverse

T_CG_inverse(*T*, *trans_gmean*, *var*, *len_scale*, *T_well=None*, *prop=1.6*)

The inverse coarse-graining Transmissivity.

See: *T_CG()*

Parameters

- **T** (numpy.ndarray) – Array with all transmissivity values where the function should be evaluated
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.
- **T_well** (float, optional) – Explicit transmissivity value at the well. Harmonic mean by default.
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns

rad – Array containing the radii belonging to the given transmissivity values

Return type

numpy.ndarray

Examples

```
>>> T_CG_inverse([7e-4,8e-4,9e-4], 0.001, 1, 10, 2)
array([3.16952925, 5.56935826, 9.67679026])
```

anaflow.tools.coarse_graining.T_CG_error

T_CG_error(*err*, *trans_gmean*, *var*, *len_scale*, *T_well=None*, *prop=1.6*)

Calculating the radial-point for given error.

Calculating the radial-point where the relative error of the farfield value is less than the given tollerance. See: *T_CG()*

Parameters

- **err** (float) – Given relative error for the farfield transmissivity
- **trans_gmean** (float) – Geometric-mean transmissivity.
- **var** (float) – Variance of log-transmissivity.
- **len_scale** (float) – Correlation-length of log-transmissivity.
- **T_well** (float, optional) – Explicit transmissivity value at the well. Harmonic mean by default.
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns

rad – Radial point, where the relative error is less than the given one.

Return type

float

Examples

```
>>> T_CG_error(0.01, 0.001, 1, 10, 2)
34.91045016779039
```

anaflow.tools.coarse_graining.K_CG

K_CG(*rad*, *cond_gmean*, *var*, *len_scale*, *anis*, *K_well='KH'*, *prop=1.6*)

The coarse-graining conductivity.

This solution was presented in ''Zech 2013"[R8].

This functions gives an effective conductivity for 3D pumpingtests in heterogenous aquifers, where the conductivity is following a log-normal distribution and a gaussian correlation function and taking vertical anisotropy into account.

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **cond_gmean** (float) – Geometric-mean conductivity.
- **var** (float) – Variance of the log-conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **anis** (float) – Anisotropy-ratio of the vertical and horizontal corralation-lengths.
- **K_well** (*string/float, optional*) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns

K_CG – Array containing the effective conductivity values.

Return type

numpy.ndarray

References

Examples

```
>>> K_CG([1,2,3], 0.001, 1, 10, 1, 2)
array([0.00063008, 0.00069285, 0.00077595])
```

anaflow.tools.coarse_graining.K_CG_inverse

K_CG_inverse(*K*, *cond_gmean*, *var*, *len_scale*, *anis*, *K_well='KH'*, *prop=1.6*)

The inverse coarse-graining conductivity.

See: *K_CG()*

Parameters

- **K** (numpy.ndarray) – Array with all conductivity values where the function should be evaluated
- **cond_gmean** (float) – Geometric-mean conductivity.

- **var** (float) – Variance of the log-conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **anis** (float) – Anisotropy-ratio of the vertical and horizontal corralation-lengths.
- **K_well** (*string/float, optional*) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns
rad – Array containing the radii belonging to the given conductivity values

Return type
numpy.ndarray

Examples

```
>>> K_CG_inverse([7e-4,8e-4,9e-4], 0.001, 1, 10, 1, 2)
array([2.09236867, 3.27914996, 4.52143956])
```

anaflow.tools.coarse_graining.K_CG_error

K_CG_error(*err*, *cond_gmean*, *var*, *len_scale*, *anis*, *K_well='KH'*, *prop=1.6*)

Calculating the radial-point for given error.

Calculating the radial-point where the relative error of the farfield value is less than the given tollerance. See: *K_CG()*

Parameters

- **err** (float) – Given relative error for the farfield conductivity
- **cond_gmean** (float) – Geometric-mean conductivity.
- **var** (float) – Variance of the log-conductivity.
- **len_scale** (float) – Corralation-length of log-conductivity.
- **anis** (float) – Anisotropy-ratio of the vertical and horizontal corralation-lengths.
- **K_well** (*string/float, optional*) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns
rad – Radial point, where the relative error is less than the given one.

Return type
float

Examples

```
>>> K_CG_error(0.01, 0.001, 1, 10, 1, 2)
19.612796453639845
```

anaflow.tools.coarse_graining.TPL_CG

TPL_CG(*rad*, *cond_gmean*, *len_scale*, *hurst*, *var=None*, *c=1.0*, *anis=1*, *dim=2.0*, *K_well='KH'*, *prop=1.6*)

The gaussian truncated power-law coarse-graining conductivity.

Parameters

- **rad** (numpy.ndarray) – Array with all radii where the function should be evaluated
- **cond_gmean** (float) – Geometric-mean conductivity
- **len_scale** (float) – upper bound of the corralation-length of conductivity-distribution
- **hurst** (float) – Hurst coefficient of the TPL model. Should be in (0, 1).
- **var** (float or None, optional) – Variance of log-conductivity If given, c will be calculated accordingly. Default: None
- **c** (float, optional) – Intensity of variation in the TPL model. Is overwritten if var is given. Default: 1.0
- **anis** (float, optional) – Anisotropy-ratio of the vertical and horizontal corralation-lengths. This is only applied in 3 dimensions. Default: 1.0
- **dim** (float, optional) – Dimension of space. Default: 2.0
- **K_well** (str or float, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns

TPL_CG – Array containing the effective conductivity values.

Return type

numpy.ndarray

anaflow.tools.coarse_graining.TPL_CG_error

TPL_CG_error(*err*, *cond_gmean*, *len_scale*, *hurst*, *var=None*, *c=1.0*, *anis=1*, *dim=2.0*, *K_well='KH'*, *prop=1.6*)

Calculating the radial-point for given error.

Calculating the radial-point where the relative error of the farfield value is less than the given tollerance. See: *TPL_CG()*

Parameters

- **err** (float) – Given relative error for the farfield conductivity
- **cond_gmean** (float) – Geometric-mean conductivity
- **len_scale** (float) – upper bound of the corralation-length of conductivity-distribution
- **hurst** (float) – Hurst coefficient of the TPL model. Should be in (0, 1).
- **var** (float or None, optional) – Variance of log-conductivity If given, c will be calculated accordingly. Default: None

- **c** (float, optional) – Intensity of variation in the TPL model. Is overwritten if var is given. Default: 1.0
- **anis** (float, optional) – Anisotropy-ratio of the vertical and horizontal corralation-lengths. This is only applied in 3 dimensions. Default: 1.0
- **dim** (float, optional) – Dimension of space. Default: 2.0
- **K_well** (str or float, optional) – Explicit conductivity value at the well. One can choose between the harmonic mean ("KH"), the arithmetic mean ("KA") or an arbitrary float value. Default: "KH"
- **prop** (float, optional) – Proportionality factor used within the upscaling procedure. Default: 1.6

Returns

rad – Radial point, where the relative error is less than the given one.

Return type

float

Functions

Annular mean

Functions to calculate dimension dependent annular means of a function.

annular_fmean(func, val_arr, f_def, f_inv[, ...])	Calculating the annular generalized f-mean.
annular_amean(func, val_arr[, ann_dim, arg_dict])	Calculating the annular arithmetic mean.
annular_gmean(func, val_arr[, ann_dim, arg_dict])	Calculating the annular geometric mean.
annular_hmean(func, val_arr[, ann_dim, arg_dict])	Calculating the annular harmonic mean.
annular_pmean(func, val_arr[, p, ann_dim, ...])	Calculating the annular p-mean.

Coarse Graining solutions

Effective Coarse Graining conductivity/transmissivity solutions.

T_CG(rad, trans_gmean, var, len_scale[, ...])	The coarse-graining Transmissivity.
K_CG(rad, cond_gmean, var, len_scale, anis)	The coarse-graining conductivity.
TPL_CG(rad, cond_gmean, len_scale, hurst[, ...])	The gaussian truncated power-law coarse-graining conductivity.

Special

Special functions.

step_f(rad, r_part, f_part)	Callalbe step function.
specialrange(val_min, val_max, steps[, typ])	Calculation of special point ranges.
specialrange_cut(val_min, val_max, steps[, ...])	Calculation of special point ranges.
neuman2004_trans(rad, trans_gmean, var, ...)	The apparent transmissivity from Neuman 2004.
aniso(e)	The anisotropy function.

Laplace

Helping functions related to the laplace-transformation

get_lap(func[, arg_dict])	Callable Laplace transform.
get_lap_inv(func[, method, method_dict, ...])	Callable Laplace inversion.

3.3 Solutions

Homogeneous

Solutions for homogeneous aquifers

thiem(rad, r_ref, transmissivity[, rate, h_ref])	The Thiem solution.
theis(time, rad, storage, transmissivity[, ...])	The Theis solution.
grf(time, rad, storage, conductivity[, dim, ...])	The general radial flow (GRF) model for a pumping test.

Heterogeneous

Solutions for heterogeneous aquifers

ext_thiem_2d(rad, r_ref, trans_gmean, var, ...)	The extended Thiem solution in 2D.
ext_thiem_3d(rad, r_ref, cond_gmean, var, ...)	The extended Thiem solution in 3D.
ext_thiem_tpl(rad, r_ref, cond_gmean, ...[, ...])	The extended Thiem solution for truncated power-law fields.
ext_thiem_tpl_3d(rad, r_ref, cond_gmean, ...)	The extended Theis solution for truncated power-law fields in 3D.
ext_theis_2d(time, rad, storage, ...[, ...])	The extended Theis solution in 2D.
ext_theis_3d(time, rad, storage, cond_gmean, ...)	The extended Theis solution in 3D.
ext_theis_tpl(time, rad, storage, ...[, ...])	The extended Theis solution for truncated power-law fields.
ext_thiem_tpl_3d(rad, r_ref, cond_gmean, ...)	The extended Theis solution for truncated power-law fields in 3D.
neuman2004(time, rad, storage, trans_gmean, ...)	The transient solution for the apparent transmissivity from [Neuman2004].
neuman2004_steady(rad, r_ref, trans_gmean, ...)	The steady solution for the apparent transmissivity from [Neuman2004].

Extended GRF

The extended general radial flow model.

ext_grf(time, rad, S_part, K_part, R_part[, ...])	The extended "General radial flow" model for transient flow.
ext_grf_steady(rad, r_ref, conductivity[, ...])	The extended "General radial flow" model for steady flow.

3.4 Laplace

Helping functions related to the laplace-transformation

get_lap(func[, arg_dict])	Callable Laplace transform.
get_lap_inv(func[, method, method_dict, ...])	Callable Laplace inversion.

3.5 Tools

Helping functions.

step_f(rad, r_part, f_part)	Callalbe step function.
specialrange(val_min, val_max, steps[, typ])	Calculation of special point ranges.
specialrange_cut(val_min, val_max, steps[, ...])	Calculation of special point ranges.

CHAPTER 4

CHANGELOG

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

4.1 [1.1.0] - 2023-04

See #11

Enhancements

- move to `src/` base package structure
- drop py36 support
- added archive support
- simplify documentation

4.2 1.0.1 - 2020-04-02

Bugfixes

- `ModuleNotFoundError` not present in py35
- `np.asscalar` deprecated, use `array.item()`
- `CHANGELOG.md` links updated

4.3 1.0.0 - 2020-03-22

Enhancements

- new TPL Solution
- new tools sub-module
- using pentapy to solve LES in laplace space
- solution for aparent transmissivity from neuman 2004
- added extended GRF model
- convenient functions for (inverse-)laplace transformation

Bugfixes

- `lat_ext` was ignored by ext_theis_3d

Changes

- py2.7 support dropped

4.4 0.4.0 - 2019-03-07

Enhancements

- the output for transient tests now preserves the shapes of time and rad (better for plotting in 3D)
- the grf model is now the default way of calculating pumping tests in laplace space
- the grf_laplace routine was optimized to estimate the radius of the cone of depression
- the grf_laplace uses now the pentapy solver, so we get rid of the umf_pack dependency
- grf_model and grf_disk are now part of the standard routines

Changes

- the input for transient tests changed from “rad, time” to “time, rad” as order of input (in line with output format)

Bugfixes

- multiple minor bugfixes

4.5 0.3.0 - 2019-02-28

Enhancements

- GRF model added
- new documetation
- added examples
- code restructured

Changes

Bugfixes

4.6 0.2.4 - 2018-04-26

Enhancements

- Released on PyPI

4.7 0.1.0 - 2018-01-05

First release of AnaFlow. Containing:

- thiem - Thiem solution for steady state pumping
- theis - Theis solution for transient pumping
- ext_thiem2D - extended Thiem solution in 2D
- ext_theis2D - extended Theis solution in 2D
- ext_thiem3D - extended Thiem solution in 3D
- ext_theis3D - extended Theis solution in 3D
- diskmodel - Solution for a diskmodel
- lap_transgwflow_cyl - Solution for a diskmodel in laplace inversion

BIBLIOGRAPHY

[Thiem1906] Thiem, G., ''Hydrologische Methoden, J.M. Gebhardt", Leipzig, 1906.

[Theis35] Theis, C., ''The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using groundwater storage", Trans. Am. Geophys. Union, 16, 519-524, 1935

[Barker88] Barker, J. ''A generalized radial flow model for hydraulic tests in fractured rock.", Water Resources Research 24.10, 1796-1804, 1988

[Zech2013] Zech, A. ''Impact of Aqifer Heterogeneity on Subsurface Flow and Salt Transport at Different Scales: from a method determine parameters of heterogeneous permeability at local scale to a large-scale model for the sedimentary basin of Thuringia." PhD thesis, Friedrich-Schiller-Universität Jena, 2013

[Zech2013] Zech, A. ''Impact of Aqifer Heterogeneity on Subsurface Flow and Salt Transport at Different Scales: from a method determine parameters of heterogeneous permeability at local scale to a large-scale model for the sedimentary basin of Thuringia." PhD thesis, Friedrich-Schiller-Universität Jena, 2013

[Neuman2004] Neuman, Shlomo P., Alberto Guadagnini, and Monica Riva. ''Type-curve estimation of statistical heterogeneity." Water resources research 40.4, 2004

[Neuman2004] Neuman, Shlomo P., Alberto Guadagnini, and Monica Riva. ''Type-curve estimation of statistical heterogeneity." Water resources research 40.4, 2004

[Barker88] Barker, J. ''A generalized radial flow model for hydraulic tests in fractured rock.", Water Resources Research 24.10, 1796-1804, 1988

[Barker88] Barker, J. ''A generalized radial flow model for hydraulic tests in fractured rock.", Water Resources Research 24.10, 1796-1804, 1988

[R1] Stehfest, H., ''Algorithm 368: Numerical inversion of laplace transforms [d5]." Communications of the ACM, 13(1):47-49, 1970

[R2] Dagan, G., ''Flow and Transport on Porous Formations", Springer Verlag, New York, 1989.

[R9] Theis, C., ''The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using groundwater storage", Trans. Am. Geophys. Union, 16, 519-524, 1935

[R3] Schneider, C. and Attinger, S., ''Beyond thiem: A new method for interpreting large scale pumping tests in heterogeneous aquifers." Water resources research, 44(4), 2008

[R8] Zech, A. ''Impact of Aqifer Heterogeneity on Subsurface Flow and Salt Transport at Different Scales: from a method determine parameters of heterogeneous permeability at local scale to a large-scale model for the sedimentary basin of Thuringia." PhD thesis, Friedrich-Schiller-Universität Jena, 2013

PYTHON MODULE INDEX

a

INDEX

T

W