pygfunction Documentation

Release 2.2.2

Massimo Cimmino

Jan 09, 2023

CONTENTS

pygfunction is a Python package that allows the calculation of thermal response factors - or *g*-functions - for geothermal borehole fields using analytical solutions.

CHAPTER

ONE

SETTING UP PYGFUNCTION

***pygfunction* uses Python 3.7, along with the following packages:**

- matplotlib (>= 3.5.1),
- numpy (>= 1.21.5)
- scipy (>= 1.7.3)
- SecondaryCoolantProps (>= 1.1)

***pygfunction*'s- documentation is built using:**

- sphinx (>= 4.4.0)
- numpydoc (>= 1.2.0)

Users - Download pip and install the latest release:

` pip install pygfunction `

Alternatively, download the latest release and run the installation script:

` pip install . `

Developers - To get the latest version of the code, you can download the repository from github or clone the project in a local directory using git:

` git clone https://github.com/MassimoCimmino/pygfunction.git `

Install *pygfunction* in development mode (this requires *pip* >= *21.1*): ` pip install --editable . `

Test that *pygfunction* is running correctly by running any of the provided examples in `../pygfunction/examples/`

CHAPTER

TWO

CITING PYGFUNCTION IN SCIENTIFIC PUBLICATIONS

2.1 pygfunction v2.2

Improvements to *pygfunction* from v1.0 to v2.2 were presented at the IGSHPA Annual Conference 2022:

Cimmino, M., & Cook, J.C. (2022). pygfunction 2.2: New features and improvements in accuracy and computational efficiency. In *Research Conference Proceedings, IGSHPA Annual Conference 2022* (pp. 45-52). International Ground Source Heat Pump Association. DOI: https://doi.org/10.22488/okstate.22.000015.

The conference paper is available here.

Here is an example of a BibTeX entry:

```
@inproceedings{2022Pygfunction-V2-2,
  author =       {Cimmino, Massimo and Cook, Jonathan C.},
  title =        {{pygfunction} 2.2: New features and improvements in accuracy and␣
↪computational efficiency},
  crossref =     {IGSHPA2022},
  pages =        {45--52},
}

@proceedings{IGSHPA2022,
  title =        "Research Conference Proceedings, IGSHPA Annual Conference 2022",
  booktitle =    "Research Conference Proceedings, IGSHPA Annual Conference 2022",
  publisher =    {International Ground Source Heat Pump Association}
  venue =        {Las Vegas NV, USA},
  month =        december,
  year =         {2022},
  doi =          {10.22488/okstate.22.000010},
}
```

2.2 pygfunction v1.0

pygfunction was introduced at the esim 2018 conference:

Cimmino, M. (2018). pygfunction: an open-source toolbox for the evaluation of thermal response factors for geothermal borehole fields. In *Proceedings of eSim 2018, the 10th conference of IBPSA-Canada* (pp. 492-501). IBPSA. ISBN 978-2-921145-88-6.

The conference paper is available here.

Here is an example of a BibTeX entry:

```
@inproceedings{2018Pygfunction-V1,
  author =       {Massimo Cimmino},
  title =        {{pygfunction}: an open-source toolbox for the evaluation of thermal␣
↪response factors for geothermal borehole fields},
  crossref =     {eSim2018},
  pages =        {492--501},
}

@proceedings{eSim2018,
  title =        "Proceedings of eSim 2018, the 10th conference of IBPSA-Canada",
  booktitle =    "Proceedings of eSim 2018, the 10th conference of IBPSA-Canada",
  publisher =    {IBPSA Canada}
  venue =        {Montréal, QC, Canada},
  month =        may,
  year =         {2018},
  isbn =         {978-2-921145-88-6},
}
```

CHAPTER

THREE

MODULES

3.1 Borehole Module

class pygfunction.boreholes.**Borehole**(*H*, *D*, *r_b*, *x*, *y*, *tilt=0.0*, *orientation=0.0*)

Bases: object

Contains information regarding the dimensions and position of a borehole.

Attributes

H
[float] Borehole length (in meters).

D
[float] Borehole buried depth (in meters).

r_b
[float] Borehole radius (in meters).

x
[float] Position (in meters) of the head of the borehole along the x-axis.

y
[float] Position (in meters) of the head of the borehole along the y-axis.

tilt
[float] Angle (in radians) from vertical of the axis of the borehole.

orientation
[float] Direction (in radians) of the tilt of the borehole.

distance(*target*)

Evaluate the distance between the current borehole and a target borehole.

Parameters

target
[Borehole object] Target borehole for which the distance is evaluated.

Returns

dis
[float] Distance (in meters) between current borehole and target borehole.

Note: The smallest distance returned is equal to the borehole radius. This means that the distance between a borehole and itself is equal to r_b.

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=0., y=0.)
>>> b2 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> b1.distance(b2)
5.0
```

is_tilted()

Returns true if the borehole is inclined.

Returns

bool
True if borehole is inclined.

is_vertical()

Returns true if the borehole is vertical.

Returns

bool
True if borehole is vertical.

position()

Returns the position of the borehole.

Returns

pos
[tuple] Position (x, y) (in meters) of the borehole.

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> b1.position()
(5.0, 0.0)
```

segments(*nSegments*, *segment_ratios=None*)

Split a borehole into segments.

Parameters

nSegments
[int] Number of segments.

segment_ratios
[array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. The shape of the array is of (nSegments,). If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

boreSegments
[list] List of borehole segments.

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> b1.segments(5)
```

pygfunction.boreholes.**L_shaped_field**(*N_1*, *N_2*, *B_1*, *B_2*, *H*, *D*, *r_b*, *tilt=0.0*, *origin=None*)

Build a list of boreholes in a L-shaped bore field configuration.

Parameters

N_1
[int] Number of borehole in the x direction.

N_2
[int] Number of borehole in the y direction.

B_1
[float] Distance (in meters) between adjacent boreholes in the x direction.

B_2
[float] Distance (in meters) between adjacent boreholes in the y direction.

H
[float] Borehole length (in meters).

D
[float] Borehole buried depth (in meters).

r_b
[float] Borehole radius (in meters).

tilt
[float, optional] Angle (in radians) from vertical of the axis of the borehole. The orientation of the tilt is orthogonal to the origin coordinate. Default is 0.

origin
[tuple, optional] A coordinate indicating the origin of reference for orientation of boreholes. Default is origin is placed at the center of an assumed rectangle.

Returns

boreField
[list of Borehole objects] List of boreholes in the L-shaped bore field.

Notes

Boreholes located at the origin will remain vertical.

Examples

```
>>> boreField = gt.boreholes.L_shaped_field(N_1=3, N_2=2, B_1=5., B_2=5.,
                                            H=100., D=2.5, r_b=0.05)
```

The bore field is constructed line by line. For N_1=3 and N_2=2, the bore field layout is as follows:

```
3

0   1   2
```

pygfunction.boreholes.**U_shaped_field**(*N_1*, *N_2*, *B_1*, *B_2*, *H*, *D*, *r_b*, *tilt=0.0*, *origin=None*)

Build a list of boreholes in a U-shaped bore field configuration.

Parameters

N_1
[int] Number of borehole in the x direction.

N_2
[int] Number of borehole in the y direction.

B_1
[float] Distance (in meters) between adjacent boreholes in the x direction.

B_2
[float] Distance (in meters) between adjacent boreholes in the y direction.

H
[float] Borehole length (in meters).

D
[float] Borehole buried depth (in meters).

r_b
[float] Borehole radius (in meters).

tilt
[float, optional] Angle (in radians) from vertical of the axis of the borehole. The orientation of the tilt is orthogonal to the origin coordinate. Default is 0.

origin
[tuple, optional] A coordinate indicating the origin of reference for orientation of boreholes. Default is the center considering an outer rectangle.

Returns

boreField
[list of Borehole objects] List of boreholes in the U-shaped bore field.

Notes

Boreholes located at the origin will remain vertical.

Examples

```
>>> boreField = gt.boreholes.U_shaped_field(N_1=3, N_2=2, B_1=5., B_2=5.,
                                            H=100., D=2.5, r_b=0.05)
```

The bore field is constructed line by line. For N_1=3 and N_2=2, the bore field layout is as follows:

```
3       4

0   1   2
```

pygfunction.boreholes.**box_shaped_field**(*N_1*, *N_2*, *B_1*, *B_2*, *H*, *D*, *r_b*, *tilt=0*, *origin=None*)

Build a list of boreholes in a box-shaped bore field configuration.

Parameters

N_1
[int] Number of borehole in the x direction.

N_2
[int] Number of borehole in the y direction.

B_1
[float] Distance (in meters) between adjacent boreholes in the x direction.

B_2
[float] Distance (in meters) between adjacent boreholes in the y direction.

H
[float] Borehole length (in meters).

D
[float] Borehole buried depth (in meters).

r_b
[float] Borehole radius (in meters).

tilt
[float, optional] Angle (in radians) from vertical of the axis of the borehole. The orientation of the tilt is orthogonal to the origin coordinate. Default is 0.

origin
[tuple, optional] A coordinate indicating the origin of reference for orientation of boreholes. Default is the center of the box.

Returns

boreField
[list of Borehole objects] List of boreholes in the box-shaped bore field.

Notes

Boreholes located at the origin will remain vertical.

Examples

```
>>> boreField = gt.boreholes.box_shaped_field(N_1=4, N_2=3, B_1=5., B_2=5.,
                                              H=100., D=2.5, r_b=0.05)
```

The bore field is constructed line by line. For N_1=4 and N_2=3, the bore field layout is as follows:

```
6   7   8   9

4           5

0   1   2   3
```

pygfunction.boreholes.**circle_field**(*N*, *R*, *H*, *D*, *r_b*, *tilt=0.0*, *origin=None*)

Build a list of boreholes in a circular field configuration.

Parameters

N
[int] Number of boreholes in the bore field.

R
[float] Distance (in meters) of the boreholes from the center of the field.

H
[float] Borehole length (in meters).

D
[float] Borehole buried depth (in meters).

r_b
[float] Borehole radius (in meters).

tilt
[float, optional] Angle (in radians) from vertical of the axis of the borehole. The orientation of the tilt is towards the exterior of the bore field. Default is 0.

origin
[tuple] A coordinate indicating the origin of reference for orientation of boreholes. Default is the origin (0, 0).

Returns

boreField
[list of Borehole objects] List of boreholes in the circular shaped bore field.

Notes

Boreholes located at the origin will remain vertical.

Examples

```
>>> boreField = gt.boreholes.circle_field(N=8, R = 5., H=100., D=2.5,
                                          r_b=0.05)
```

The bore field is constructed counter-clockwise. For N=8, the bore field layout is as follows:

```
      2
  3       1

4           0

  5       7
      6
```

pygfunction.boreholes.**field_from_file**(*filename*)

Build a list of boreholes given coordinates and dimensions provided in a text file.

Parameters

filename
[str] Absolute path to text file.

Returns

boreField
[list of Borehole objects] List of boreholes in the bore field.

Notes

The text file should be formatted as follows:

```
# x   y     H     D     r_b     tilt   orientation
0.    0.    100.  2.5   0.075   0.     0.
5.    0.    100.  2.5   0.075   0.     0.
0.    5.    100.  2.5   0.075   0.     0.
0.    10.   100.  2.5   0.075   0.     0.
0.    20.   100.  2.5   0.075   0.     0.
```

pygfunction.boreholes.**find_duplicates**(*boreField*, *disp=False*)

The distance method *Borehole.distance()* is utilized to find all duplicate boreholes in a boreField. This function considers a duplicate to be any pair of points that fall within each others radius. The lower index (i) is always stored in the 0 position of the tuple, while the higher index (j) is stored in the 1 position.

Parameters

boreField
[list] A list of *Borehole* objects

disp
[bool, optional] Set to true to print progression messages. Default is False.

Returns

duplicate_pairs
[list] A list of tuples where the tuples are pairs of duplicates

pygfunction.boreholes.**rectangle_field**(*N_1*, *N_2*, *B_1*, *B_2*, *H*, *D*, *r_b*, *tilt=0.0*, *origin=None*)

Build a list of boreholes in a rectangular bore field configuration.

Parameters

N_1
[int] Number of borehole in the x direction.

N_2
[int] Number of borehole in the y direction.

B_1
[float] Distance (in meters) between adjacent boreholes in the x direction.

B_2
[float] Distance (in meters) between adjacent boreholes in the y direction.

H
[float] Borehole length (in meters).

D
[float] Borehole buried depth (in meters).

r_b
[float] Borehole radius (in meters).

tilt
[float, optional] Angle (in radians) from vertical of the axis of the borehole. The orientation of the tilt is orthogonal to the origin coordinate. Default is 0.

origin
[tuple, optional] A coordinate indicating the origin of reference for orientation of boreholes. Default is the center of the rectangle.

Returns

boreField
[list of Borehole objects] List of boreholes in the rectangular bore field.

Notes

Boreholes located at the origin will remain vertical.

Examples

```
>>> boreField = gt.boreholes.rectangle_field(N_1=3, N_2=2, B_1=5., B_2=5.,
                                             H=100., D=2.5, r_b=0.05)
```

The bore field is constructed line by line. For N_1=3 and N_2=2, the bore field layout is as follows:

```
3   4   5

0   1   2
```

pygfunction.boreholes.**remove_duplicates**(*boreField*, *disp=False*)

Removes all of the duplicates found from the duplicate pairs returned in `check_duplicates()`.

For each pair of duplicates, the first borehole (with the lower index) is kept and the other (with the higher index) is removed.

Parameters

boreField
[list] A list of *Borehole* objects

disp
[bool, optional] Set to true to print progression messages. Default is False.

Returns

new_boreField
[list] A boreField without duplicates

pygfunction.boreholes.**visualize_field**(*borefield*, *viewTop=True*, *view3D=True*, *labels=True*, *showTilt=True*)

Plot the top view and 3D view of borehole positions.

Parameters

borefield
[list] List of boreholes in the bore field.

viewTop
[bool, optional] Set to True to plot top view. Default is True

view3D
[bool, optional] Set to True to plot 3D view. Default is True

labels
[bool, optional] Set to True to annotate borehole indices to top view plot. Default is True

showTilt
[bool, optional] Set to True to show borehole inclination on top view plot. Default is True

Returns

fig
[figure] Figure object (matplotlib).

3.2 g-Function Module

pygfunction.gfunction.**equal_inlet_temperature**(*boreholes*, *UTubes*, *m_flow_borehole*, *cp_f*, *time*, *alpha*, *kind='linear'*, *nSegments=8*, *segment_ratios=<function segment_ratios>*, *use_similarities=True*, *disTol=0.01*, *tol=1e-06*, *dtype=<class 'numpy.float64'>*, *disp=False*, ***kwargs*)

Evaluate the g-function with equal inlet fluid temperatures.

This function superimposes the finite line source (FLS) solution to estimate the g-function of a geothermal bore field. Each borehole is modeled as a series of finite line source segments, as proposed in[1].

Parameters

boreholes
[list of Borehole objects] List of boreholes included in the bore field.

UTubes
[list of pipe objects] Model for pipes inside each borehole.

m_flow_borehole
[float or array] Fluid mass flow rate per borehole (in kg/s).

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.K).

time
[float or array] Values of time (in seconds) for which the g-function is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

nSegments
[int or list, optional] Number of line segments used per borehole, or list of number of line segments used for each borehole. Default is 8.

segment_ratios
[array, list of arrays, or callable, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. The shape of the array is of (nSegments,) or list of (nSegments[i],). If segment_ratios==None, segments of equal lengths are considered. If a callable is provided, it must return an array of size (nSegments,) when provided with nSegments (of type int) as an argument, or an array of size (nSegments[i],) when provided with an element of nSegments (of type list). Default is `utilities.segment_ratios()`.

kind
[string, optional] Interpolation method used for segment-to-segment thermal response factors. See documentation for scipy.interpolate.interp1d. Default is 'linear'.

use_similarities
[bool, optional] True if similarities are used to limit the number of FLS evaluations. Default is True.

[1] Cimmino, M. (2015). The effects of borehole thermal resistances and fluid flow rate on the g-functions of geothermal bore fields. International Journal of Heat and Mass Transfer, 91, 1119-1127.

disTol
[float, optional] Relative tolerance on radial distance. Two distances (d1, d2) between two pairs of boreholes are considered equal if the difference between the two distances (abs(d1-d2)) is below tolerance. Default is 0.01.

tol
[float, optional] Relative tolerance on length and depth. Two lengths H1, H2 (or depths D1, D2) are considered equal if abs(H1 - H2)/H2 < tol. Default is 1.0e-6.

dtype
[numpy dtype, optional] numpy data type used for matrices and vectors. Should be one of numpy.single or numpy.double. Default is numpy.double.

disp
[bool, optional] Set to true to print progression messages. Default is False.

Returns

gFunction
[float or array] Values of the g-function

References

class pygfunction.gfunction.**gFunction**(*boreholes_or_network*, *alpha*, *time=None*, *method='equivalent'*, *boundary_condition=None*, *options={}*)

Bases: `object`

Class for the calculation and visualization of the g-functions of geothermal bore fields.

This class superimposes the finite line source (FLS) solution to estimate the g-function of a geothermal bore field. Each borehole is modeled as a series of finite line source segments, as proposed in[2]. Different boundary conditions and solution methods are implemented.

Notes

- The 'equivalent' solver does not support the 'MIFT' boundary condition when boreholes are connected in series.
- The 'equivalent' solver does not support inclined boreholes.
- The g-function is linearized for times *t < r_b**2 / (25 * self.alpha)*. The g-function value is then interpolated between 0 and its value at the threshold.

References

Attributes

boreholes_or_network
[list of Borehole objects or Network object] List of boreholes included in the bore field, or network of boreholes and pipes.

alpha
[float] Soil thermal diffusivity (in m2/s).

[2] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

time
[float or array, optional] Values of time (in seconds) for which the g-function is evaluated. The g-function is only evaluated at initialization if a value is provided. Default is None.

method
[str, optional] Method for the evaluation of the g-function. Should be one of

- **'similarities' :**
  The accelerated method of Cimmino (2018)[4], using similarities in the bore field to decrease the number of evaluations of the FLS solution.
- **'detailed' :**
  The classical superposition of the FLS solution. The FLS solution is evaluated for all pairs of segments in the bore field.
- **'equivalent' :**
  The equivalent borehole method of Prieto and Cimmino (2021)[6]. Boreholes are assembled into groups of boreholes that share similar borehole wall temperatures and heat extraction rates. Each group is represented by an equivalent borehole and the group-to-group thermal interactions are calculated by the FLS solution. This is an approximation of the 'similarities' method.

Default is 'equivalent'.

boundary_condition
[str, optional] Boundary condition for the evaluation of the g-function. Should be one of

- **'UHTR' :**
  Uniform heat transfer rate. This is corresponds to boundary condition *BC-I* as defined by Cimmino and Bernier (2014)[Page 16, 2].
- **'UBWT' :**
  Uniform borehole wall temperature. This is corresponds to boundary condition *BC-III* as defined by Cimmino and Bernier (2014)[Page 16, 2].
- **'MIFT' :**
  Mixed inlet fluid temperatures. This boundary condition was introduced by Cimmino (2015)[3] for parallel-connected boreholes and extended to mixed configurations by Cimmino (2019)[5].

If not given, chosen to be 'UBWT' if a list of boreholes is provided or 'MIFT' if a Network object is provided.

options
[dict, optional] A dictionary of solver options. All methods accept the following generic options:

nSegments
[int or list, optional] Number of line segments used per borehole, or list of number of line segments used for each borehole. Default is 8.

segment_ratios
[array, list of arrays, or callable, optional] Ratio of the borehole length represented

[4] Cimmino, M. (2018). Fast calculation of the g-functions of geothermal borehole fields using similarities in the evaluation of the finite line source solution. Journal of Building Performance Simulation, 11 (6), 655-668.

[6] Prieto, C., & Cimmino, M. (2021). Thermal interactions in large irregular fields of geothermal boreholes: the method of equivalent borehole. Journal of Building Performance Simulation, 14 (4), 446-460.

[3] Cimmino, M. (2015). The effects of borehole thermal resistances and fluid flow rate on the g-functions of geothermal bore fields. International Journal of Heat and Mass Transfer, 91, 1119-1127.

[5] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022.

by each segment. The sum of ratios must be equal to 1. The shape of the array is of (nSegments,) or list of (nSegments[i],). If segment_ratios==None, segments of equal lengths are considered. If a callable is provided, it must return an array of size (nSegments,) when provided with nSegments (of type int) as an argument, or an array of size (nSegments[i],) when provided with an element of nSegments (of type list). Default is `utilities.segment_ratios()`.

approximate_FLS
[bool, optional] Set to true to use the approximation of the FLS solution of Cimmino (2021)[7]. This approximation does not require the numerical evaluation of any integral. When using the 'equivalent' solver, the approximation is only applied to the thermal response at the borehole radius. Thermal interaction between boreholes is evaluated using the FLS solution. Default is False.

nFLS
[int, optional] Number of terms in the approximation of the FLS solution. This parameter is unused if *approximate_FLS* is set to False. Default is 10. Maximum is 25.

mQuad
[int, optional] Number of Gauss-Legendre sample points for the integral over u in the inclined FLS solution. Default is 11.

linear_threshold
[float, optional] Threshold time (in seconds) under which the g-function is linearized. The g-function value is then interpolated between 0 and its value at the threshold. If linear_threshold==None, the g-function is linearized for times *t < r_b**2 / (25 * self.alpha)*. Default is None.

disp
[bool, optional] Set to true to print progression messages. Default is False.

profiles
[bool, optional] Set to true to keep in memory the temperatures and heat extraction rates. Default is False.

kind
[str, optional] Interpolation method used for segment-to-segment thermal response factors. See documentation for scipy.interpolate.interp1d. Default is 'linear'.

dtype
[numpy dtype, optional] numpy data type used for matrices and vectors. Should be one of numpy.single or numpy.double. Default is numpy.double.

The 'similarities' solver accepts the following method-specific options:

disTol
[float, optional] Relative tolerance on radial distance. Two distances (d1, d2) between two pairs of boreholes are considered equal if the difference between the two distances (abs(d1-d2)) is below tolerance. Default is 0.01.

tol
[float, optional] Relative tolerance on length and depth. Two lengths H1, H2 (or depths D1, D2) are considered equal if abs(H1 - H2)/H2 < tol. Default is 1.0e-6.

The 'equivalent' solver accepts the following method-specific options:

[7] Cimmino, M. (2021). An approximation of the finite line source solution to model thermal interactions between geothermal boreholes. International Communications in Heat and Mass Transfer, 127, 105496.

disTol
[float, optional] Relative tolerance on radial distance. Two distances (d1, d2) between two pairs of boreholes are considered equal if the difference between the two distances (abs(d1-d2)) is below tolerance. Default is 0.01.

tol
[float, optional] Relative tolerance on length and depth. Two lengths H1, H2 (or depths D1, D2) are considered equal if abs(H1 - H2)/H2 < tol. Default is 1.0e-6.

kClusters
[int, optional] Increment on the minimum number of equivalent boreholes determined by cutting the dendrogram of the bore field given by the hierarchical agglomerative clustering method. Increasing the value of this parameter increases the accuracy of the method. Default is 1.

`evaluate_g_function`(*time*)

Evaluate the g-function.

Parameters

time
[float or array] Values of time (in seconds) for which the g-function is evaluated.

Returns

gFunction
[float or array] Values of the g-function

`visualize_g_function`()

Plot the g-function of the borefield.

Returns

fig
[figure] Figure object (matplotlib).

`visualize_heat_extraction_rate_profiles`(*time=None*, *iBoreholes=None*, *showTilt=True*)

Plot the heat extraction rate profiles at chosen time.

Parameters

time
[float] Values of time (in seconds) to plot heat extraction rate profiles. If time is None, heat extraction rates are plotted at the last time step. Default is None.

iBoreholes
[list of int] Borehole indices to plot heat extraction rate profiles. If iBoreholes is None, heat extraction rates are plotted for all boreholes. Default is None.

showTilt
[bool] Set to True to show borehole inclination. Default is True

Returns

fig
[figure] Figure object (matplotlib).

`visualize_heat_extraction_rates`(*iBoreholes=None*, *showTilt=True*)

Plot the time-variation of the average heat extraction rates.

Parameters

iBoreholes
[list of int] Borehole indices to plot heat extraction rates. If iBoreholes is None, heat extraction rates are plotted for all boreholes. Default is None.

showTilt
[bool] Set to True to show borehole inclination. Default is True

Returns

fig
[figure] Figure object (matplotlib).

`visualize_temperature_profiles`(*time=None*, *iBoreholes=None*, *showTilt=True*)

Plot the borehole wall temperature profiles at chosen time.

Parameters

time
[float] Values of time (in seconds) to plot temperature profiles. If time is None, temperatures are plotted at the last time step. Default is None.

iBoreholes
[list of int] Borehole indices to plot temperature profiles. If iBoreholes is None, temperatures are plotted for all boreholes. Default is None.

showTilt
[bool] Set to True to show borehole inclination. Default is True

Returns

fig
[figure] Figure object (matplotlib).

`visualize_temperatures`(*iBoreholes=None*, *showTilt=True*)

Plot the time-variation of the average borehole wall temperatures.

Parameters

iBoreholes
[list of int] Borehole indices to plot temperatures. If iBoreholes is None, temperatures are plotted for all boreholes. Default is None.

showTilt
[bool] Set to True to show borehole inclination. Default is True

Returns

fig
[figure] Figure object (matplotlib).

pygfunction.gfunction.`mixed_inlet_temperature`(*network*, *m_flow_network*, *cp_f*, *time*, *alpha*, *kind='linear'*, *nSegments=8*, *segment_ratios=<function segment_ratios>*, *use_similarities=True*, *disTol=0.01*, *tol=1e-06*, *dtype=<class 'numpy.float64'>*, *disp=False*, ***kwargs*)

Evaluate the g-function with mixed inlet fluid temperatures.

This function superimposes the finite line source (FLS) solution to estimate the g-function of a geothermal bore field. Each borehole is modeled as a series of finite line source segments, as proposed in[8]. The piping configurations between boreholes can be any combination of series and parallel connections.

[8] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022

Parameters

network
[Network objects] List of boreholes included in the bore field.

m_flow_network
[float or array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float or array] Fluid specific isobaric heat capacity (in J/kg.degC). Must be the same for all circuits (a single float can be supplied).

time
[float or array] Values of time (in seconds) for which the g-function is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

nSegments
[int or list, optional] Number of line segments used per borehole, or list of number of line segments used for each borehole. Default is 8.

segment_ratios
[array, list of arrays, or callable, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. The shape of the array is of (nSegments,) or list of (nSegments[i],). If segment_ratios==None, segments of equal lengths are considered. Default is None.

kind
[string, optional] Interpolation method used for segment-to-segment thermal response factors. See documentation for scipy.interpolate.interp1d. Default is 'linear'.

use_similarities
[bool, optional] True if similarities are used to limit the number of FLS evaluations. Default is True.

disTol
[float, optional] Relative tolerance on radial distance. Two distances (d1, d2) between two pairs of boreholes are considered equal if the difference between the two distances (abs(d1-d2)) is below tolerance. Default is 0.01.

tol
[float, optional] Relative tolerance on length and depth. Two lengths H1, H2 (or depths D1, D2) are considered equal if abs(H1 - H2)/H2 < tol. Default is 1.0e-6.

dtype
[numpy dtype, optional] numpy data type used for matrices and vectors. Should be one of numpy.single or numpy.double. Default is numpy.double.

disp
[bool, optional] Set to true to print progression messages. Default is False.

Returns

gFunction
[float or array] Values of the g-function

References

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=0., y=0.)
>>> b2 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> Utube1 = gt.pipes.SingleUTube(pos=[(-0.05, 0), (0, -0.05)],
                                  r_in=0.015, r_out=0.02,
                                  borehole=b1,k_s=2, k_g=1, R_fp=0.1)
>>> Utube2 = gt.pipes.SingleUTube(pos=[(-0.05, 0), (0, -0.05)],
                                  r_in=0.015, r_out=0.02,
                                  borehole=b1,k_s=2, k_g=1, R_fp=0.1)
>>> bore_connectivity = [-1, 0]
>>> network = gt.networks.Network([b1, b2], [Utube1, Utube2], bore_connectivity)
>>> time = np.array([1.0*10**i for i in range(4, 12)])
>>> m_flow_network = 0.25
>>> cp_f = 4000.
>>> alpha = 1.0e-6
>>> gt.gfunction.mixed_inlet_temperature(network, m_flow_network, cp_f, time, alpha)
array([0.63782415, 1.63304116, 2.72191316, 4.04091713, 5.98240458,
   7.77216202, 8.66195828, 8.77567215])
```

pygfunction.gfunction.**uniform_heat_extraction**(*boreholes*, *time*, *alpha*, *use_similarities=True*, *disTol=0.01*, *tol=1e-06*, *dtype=<class 'numpy.float64'>*, *disp=False*, ***kwargs*)

Evaluate the g-function with uniform heat extraction along boreholes.

This function superimposes the finite line source (FLS) solution to estimate the g-function of a geothermal bore field. This boundary condition correponds to *BC-I*, as defined by[9].

Parameters

boreholes
[list of Borehole objects] List of boreholes included in the bore field.

time
[float or array] Values of time (in seconds) for which the g-function is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

use_similarities
[bool, optional] True if similarities are used to limit the number of FLS evaluations. Default is True.

disTol
[float, optional] Relative tolerance on radial distance. Two distances (d1, d2) between two pairs of boreholes are considered equal if the difference between the two distances (abs(d1-d2)) is below tolerance. Default is 0.01.

tol
[float, optional] Relative tolerance on length and depth. Two lengths H1, H2 (or depths D1, D2) are considered equal if abs(H1 - H2)/H2 < tol. Default is 1.0e-6.

[9] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

dtype
[numpy dtype, optional] numpy data type used for matrices and vectors. Should be one of numpy.single or numpy.double. Default is numpy.double.

disp
[bool, optional] Set to true to print progression messages. Default is False.

Returns

gFunction
[float or array] Values of the g-function

References

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=0., y=0.)
>>> b2 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> alpha = 1.0e-6
>>> time = np.array([1.0*10**i for i in range(4, 12)])
>>> gt.gfunction.uniform_heat_extraction([b1, b2], time, alpha)
array([ 0.75978163,  1.84860837,  2.98861057,  4.33496051,  6.29199383,
    8.13636888,  9.08401497,  9.20736188])
```

pygfunction.gfunction.**uniform_temperature**(*boreholes*, *time*, *alpha*, *nSegments=8*, *segment_ratios=<function segment_ratios>*, *kind='linear'*, *use_similarities=True*, *disTol=0.01*, *tol=1e-06*, *dtype=<class 'numpy.float64'>*, *disp=False*, ***kwargs*)

Evaluate the g-function with uniform borehole wall temperature.

This function superimposes the finite line source (FLS) solution to estimate the g-function of a geothermal bore field. Each borehole is modeled as a series of finite line source segments. This boundary condition correponds to *BC-III*, as defined by[10].

Parameters

boreholes
[list of Borehole objects] List of boreholes included in the bore field.

time
[float or array] Values of time (in seconds) for which the g-function is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

nSegments
[int or list, optional] Number of line segments used per borehole, or list of number of line segments used for each borehole. Default is 8.

segment_ratios
[array, list of arrays, or callable, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. The shape of the array is of (nSegments,) or list of (nSegments[i],). If segment_ratios==None, segments of equal lengths are considered. If a callable is provided, it must return an array of size (nSegments,) when provided with

[10] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

nSegments (of type int) as an argument, or an array of size (nSegments[i],) when provided with an element of nSegments (of type list). Default is `utilities.segment_ratios()`.

kind
[string, optional] Interpolation method used for segment-to-segment thermal response factors. See documentation for scipy.interpolate.interp1d. Default is 'linear'.

use_similarities
[bool, optional] True if similarities are used to limit the number of FLS evaluations. Default is True.

disTol
[float, optional] Relative tolerance on radial distance. Two distances (d1, d2) between two pairs of boreholes are considered equal if the difference between the two distances (abs(d1-d2)) is below tolerance. Default is 0.01.

tol
[float, optional] Relative tolerance on length and depth. Two lengths H1, H2 (or depths D1, D2) are considered equal if abs(H1 - H2)/H2 < tol. Default is 1.0e-6.

dtype
[numpy dtype, optional] numpy data type used for matrices and vectors. Should be one of numpy.single or numpy.double. Default is numpy.double.

disp
[bool, optional] Set to true to print progression messages. Default is False.

Returns

gFunction
[float or array] Values of the g-function

References

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=0., y=0.)
>>> b2 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> alpha = 1.0e-6
>>> time = np.array([1.0*10**i for i in range(4, 12)])
>>> gt.gfunction.uniform_temperature([b1, b2], time, alpha)
array([ 0.75978079,  1.84859851,  2.98852756,  4.33406497,  6.27830732,
    8.05746656,  8.93697282,  9.04925079])
```

3.3 Heat Transfer Module

pygfunction.heat_transfer.**finite_line_source**(*time*, *alpha*, *borehole1*, *borehole2*, *reaSource=True*, *imgSource=True*, *approximation=False*, *M=11*, *N=10*)

Evaluate the Finite Line Source (FLS) solution.

This function uses a numerical quadrature to evaluate the one-integral form of the FLS solution. For vertical boreholes, the FLS solution was proposed by Claesson and Javed[1] and extended to boreholes with different

[1] Claesson, J., & Javed, S. (2011). An analytical method to calculate borehole fluid temperatures for time-scales from minutes to decades. ASHRAE Transactions, 117(2), 279-288.

vertical positions by Cimmino and Bernier[2]. The FLS solution is given by:

$$
\begin{aligned}
h_{1\to2}(t) &= \frac{1}{2H_2}\int_{\frac{1}{\sqrt{4\alpha t}}}^{\infty} e^{-d_{12}^2 s^2}(I_{real}(s)+I_{imag}(s))ds \\
d_{12} &= \sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \\
I_{real}(s) &= erfint((D_2-D_1+H_2)s) - erfint((D_2-D_1)s) \\
&\quad + erfint((D_2-D_1-H_1)s) - erfint((D_2-D_1+H_2-H_1)s) \\
I_{imag}(s) &= erfint((D_2+D_1+H_2)s) - erfint((D_2+D_1)s) \\
&\quad + erfint((D_2+D_1+H_1)s) - erfint((D_2+D_1+H_2+H_1)s) \\
erfint(X) &= \int_0^X erf(x)dx \\
&= Xerf(X) - \frac{1}{\sqrt{\pi}}(1-e^{-X^2})
\end{aligned}
$$

For inclined boreholes, the FLS solution was proposed by Lazzarotto[4] and Lazzarotto and Björk[5]. The FLS solution is given by:

$$
\begin{aligned}
h_{1\to2}(t) &= \frac{H_1}{2H_2}\int_{\frac{1}{\sqrt{4\alpha t}}}^{\infty}\frac{1}{s}\int_0^1 (I_{real}(u,s)+I_{imag}(u,s))duds \\
I_{real}(u,s) &= e^{-((x_1-x_2)^2+(y_1-y_2)^2+(D_1-D_2)^2)s^2} \\
&\quad \cdot (erf((uH_1k_{0,real}+k_{2,real})s) - erf((uH_1k_{0,real}+k_{2,real}-H_2)s)) \\
&\quad \cdot e^{(u^2H_1^2(k_{0,real}^2-1)+2uH_1(k_{0,real}k_{2,real}-k_{1,real})+k_{2,real}^2)s^2}duds \\
I_{imag}(u,s) &= -e^{-((x_1-x_2)^2+(y_1-y_2)^2+(D_1+D_2)^2)s^2} \\
&\quad \cdot (erf((uH_1k_{0,imag}+k_{2,imag})s) - erf((uH_1k_{0,imag}+k_{2,imag}-H_2)s)) \\
&\quad \cdot e^{(u^2H_1^2(k_{0,imag}^2-1)+2uH_1(k_{0,imag}k_{2,imag}-k_1)+k_{2,imag}^2)s^2}duds \\
k_{0,real} &= sin(\beta_1)sin(\beta_2)cos(\theta_1-\theta_2)+cos(\beta_1)cos(\beta_2) \\
k_{0,imag} &= sin(\beta_1)sin(\beta_2)cos(\theta_1-\theta_2)-cos(\beta_1)cos(\beta_2) \\
k_{1,real} &= sin(\beta_1)(cos(\theta_1)(x_1-x_2)+sin(\theta_1)(y_1-y_2))+cos(\beta_1)(D_1-D_2) \\
k_{1,imag} &= sin(\beta_1)(cos(\theta_1)(x_1-x_2)+sin(\theta_1)(y_1-y_2))+cos(\beta_1)(D_1+D_2) \\
k_{2,real} &= sin(\beta_2)(cos(\theta_2)(x_1-x_2)+sin(\theta_2)(y_1-y_2))+cos(\beta_2)(D_1-D_2) \\
k_{2,imag} &= sin(\beta_2)(cos(\theta_2)(x_1-x_2)+sin(\theta_2)(y_1-y_2))-cos(\beta_2)(D_1+D_2)
\end{aligned}
$$

where β_1 and β_2 are the tilt angle of the boreholes (relative to vertical), and θ_1 and θ_2 are the orientation of the boreholes (relative to the x-axis).

Note: The reciprocal thermal response factor $h_{2\to1}(t)$ can be conveniently calculated by:

$$h_{2\to1}(t) = \frac{H_2}{H_1}h_{1\to2}(t)$$

[2] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

[4] Lazzarotto, A. (2016). A methodology for the calculation of response functions for geothermal fields with arbitrarily oriented boreholes – Part 1, Renewable Energy, 86, 1380-1393.

[5] Lazzarotto, A., & Björk, F. (2016). A methodology for the calculation of response functions for geothermal fields with arbitrarily oriented boreholes – Part 2, Renewable Energy, 86, 1353-1361.

Parameters

time
[float or array, shape (K)] Value of time (in seconds) for which the FLS solution is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

borehole1
[Borehole object or list of Borehole objects, length (N)] Borehole object of the borehole extracting heat.

borehole2
[Borehole object or list of Borehole objects, length (M)] Borehole object for which the FLS is evaluated.

reaSource
[bool] True if the real part of the FLS solution is to be included. Default is True.

imgSource
[bool, optional] True if the image part of the FLS solution is to be included. Default is True.

approximation
[bool, optional] Set to true to use the approximation of the FLS solution of Cimmino (2021)[3]. This approximation does not require the numerical evaluation of any integral. Default is False.

M
[int, optional] Number of Gauss-Legendre sample points for the quadrature over u. This is only used for inclined boreholes. Default is 11.

N
[int, optional] Number of terms in the approximation of the FLS solution. This parameter is unused if *approximation* is set to False. Default is 10. Maximum is 25.

Returns

h
[float or array, shape (M, N, K), (M, N) or (K)] Value of the FLS solution. The average (over the length) temperature drop on the wall of borehole2 due to heat extracted from borehole1 is:

$$\Delta T_{b,2} = T_g - \frac{Q_1}{2\pi k_s H_2} h$$

Notes

The function returns a float if time is a float and borehole1 and borehole2 are Borehole objects. If time is a float and any of borehole1 and borehole2 are lists, the function returns an array, shape (M, N), If time is an array and borehole1 and borehole2 are Borehole objects, the function returns an array, shape (K).If time is an array and any of borehole1 and borehole2 are are lists, the function returns an array, shape (M, N, K).

[3] Cimmino, M. (2021). An approximation of the finite line source solution to model thermal interactions between geothermal boreholes. International Communications in Heat and Mass Transfer, 127, 105496.

References

Examples

```
>>> b1 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=0., y=0.)
>>> b2 = gt.boreholes.Borehole(H=150., D=4., r_b=0.075, x=5., y=0.)
>>> h = gt.heat_transfer.finite_line_source(4*168*3600., 1.0e-6, b1, b2)
h = 0.0110473635393
>>> h = gt.heat_transfer.finite_line_source(
    4*168*3600., 1.0e-6, b1, b2, approximation=True, N=10)
h = 0.0110474667731
>>> b3 = gt.boreholes.Borehole(
    H=150., D=4., r_b=0.075, x=5., y=0., tilt=3.1415/15, orientation=0.)
>>> h = gt.heat_transfer.finite_line_source(
    4*168*3600., 1.0e-6, b1, b3, M=21)
h = 0.0002017450051
```

pygfunction.heat_transfer.**finite_line_source_approximation**(*time*, *alpha*, *dis*, *H1*, *D1*, *H2*, *D2*, *reaSource=True*, *imgSource=True*, *N=10*)

Evaluate the Finite Line Source (FLS) solution using the approximation of Cimmino (2021)[6].

Parameters

time
[float or array, shape (K)] Value of time (in seconds) for which the FLS solution is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

dis
[float or array] Radial distances to evaluate the FLS solution.

H1
[float or array] Lengths of the emitting heat sources.

D1
[float or array] Buried depths of the emitting heat sources.

H2
[float or array] Lengths of the receiving heat sources.

D2
[float or array] Buried depths of the receiving heat sources.

reaSource
[bool, optional] True if the real part of the FLS solution is to be included. Default is True.

imgSource
[bool, optional] True if the image part of the FLS solution is to be included. Default is True.

N
[int, optional] Number of terms in the approximation of the FLS solution. This parameter is unused if *approximation* is set to False. Default is 10. Maximum is 25.

Returns

[6] Cimmino, M. (2021). An approximation of the finite line source solution to model thermal interactions between geothermal boreholes. International Communications in Heat and Mass Transfer, 127, 105496.

h
[float] Value of the FLS solution. The average (over the length) temperature drop on the wall of borehole2 due to heat extracted from borehole1 is:

$$\Delta T_{b,2} = T_g - \frac{Q_1}{2\pi k_s H_2} h$$

References

pygfunction.heat_transfer.**finite_line_source_equivalent_boreholes_vectorized**(*time*, *alpha*, *dis*, *wDis*, *H1*, *D1*, *H2*, *D2*, *N2*, *reaSource=True*, *imgSource=True*)

Evaluate the equivalent Finite Line Source (FLS) solution.

This function uses a numerical quadrature to evaluate the one-integral form of the FLS solution, as proposed by Prieto and Cimmino[7]. The equivalent FLS solution is given by:

$$\begin{aligned}
h_{1\to2}(t) &= \frac{1}{2H_2 N_{b,2}} \int_{\frac{1}{\sqrt{4\alpha t}}}^{\infty} \sum_{G_1} \sum_{G_2} e^{-d_{12}^2 s^2} (I_{real}(s) + I_{imag}(s)) ds \\
I_{real}(s) &= erfint((D_2 - D_1 + H_2)s) - erfint((D_2 - D_1)s) \\
&\quad + erfint((D_2 - D_1 - H_1)s) - erfint((D_2 - D_1 + H_2 - H_1)s) \\
I_{imag}(s) &= erfint((D_2 + D_1 + H_2)s) - erfint((D_2 + D_1)s) \\
&\quad + erfint((D_2 + D_1 + H_1)s) - erfint((D_2 + D_1 + H_2 + H_1)s) \\
erfint(X) &= \int_0^X erf(x) dx \\
&= X erf(X) - \frac{1}{\sqrt{\pi}}(1 - e^{-X^2})
\end{aligned}$$

Note: The reciprocal thermal response factor $h_{2\to1}(t)$ can be conveniently calculated by:

$$h_{2\to1}(t) = \frac{H_2 N_{b,2}}{H_1 N_{b,1}} h_{1\to2}(t)$$

Parameters

time
[float or array, shape (K)] Value of time (in seconds) for which the FLS solution is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

dis
[array] Unique radial distances to evaluate the FLS solution.

[7] Prieto, C., & Cimmino, M. (2021). Thermal interactions in large irregular fields of geothermal boreholes: the method of equivalent borehole. Journal of Building Performance Simulation, 14 (4), 446-460.

wDis
[array] Number of instances of each unique radial distances.

H1
[float or array] Lengths of the emitting heat sources.

D1
[float or array] Buried depths of the emitting heat sources.

H2
[float or array] Lengths of the receiving heat sources.

D2
[float or array] Buried depths of the receiving heat sources.

N2
[float or array,] Number of segments represented by the receiving heat sources.

reaSource
[bool] True if the real part of the FLS solution is to be included. Default is True.

imgSource
[bool] True if the image part of the FLS solution is to be included. Default is True.

Returns

h
[float] Value of the FLS solution. The average (over the length) temperature drop on the wall of borehole2 due to heat extracted from borehole1 is:

$$\Delta T_{b,2} = T_g - \frac{Q_1}{2\pi k_s H_2} h$$

Notes

This is a vectorized version of the *finite_line_source()* function using scipy.integrate.quad_vec to speed up calculations. All arrays (dis, H1, D1, H2, D2) must follow numpy array broadcasting rules. If time is an array, the integrals for different time values are stacked on the last axis.

References

pygfunction.heat_transfer.**finite_line_source_inclined_approximation**(*time*, *alpha*, *rb1*, *x1*, *y1*, *H1*, *D1*, *tilt1*, *orientation1*, *x2*, *y2*, *H2*, *D2*, *tilt2*, *orientation2*, *reaSource=True*, *imgSource=True*, *M=11*, *N=10*)

Evaluate the inclined Finite Line Source (FLS) solution using the approximation method of Cimmino (2021)[8].

Parameters

time
[float or array, shape (K)] Value of time (in seconds) for which the FLS solution is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

[8] Cimmino, M. (2021). An approximation of the finite line source solution to model thermal interactions between geothermal boreholes. International Communications in Heat and Mass Transfer, 127, 105496.

rb1
[array] Radii of the emitting heat sources.

x1
[float or array] x-Positions of the emitting heat sources.

y1
[float or array] y-Positions of the emitting heat sources.

H1
[float or array] Lengths of the emitting heat sources.

D1
[float or array] Buried depths of the emitting heat sources.

tilt1
[float or array] Angles (in radians) from vertical of the emitting heat sources.

orientation1
[float or array] Directions (in radians) of the tilt the emitting heat sources.

x2
[array] x-Positions of the receiving heat sources.

y2
[array] y-Positions of the receiving heat sources.

H2
[float or array] Lengths of the receiving heat sources.

D2
[float or array] Buried depths of the receiving heat sources.

tilt2
[float or array] Angles (in radians) from vertical of the receiving heat sources.

orientation2
[float or array] Directions (in radians) of the tilt the receiving heat sources.

reaSource
[bool, optional] True if the real part of the FLS solution is to be included. Default is True.

imgSource
[bool, optional] True if the image part of the FLS solution is to be included. Default is true.

M
[int, optional] Number of points for the Gauss-Legendre quadrature rule along the receiving heat sources. Default is 21.

N
[int, optional] Number of terms in the approximation of the FLS solution. Default is 10. Maximum is 25.

Returns

h
[float] Value of the FLS solution. The average (over the length) temperature drop on the wall of borehole2 due to heat extracted from borehole1 is:

$$\Delta T_{b,2} = T_g - \frac{Q_1}{2\pi k_s H_2} h$$

References

pygfunction.heat_transfer.**finite_line_source_inclined_vectorized**(*time*, *alpha*, *rb1*, *x1*, *y1*, *H1*, *D1*, *tilt1*, *orientation1*, *x2*, *y2*, *H2*, *D2*, *tilt2*, *orientation2*, *reaSource=True*, *imgSource=True*, *M=11*, *approximation=False*, *N=10*)

Evaluate the inclined Finite Line Source (FLS) solution.

This function uses a numerical quadrature to evaluate the inclined FLS solution, as proposed by Lazzarotto[9]. The inclined FLS solution is given by:

$$
\begin{aligned}
h_{1\to2}(t) &= \frac{H_1}{2H_2}\int_{\frac{1}{\sqrt{4\alpha t}}}^{\infty}\frac{1}{s}\int_0^1 (I_{real}(u,s)+I_{imag}(u,s))duds \\
I_{real}(u,s) &= e^{-((x_1-x_2)^2+(y_1-y_2)^2+(D_1-D_2)^2)s^2} \\
&\quad\cdot(erf((uH_1k_{0,real}+k_{2,real})s)-erf((uH_1k_{0,real}+k_{2,real}-H_2)s)) \\
&\quad\cdot e^{(u^2H_1^2(k_{0,real}^2-1)+2uH_1(k_{0,real}k_{2,real}-k_{1,real})+k_{2,real}^2)s^2}duds \\
I_{imag}(u,s) &= -e^{-((x_1-x_2)^2+(y_1-y_2)^2+(D_1+D_2)^2)s^2} \\
&\quad\cdot(erf((uH_1k_{0,imag}+k_{2,imag})s)-erf((uH_1k_{0,imag}+k_{2,imag}-H_2)s)) \\
&\quad\cdot e^{(u^2H_1^2(k_{0,imag}^2-1)+2uH_1(k_{0,imag}k_{2,imag}-k_1)+k_{2,imag}^2)s^2}duds \\
k_{0,real} &= sin(\beta_1)sin(\beta_2)cos(\theta_1-\theta_2)+cos(\beta_1)cos(\beta_2) \\
k_{0,imag} &= sin(\beta_1)sin(\beta_2)cos(\theta_1-\theta_2)-cos(\beta_1)cos(\beta_2) \\
k_{1,real} &= sin(\beta_1)(cos(\theta_1)(x_1-x_2)+sin(\theta_1)(y_1-y_2))+cos(\beta_1)(D_1-D_2) \\
k_{1,imag} &= sin(\beta_1)(cos(\theta_1)(x_1-x_2)+sin(\theta_1)(y_1-y_2))+cos(\beta_1)(D_1+D_2) \\
k_{2,real} &= sin(\beta_2)(cos(\theta_2)(x_1-x_2)+sin(\theta_2)(y_1-y_2))+cos(\beta_2)(D_1-D_2) \\
k_{2,imag} &= sin(\beta_2)(cos(\theta_2)(x_1-x_2)+sin(\theta_2)(y_1-y_2))-cos(\beta_2)(D_1+D_2)
\end{aligned}
$$

where β_1 and β_2 are the tilt angle of the boreholes (relative to vertical), and θ_1 and θ_2 are the orientation of the boreholes (relative to the x-axis).

Note: The reciprocal thermal response factor $h_{2\to1}(t)$ can be conveniently calculated by:

$$h_{2\to1}(t) = \frac{H_2}{H_1}h_{1\to2}(t)$$

Parameters

time
[float or array, shape (K)] Value of time (in seconds) for which the FLS solution is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

[9] Lazzarotto, A. (2016). A methodology for the calculation of response functions for geothermal fields with arbitrarily oriented boreholes – Part 1, Renewable Energy, 86, 1380-1393.

rb1
[array] Radii of the emitting heat sources.

x1
[float or array] x-Positions of the emitting heat sources.

y1
[float or array] y-Positions of the emitting heat sources.

H1
[float or array] Lengths of the emitting heat sources.

D1
[float or array] Buried depths of the emitting heat sources.

tilt1
[float or array] Angles (in radians) from vertical of the emitting heat sources.

orientation1
[float or array] Directions (in radians) of the tilt the emitting heat sources.

x2
[array] x-Positions of the receiving heat sources.

y2
[array] y-Positions of the receiving heat sources.

H2
[float or array] Lengths of the receiving heat sources.

D2
[float or array] Buried depths of the receiving heat sources.

tilt2
[float or array] Angles (in radians) from vertical of the receiving heat sources.

orientation2
[float or array] Directions (in radians) of the tilt the receiving heat sources.

reaSource
[bool, optional] True if the real part of the FLS solution is to be included. Default is True.

imgSource
[bool, optional] True if the image part of the FLS solution is to be included. Default is true.

M
[int, optional] Number of points for the Gauss-Legendre quadrature rule along the receiving heat sources. Default is 21.

approximation
[bool, optional] Set to true to use the approximation of the FLS solution of Cimmino (2021)[12]. This approximation does not require the numerical evaluation of any integral. Default is False.

N
[int, optional] Number of terms in the approximation of the FLS solution. This parameter is unused if *approximation* is set to False. Default is 10. Maximum is 25.

Returns

[12] Cimmino, M. (2021). An approximation of the finite line source solution to model thermal interactions between geothermal boreholes. International Communications in Heat and Mass Transfer, 127, 105496.

f
[callable] Integrand of the finite line source solution. Can be vector-valued.

Notes

This is a vectorized version of the *finite_line_source()* function using scipy.integrate.quad_vec to speed up calculations. All arrays (x1, y1, H1, D1, tilt1, orientation1, x2, y2, H2, D2, tilt2, orientation2) must follow numpy array broadcasting rules.

References

pygfunction.heat_transfer.**finite_line_source_vectorized**(*time*, *alpha*, *dis*, *H1*, *D1*, *H2*, *D2*, *reaSource=True*, *imgSource=True*, *approximation=False*, *N=10*)

Evaluate the Finite Line Source (FLS) solution.

This function uses a numerical quadrature to evaluate the one-integral form of the FLS solution, as proposed by Claesson and Javed[10] and extended to boreholes with different vertical positions by Cimmino and Bernier[11]. The FLS solution is given by:

$$
\begin{aligned}
h_{1\to2}(t) &= \frac{1}{2H_2}\int_{\frac{1}{\sqrt{4\alpha t}}}^{\infty} e^{-d_{12}^2 s^2}(I_{real}(s)+I_{imag}(s))ds \\
I_{real}(s) &= erfint((D_2-D_1+H_2)s) - erfint((D_2-D_1)s) \\
&\quad + erfint((D_2-D_1-H_1)s) - erfint((D_2-D_1+H_2-H_1)s) \\
I_{imag}(s) &= erfint((D_2+D_1+H_2)s) - erfint((D_2+D_1)s) \\
&\quad + erfint((D_2+D_1+H_1)s) - erfint((D_2+D_1+H_2+H_1)s) \\
erfint(X) &= \int_0^X erf(x)dx \\
&= Xerf(X) - \frac{1}{\sqrt{\pi}}(1-e^{-X^2})
\end{aligned}
$$

Note: The reciprocal thermal response factor $h_{2\to1}(t)$ can be conveniently calculated by:

$$h_{2\to1}(t) = \frac{H_2}{H_1}h_{1\to2}(t)$$

Parameters

time
[float or array, shape (K)] Value of time (in seconds) for which the FLS solution is evaluated.

alpha
[float] Soil thermal diffusivity (in m2/s).

[10] Claesson, J., & Javed, S. (2011). An analytical method to calculate borehole fluid temperatures for time-scales from minutes to decades. ASHRAE Transactions, 117(2), 279-288.

[11] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

dis
[float or array] Radial distances to evaluate the FLS solution.

H1
[float or array] Lengths of the emitting heat sources.

D1
[float or array] Buried depths of the emitting heat sources.

H2
[float or array] Lengths of the receiving heat sources.

D2
[float or array] Buried depths of the receiving heat sources.

reaSource
[bool] True if the real part of the FLS solution is to be included. Default is True.

imgSource
[bool] True if the image part of the FLS solution is to be included. Default is True.

approximation
[bool, optional] Set to true to use the approximation of the FLS solution of Cimmino (2021)[Page 32, 12]. This approximation does not require the numerical evaluation of any integral. Default is False.

N
[int, optional] Number of terms in the approximation of the FLS solution. This parameter is unused if *approximation* is set to False. Default is 10. Maximum is 25.

Returns

h
[float] Value of the FLS solution. The average (over the length) temperature drop on the wall of borehole2 due to heat extracted from borehole1 is:

$$\Delta T_{b,2} = T_g - \frac{Q_1}{2\pi k_s H_2} h$$

Notes

This is a vectorized version of the *finite_line_source()* function using scipy.integrate.quad_vec to speed up calculations. All arrays (dis, H1, D1, H2, D2) must follow numpy array broadcasting rules. If time is an array, the integrals for different time values are stacked on the last axis.

References

3.4 Load Aggregation Module

class pygfunction.load_aggregation.**ClaessonJaved**(*dt*, *tmax*, *nSources=1*, *cells_per_level=5*, ***kwargs*)

Bases: _LoadAggregation

Load aggregation algorithm of Claesson and Javed[1].

[1] Claesson, J., & Javed, S. (2012). A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. ASHRAE Transactions, 118 (1): 530–539.

References

Attributes

dt
[float] Simulation time step (in seconds).

tmax
[float] Maximum simulation time (in seconds).

nSources
[int, optional] Number of heat sources with independent load histories. Default is 1.

cells_per_level
[int, optional] Number of load aggregation cells per level of aggregation. Cell widths double every cells_per_level cells. Default is 5.

`get_thermal_response_factor_increment()`

Returns an array of the **dimensional** thermal response factors.

Returns

dg
[array] Array of **dimensional** thermal response factor increments used for temporal superposition ($g(t_{i+1})/(2\pi k_s) - g(t_i)/(2\pi k_s)$), in correspondance with the intialized values of the thermal response factors in `initialize()`. The output size of the array is (nSources, nSources, Nt) if nSources>1. If nSources=1, then the method returns a 1d array.

`get_times_for_simulation()`

Returns a vector of time values at which the thermal response factors are required.

Returns

time_req
[array] Time values at which the thermal response factors are required (in seconds).

`initialize(`*g_d*`)`

Initialize the thermal aggregation scheme.

Creates a matrix of thermal response factor increments for later use in temporal superposition.

Parameters

g_d
[array] Matrix of **dimensional** thermal response factors for temporal superposition ($g/(2\pi k_s)$). The expected size is (nSources, nSources, Nt), where Nt is the number of time values at which the thermal response factors are required. The time values are returned by `get_times_for_simulation()`. If nSources=1, g_d can be 1 dimensional.

`next_time_step(`*time*`)`

Shifts aggregated loads by one time step.

Parameters

time
[float] Current value of time (in seconds).

`set_current_load(`*q_b*`)`

Set the load at the current time step.

Parameters

q_b
[float or array] Current value of heat extraction rates per unit borehole length (in watts per meter).

`temporal_superposition()`

Returns the borehole wall temperature variations at the current time step from the temporal superposition of past loads.

Returns

deltaT
[array] Values of borehole wall temperature drops at the current time step (in degC).

Note: *pygfunction* assumes positive values for heat **extraction** and for borehole wall temperature **drops**. The borehole wall temperature are thus given by : $T_b = T_g - \Delta T_b$.

`class pygfunction.load_aggregation.Liu(dt, tmax, nSources=1, N1=24, N2=5, N3=73, W1=12, W2=3, W3=40)`

Bases: `_LoadAggregation`

Hierarchical load aggregation algorithm of Liu[2].

References

Attributes

dt
[float] Simulation time step (in seconds).

tmax
[float] Maximum simulation time (in seconds).

nSources
[int, optional] Number of heat sources with independent load histories. Default is 1.

N1
[int, optional] Size (number of time steps) of small load aggregation cell. Default is 24.

N2
[int, optional] Size (number of small cells) of medium load aggregation cell. Default is 5.

N3
[int, optional] Size (number of medium cells) of large load aggregation cell. Default is 73.

W1
[int, optional] Waiting period (number of time steps) for the creation of a small load aggregation cell. Default is 12.

W2
[int, optional] Waiting period (number of small cells) for the creation of a medium load aggregation cell. Default is 3.

W3
[int, optional] Waiting period (number of medium cells) for the creation of a large load aggregation cell. Default is 40.

[2] Liu, X. (2005). Development and experimental validation of simulation of hydronic snow melting systems for bridges. Ph.D. Thesis. Oklahoma State University.

`get_times_for_simulation()`

Returns a vector of time values at which the thermal response factors are required.

Returns

time_req
[array] Time values at which the thermal response factors are required (in seconds).

`initialize(g_d)`

Initialize the load aggregation scheme.

Creates a matrix of thermal response factor for later use in temporal superposition.

Parameters

g_d
[array] Matrix of **dimensional** thermal response factors for temporal superposition ($g/(2\pi k_s)$). The expected size is (nSources, nSources, Nt), where Nt is the number of time values at which the thermal response factors are required. The time values are returned by `get_times_for_simulation()`. If nSources=1, g_d can be 1 dimensional.

`next_time_step(time)`

Shifts aggregated loads by one time step.

Parameters

time
[float] Current value of time (in seconds).

`set_current_load(q_b)`

Set the load at the current time step.

Parameters

q_b
[array] Current value of heat extraction rates per unit borehole length (in watts per meter).

`temporal_superposition()`

Returns the borehole wall temperature variations at the current time step from the temporal superposition of past loads.

Returns

deltaT
[array] Values of borehole wall temperature drops at the current time step (in degC).

Note: *pygfunction* assumes positive values for heat **extraction** and for borehole wall temperature **drops**. The borehole wall temperature are thus given by : $T_b = T_g - \Delta T_b$.

`class pygfunction.load_aggregation.MLAA(dt, tmax, nSources=1, N0=12, N1=48, N2=168, N3=360, **kwargs)`

Bases: `_LoadAggregation`

Multiple load aggregation algorithm (MLAA) of Bernier et al.[3].

[3] Bernier, M., Pinel, P., Labib, R. and Paillot, R. (2004). A multiple load aggregation algorithm for annual hourly simulations of GCHP systems. HVAC&R Research 10 (4): 471–487.

References

Attributes

dt
[float] Simulation time step (in seconds).

tmax
[float] Maximum simulation time (in seconds).

nSources
[int, optional] Number of heat sources with independent load histories. Default is 1.

N0
[int, optional] Number of non-aggregated loads. Default is 12.

N1
[int, optional] Number of time steps in first aggregation cell. Default is 48.

N2
[int, optional] Number of time steps in second aggregation cell. Default is 168.

N3
[int, optional] Number of time steps in third aggregation cell. Default is 360.

`get_times_for_simulation()`

Returns a vector of time values at which the thermal response factors are required.

Returns

time_req
[array] Time values at which the thermal response factors are required (in seconds).

`initialize(`g_d`)`

Initialize the load aggregation scheme.

Creates a matrix of thermal response factor for later use in temporal superposition.

Parameters

g_d
[array] Matrix of **dimensional** thermal response factors for temporal superposition ($g/(2\pi k_s)$). The expected size is (nSources, nSources, Nt), where Nt is the number of time values at which the thermal response factors are required. The time values are returned by `get_times_for_simulation()`. If nSources=1, g_d can be 1 dimensional.

`next_time_step(`*time*`)`

Shifts aggregated loads by one time step.

Parameters

time
[float] Current value of time (in seconds).

`set_current_load(`q_b`)`

Set the load at the current time step.

Parameters

q_b
[array] Current value of heat extraction rates per unit borehole length (in watts per meter).

`temporal_superposition()`

Returns the borehole wall temperature variations at the current time step from the temporal superposition of past loads.

Returns

deltaT
[array] Values of borehole wall temperature drops at the current time step (in degC).

Note: *pygfunction* assumes positive values for heat **extraction** and for borehole wall temperature **drops**. The borehole wall temperature are thus given by : $T_b = T_g - \Delta T_b$.

3.5 Media Module

`class pygfunction.media.Fluid(`*fluid_str: str*, *percent: float*, *T: float = 20.0*`)`

Bases: `object`

An object for handling the fluid properties

Parameters

fluid_str: str

The mixer for this application should be one of:

- 'Water' - Complete water solution
- 'MEG' - Ethylene glycol mixed with water
- 'MPG' - Propylene glycol mixed with water
- 'MEA' - Ethanol mixed with water
- 'MMA' - Methanol mixed with water

percent: float
Mass fraction of the mixing fluid added to water (in %). Lower bound = 0. Upper bound is dependent on the mixture.

T: float, optional
The temperature of the fluid (in Celcius). Default is 20 degC.

Examples

```
>>> import pygfunction as gt
>>> T_f = 20.      # Temp at 20 C
```

```
>>> # complete water solution
>>> fluid_str = 'Water'
>>> percent = 0
>>> fluid = gt.media.Fluid(fluid_str, percent, T=T_f)
>>> print(fluid)
```

```
>>> # 20 % propylene glycol mixed with water
>>> fluid_str = 'MPG'
>>> percent = 20
>>> fluid = gt.media.Fluid(fluid_str, percent, T=T_f)
```

```
>>> # 60% ethylene glycol mixed with water
>>> fluid_str = 'MEG'
>>> percent = 60
>>> fluid = gt.media.Fluid(fluid_str, percent, T=T_f)
>>> print(fluid)
```

```
>>> # 5% methanol mixed with water
>>> fluid_str = 'MMA'
>>> percent = 5
>>> fluid = gt.media.Fluid(fluid_str, percent, T=T_f)
>>> print(fluid)
```

```
>>> # ethanol / water
>>> fluid_str = 'MEA'
>>> percent = 10
>>> fluid = gt.media.Fluid(fluid_str, percent, T=T_f)
>>> print(fluid)
```

Prandlt_number()

Returns the Prandtl of the fluid.

Returns

Pr
[float] Prandlt number.

append_to_dict(*dnary*)

density()

Returns the density of the fluid (in kg/m3).

Returns

rho
[float] Density (in kg/m3).

dynamic_viscosity()

Returns the dynamic viscosity of the fluid (in Pa.s, or N.s/m2).

Returns

mu
[float] Dynamic viscosity (in Pa.s, or N.s/m2).

kinematic_viscosity()

Returns the kinematic viscosity of the fluid (in m2/s).

Returns

nu
[float] Kinematic viscosity (in m2/s).

`specific_heat_capacity()`

Returns the specific isobaric heat capacity of the fluid (J/kg.K).

Returns

cp
[float] Specific isobaric heat capacity (J/kg.K).

`thermal_conductivity()`

Returns the thermal conductivity of the fluid (in W/m.K).

Returns

k
[float] Thermal conductivity (in W/m.K).

`volumetric_heat_capacity()`

Returns the volumetric heat capacity of the fluid (J/m3.K).

Returns

rhoCp
[float] Volumetric heat capacity (in J/m3.K).

3.6 Networks Module

`class pygfunction.networks.Network(`*boreholes*, *pipes*, *bore_connectivity=None*, *m_flow_network=None*, *cp_f=None*, *nSegments=None*, *segment_ratios=None*`)`

Bases: `object`

Class for networks of boreholes with series, parallel, and mixed connections between the boreholes.

Contains information regarding the physical dimensions and thermal characteristics of the pipes and the grout material in each boreholes, the topology of the connections between boreholes, as well as methods to evaluate fluid temperatures and heat extraction rates based on the work of Cimmino (2018, 2019)[1],[2].

Notes

The expected array shapes of input parameters and outputs are documented for each class method. *nInlets* and *nOutlets* are the number of inlets and outlets to the network, and both correspond to the number of parallel circuits. *nTotalSegments* is the sum of the number of discretized segments along every borehole. *nBoreholes* is the total number of boreholes in the network.

[1] Cimmino, M. (2018). g-Functions for bore fields with mixed parallel and series connections considering the axial fluid temperature variations. Proceedings of the IGSHPA Sweden Research Track 2018. Stockholm, Sweden. pp. 262-270.

[2] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022.

References

Attributes

boreholes
[list of Borehole objects] List of boreholes included in the bore field.

pipes
[list of pipe objects] List of pipes included in the bore field.

bore_connectivity
[list, optional] Index of fluid inlet into each borehole. -1 corresponds to a borehole connected to the bore field inlet. If this parameter is not provided, parallel connections between boreholes is used. Default is None.

m_flow_network
[float or array, optional] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits. This parameter is used to initialize the coefficients if it is provided. Default is None.

cp_f
[float, optional] Fluid specific isobaric heat capacity (in J/kg.degC). This parameter is used to initialize the coefficients if it is provided. Default is None.

nSegments
[int, optional] Number of line segments used per borehole. This parameter is used to initialize the coefficients if it is provided. Default is None.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

`coefficients_borehole_heat_extraction_rate`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates of all boreholes segments.

Returns coefficients for the relation:

$$\mathbf{Q_b} = \mathbf{a_{in}} T_{f,network,in} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nTotalSegments, 1,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nTotalSegments, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_fluid_heat_extraction_rate`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates of all boreholes.

Returns coefficients for the relation:

$$\mathbf{Q_f} = \mathbf{a_{in}} T_{f,network,in} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nBoreholes, 1,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nBoreholes, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_inlet_temperature`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate intlet fluid temperatures of all boreholes.

Returns coefficients for the relation:

$$\mathbf{T_{f,borehole,in}} = \mathbf{a_{in}} T_{f,network,in} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nBoreholes, 1,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nBoreholes, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_network_heat_extraction_rate`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate total heat extraction rate of the network.

Returns coefficients for the relation:

$$\mathbf{Q_n etwork} = \mathbf{a_{in}} T_{f,network,in} + \mathbf{a_b T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(1, 1,) array] Array of coefficients for inlet fluid temperature.

a_b
[(1, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_network_inlet_temperature`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate intlet fluid temperature of the network.

Returns coefficients for the relation:

$$\mathbf{T_{f,network,in}} = \mathbf{a_{q,f}} Q_f + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_qf
[(1, 1,) array] Array of coefficients for total heat extraction rate.

a_b
[(1, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_network_outlet_temperature`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate outlet fluid temperature of the network.

Returns coefficients for the relation:

$$\mathbf{T_{f,network,out}} = \mathbf{a_{in}} T_{f,network,in} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(1, 1,) array] Array of coefficients for inlet fluid temperature.

a_b
[(1, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_outlet_temperature`(*m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate outlet fluid temperatures of all boreholes.

Returns coefficients for the relation:

$$\mathbf{T_{f,borehole,out}} = \mathbf{a_{in}} T_{f,network,in} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nBoreholes, 1,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nBoreholes, nTotalSegments,) array] Array of coefficients for borehole wall temperatures.

`get_borehole_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the heat extraction rates of all boreholes.

Parameters

T_f_in
[float or (1,) array] Inlet fluid temperatures into network (in Celsius).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_b
[(nTotalSegments,) array] Heat extraction rates along each borehole segment (in Watts).

`get_fluid_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the total heat extraction rates of all boreholes.

Parameters

T_f_in
[float or (1,) array] Inlet fluid temperatures into network (in Celsius).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_f
[(nBoreholes,) array] Total heat extraction rates from each borehole (in Watts).

`get_inlet_temperature`(*T_f_in*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the inlet fluid temperatures of all boreholes.

Parameters

T_f_in
[float or (1,) array] Inlet fluid temperatures into network (in Celsius).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_in
[(nBoreholes,) array] Inlet fluid temperature (in Celsius) into each borehole.

`get_network_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the total heat extraction rate of the network.

Parameters

T_f_in
[float or (1,) array] Inlet fluid temperatures into network (in Celsius).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_t
[float or (1,) array] Heat extraction rate of the network (in Watts). The returned type corresponds to the type of the parameter *Tin*.

`get_network_inlet_temperature`(*Q_t*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the inlet fluid temperature of the network.

Parameters

Q_t
[float or (1,) array] Total heat extraction rate from the network (in Watts).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_in
[float or (1,) array] Inlet fluid temperature (in Celsius) into the network. The returned type corresponds to the type of the parameter *Qt*.

`get_network_outlet_temperature`(*T_f_in*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the outlet fluid temperature of the network.

Parameters

T_f_in
[float or (1,) array] Inlet fluid temperatures into network (in Celsius).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_out
[float or (1,) array] Outlet fluid temperature (in Celsius) from the network. The returned type corresponds to the type of the parameter *Tin*.

`get_outlet_temperature`(*T_f_in*, *T_b*, *m_flow_network*, *cp_f*, *nSegments*, *segment_ratios=None*)

Returns the outlet fluid temperatures of all boreholes.

Parameters

T_f_in
[float or (1,) array] Inlet fluid temperatures into network (in Celsius).

T_b
[float or (nTotalSegments,) array] Borehole wall temperatures (in Celsius). If a float is supplied, the same temperature is applied to all segments of all boreholes.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int or list] Number of borehole segments for each borehole. If an int is supplied, all boreholes are considered to have the same number of segments.

segment_ratios
(nSegments,) array or list of (nSegments[i],) arrays, optional
Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_out
[(nBoreholes,) array] Outlet fluid temperatures (in Celsius) from each borehole.

pygfunction.networks.`network_thermal_resistance`(*network*, *m_flow_network*, *cp_f*)

Evaluate the effective bore field thermal resistance.

As proposed in Cimmino (2018, 2019)[Page 41, 1],[Page 41, 2].

Parameters

network
[network object] Model of the network.

m_flow_network
[float or (nInlets,) array] Total mass flow rate into the network or inlet mass flow rates into each circuit of the network (in kg/s). If a float is supplied, the total mass flow rate is split equally into all circuits.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.degC).

Returns

R_field
[float] Effective bore field thermal resistance (m.K/W).

3.7 Pipes Module

class pygfunction.pipes.**Coaxial**(*pos*, *r_in*, *r_out*, *borehole*, *k_s*, *k_g*, *R_ff*, *R_fp*, *J=2*)

Bases: *SingleUTube*

Class for coaxial boreholes.

Contains information regarding the physical dimensions and thermal characteristics of the pipes and the grout material, as well as methods to evaluate fluid temperatures and heat extraction rates based on the work of Hellstrom[1]. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom[2].

Notes

The expected array shapes of input parameters and outputs are documented for each class method. *nInlets* and *nOutlets* are the number of inlets and outlets to the borehole, and both are equal to 1 for a coaxial borehole. *nSegments* is the number of discretized segments along the borehole. *nPipes* is the number of pipes (i.e. the number of U-tubes) in the borehole, equal to 1. *nDepths* is the number of depths at which temperatures are evaluated.

The effective borehole thermal resistance is evaluated using the method of Cimmino[3]. This is valid for any number of pipes.

References

Attributes

pos
[tuple] Position (x, y) (in meters) of the pipes inside the borehole.

r_in
[(2,) array] Inner radii (in meters) of the coaxial pipes. The first element of the array corresponds to the inlet pipe.

r_out
[(2,) array] Outer radii (in meters) of the coaxial pipes. The first element of the array corresponds to the inlet pipe.

borehole
[Borehole object] Borehole class object of the borehole containing the U-Tube.

k_s
[float] Soil thermal conductivity (in W/m-K).

[1] Hellstrom, G. (1991). Ground heat storage. Thermal Analyses of Duct Storage Systems I: Theory. PhD Thesis. University of Lund, Department of Mathematical Physics. Lund, Sweden.

[2] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

[3] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022.

k_g
[float] Grout thermal conductivity (in W/m-K).

R_ff
[float] Fluid to fluid thermal resistance of the inner pipe to the outer pipe (in m-K/W).

R_fp
[float] Fluid to outer pipe wall thermal resistance of the outer pipe in contact with the grout (in m-K/W).

J
[int, optional] Number of multipoles per pipe to evaluate the thermal resistances. Default is 2.

nPipes
[int] Number of U-Tubes, equals to 1.

nInlets
[int] Total number of pipe inlets, equals to 1.

nOutlets
[int] Total number of pipe outlets, equals to 1.

`coefficients_borehole_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_b} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nSegments, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nSegments, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_fluid_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_f} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_inlet_temperature`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate inlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,in}} = \mathbf{a_{q,f}} \mathbf{Q_f} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_qf
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

coefficients_outlet_temperature(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate outlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,out}} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

coefficients_temperature(*z*, *m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate fluid temperatures at a depth (z).

Returns coefficients for the relation:

$$\mathbf{T_f}(z) = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperature coefficients.

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
(2*nPipes, nInlets,) array, or (nDepths, 2*nPipes, nInlets,) array
Array of coefficients for inlet fluid temperature.

a_b
(2*nPipes, nSegments,) array, or (nDepths, 2*nPipes, nSegments,) array
Array of coefficients for borehole wall temperatures.

`effective_borehole_thermal_resistance`(*m_flow_borehole*, *cp_f*)

Evaluate the effective borehole thermal resistance, defined by:

$$\frac{Q_b}{H} = \frac{T_b^* - \bar{T}_f}{R_b^*}$$
$$\bar{T}_f = \frac{1}{2}(T_{f,in} + T_{f,out})$$

where Q_b is the borehole heat extraction rate (in Watts), H is the borehole length, T_b^* is the effective borehole wall temperature, R_b^* is the effective borehole thermal resistance, $T_{f,in}$ is the inlet fluid temperature, and $T_{f,out}$ is the outlet fluid temperature.

Parameters

m_flow_borehole
[float] Fluid mass flow rate (in kg/s) into the borehole.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.K)

Returns

R_b
[float] Effective borehole thermal resistance (in m.K/W).

`get_borehole_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_b
[float or (nSegments,) array] Heat extraction rates along each borehole segment (in Watts). The returned type corresponds to the type of the parameter *T_b*.

`get_fluid_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_f
[float or (nOutlets,) array] Heat extraction rates from each fluid circuit (in Watts). The returned type corresponds to the type of the parameter *T_f_in*.

`get_inlet_temperature`(*Q_f*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the inlet fluid temperatures of the borehole.

Parameters

Q_f
[float or (nInlets,) array] Heat extraction from the fluid circuits (in Watts).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_in
[float or (nOutlets,) array] Inlet fluid temperatures (in Celsius) into each inlet pipe. The returned type corresponds to the type of the parameter *Q_f*.

`get_outlet_temperature`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the outlet fluid temperatures of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_out
[float or (nOutlets,) array] Outlet fluid temperatures (in Celsius) from each outlet pipe. The returned type corresponds to the type of the parameter *T_f_in*.

`get_temperature`(*z*, *T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the fluid temperatures of the borehole at a depth (z).

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperatures.

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f
[(2*nPipes,) or (nDepths, 2*nPipes,) array] Fluid temperature (in Celsius) in each pipe. The returned shape depends on the type of the parameter *z*.

`get_total_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the total heat extraction rate of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_t
[float] Total net heat extraction rate of the borehole (in Watts).

`local_borehole_thermal_resistance()`

Evaluate the local (cross-sectional) borehole thermal resistance, defined by:

$$Q_b'(z) = \frac{T_b(z) - \bar{T}_f(z)}{R_b}$$

where $Q_b'(z)$ is the borehole heat extraction rate per unit depth at a depth (z), $T_b(z)$ is the borehole wall temperature, $\bar{T}_f(z)$ is the arithmetic mean fluid temperature and R_b is the local borehole thermal resistance.

Returns

R_b
[float] Local borehole thermal resistance (in m.K/W).

`update_thermal_resistances(R_ff, R_fp)`

Update the delta-circuit of thermal resistances.

This methods updates the values of the delta-circuit thermal resistances based on the provided fluid to fluid and fluid to outer pipe wall thermal resistances.

Parameters

R_ff
[float] Fluid to fluid thermal resistance of the inner pipe to the outer pipe (in m-K/W).

R_fp
[float] Fluid to outer pipe wall thermal resistance of the outer pipe in contact with the grout (in m-K/W).

`visualize_pipes()`

Plot the cross-section view of the borehole.

Returns

fig
[figure] Figure object (matplotlib).

`class pygfunction.pipes.IndependentMultipleUTube(pos, r_in, r_out, borehole, k_s, k_g, R_fp, nPipes, J=2)`

Bases: *MultipleUTube*

Class for multiple U-Tube boreholes with independent U-tubes.

Contains information regarding the physical dimensions and thermal characteristics of the pipes and the grout material, as well as methods to evaluate fluid temperatures and heat extraction rates based on the work of Cimmino[4] for boreholes with any number of U-tubes. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom[5].

Notes

The expected array shapes of input parameters and outputs are documented for each class method. *nInlets* and *nOutlets* are the number of inlets and outlets to the borehole, and both are equal to the number of pipes. *nSegments* is the number of discretized segments along the borehole. *nPipes* is the number of pipes (i.e. the number of U-tubes) in the borehole. *nDepths* is the number of depths at which temperatures are evaluated.

References

Attributes

pos
[list of tuples] Position (x, y) (in meters) of the pipes inside the borehole.

r_in
[float] Inner radius (in meters) of the U-Tube pipes.

r_out
[float] Outer radius (in meters) of the U-Tube pipes.

borehole
[Borehole object] Borehole class object of the borehole containing the U-Tube.

k_s
[float] Soil thermal conductivity (in W/m-K).

k_g
[float] Grout thermal conductivity (in W/m-K).

R_fp
[float] Fluid to outer pipe wall thermal resistance (m-K/W).

J
[int, optional] Number of multipoles per pipe to evaluate the thermal resistances. Default is 2.

nPipes
[int] Number of U-Tubes.

nInlets
[int] Total number of pipe inlets, equals to nPipes.

nOutlets
[int] Total number of pipe outlets, equals to nPipes.

`coefficients_borehole_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

[4] Cimmino, M. (2016). Fluid and borehole wall temperature profiles in vertical geothermal boreholes with multiple U-tubes. Renewable Energy, 96, 137-147.

[5] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_b} = \mathbf{a_{in}}\mathbf{T_{f,in}} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nSegments, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nSegments, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_fluid_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_f} = \mathbf{a_{in}}\mathbf{T_{f,in}} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_inlet_temperature`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate inlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,in}} = \mathbf{a_{q,f}} \mathbf{Q_f} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_qf
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_outlet_temperature`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate outlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,out}} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_temperature`(*z*, *m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate fluid temperatures at a depth (z).

Returns coefficients for the relation:

$$\mathbf{T_f}(z) = \mathbf{a_{in}}\mathbf{T_{f,in}} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperature coefficients.

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
(2*nPipes, nInlets,) array, or (nDepths, 2*nPipes, nInlets,) array
Array of coefficients for inlet fluid temperature.

a_b
(2*nPipes, nSegments,) array, or (nDepths, 2*nPipes, nSegments,) array
Array of coefficients for borehole wall temperatures.

`effective_borehole_thermal_resistance`(*m_flow_borehole*, *cp_f*)

Evaluate the effective borehole thermal resistance, defined by:

$$\frac{Q_b}{H} = \frac{T_b^* - \bar{T}_f}{R_b^*}$$

$$\bar{T}_f = \frac{1}{2}(T_{f,in} + T_{f,out})$$

where Q_b is the borehole heat extraction rate (in Watts), H is the borehole length, T_b^* is the effective borehole wall temperature, R_b^* is the effective borehole thermal resistance, $T_{f,in}$ is the inlet fluid temperature, and $T_{f,out}$ is the outlet fluid temperature.

Parameters

m_flow_borehole
[float] Fluid mass flow rate (in kg/s) into the borehole.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.K)

Returns

R_b
[float] Effective borehole thermal resistance (in m.K/W).

`get_borehole_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_b
[float or (nSegments,) array] Heat extraction rates along each borehole segment (in Watts). The returned type corresponds to the type of the parameter *T_b*.

`get_fluid_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment.

The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_f
[float or (nOutlets,) array] Heat extraction rates from each fluid circuit (in Watts). The returned type corresponds to the type of the parameter *T_f_in*.

`get_inlet_temperature`(*Q_f*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the inlet fluid temperatures of the borehole.

Parameters

Q_f
[float or (nInlets,) array] Heat extraction from the fluid circuits (in Watts).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_in
[float or (nOutlets,) array] Inlet fluid temperatures (in Celsius) into each inlet pipe. The returned type corresponds to the type of the parameter *Q_f*.

`get_outlet_temperature`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the outlet fluid temperatures of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_out
[float or (nOutlets,) array] Outlet fluid temperatures (in Celsius) from each outlet pipe. The returned type corresponds to the type of the parameter *T_f_in*.

`get_temperature`(*z*, *T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the fluid temperatures of the borehole at a depth (z).

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperatures.

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f
[(2*nPipes,) or (nDepths, 2*nPipes,) array] Fluid temperature (in Celsius) in each pipe. The returned shape depends on the type of the parameter *z*.

`get_total_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the total heat extraction rate of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_t
[float] Total net heat extraction rate of the borehole (in Watts).

`local_borehole_thermal_resistance`()

Evaluate the local (cross-sectional) borehole thermal resistance, defined by:

$$Q'_b(z) = \frac{T_b(z) - \bar{T}_f(z)}{R_b}$$

where $Q'_b(z)$ is the borehole heat extraction rate per unit depth at a depth (z), $T_b(z)$ is the borehole wall temperature, $\bar{T}_f(z)$ is the arithmetic mean fluid temperature and R_b is the local borehole thermal resistance.

Returns

R_b
[float] Local borehole thermal resistance (in m.K/W).

`update_thermal_resistances`(*R_fp*)

Update the delta-circuit of thermal resistances.

This methods updates the values of the delta-circuit thermal resistances based on the provided fluid to outer pipe wall thermal resistance.

Parameters

R_fp
[float] Fluid to outer pipe wall thermal resistance (m-K/W).

`visualize_pipes`()

Plot the cross-section view of the borehole.

Returns

fig
[figure] Figure object (matplotlib).

`class pygfunction.pipes.MultipleUTube`(*pos*, *r_in*, *r_out*, *borehole*, *k_s*, *k_g*, *R_fp*, *nPipes*, *config='parallel'*, *J=2*)

Bases: `_BasePipe`

Class for multiple U-Tube boreholes.

Contains information regarding the physical dimensions and thermal characteristics of the pipes and the grout material, as well as methods to evaluate fluid temperatures and heat extraction rates based on the work of Cimmino[6] for boreholes with any number of U-tubes. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom[7].

Notes

The expected array shapes of input parameters and outputs are documented for each class method. *nInlets* and *nOutlets* are the number of inlets and outlets to the borehole, and both are equal to 1 for a multiple U-tube borehole. *nSegments* is the number of discretized segments along the borehole. *nPipes* is the number of pipes (i.e. the number of U-tubes) in the borehole. *nDepths* is the number of depths at which temperatures are evaluated.

The effective borehole thermal resistance is evaluated using the method of Cimmino[8]. This is valid for any number of pipes.

[6] Cimmino, M. (2016). Fluid and borehole wall temperature profiles in vertical geothermal boreholes with multiple U-tubes. Renewable Energy, 96, 137-147.

[7] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

[8] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022.

References

Attributes

pos
[list of tuples] Position (x, y) (in meters) of the pipes inside the borehole.

r_in
[float] Inner radius (in meters) of the U-Tube pipes.

r_out
[float] Outer radius (in meters) of the U-Tube pipes.

borehole
[Borehole object] Borehole class object of the borehole containing the U-Tube.

k_s
[float] Soil thermal conductivity (in W/m-K).

k_g
[float] Grout thermal conductivity (in W/m-K).

R_fp
[float] Fluid to outer pipe wall thermal resistance (m-K/W).

J
[int, optional] Number of multipoles per pipe to evaluate the thermal resistances. Default is 2.

nPipes
[int] Number of U-Tubes.

config
[str, defaults to 'parallel']

Configuration of the U-Tube pipes:
'parallel' : U-tubes are connected in parallel. 'series' : U-tubes are connected in series.

nInlets
[int] Total number of pipe inlets, equals to 1.

nOutlets
[int] Total number of pipe outlets, equals to 1.

`coefficients_borehole_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_b} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nSegments, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nSegments, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_fluid_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_f} = \mathbf{a_{in}}\mathbf{T_{f,in}} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_inlet_temperature`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate inlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,in}} = \mathbf{a_{q,f}}\mathbf{Q_f} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_qf
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_outlet_temperature`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate outlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,out}} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_temperature`(*z*, *m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate fluid temperatures at a depth (z).

Returns coefficients for the relation:

$$\mathbf{T_f}(z) = \mathbf{a_{in}}\mathbf{T_{f,in}} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperature coefficients.

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
(2*nPipes, nInlets,) array, or (nDepths, 2*nPipes, nInlets,) array
Array of coefficients for inlet fluid temperature.

a_b
(2*nPipes, nSegments,) array, or (nDepths, 2*nPipes, nSegments,) array
Array of coefficients for borehole wall temperatures.

`effective_borehole_thermal_resistance`(*m_flow_borehole*, *cp_f*)

Evaluate the effective borehole thermal resistance, defined by:

$$\frac{Q_b}{H} = \frac{T_b^* - \bar{T}_f}{R_b^*}$$
$$\bar{T}_f = \frac{1}{2}(T_{f,in} + T_{f,out})$$

where Q_b is the borehole heat extraction rate (in Watts), H is the borehole length, T_b^* is the effective borehole wall temperature, R_b^* is the effective borehole thermal resistance, $T_{f,in}$ is the inlet fluid temperature, and $T_{f,out}$ is the outlet fluid temperature.

Parameters

m_flow_borehole
[float] Fluid mass flow rate (in kg/s) into the borehole.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.K)

Returns

R_b
[float] Effective borehole thermal resistance (in m.K/W).

`get_borehole_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_b
[float or (nSegments,) array] Heat extraction rates along each borehole segment (in Watts). The returned type corresponds to the type of the parameter *T_b*.

`get_fluid_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_f
[float or (nOutlets,) array] Heat extraction rates from each fluid circuit (in Watts). The returned type corresponds to the type of the parameter *T_f_in*.

`get_inlet_temperature`(*Q_f*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the inlet fluid temperatures of the borehole.

Parameters

Q_f
[float or (nInlets,) array] Heat extraction from the fluid circuits (in Watts).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_in
[float or (nOutlets,) array] Inlet fluid temperatures (in Celsius) into each inlet pipe. The returned type corresponds to the type of the parameter *Q_f*.

`get_outlet_temperature`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the outlet fluid temperatures of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_out
[float or (nOutlets,) array] Outlet fluid temperatures (in Celsius) from each outlet pipe. The returned type corresponds to the type of the parameter *T_f_in*.

`get_temperature`(*z*, *T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the fluid temperatures of the borehole at a depth (z).

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperatures.

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f
[(2*nPipes,) or (nDepths, 2*nPipes,) array] Fluid temperature (in Celsius) in each pipe. The returned shape depends on the type of the parameter *z*.

`get_total_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the total heat extraction rate of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_t
[float] Total net heat extraction rate of the borehole (in Watts).

`local_borehole_thermal_resistance`()

Evaluate the local (cross-sectional) borehole thermal resistance, defined by:

$$Q_b'(z) = \frac{T_b(z) - \bar{T}_f(z)}{R_b}$$

where $Q_b'(z)$ is the borehole heat extraction rate per unit depth at a depth (z), $T_b(z)$ is the borehole wall temperature, $\bar{T}_f(z)$ is the arithmetic mean fluid temperature and R_b is the local borehole thermal resistance.

Returns

R_b
[float] Local borehole thermal resistance (in m.K/W).

`update_thermal_resistances`(*R_fp*)

Update the delta-circuit of thermal resistances.

This methods updates the values of the delta-circuit thermal resistances based on the provided fluid to outer pipe wall thermal resistance.

Parameters

R_fp
[float] Fluid to outer pipe wall thermal resistance (m-K/W).

`visualize_pipes`()

Plot the cross-section view of the borehole.

Returns

fig
[figure] Figure object (matplotlib).

`class pygfunction.pipes.SingleUTube`(*pos*, *r_in*, *r_out*, *borehole*, *k_s*, *k_g*, *R_fp*, *J=2*)

Bases: `_BasePipe`

Class for single U-Tube boreholes.

Contains information regarding the physical dimensions and thermal characteristics of the pipes and the grout material, as well as methods to evaluate fluid temperatures and heat extraction rates based on the work of Hellstrom[9]. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom[10].

Notes

The expected array shapes of input parameters and outputs are documented for each class method. *nInlets* and *nOutlets* are the number of inlets and outlets to the borehole, and both are equal to 1 for a single U-tube borehole. *nSegments* is the number of discretized segments along the borehole. *nPipes* is the number of pipes (i.e. the number of U-tubes) in the borehole, equal to 1. *nDepths* is the number of depths at which temperatures are evaluated.

The effective borehole thermal resistance is evaluated using the method of Cimmino[11]. This is valid for any number of pipes.

References

Attributes

pos
[list of tuples] Position (x, y) (in meters) of the pipes inside the borehole.

r_in
[float] Inner radius (in meters) of the U-Tube pipes.

r_out
[float] Outer radius (in meters) of the U-Tube pipes.

[9] Hellstrom, G. (1991). Ground heat storage. Thermal Analyses of Duct Storage Systems I: Theory. PhD Thesis. University of Lund, Department of Mathematical Physics. Lund, Sweden.

[10] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

[11] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022.

borehole
[Borehole object] Borehole class object of the borehole containing the U-Tube.

k_s
[float] Soil thermal conductivity (in W/m-K).

k_g
[float] Grout thermal conductivity (in W/m-K).

R_fp
[float] Fluid to outer pipe wall thermal resistance (m-K/W).

J
[int, optional] Number of multipoles per pipe to evaluate the thermal resistances. Default is 2.

nPipes
[int] Number of U-Tubes, equals to 1.

nInlets
[int] Total number of pipe inlets, equals to 1.

nOutlets
[int] Total number of pipe outlets, equals to 1.

`coefficients_borehole_heat_extraction_rate`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_b} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nSegments, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nSegments, nSegments,) array] Array of coefficients for borehole wall temperatures.

coefficients_fluid_heat_extraction_rate(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate heat extraction rates.

Returns coefficients for the relation:

$$\mathbf{Q_f} = \mathbf{a_{in}}\mathbf{T_{f,in}} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

coefficients_inlet_temperature(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate inlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,in}} = \mathbf{a_{q,f}}\mathbf{Q_f} + \mathbf{a_b}\mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_qf
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_outlet_temperature`(*m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate outlet fluid temperature.

Returns coefficients for the relation:

$$\mathbf{T_{f,out}} = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
[(nOutlets, nInlets,) array] Array of coefficients for inlet fluid temperature.

a_b
[(nOutlets, nSegments,) array] Array of coefficients for borehole wall temperatures.

`coefficients_temperature`(*z*, *m_flow_borehole*, *cp_f*, *nSegments*, *segment_ratios=None*)

Build coefficient matrices to evaluate fluid temperatures at a depth (z).

Returns coefficients for the relation:

$$\mathbf{T_f}(z) = \mathbf{a_{in}} \mathbf{T_{f,in}} + \mathbf{a_b} \mathbf{T_b}$$

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperature coefficients.

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rate (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

nSegments
[int] Number of borehole segments.

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

a_in
(2*nPipes, nInlets,) array, or (nDepths, 2*nPipes, nInlets,) array
Array of coefficients for inlet fluid temperature.

a_b
(2*nPipes, nSegments,) array, or (nDepths, 2*nPipes, nSegments,) array
Array of coefficients for borehole wall temperatures.

`effective_borehole_thermal_resistance`(*m_flow_borehole*, *cp_f*)

Evaluate the effective borehole thermal resistance, defined by:

$$\frac{Q_b}{H} = \frac{T_b^* - \bar{T}_f}{R_b^*}$$
$$\bar{T}_f = \frac{1}{2}(T_{f,in} + T_{f,out})$$

where Q_b is the borehole heat extraction rate (in Watts), H is the borehole length, T_b^* is the effective borehole wall temperature, R_b^* is the effective borehole thermal resistance, $T_{f,in}$ is the inlet fluid temperature, and $T_{f,out}$ is the outlet fluid temperature.

Parameters

m_flow_borehole
[float] Fluid mass flow rate (in kg/s) into the borehole.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.K)

Returns

R_b
[float] Effective borehole thermal resistance (in m.K/W).

`get_borehole_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment.

The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_b
[float or (nSegments,) array] Heat extraction rates along each borehole segment (in Watts). The returned type corresponds to the type of the parameter *T_b*.

`get_fluid_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the heat extraction rates of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_f
[float or (nOutlets,) array] Heat extraction rates from each fluid circuit (in Watts). The returned type corresponds to the type of the parameter *T_f_in*.

`get_inlet_temperature`(*Q_f*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the inlet fluid temperatures of the borehole.

Parameters

Q_f
[float or (nInlets,) array] Heat extraction from the fluid circuits (in Watts).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_in
[float or (nOutlets,) array] Inlet fluid temperatures (in Celsius) into each inlet pipe. The returned type corresponds to the type of the parameter *Q_f*.

`get_outlet_temperature`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the outlet fluid temperatures of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f_out
[float or (nOutlets,) array] Outlet fluid temperatures (in Celsius) from each outlet pipe. The returned type corresponds to the type of the parameter *T_f_in*.

`get_temperature`(*z*, *T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the fluid temperatures of the borehole at a depth (z).

Parameters

z
[float or (nDepths,) array] Depths (in meters) to evaluate the fluid temperatures.

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

T_f
[(2*nPipes,) or (nDepths, 2*nPipes,) array] Fluid temperature (in Celsius) in each pipe. The returned shape depends on the type of the parameter *z*.

`get_total_heat_extraction_rate`(*T_f_in*, *T_b*, *m_flow_borehole*, *cp_f*, *segment_ratios=None*)

Returns the total heat extraction rate of the borehole.

Parameters

T_f_in
[float or (nInlets,) array] Inlet fluid temperatures (in Celsius).

T_b
[float or (nSegments,) array] Borehole wall temperatures (in Celsius).

m_flow_borehole
[float or (nInlets,) array] Inlet mass flow rates (in kg/s) into the borehole.

cp_f
[float or (nInlets,) array] Fluid specific isobaric heat capacity (in J/kg.degC).

segment_ratios
[(nSegments,) array, optional] Ratio of the borehole length represented by each segment. The sum of ratios must be equal to 1. If segment_ratios==None, segments of equal lengths are considered. Default is None.

Returns

Q_t
[float] Total net heat extraction rate of the borehole (in Watts).

`local_borehole_thermal_resistance()`

Evaluate the local (cross-sectional) borehole thermal resistance, defined by:

$$Q_b'(z) = \frac{T_b(z) - \bar{T}_f(z)}{R_b}$$

where $Q_b'(z)$ is the borehole heat extraction rate per unit depth at a depth (z), $T_b(z)$ is the borehole wall temperature, $\bar{T}_f(z)$ is the arithmetic mean fluid temperature and R_b is the local borehole thermal resistance.

Returns

R_b
[float] Local borehole thermal resistance (in m.K/W).

`update_thermal_resistances(R_fp)`

Update the delta-circuit of thermal resistances.

This methods updates the values of the delta-circuit thermal resistances based on the provided fluid to outer pipe wall thermal resistance.

Parameters

R_fp
[float] Fluid to outer pipe wall thermal resistance (m-K/W).

`visualize_pipes()`

Plot the cross-section view of the borehole.

Returns

fig
[figure] Figure object (matplotlib).

`pygfunction.pipes.borehole_thermal_resistance(pipe, m_flow_borehole, cp_f)`

Evaluate the effective borehole thermal resistance, defined by:

$$\frac{Q_b}{H} = \frac{T_b^* - \bar{T}_f}{R_b^*}$$

$$\bar{T}_f = \frac{1}{2}(T_{f,in} + T_{f,out})$$

where Q_b is the borehole heat extraction rate (in Watts), H is the borehole length, T_b^* is the effective borehole wall temperature, R_b^* is the effective borehole thermal resistance, $T_{f,in}$ is the inlet fluid temperature, and $T_{f,out}$ is the outlet fluid temperature.

Parameters

pipe
[pipe object] Model for pipes inside the borehole.

m_flow_borehole
[float] Fluid mass flow rate (in kg/s) into the borehole.

cp_f
[float] Fluid specific isobaric heat capacity (in J/kg.K)

Returns

R_b
[float] Effective borehole thermal resistance (m.K/W).

Notes

The effective borehole thermal resistance is evaluated using the method of Cimmino[12]. This is valid for any number of pipes.

References

pygfunction.pipes.**conduction_thermal_resistance_circular_pipe**(*r_in*, *r_out*, *k_p*)

Evaluate the conduction thermal resistance for circular pipes.

Parameters

r_in
[float] Inner radius of the pipes (in meters).

r_out
[float] Outer radius of the pipes (in meters).

k_p
[float] Pipe thermal conductivity (in W/m-K).

Returns

R_p
[float] Conduction thermal resistance (in m-K/W).

[12] Cimmino, M. (2019). Semi-analytical method for g-function calculation of bore fields with series- and parallel-connected boreholes. Science and Technology for the Built Environment, 25 (8), 1007-1022.

pygfunction.pipes.**convective_heat_transfer_coefficient_circular_pipe**(*m_flow_pipe*, *r_in*, *mu_f*, *rho_f*, *k_f*, *cp_f*, *epsilon*)

Evaluate the convective heat transfer coefficient for circular pipes.

The Nusselt number must first be determined to find the convection coefficient. Determination of the Nusselt number in turbulent flow is done by calling `_Nusselt_number_turbulent_flow()`. An analytical solution for constant pipe wall surface temperature is used for laminar flow.

Since `_Nusselt_number_turbulent_flow()` is only valid for Re > 3000. and to avoid dicontinuities in the values of the convective heat transfer coefficient near the onset of the turbulence region (approximately Re = 2300.), linear interpolation is used over the range 2300 < Re < 4000 for the evaluation of the Nusselt number.

This approach was verified by Gnielinski (2013)[13].

Parameters

m_flow_pipe
[float] Fluid mass flow rate (in kg/s) into the pipe.

r_in
[float] Inner radius of the pipe (in meters).

mu_f
[float] Fluid dynamic viscosity (in kg/m-s).

rho_f
[float] Fluid density (in kg/m3).

k_f
[float] Fluid thermal conductivity (in W/m-K).

cp_f
[float] Fluid specific heat capacity (in J/kg-K).

epsilon
[float] Pipe roughness (in meters).

Returns

h_fluid
[float] Convective heat transfer coefficient (in W/m2-K).

References

pygfunction.pipes.**convective_heat_transfer_coefficient_concentric_annulus**(*m_flow_pipe*, *r_a_in*, *r_a_out*, *mu_f*, *rho_f*, *k_f*, *cp_f*, *epsilon*)

Evaluate the inner and outer convective heat transfer coefficient for the annulus region of a concentric pipe.

Grundmann (2016) referenced Hellström (1991)[17] in the discussion about inner and outer convection coefficients in an annulus region of a concentric pipe arrangement.

The following is valid for $Re < 2300$ and $0.1 \leq Pr \leq 1000$:

[13] Gnielinski, V. (2013). On heat transfer in tubes. International Journal of Heat and Mass Transfer, 63, 134–140. https://doi.org/10.1016/j.ijheatmasstransfer.2013.04.015

[17] Hellstrom, G. (1991). Ground heat storage. Thermal Analyses of Duct Storage Systems I: Theory. PhD Thesis. University of Lund, Department of Mathematical Physics. Lund, Sweden.

$$\mathrm{Nu}_{a,in} = 3.66 + 1.2(r^*)^{-0.8}$$

$$\mathrm{Nu}_{a,out} = 3.66 + 1.2(r^*)^{0.5}$$

where $r^* = r_{a,in}/r_{a,out}$ is the ratio of the inner over the outer annulus radius (Cengel and Ghajar 2015, pg. 476).

Cengel and Ghajar (2015)[15] state that inner and outer Nusselt numbers are approximately equivalent for turbulent flow. They additionally state that Gnielinski `_Nusselt_number_turbulent_flow()` can be used for turbulent flow. Linear interpolation is used over the range 2300 < Re < 4000 for the evaluation of the Nusselt number, as proposed by Gnielinski (2013)[Page 83, 13].

Parameters

m_flow_pipe: float
Fluid mass flow rate (in kg/s) into the pipe.

r_a_in: float
Pipe annulus inner radius (in meters).

r_a_out: float
Pipe annulus outer radius (in meters).

mu_f
[float] Fluid dynamic viscosity (in kg/m-s).

rho_f
[float] Fluid density (in kg/m3).

k_f
[float] Fluid thermal conductivity (in W/m-K).

cp_f
[float] Fluid specific heat capacity (in J/kg-K).

epsilon
[float] Pipe roughness (in meters).

Returns

h_fluid_a_in: float
Convective heat transfer coefficient of the inner pipe annulus region (in W/m2-K).

h_fluid_a_out: float
Convective heat transfer coefficient of the outer pipe annulus region (in W/m2-K).

References

pygfunction.pipes.**fluid_friction_factor_circular_pipe**(*m_flow_pipe*, *r_in*, *mu_f*, *rho_f*, *epsilon*, *tol=1e-06*)

Evaluate the Darcy-Weisbach friction factor.

Parameters

m_flow_pipe
[float] Fluid mass flow rate (in kg/s) into the pipe.

[15] Cengel, Y.A., & Ghajar, A.J. (2015). Heat and mass transfer: fundamentals & applications (Fifth edition.). McGraw-Hill.

r_in
[float] Inner radius of the pipes (in meters).

mu_f
[float] Fluid dynamic viscosity (in kg/m-s).

rho_f
[float] Fluid density (in kg/m3).

epsilon
[float] Pipe roughness (in meters).

tol
[float] Relative convergence tolerance on Darcy friction factor. Default is 1.0e-6.

Returns

fDarcy
[float] Darcy friction factor.

pygfunction.pipes.**multipole**(*pos*, *r_out*, *r_b*, *k_s*, *k_g*, *R_fp*, *T_b*, *q_p*, *J*, *x_T=array([], dtype=float64)*, *y_T=array([], dtype=float64)*, *eps=1e-05*, *it_max=100*)

Multipole method to calculate borehole thermal resistances in a borehole heat exchanger.

Adapted from the work of Claesson and Hellstrom[16].

Parameters

pos
[list] List of positions (x,y) (in meters) of pipes around the center of the borehole.

r_out
[float or array] Outer radius of the pipes (in meters).

r_b
[float] Borehole radius (in meters).

k_s
[float] Soil thermal conductivity (in W/m-K).

k_g
[float] Grout thermal conductivity (in W/m-K).

R_fp
[float or array] Fluid-to-outer-pipe-wall thermal resistance (in m-K/W).

J
[int] Number of multipoles per pipe to evaluate the thermal resistances. J=1 or J=2 usually gives sufficient accuracy. J=0 corresponds to the line source approximation.

q_p
[array] Thermal energy flows (in W/m) from pipes.

T_b
[float] Average borehole wall temperature (in degC).

eps
[float, optional] Iteration relative accuracy. Default is 1e-5.

it_max
[int, optional] Maximum number of iterations. Default is 100.

[16] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

x_T
[array, optional] x-coordinates (in meters) to calculate temperatures. Default is np.empty(0).

y_T
[array, optional] y-coordinates (in meters) to calculate temperatures. Default is np.empty(0).

Returns

T_f
[array] Fluid temperatures (in degC) in the pipes.

T
[array] Requested temperatures (in degC).

it
[int] Total number of iterations

eps_max
[float] Maximum error.

References

pygfunction.pipes.**thermal_resistances**(*pos*, *r_out*, *r_b*, *k_s*, *k_g*, *R_fp*, *J=2*)

Evaluate thermal resistances and delta-circuit thermal resistances.

This function evaluates the thermal resistances and delta-circuit thermal resistances between pipes in a borehole using the multipole method[18]. Thermal resistances are defined by:

$$\mathbf{T_f} - T_b = \mathbf{R} \cdot \mathbf{Q_p}$$

Delta-circuit thermal resistances are defined by:

$$q_{p,i,j} = \frac{T_{f,i} - T_{f,j}}{R^{\Delta}_{i,j}}$$
$$q_{p,i,i} = \frac{T_{f,i} - T_b}{R^{\Delta}_{i,i}}$$

Parameters

pos
[list] List of positions (x,y) (in meters) of pipes around the center of the borehole.

r_out
[float or array] Outer radius of the pipes (in meters).

r_b
[float] Borehole radius (in meters).

k_s
[float] Soil thermal conductivity (in W/m-K).

k_g
[float] Grout thermal conductivity (in W/m-K).

R_fp
[float or array] Fluid-to-outer-pipe-wall thermal resistance (in m-K/W).

[18] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

J
[int, optional] Number of multipoles per pipe to evaluate the thermal resistances. J=1 or J=2 usually gives sufficient accuracy. J=0 corresponds to the line source approximation[Page 83, 17]. Default is 2.

Returns

R
[array] Thermal resistances (in m-K/W).

Rd
[array] Delta-circuit thermal resistances (in m-K/W).

References

Examples

```
>>> pos = [(-0.06, 0.), (0.06, 0.)]
>>> R, Rd = gt.pipes.thermal_resistances(pos, 0.01, 0.075, 2., 1., 0.1,
                                         J=0)
R = [[ 0.36648149, -0.04855895],
     [-0.04855895,  0.36648149]]
Rd = [[ 0.31792254, -2.71733044],
      [-2.71733044,  0.31792254]]
```

3.8 Utilities Module

pygfunction.utilities.**cardinal_point**(*direction*)

pygfunction.utilities.**erfint**(*x*)
Integral of the error function.

Parameters

x
[float or array] Argument.

Returns

float or array
Integral of the error function.

pygfunction.utilities.**exp1**(*x*)
Exponential integral E1.

Parameters

x
[float or array] Argument.

Returns

E1
[float or array] Exponential integral.

References

pygfunction.utilities.**segment_ratios**(*nSegments*, *end_length_ratio=0.02*)

Discretize a borehole into segments of different lengths using a geometrically expanding mesh from the provided end-length-ratio towards the middle of the borehole. Eskilson (1987)[2] proposed that segment lengths increase with a factor of sqrt(2) towards the middle of the borehole. Here, the expansion factor is inferred from the provided number of segments and end-length-ratio.

Parameters

nSegments
[int] Number of line segments along the borehole.

end_length_ratio: float, optional
The ratio of the height of the borehole that accounts for the end segment lengths. Default is 0.02.

Returns

segment_ratios
[array] The segment ratios along the borehole, from top to bottom.

References

Examples

```
>>> gt.utilities.segment_ratios(5)
array([0.02, 0.12, 0.72, 0.12, 0.02])
```

pygfunction.utilities.**time_ClaessonJaved**(*dt*, *tmax*, *cells_per_level=5*)

Build a time vector of expanding cell width following the method of Claesson and Javed[3].

Parameters

dt
[float] Simulation time step (in seconds).

tmax
[float] Maximum simulation time (in seconds).

cells_per_level
[int, optional] Number of time steps cells per level. Cell widths double every cells_per_level cells. Default is 5.

Returns

time
[array] Time vector.

[2] Eskilson, P. (1987). Thermal analysis of heat extraction boreholes. PhD Thesis. University of Lund, Department of Mathematical Physics. Lund, Sweden.

[3] Claesson, J., & Javed, S. (2012). A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. ASHRAE Transactions, 118 (1): 530-539.

References

Examples

```
>>> time = gt.utilities.time_ClaessonJaved(3600., 12*3600.)
array([3600.0, 7200.0, 10800.0, 14400.0, 18000.0, 25200.0, 32400.0,
       39600.0, 46800.0])
```

pygfunction.utilities.**time_MarcottePasquier**(*dt*, *tmax*, *non_expanding_cells=48*)

Build a time vector of expanding cell width following the method of Marcotte and Pasquier[4].

Parameters

dt
[float] Simulation time step (in seconds).

tmax
[float] Maximum simulation time (in seconds).

non_expanding_cells
[int, optional] Number of cells before geomteric expansion starts. Default is 48.

Returns

time
[array] Time vector.

References

Examples

```
>>> time = gt.utilities.time_MarcottePasquier(3600., 13*3600.,
                                              non_expanding_cells=6)
array([3600., 7200., 10800., 14400., 18000., 21600., 28800., 43200.,
       72000.])
```

pygfunction.utilities.**time_geometric**(*dt*, *tmax*, *Nt*)

Build a time vector of geometrically expanding cell width.

Parameters

dt
[float] Simulation time step (in seconds).

tmax
[float] Maximum simulation time (in seconds).

Nt
[int] Total number of time steps.

Returns

time
[array] Time vector.

[4] Marcotte, D., & Pasquier, P. (2008). Fast fluid and ground temperature computation for geothermal ground-loop heat exchanger systems. Geothermics, 37: 651-665.

Examples

```
>>> time = gt.utilities.time_geometric(3600., 13*3600., 5)
array([3600., 8971.99474335, 16988.19683297, 28950.14002383, 46800.])
```

CHAPTER

FOUR

EXAMPLES

4.1 Definition of a bore field using pre-defined configurations

This example demonstrates the use of the *borehole* module to define the positions of the boreholes within a bore field using pre-define bore field configurations.

The following script generates rectangular, box-shaped, U-shaped and L-shaped bore fields in a 4 x 3 configuration as well as a circular field of 8 boreholes. Each field is then plotted on a separate figure.

The script is located in: *pygfunction/examples/regular_bore_field.py*

```
# -*- coding: utf-8 -*-
""" Example of definition of a bore field using pre-defined configurations.

"""
import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0             # Borehole buried depth (m)
    H = 150.0           # Borehole length (m)
    r_b = 0.075         # Borehole radius (m)
    B = 7.5             # Borehole spacing (m)
    N_1 = 4             # Number of boreholes in the x-direction (columns)
    N_2 = 3             # Number of boreholes in the y-direction (rows)

    # Circular field
    N_b = 8     # Number of boreholes
    R = 5.0     # Distance of the boreholes from the center of the field (m)

    # -------------------------------------------------------------------------
    # Borehole fields
    # -------------------------------------------------------------------------

    # Rectangular field of 4 x 3 boreholes
    rectangularField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
```

(continues on next page)

(continued from previous page)

```

    # Box-shaped field of 4 x 3 boreholes
    boxField = gt.boreholes.box_shaped_field(N_1, N_2, B, B, H, D, r_b)

    # U-shaped field of 4 x 3 boreholes
    UField = gt.boreholes.U_shaped_field(N_1, N_2, B, B, H, D, r_b)

    # L-shaped field of 4 x 3 boreholes
    LField = gt.boreholes.L_shaped_field(N_1, N_2, B, B, H, D, r_b)

    # Circular field of 8 boreholes
    circleField = gt.boreholes.circle_field(N_b, R, H, D, r_b)

    # -------------------------------------------------------------------------
    # Draw bore fields
    # -------------------------------------------------------------------------
    for field in [rectangularField, boxField, UField, LField, circleField]:
        gt.boreholes.visualize_field(field)

    return


# Main function
if __name__ == '__main__':
    main()
```

4.2 Definition of a bore field using custom borehole positions

This example demonstrates the use of the *borehole* module to define the positions of the boreholes within a bore field from a list of borehole positions.

Two borehole positions (1 and 2) are intentionally added as duplicates and are removed by calling the *pygfunction.boreholes.remove_duplicates()* function.

The following script generates a bore field with 5 boreholes. The field is then plotted on a figure.

The script is located in: *pygfunction/examples/custom_bore_field.py*

```
# -*- coding: utf-8 -*-
""" Example of definition of a bore field using custom borehole positions.

"""
import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
```

(continues on next page)

(continued from previous page)

```
    D = 4.0             # Borehole buried depth (m)
    H = 150.0           # Borehole length (m)
    r_b = 0.075         # Borehole radius (m)

    # Borehole positions
    # Note: Two duplicate boreholes have been added to this list of positions.
    #       Position 1 has a borehole that is directly on top of another bore
    #       Position 2 has a borehole with radius inside of another bore
    #       The duplicates will be removed with the remove_duplicates function
    pos = [(0.0, 0.0),
           (0.0, 0.0),  # Duplicate (for example purposes)
           (0.03, 0.0),   # Duplicate (for example purposes)
           (5.0, 0.),
           (3.5, 4.0),
           (1.0, 7.0),
           (5.5, 5.5)]

    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    # Build list of boreholes
    field = [gt.boreholes.Borehole(H, D, r_b, x, y) for (x, y) in pos]

    # -------------------------------------------------------------------------
    # Find and remove duplicates from borehole field
    # -------------------------------------------------------------------------

    field = gt.boreholes.remove_duplicates(field, disp=True)

    # -------------------------------------------------------------------------
    # Draw bore field
    # -------------------------------------------------------------------------

    gt.boreholes.visualize_field(field)

    return


# Main function
if __name__ == '__main__':
    main()
```

4.3 Definition of a bore field using external text file

This example demonstrates the use of the *borehole* module to define the positions of the boreholes within a bore field from an external text file.

The following script generates a bore field with 32 boreholes. The field is then plotted on a figure.

The script is located in: *./pygfunction/examples/custom_bore_field_from_file.py*

```
# -*- coding: utf-8 -*-
""" Example of definition of a bore field using custom borehole positions.

"""
import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Parameters
    # -------------------------------------------------------------------------

    # Filepath to bore field text file
    filename = './data/custom_field_32_boreholes.txt'

    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    # Build list of boreholes
    field = gt.boreholes.field_from_file(filename)

    # -------------------------------------------------------------------------
    # Draw bore field
    # -------------------------------------------------------------------------

    gt.boreholes.visualize_field(field)

    return


# Main function
if __name__ == '__main__':
    main()
```

4.4 Definition and visualization of a borehole

This example demonstrates the use of the *borehole* module to define the positions of pipes inside a borehole and visualize the top view of a borehole.

The following script generates boreholes with a single U-tube, a double U-tube (in series and parallel configurations), and a coaxial borehole. The borehole cross-sections are then plotted.

The script is located in: *pygfunction/examples/custom_borehole.py*

```
# -*- coding: utf-8 -*-
""" Example definition of a borehole. A top-view plot of the borehole is
    created and the borehole resistance is computed.

"""
import numpy as np
from scipy.constants import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 5.          # Borehole buried depth (m)
    H = 400.        # Borehole length (m)
    r_b = 0.0875    # Borehole radius (m)

    # Pipe dimensions (all configurations)
    epsilon = 1.0e-6    # Pipe roughness (m)

    # Pipe dimensions (single U-tube and double U-tube)
    r_out = 0.0211      # Pipe outer radius (m)
    r_in = 0.0147       # Pipe inner radius (m)
    D_s = 0.052         # Shank spacing (m)

    # Pipe dimensions (coaxial)
    r_in_in = 0.0221    # Inside pipe inner radius (m)
    r_in_out = 0.025    # Inside pipe outer radius (m)
    r_out_in = 0.0487   # Outer pipe inside radius (m)
    r_out_out = 0.055   # Outer pipe outside radius (m)
    # Vectors of inner and outer pipe radii
    # Note : The dimensions of the inlet pipe are the first elements of
    #        the vectors. In this example, the inlet pipe is the inside pipe.
    r_inner = np.array([r_in_in, r_out_in])     # Inner pipe radii (m)
    r_outer = np.array([r_in_out, r_out_out])   # Outer pip radii (m)

    # Ground properties
    k_s = 2.0           # Ground thermal conductivity (W/m.K)

    # Grout properties
```

(continues on next page)

(continued from previous page)

```
    k_g = 1.0           # Grout thermal conductivity (W/m.K)

    # Pipe properties
    k_p = 0.4           # Pipe thermal conductivity (W/m.K)

    # Fluid properties
    # Total fluid mass flow rate per borehole (kg/s)
    m_flow_borehole = 1.0
    # The fluid is propylene-glycol (20 %) at 20 degC
    fluid = gt.media.Fluid('MPG', 20.)
    cp_f = fluid.cp     # Fluid specific isobaric heat capacity (J/kg.K)
    rho_f = fluid.rho   # Fluid density (kg/m3)
    mu_f = fluid.mu     # Fluid dynamic viscosity (kg/m.s)
    k_f = fluid.k       # Fluid thermal conductivity (W/m.K)

    # -------------------------------------------------------------------------
    # Initialize borehole model
    # -------------------------------------------------------------------------

    borehole = gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)

    # -------------------------------------------------------------------------
    # Define a single U-tube borehole
    # -------------------------------------------------------------------------

    # Pipe positions
    # Single U-tube [(x_in, y_in), (x_out, y_out)]
    pos_single = [(-D_s, 0.), (D_s, 0.)]

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
        r_in, r_out, k_p)

    # Fluid to inner pipe wall thermal resistance
    m_flow_pipe = m_flow_borehole
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f = 1.0 / (h_f * 2 * pi * r_in)

    # Single U-tube GHE in borehole
    SingleUTube = gt.pipes.SingleUTube(
        pos_single, r_in, r_out, borehole, k_s, k_g, R_f + R_p)

    # Check the geometry to make sure it is physically possible

    # This class method is automatically called at the instanciation of the
    # pipe object and raises an error if the pipe geometry is invalid. It is
    # manually called here for demonstration.
    check_single = SingleUTube._check_geometry()
    print(f'The geometry of the borehole is valid (realistic/possible): '
          f'{check_single!s}.')

```

(continues on next page)

(continued from previous page)

```
    # Evaluate and print the effective borehole thermal resistance
    R_b = SingleUTube.effective_borehole_thermal_resistance(
        m_flow_borehole, fluid.cp)
    print(f'Single U-tube Borehole thermal resistance: '
          f'{R_b:.4f} m.K/W')

    # Visualize the borehole geometry and save the figure
    fig_single = SingleUTube.visualize_pipes()
    fig_single.savefig('single-u-tube-borehole.png')

    # -------------------------------------------------------------------------
    # Define a double U-tube borehole
    # -------------------------------------------------------------------------

    # Pipe positions
    # Double U-tube [(x_in1, y_in1), (x_in2, y_in2),
    #                (x_out1, y_out1), (x_out2, y_out2)]
    # Note: in series configuration, fluid enters pipe (in,1), exits (out,1),
    # then enters (in,2) and finally exits (out,2)
    # (if you view visualize_pipe, series is 1->3->2->4)
    pos_double = [(-D_s, 0.), (0., -D_s), (D_s, 0.), (0., D_s)]

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
            r_in, r_out, k_p)

    # Fluid to inner pipe wall thermal resistance
    # Double U-tube in series
    m_flow_pipe_series = m_flow_borehole
    h_f_series = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow_pipe_series, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f_series = 1.0 / (h_f_series * 2 * pi * r_in)
    # Double U-tube in parallel
    m_flow_pipe_parallel = m_flow_borehole / 2
    h_f_parallel = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow_pipe_parallel, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f_parallel = 1.0 / (h_f_parallel * 2 * pi * r_in)

    # Double U-tube GHE in borehole
    # Double U-tube in series
    DoubleUTube_series = gt.pipes.MultipleUTube(
        pos_double, r_in, r_out, borehole, k_s, k_g, R_p + R_f_series, 2,
        config='series')
    # Double U-tube in parallel
    DoubleUTube_parallel = gt.pipes.MultipleUTube(
        pos_double, r_in, r_out, borehole, k_s, k_g, R_p + R_f_parallel, 2,
        config='parallel')

    # Evaluate and print the effective borehole thermal resistance
    R_b_series = DoubleUTube_series.effective_borehole_thermal_resistance(
        m_flow_borehole, fluid.cp)
    print(f'Double U-tube (series) Borehole thermal resistance: '
```

(continues on next page)

(continued from previous page)

```
        f'{R_b_series:.4f} m.K/W')
R_b_parallel = DoubleUTube_parallel.effective_borehole_thermal_resistance(
    m_flow_borehole, fluid.cp)
print(f'Double U-tube (parallel) Borehole thermal resistance: '
      f'{R_b_parallel:.4f} m.K/W')

# Visualize the borehole geometry and save the figure
fig_double = DoubleUTube_series.visualize_pipes()
fig_double.savefig('double-u-tube-borehole.png')

# -----------------------------------------------------------------------------
# Define a coaxial borehole
# -----------------------------------------------------------------------------

# Pipe positions
# Coaxial pipe (x, y)
pos = (0., 0.)

# Pipe thermal resistance
# (the two pipes have the same thermal conductivity, k_p)
# Inner pipe
R_p_in = gt.pipes.conduction_thermal_resistance_circular_pipe(
    r_in_in, r_in_out, k_p)
# Outer pipe
R_p_out = gt.pipes.conduction_thermal_resistance_circular_pipe(
    r_out_in, r_out_out, k_p)

# Fluid-to-fluid thermal resistance
# Inner pipe
h_f_in = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
    m_flow_borehole, r_in_in, mu_f, rho_f, k_f, cp_f, epsilon)
R_f_in = 1.0 / (h_f_in * 2 * pi * r_in_in)
# Outer pipe
h_f_a_in, h_f_a_out = \
    gt.pipes.convective_heat_transfer_coefficient_concentric_annulus(
        m_flow_borehole, r_in_out, r_out_in, mu_f, rho_f, k_f, cp_f,
        epsilon)
R_f_out_in = 1.0 / (h_f_a_in * 2 * pi * r_in_out)
R_ff = R_f_in + R_p_in + R_f_out_in

# Coaxial GHE in borehole
R_f_out_out = 1.0 / (h_f_a_out * 2 * pi * r_out_in)
R_fp = R_p_out + R_f_out_out
Coaxial = gt.pipes.Coaxial(
    pos, r_inner, r_outer, borehole, k_s, k_g, R_ff, R_fp, J=2)

# Evaluate and print the effective borehole thermal resistance
R_b = Coaxial.effective_borehole_thermal_resistance(
    m_flow_borehole, fluid.cp)
print(f'Coaxial tube Borehole thermal resistance: {R_b:.4f} m.K/W')

# Visualize the borehole geometry and save the figure
```

(continues on next page)

(continued from previous page)

```
    fig_coaxial = Coaxial.visualize_pipes()
    fig_coaxial.savefig('coaxial-borehole.png')


if __name__ == '__main__':
    main()
```

4.5 Calculation of g-functions with uniform borehole heat extraction rates

This example demonstrates the use of the *g-function* module to calculate *g*-functions using a boundary condition of uniform and equal heat extraction rate for all boreholes, constant in time.

The following script generates the *g*-functions of rectangular fields of 3 x 2, 6 x 4 and 10 x 10 boreholes. *g*-Functions are verified against the *g*-functions presented by Cimmino and Bernier[1].

The script is located in: *pygfunction/examples/uniform_heat_extraction_rate.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of g-functions using uniform heat extraction rates.

    The g-functions of fields of 3x2, 6x4 and 10x10 boreholes are calculated
    for boundary condition of uniform heat extraction rate along the boreholes,
    equal for all boreholes.

"""
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import AutoMinorLocator

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0             # Borehole buried depth (m)
    H = 150.0           # Borehole length (m)
    r_b = 0.075         # Borehole radius (m)
    B = 7.5             # Borehole spacing (m)

    # Thermal properties
    alpha = 1.0e-6      # Ground thermal diffusivity (m2/s)

```

(continues on next page)

[1] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

(continued from previous page)

```
    # Path to validation data
    filePath = './data/CiBe14_uniform_heat_extraction_rate.txt'

    # g-Function calculation options
    # The second field is evaluated with more segments to draw the
    # temperature profiles. A uniform discretization is used to compare results
    # with Cimmino and Bernier (2014).
    options = [{'nSegments': 1,
                'segment_ratios': None,
                'disp': True,
                'profiles': True},
               {'nSegments': 12,
                'segment_ratios': None,
                'disp': True,
                'profiles': True},
               {'nSegments': 1,
                'segment_ratios': None,
                'disp': True,
                'profiles': True}]

    # The 'similarities' method is used to consider unequal numbers of segments
    # per borehole and to plot heat extraction rate profiles along
    # individual boreholes
    method = 'similarities'

    # Geometrically expanding time vector.
    dt = 100*3600.                  # Time step
    tmax = 3000. * 8760. * 3600.    # Maximum time
    Nt = 25                         # Number of time steps
    ts = H**2/(9.*alpha)            # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)

    # -------------------------------------------------------------------------
    # Borehole fields
    # -------------------------------------------------------------------------

    # Field of 3x2 (n=6) boreholes
    N_1 = 3
    N_2 = 2
    boreField1 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # Field of 6x4 (n=24) boreholes
    N_1 = 6
    N_2 = 4
    boreField2 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # Field of 10x10 (n=100) boreholes
    N_1 = 10
    N_2 = 10
    boreField3 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # -------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
    # Load data from Cimmino and Bernier (2014)
    # -------------------------------------------------------------------------
    data = np.loadtxt(filePath, skiprows=55)

    # -------------------------------------------------------------------------
    # Evaluate g-functions for all fields
    # -------------------------------------------------------------------------
    for i, field in enumerate([boreField1, boreField2, boreField3]):
        gfunc = gt.gfunction.gFunction(
            field, alpha, time=time, boundary_condition='UHTR',
            options=options[i], method=method)
        # Draw g-function
        ax = gfunc.visualize_g_function().axes[0]
        # Draw reference g-function
        ax.plot(data[:,0], data[:,i+1], 'bx')
        ax.legend(['pygfunction', 'Cimmino and Bernier (2014)'])
        ax.set_title('Field of {} boreholes'.format(len(field)))
        plt.tight_layout()

        # For the second borefield, draw the evolution of heat extraction rates
        if i == 1:
            gfunc.visualize_temperatures(iBoreholes=[18, 12, 14])
            gfunc.visualize_temperature_profiles(iBoreholes=[14])

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.6 Calculation of g-functions with uniform borehole wall temperature

This example demonstrates the use of the *g-function* module to calculate g-functions using a boundary condition of uniform and equal borehole wall temperature for all boreholes. The total rate of heat extraction in the bore field is constant.

The following script generates the g-functions of rectangular fields of 3 x 2, 6 x 4 and 10 x 10 boreholes. g-Functions are verified against the g-functions presented by Cimmino and Bernier[1].

The script is located in: *pygfunction/examples/uniform_temperature.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of g-functions using uniform and equal borehole
    wall temperatures.

    The g-functions of fields of 3x2, 6x4 and 10x10 boreholes are calculated
```

(continues on next page)

[1] Cimmino, M., & Bernier, M. (2014). A semi-analytical method to generate g-functions for geothermal bore fields. International Journal of Heat and Mass Transfer, 70, 641-650.

(continued from previous page)

```
    for boundary condition of uniform borehole wall temperature along the
    boreholes, equal for all boreholes.

"""
import matplotlib.pyplot as plt
import numpy as np
from time import perf_counter

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)
    B = 7.5              # Borehole spacing (m)

    # Thermal properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)

    # Path to validation data
    filePath = './data/CiBe14_uniform_temperature.txt'

    # g-Function calculation options
    # A uniform discretization is used to compare results with Cimmino and
    # Bernier (2014).
    options = {'nSegments': 12,
               'segment_ratios': None,
               'disp': True,
               'profiles': True}

    # Geometrically expanding time vector.
    dt = 100*3600.                   # Time step
    tmax = 3000. * 8760. * 3600.     # Maximum time
    Nt = 25                          # Number of time steps
    ts = H**2/(9.*alpha)             # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)
    lntts = np.log(time/ts)

    # -------------------------------------------------------------------------
    # Borehole fields
    # -------------------------------------------------------------------------

    # Field of 3x2 (n=6) boreholes
    N_1 = 3
    N_2 = 2
    boreField1 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
```

(continues on next page)

(continued from previous page)

```
    # Field of 6x4 (n=24) boreholes
    N_1 = 6
    N_2 = 4
    boreField2 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # Field of 10x10 (n=100) boreholes
    N_1 = 10
    N_2 = 10
    boreField3 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # -------------------------------------------------------------------------
    # Load data from Cimmino and Bernier (2014)
    # -------------------------------------------------------------------------
    data = np.loadtxt(filePath, skiprows=55)

    # -------------------------------------------------------------------------
    # Evaluate g-functions for all fields
    # -------------------------------------------------------------------------
    for i, field in enumerate([boreField1, boreField2, boreField3]):
        nBoreholes = len(field)
        # Compare 'similarities' and 'equivalent' solvers
        t0 = perf_counter()
        gfunc_similarities = gt.gfunction.gFunction(
            field, alpha, time=time, options=options, method='similarities')
        t1 = perf_counter()
        t_similarities = t1 - t0
        gfunc_equivalent = gt.gfunction.gFunction(
            field, alpha, time=time, options=options, method='equivalent')
        t2 = perf_counter()
        t_equivalent = t2 - t1
        # Draw g-function
        ax = gfunc_similarities.visualize_g_function().axes[0]
        ax.plot(lntts, gfunc_equivalent.gFunc)
        # Draw reference g-function
        ax.plot(data[:,0], data[:,i+1], 'o')
        ax.legend([f'similarities (t = {t_similarities:.3f} sec)',
                   f'equivalent (t = {t_equivalent:.3f} sec)',
                   'Cimmino and Bernier (2014)'])
        ax.set_title(f'Field of {nBoreholes} boreholes')
        plt.tight_layout()

        # For the second borefield, draw the evolution of heat extraction rates
        if i == 1:
            fig = gfunc_similarities.visualize_heat_extraction_rates(
                iBoreholes=[18, 12, 14])
            fig.suptitle(f"Field of {nBoreholes} boreholes: 'similarities' "
                         f"solver")
            fig.tight_layout()

            fig = gfunc_equivalent.visualize_heat_extraction_rates()
            fig.suptitle(f"Field of {nBoreholes} boreholes: 'equivalent' "
```

(continues on next page)

(continued from previous page)

```
                        f"solver")
            fig.tight_layout()

            fig = gfunc_similarities.visualize_heat_extraction_rate_profiles(
                iBoreholes=[18, 12, 14])
            fig.suptitle(f"Field of {nBoreholes} boreholes: 'similarities' "
                        f"solver")
            fig.tight_layout()

            fig = gfunc_equivalent.visualize_heat_extraction_rate_profiles()
            fig.suptitle(f"Field of {nBoreholes} boreholes: 'equivalent' "
                        f"solver")
            fig.tight_layout()

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.7 Compare the accuracy and speed of different g-function solvers

This example compares the simulation times and the accuracy of different solvers for the evaluation of g-functions.

The g-function of a field of 6 by4 boreholes is first calculated for a boundary condition of uniform borehole wall temperature along the boreholes, equal for all boreholes. Three different solvers are compared : 'detailed', 'similarities'[1] and 'equivalent'[2]. Their accuracy and calculation time are compared using the 'detailed' solver as a reference. This shows that the 'similarities' solver can evaluate g-functions with high accuracy.

The g-function of a field of 12 by 10 boreholes is then calculated for a boundary condition of uniform borehole wall temperature along the boreholes, equal for all boreholes. Two different solvers are compared : 'similarities' and 'equivalent'. The accuracy and calculation time of the 'equivalent' is compared using the 'similarities' solver as a reference. This shows that the 'equivalent' solver evaluates g-functions at a very high calculation speed while maintaining reasonable accuracy.

The script is located in: *pygfunction/examples/comparison_gfunction_solvers.py*

```
# -*- coding: utf-8 -*-
""" Comparison of solvers for the evaluation of g-functions using uniform and
    equal borehole wall temperatures.

    The g-function of a field of 6x4 boreholes is calculated for a boundary
    condition of uniform borehole wall temperature along the boreholes, equal
    for all boreholes. Three different solvers are compared : 'detailed',
    'similarities' and 'equivalent'. Their accuracy and calculation time are
```

(continues on next page)

[1] Cimmino, M. (2018). Fast calculation of the g-functions of geothermal borehole fields using similarities in the evaluation of the finite line source solution. Journal of Building Performance Simulation, 11 (6), 655-668.

[2] Prieto, C., & Cimmino, M. (2021). Thermal interactions in large irregular fields of geothermal boreholes: the method of equivalent borehole. Journal of Building Performance Simulation, 14 (4), 446-460.

(continued from previous page)

```
    compared using the 'detailed' solver as a reference. This shows that the
    'similarities' solver can evaluate g-functions with high accuracy.

    The g-function of a field of 12x10 boreholes is calculated for a boundary
    condition of uniform borehole wall temperature along the boreholes, equal
    for all boreholes. Two different solvers are compared : 'similarities' and
    'equivalent'. The accuracy and calculation time of the 'equivalent' is
    compared using the 'similarities' solver as a reference. This shows that
    the 'equivalent' solver evaluates g-functions at a very high calculation
    speed while maintaining reasonable accuracy.

"""
import matplotlib.pyplot as plt
import numpy as np
from time import perf_counter

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)
    B = 7.5              # Borehole spacing (m)

    # Thermal properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)

    # g-Function calculation options
    options = {'nSegments': 8,
               'disp': True}

    # Geometrically expanding time vector.
    dt = 100*3600.                    # Time step
    tmax = 3000. * 8760. * 3600.      # Maximum time
    Nt = 15                           # Number of time steps
    ts = H**2/(9.*alpha)              # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)
    lntts = np.log(time/ts)

    # -------------------------------------------------------------------------
    # Borehole field (First bore field)
    # -------------------------------------------------------------------------

    # Field of 6x4 (n=24) boreholes
    N_1 = 6
    N_2 = 4
```

(continues on next page)

(continued from previous page)

```
    field = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # -------------------------------------------------------------------------
    # Evaluate g-functions
    # -------------------------------------------------------------------------
    t0 = perf_counter()
    gfunc_detailed = gt.gfunction.gFunction(
        field, alpha, time=time, options=options, method='detailed')
    t1 = perf_counter()
    t_detailed = t1 - t0
    gfunc_similarities = gt.gfunction.gFunction(
        field, alpha, time=time, options=options, method='similarities')
    t2 = perf_counter()
    t_similarities = t2 - t1
    gfunc_equivalent = gt.gfunction.gFunction(
        field, alpha, time=time, options=options, method='equivalent')
    t3 = perf_counter()
    t_equivalent = t3 - t2

    # -------------------------------------------------------------------------
    # Plot results
    # -------------------------------------------------------------------------
    # Draw g-functions
    ax = gfunc_detailed.visualize_g_function().axes[0]
    ax.plot(lntts, gfunc_similarities.gFunc, 'bx')
    ax.plot(lntts, gfunc_equivalent.gFunc, 'ro')
    ax.legend([f'detailed (t = {t_detailed:.3f} sec)',
               f'similarities (t = {t_similarities:.3f} sec)',
               f'equivalent (t = {t_equivalent:.3f} sec)'])
    ax.set_title(f'Field of {N_1} by {N_2} boreholes')
    plt.tight_layout()

    # Draw absolute error
    # Configure figure and axes
    fig = gt.utilities._initialize_figure()
    ax = fig.add_subplot(111)
    # Axis labels
    ax.set_xlabel(r'ln$(t/t_s)$')
    ax.set_ylabel(r'Absolute error')
    gt.utilities._format_axes(ax)
    # Absolute error
    ax.plot(lntts, np.abs(gfunc_similarities.gFunc - gfunc_detailed.gFunc),
            '-', label='similarities')
    ax.plot(lntts, np.abs(gfunc_equivalent.gFunc - gfunc_detailed.gFunc),
            '--', label='equivalent')
    ax.legend()
    ax.set_title(f"Absolute error relative to the 'detailed' solver "
                 f"(Field of {N_1} by {N_2} boreholes)")
    # Adjust to plot window
    fig.tight_layout()

    # Draw relative error
```

(continues on next page)

(continued from previous page)

```
    # Configure figure and axes
    fig = gt.utilities._initialize_figure()
    ax = fig.add_subplot(111)
    # Axis labels
    ax.set_xlabel(r'ln$(t/t_s)$')
    ax.set_ylabel(r'Relative error')
    gt.utilities._format_axes(ax)
    # Relative error
    gFunc_ref = gfunc_detailed.gFunc  # reference g-function
    ax.plot(lntts, (gfunc_similarities.gFunc - gFunc_ref) / gFunc_ref,
            '-', label='similarities')
    ax.plot(lntts, (gfunc_equivalent.gFunc - gFunc_ref) / gFunc_ref,
            '--', label='equivalent')
    ax.legend()
    ax.set_title(f"Relative error relative to the 'detailed' solver "
                 f"(Field of {N_1} by {N_2} boreholes)")
    # Adjust to plot window
    fig.tight_layout()

    # -------------------------------------------------------------------------
    # Borehole field (Second bore field)
    # -------------------------------------------------------------------------

    # Field of 6x4 (n=24) boreholes
    N_1 = 12
    N_2 = 10
    field = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

    # -------------------------------------------------------------------------
    # Evaluate g-functions
    # -------------------------------------------------------------------------
    gfunc_similarities = gt.gfunction.gFunction(
        field, alpha, time=time, options=options, method='similarities')
    t2 = perf_counter()
    t_similarities = t2 - t1
    gfunc_equivalent = gt.gfunction.gFunction(
        field, alpha, time=time, options=options, method='equivalent')
    t3 = perf_counter()
    t_equivalent = t3 - t2

    # -------------------------------------------------------------------------
    # Plot results
    # -------------------------------------------------------------------------
    # Draw g-functions
    ax = gfunc_similarities.visualize_g_function().axes[0]
    ax.plot(lntts, gfunc_equivalent.gFunc, 'ro')
    ax.legend([f'similarities (t = {t_similarities:.3f} sec)',
               f'equivalent (t = {t_equivalent:.3f} sec)'])
    ax.set_title(f'Field of {N_1} by {N_2} boreholes')
    plt.tight_layout()

    # Draw absolute error
```

(continues on next page)

(continued from previous page)

```
    # Configure figure and axes
    fig = gt.utilities._initialize_figure()
    ax = fig.add_subplot(111)
    # Axis labels
    ax.set_xlabel(r'ln$(t/t_s)$')
    ax.set_ylabel(r'Absolute error')
    gt.utilities._format_axes(ax)
    # Absolute error
    ax.plot(lntts, np.abs(gfunc_equivalent.gFunc - gfunc_similarities.gFunc),
            label='equivalent')
    ax.legend()
    ax.set_title(f"Absolute error relative to the 'similarities' solver "
                 f"(Field of {N_1} by {N_2} boreholes)")
    # Adjust to plot window
    fig.tight_layout()

    # Draw relative error
    # Configure figure and axes
    fig = gt.utilities._initialize_figure()
    ax = fig.add_subplot(111)
    # Axis labels
    ax.set_xlabel(r'ln$(t/t_s)$')
    ax.set_ylabel(r'Relative error')
    gt.utilities._format_axes(ax)
    # Relative error
    ax.plot(lntts, (gfunc_equivalent.gFunc - gfunc_similarities.gFunc) / gfunc_
↪similarities.gFunc,
            label='equivalent')
    ax.legend()
    ax.set_title(f"Relative error relative to the 'similarities' solver "
                 f"(Field of {N_1} by {N_2} boreholes)")
    # Adjust to plot window
    fig.tight_layout()

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.8 Calculation of g-functions with unequal numbers of segments

This example demonstrates the use of the *g-function* module to calculate g-functions using a boundary condition of uniform and equal borehole wall temperature for all boreholes. The total rate of heat extraction in the bore field is constant. The discretization along three of the boreholes is refined for the calculation of the g-function and to draw the heat extraction rate profiles along their lengths.

The following script generates the g-functions of a rectangular field of 6 x 4. g-Functions using equal and unequal

numbers of segments are compared.

The script is located in: *pygfunction/examples/unequal_segments.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of g-functions with varied declaration of segments
    using uniform borehole wall temperature.

    The g-functions of a field of 6x4 boreholes is calculated with unequal
    number of segments.
"""
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
import numpy as np

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)
    B = 7.5              # Borehole spacing (m)

    # Thermal properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)

    # Geometrically expanding time vector.
    dt = 100*3600.                   # Time step
    tmax = 3000. * 8760. * 3600.     # Maximum time
    Nt = 25                          # Number of time steps
    ts = H**2/(9.*alpha)             # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)

    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    # Field of 6x4 (n=24) boreholes
    N_1 = 6
    N_2 = 4
    boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
    gt.boreholes.visualize_field(boreField)

    # -------------------------------------------------------------------------
    # Evaluate g-functions with different segment options
    # -------------------------------------------------------------------------

    # The 'similarities' method is used to consider unequal numbers of segments
```

(continues on next page)

(continued from previous page)

```
    # per borehole and to plot heat extraction rate profiles along
    # individual boreholes
    method = 'similarities'

    # Calculate g-function with equal number of segments for all boreholes and
    # uniform segment lengths
    nSegments = 24
    options = {'nSegments': nSegments,
               'segment_ratios': None,
               'disp': True}

    gfunc_equal = gt.gfunction.gFunction(
        boreField, alpha, time=time, options=options, method=method)

    # Calculate g-function with predefined number of segments for each
    # borehole, the segment lengths will be uniform along each borehole, but
    # the number of segments for the boreholes can be unequal.

    # Boreholes 12, 14 and 18 have more segments than the others and their
    # heat extraction rate profiles are plotted.
    nSegments = [12] * len(boreField)
    nSegments[12] = 24
    nSegments[14] = 24
    nSegments[18] = 24
    options = {'nSegments': nSegments,
               'segment_ratios': None,
               'disp': True,
               'profiles': True}

    gfunc_unequal = gt.gfunction.gFunction(
        boreField, alpha, time=time, options=options, method=method)

    # Calculate g-function with equal number of segments for each borehole,
    # unequal segment lengths along the length of the borehole defined by
    # segment ratios. The segment ratios for each borehole are the same.

    nSegments = 8
    # Define the segment ratios for each borehole in each segment
    # the segment lengths are defined top to bottom left to right
    segment_ratios = np.array([0.05, 0.10, 0.10, 0.25, 0.25, 0.10, 0.10, 0.05])
    options = {'nSegments': nSegments,
               'segment_ratios': segment_ratios,
               'disp': True,
               'profiles': True}

    g_func_predefined = gt.gfunction.gFunction(
        boreField, alpha, time=time, options=options, method=method)

    # -------------------------------------------------------------------------
    # Plot g-functions
    # -------------------------------------------------------------------------

```

(continues on next page)

(continued from previous page)

```
    ax = gfunc_equal.visualize_g_function().axes[0]
    ax.plot(np.log(time/ts), gfunc_unequal.gFunc, 'r-.')
    ax.plot(np.log(time/ts), g_func_predefined.gFunc, 'k-.')
    ax.legend(['Equal number of segments',
               'Unequal number of segments',
               'Unequal segment lengths'])
    plt.tight_layout()

    # Heat extraction rate profiles
    fig = gfunc_unequal.visualize_heat_extraction_rates(
        iBoreholes=[18, 12, 14])
    fig.suptitle('Heat extraction rates (unequal number of segments)')
    fig.tight_layout()
    fig = g_func_predefined.visualize_heat_extraction_rates(
        iBoreholes=[18, 12, 14])
    fig.suptitle('Heat extraction rates (unequal segment lengths)')
    fig.tight_layout()
    fig = gfunc_unequal.visualize_heat_extraction_rate_profiles(
        iBoreholes=[18, 12, 14])
    fig.suptitle('Heat extraction rate profiles (unequal number of segments)')
    fig.tight_layout()
    fig = g_func_predefined.visualize_heat_extraction_rate_profiles(
        iBoreholes=[18, 12, 14])
    fig.suptitle('Heat extraction rate profiles (unequal segment lengths)')
    fig.tight_layout()

    return


# Main function
if __name__ == '__main__':
    main()
```

4.9 Calculation of g-Functions computed with non-uniform segment lengths

This example demonstrates the use of the `utilities` module to determine discretized segment ratios along a borehole.

The following script computes g-Functions for a field of 6x4 boreholes utilizing the MIFT and UBWT boundary conditions with a 48 segments per borehole and equal segment lengths. The MIFT and UBWT g-functions are computed with only 8 segments per borehole and non-uniform segment lengths. RMSE values are compared. It is shown that g-functions can be calculated accurately using a small number of segments.

The script is located in: *pygfunction/examples/discretize_boreholes.py*

```
# -*- coding: utf-8 -*-
""" Example of g-function calculation using non-uniform segment lengths along
    the boreholes.

    The g-functions of a field of 6x4 boreholes are calculated for two
```

(continues on next page)

(continued from previous page)

```
    boundary conditions : (1) a uniform borehole wall temperature along the
    boreholes equal for all boreholes, and (2) an equal inlet fluid
    temperature into the boreholes. g-Functions using 8 segments in a
    non-uniform discretization are compared to reference g-functions
    calculated using 48 segments of equal lengths. It is shown that g-functions
    can be calculated accurately using a small number of segments.
"""

import pygfunction as gt
from numpy import pi
import matplotlib.pyplot as plt
import numpy as np


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0             # Borehole buried depth (m)
    H = 150.0           # Borehole length (m)
    r_b = 0.075         # Borehole radius (m)
    B = 7.5             # Borehole spacing (m)

    # Pipe dimensions
    r_out = 0.0211      # Pipe outer radius (m)
    r_in = 0.0147       # Pipe inner radius (m)
    D_s = 0.052         # Shank spacing (m)
    epsilon = 1.0e-6    # Pipe roughness (m)

    # Pipe positions
    # Single U-tube [(x_in, y_in), (x_out, y_out)]
    pos_pipes = [(-D_s, 0.), (D_s, 0.)]

    # Ground properties
    alpha = 1.0e-6      # Ground thermal diffusivity (m2/s)
    k_s = 2.0           # Ground thermal conductivity (W/m.K)

    # Grout properties
    k_g = 1.0           # Grout thermal conductivity (W/m.K)

    # Pipe properties
    k_p = 0.4           # Pipe thermal conductivity (W/m.K)

    # Fluid properties
    m_flow_borehole = 0.25  # Total fluid mass flow rate per borehole (kg/s)
    # The fluid is propylene-glycol (20 %) at 20 degC
    fluid = gt.media.Fluid('MPG', 20.)
    cp_f = fluid.cp     # Fluid specific isobaric heat capacity (J/kg.K)
    rho_f = fluid.rho   # Fluid density (kg/m3)
    mu_f = fluid.mu     # Fluid dynamic viscosity (kg/m.s)
```

(continues on next page)

(continued from previous page)

```
    k_f = fluid.k       # Fluid thermal conductivity (W/m.K)

    # g-Function calculation options

    # Number of segments used in the reference calculation with uniform
    # discretization
    nSegments_uniform = 48
    options_uniform = {'nSegments': nSegments_uniform,
                       'segment_ratios': None,
                       'disp': True}
    # Number of segments used in the calculation with non-uniform
    # discretization
    nSegments_unequal = 8
    segment_ratios = gt.utilities.segment_ratios(
        nSegments_unequal, end_length_ratio=0.02)
    options_unequal = {'nSegments': nSegments_unequal,
                       'segment_ratios': segment_ratios,
                       'disp':True}

    # Geometrically expanding time vector.
    dt = 100*3600.                  # Time step
    tmax = 3000. * 8760. * 3600.    # Maximum time
    Nt = 25                         # Number of time steps
    ts = H**2/(9.*alpha)            # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)

    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    # Field of 6x4 (n=24) boreholes
    N_1 = 6
    N_2 = 4
    boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
    gt.boreholes.visualize_field(boreField)
    nBoreholes = len(boreField)

    # -------------------------------------------------------------------------
    # Initialize pipe model
    # -------------------------------------------------------------------------

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
        r_in, r_out, k_p)
    # Fluid to inner pipe wall thermal resistance (Single U-tube)
    m_flow_pipe = m_flow_borehole
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f = 1.0/(h_f*2*pi*r_in)

    # Single U-tube, same for all boreholes in the bore field
    UTubes = []
```

(continues on next page)

(continued from previous page)

```
    for borehole in boreField:
        SingleUTube = gt.pipes.SingleUTube(
            pos_pipes, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
        UTubes.append(SingleUTube)
    m_flow_network = m_flow_borehole*nBoreholes

    # Network of boreholes connected in parallel
    network = gt.networks.Network(
        boreField, UTubes, m_flow_network=m_flow_network, cp_f=cp_f)

    # -------------------------------------------------------------------------
    # Evaluate the g-functions for the borefield
    # -------------------------------------------------------------------------

    # Compute g-function for the converged MIFT case with equal number of
    # segments per borehole, and equal segment lengths along the boreholes
    gfunc_MIFT_uniform = gt.gfunction.gFunction(
        network, alpha, time=time, boundary_condition='MIFT',
        options=options_uniform)

    # Calculate the g-function for uniform borehole wall temperature
    gfunc_UBWT_uniform = gt.gfunction.gFunction(
        boreField, alpha, time=time, boundary_condition='UBWT',
        options=options_uniform)

    # Compute g-function for the MIFT case with equal number of segments per
    # borehole, and non-uniform segment lengths along the boreholes
    gfunc_MIFT_unequal = gt.gfunction.gFunction(
        network, alpha, time=time, boundary_condition='MIFT',
        options=options_unequal)

    # Calculate the g-function for uniform borehole wall temperature
    gfunc_UBWT_unequal = gt.gfunction.gFunction(
        boreField, alpha, time=time, boundary_condition='UBWT',
        options=options_unequal)

    # Compute the rmse between the reference cases and the discretized
    # (predicted) cases
    RMSE_MIFT = RMSE(gfunc_MIFT_uniform.gFunc, gfunc_MIFT_unequal.gFunc)
    print(f'RMSE (MIFT) = {RMSE_MIFT:.5f}')
    RMSE_UBWT = RMSE(gfunc_UBWT_uniform.gFunc, gfunc_UBWT_unequal.gFunc)
    print(f'RMSE (UBWT) = {RMSE_UBWT:.5f}')

    # -------------------------------------------------------------------------
    # Plot g-functions
    # -------------------------------------------------------------------------

    ax = gfunc_MIFT_uniform.visualize_g_function().axes[0]
    ax.plot(np.log(time / ts), gfunc_UBWT_uniform.gFunc)
    ax.plot(np.log(time / ts), gfunc_MIFT_unequal.gFunc, 'o')
    ax.plot(np.log(time / ts), gfunc_UBWT_unequal.gFunc, 'o')
    ax.legend(
```

(continues on next page)

(continued from previous page)

```
        ['Equal inlet temperature (uniform segments)',
         'Uniform borehole wall temperature (uniform segments)',
         'Equal inlet temperature (non-uniform segments)',
         'Uniform borehole wall temperature (non-uniform segments)'])
    plt.tight_layout()

    return


def RMSE(reference, predicted):
    rmse = np.linalg.norm(predicted - reference) / len(reference)
    return rmse


# Main function
if __name__ == '__main__':
    main()
```

4.10 Simulation of a borehole using load aggregation

This example demonstrates the use of the *load agggregation* module to predict the borehole wall temperature of a single temperature with known heat extraction rates.

The g-function of a single borehole is first calculated. Then, the borehole wall temperature variations are calculated using the load aggregation scheme of Claesson and Javed[1]. The time-variation of heat extraction rates is given by the synthetic load profile of Bernier et al.[2].

The following script validates the load aggregation scheme with the exact solution obtained from convolution in the Fourier domain (see ref.[3]).

The script is located in: *pygfunction/examples/load_aggregation.py*

```
# -*- coding: utf-8 -*-
""" Example of simulation of a geothermal system.

    The g-function of a single borehole is calculated for boundary condition of
    uniform borehole wall temperature along the borehole. Then, the borehole
    wall temperature variations resulting from a time-varying load profile
    are simulated using the aggregation method of Claesson and Javed (2012).

"""
import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import pi
from scipy.interpolate import interp1d
from scipy.signal import fftconvolve
```

(continues on next page)

[1] Claesson, J., & Javed, S. (2011). A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. ASHRAE Transactions, 118 (1): 530–539.

[2] Bernier, M., Pinel, P., Labib, R. and Paillot, R. (2004). A multiple load aggregation algorithm for annual hourly simulations of GCHP systems. HVAC&R Research 10 (4): 471–487.

[3] Marcotte, D., & Pasquier, P. (2008). Fast fluid and ground temperature computation for geothermal ground-loop heat exchanger systems. Geothermics, 37 (6) : 651-665.

(continued from previous page)

```

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)

    # Ground properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)
    k_s = 2.0            # Ground thermal conductivity (W/m.K)
    T_g = 10.0           # Undisturbed ground temperature (degC)

    # g-Function calculation options
    options = {'nSegments': 8,
               'disp': True}

    # Simulation parameters
    dt = 3600.                   # Time step (s)
    tmax = 20.*8760. * 3600.     # Maximum time (s)
    Nt = int(np.ceil(tmax/dt))  # Number of time steps
    time = dt * np.arange(1, Nt+1)

    # Evaluate heat extraction rate
    Q_b = synthetic_load(time/3600.)

    # Load aggregation scheme
    LoadAgg = gt.load_aggregation.ClaessonJaved(dt, tmax)

    # -------------------------------------------------------------------------
    # Calculate g-function
    # -------------------------------------------------------------------------

    # The field contains only one borehole
    boreField = [gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)]
    # Get time values needed for g-function evaluation
    time_req = LoadAgg.get_times_for_simulation()
    # Calculate g-function
    gFunc = gt.gfunction.gFunction(
        boreField, alpha, time=time_req, options=options)
    # Initialize load aggregation scheme
    LoadAgg.initialize(gFunc.gFunc/(2*pi*k_s))

    # -------------------------------------------------------------------------
    # Simulation
    # -------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
    T_b = np.zeros(Nt)
    for i, (t, Q_b_i) in enumerate(zip(time, Q_b)):
        # Increment time step by (1)
        LoadAgg.next_time_step(t)

        # Apply current load
        LoadAgg.set_current_load(Q_b_i/H)

        # Evaluate borehole wall temeprature
        deltaT_b = LoadAgg.temporal_superposition()
        T_b[i] = T_g - deltaT_b

    # -------------------------------------------------------------------------
    # Calculate exact solution from convolution in the Fourier domain
    # -------------------------------------------------------------------------

    # Heat extraction rate increment
    dQ = np.zeros(Nt)
    dQ[0] = Q_b[0]
    dQ[1:] = Q_b[1:] - Q_b[:-1]
    # Interpolated g-function
    g = interp1d(time_req, gFunc.gFunc)(time)

    # Convolution in Fourier domain
    T_b_exact = T_g - fftconvolve(dQ, g/(2.0*pi*k_s*H), mode='full')[0:Nt]

    # -------------------------------------------------------------------------
    # plot results
    # -------------------------------------------------------------------------

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(311)
    # Axis labels
    ax1.set_xlabel(r'$t$ [hours]')
    ax1.set_ylabel(r'$Q_b$ [W]')
    gt.utilities._format_axes(ax1)

    hours = np.arange(1, Nt+1) * dt / 3600.
    ax1.plot(hours, Q_b)

    ax2 = fig.add_subplot(312)
    # Axis labels
    ax2.set_xlabel(r'$t$ [hours]')
    ax2.set_ylabel(r'$T_b$ [degC]')
    gt.utilities._format_axes(ax2)

    ax2.plot(hours, T_b)
    ax2.plot(hours, T_b_exact, 'k.')

```

(continues on next page)

(continued from previous page)

```
    ax3 = fig.add_subplot(313)
    # Axis labels
    ax3.set_xlabel(r'$t$ [hours]')
    ax3.set_ylabel(r'Error [degC]')
    gt.utilities._format_axes(ax3)

    ax3.plot(hours, T_b - T_b_exact)

    # Adjust to plot window
    plt.tight_layout()

    return


def synthetic_load(x):
    """
    Synthetic load profile of Bernier et al. (2004).

    Returns load y (in watts) at time x (in hours).
    """
    A = 2000.0
    B = 2190.0
    C = 80.0
    D = 2.0
    E = 0.01
    F = 0.0
    G = 0.95

    func = (168.0-C)/168.0
    for i in [1,2,3]:
        func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0)-1.0) \
                          *(np.sin(pi*i/84.0*(x-B)))
    func = func*A*np.sin(pi/12.0*(x-B)) \
           *np.sin(pi/4380.0*(x-B))

    y = func + (-1.0)**np.floor(D/8760.0*(x-B))*abs(func) \
      + E*(-1.0)**np.floor(D/8760.0*(x-B))/np.sign(np.cos(D*pi/4380.0*(x-F))+G)
    return -y


# Main function
if __name__ == '__main__':
    main()
```

References

4.11 Compare the accuracy and speed of different load aggregation algorithms

This example compares the simulation times and the accuracy of borehole wall temperature predictions of different load aggregation algorithms implemented into the *load agggregation* module.

The g-function of a single borehole is first calculated. Then, the borehole wall temperature variations are calculated using the load aggregation schemes of Bernier et al.[1], Liu[2], and Claesson and Javed[3],. The time-variation of heat extraction rates is given by the synthetic load profile of Bernier et al.[Page 119, 1].

The following script validates the load aggregation schemes with the exact solution obtained from convolution in the Fourier domain (see ref.[4]).

The script is located in: *pygfunction/examples/comparison_load_aggregation.py*

```
# -*- coding: utf-8 -*-
""" Comparison of the accuracy and computational speed of different load
    aggregation algorithms.

    The g-function of a single borehole is calculated for boundary condition of
    uniform borehole wall temperature along the borehole. Then, the borehole
    wall temperature variations resulting from a time-varying load profile
    are simulated using the aggregation methods of Bernier et al. (2004),
    Liu (2005), and Claesson and Javed (2012). Results are compared to the
    exact solution obtained by convolution in the Fourier domain.

    Default parameters are used for each of the aggregation schemes.

"""
from time import perf_counter

import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import pi
from scipy.interpolate import interp1d
from scipy.signal import fftconvolve

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------
```

(continues on next page)

[1] Bernier, M., Pinel, P., Labib, R. and Paillot, R. (2004). A multiple load aggregation algorithm for annual hourly simulations of GCHP systems. HVAC&R Research 10 (4): 471–487.

[2] Liu, X. (2005). Development and experimental validation of simulation of hydronic snow melting systems for bridges. Ph.D. Thesis. Oklahoma State University.

[3] Claesson, J., & Javed, S. (2011). A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. ASHRAE Transactions, 118 (1): 530–539.

[4] Marcotte, D., & Pasquier, P. (2008). Fast fluid and ground temperature computation for geothermal ground-loop heat exchanger systems. Geothermics, 37 (6) : 651-665.

(continued from previous page)

```
    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)

    # Ground properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)
    k_s = 2.0            # Ground thermal conductivity (W/m.K)
    T_g = 10.0           # Undisturbed ground temperature (degC)

    # g-Function calculation options
    options = {'nSegments': 8,
               'disp': True}

    # Simulation parameters
    dt = 3600.                   # Time step (s)
    tmax = 20.*8760. * 3600.     # Maximum time (s)
    Nt = int(np.ceil(tmax/dt))  # Number of time steps
    time = dt * np.arange(1, Nt+1)

    # Evaluate heat extraction rate
    Q_b = synthetic_load(time/3600.)

    # Load aggregation schemes
    ClaessonJaved = gt.load_aggregation.ClaessonJaved(dt, tmax)
    MLAA = gt.load_aggregation.MLAA(dt, tmax)
    Liu = gt.load_aggregation.Liu(dt, tmax)
    LoadAggSchemes = [ClaessonJaved, MLAA, Liu]
    loadAgg_labels = ['Claesson and Javed', 'MLAA', 'Liu']
    loadAgg_lines = ['b-', 'k--', 'r-.']

    # -------------------------------------------------------------------------
    # Calculate g-function
    # -------------------------------------------------------------------------

    # The field contains only one borehole
    boreField = [gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)]
    # Evaluate the g-function on a geometrically expanding time grid
    time_gFunc = gt.utilities.time_geometric(dt, tmax, 50)
    # Calculate g-function
    gFunc = gt.gfunction.gFunction(
        boreField, alpha, time=time_gFunc, options=options)

    # -------------------------------------------------------------------------
    # Simulation
    # -------------------------------------------------------------------------
    nLoadAgg = len(LoadAggSchemes)
    T_b = np.zeros((nLoadAgg, Nt))

    t_calc = np.zeros(nLoadAgg)
    for n, (LoadAgg, label) in enumerate(zip(LoadAggSchemes, loadAgg_labels)):
```

(continues on next page)

(continued from previous page)

```
        print(f'Simulation using {label} ...')
        # Interpolate g-function at required times
        time_req = LoadAgg.get_times_for_simulation()
        gFunc_int = interp1d(np.hstack([0., time_gFunc]),
                             np.hstack([0., gFunc.gFunc]),
                             kind='cubic',
                             bounds_error=False,
                             fill_value=(0., gFunc.gFunc[-1]))(time_req)
        # Initialize load aggregation scheme
        LoadAgg.initialize(gFunc_int/(2*pi*k_s))

        tic = perf_counter()
        for i in range(Nt):
            # Increment time step by (1)
            LoadAgg.next_time_step(time[i])

            # Apply current load
            LoadAgg.set_current_load(Q_b[i]/H)

            # Evaluate borehole wall temeprature
            deltaT_b = LoadAgg.temporal_superposition()
            T_b[n,i] = T_g - deltaT_b
        toc = perf_counter()
        t_calc[n] = toc - tic

    # -------------------------------------------------------------------------
    # Calculate exact solution from convolution in the Fourier domain
    # -------------------------------------------------------------------------

    # Heat extraction rate increment
    dQ = np.zeros(Nt)
    dQ[0] = Q_b[0]
    dQ[1:] = Q_b[1:] - Q_b[:-1]
    # Interpolated g-function
    g = interp1d(time_gFunc, gFunc.gFunc)(time)

    # Convolution in Fourier domain
    T_b_exact = T_g - fftconvolve(dQ, g/(2.0*pi*k_s*H), mode='full')[0:Nt]

    # -------------------------------------------------------------------------
    # plot results
    # -------------------------------------------------------------------------

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(311)
    # Axis labels
    ax1.set_xlabel(r'$t$ [hours]')
    ax1.set_ylabel(r'$Q_b$ [W]')
    gt.utilities._format_axes(ax1)
    hours = np.array([(j+1)*dt/3600. for j in range(Nt)])
```

(continues on next page)

(continued from previous page)

```
    ax1.plot(hours, Q_b)

    ax2 = fig.add_subplot(312)
    # Axis labels
    ax2.set_xlabel(r'$t$ [hours]')
    ax2.set_ylabel(r'$T_b$ [degC]')
    gt.utilities._format_axes(ax2)
    for T_b_n, line, label in zip(T_b, loadAgg_lines, loadAgg_labels):
        ax2.plot(hours, T_b_n, line, label=label)
    ax2.plot(hours, T_b_exact, 'k.', label='exact')
    ax2.legend()

    ax3 = fig.add_subplot(313)
    # Axis labels
    ax3.set_xlabel(r'$t$ [hours]')
    ax3.set_ylabel(r'Error [degC]')
    gt.utilities._format_axes(ax3)
    for T_b_n, line, label in zip(T_b, loadAgg_lines, loadAgg_labels):
        ax3.plot(hours, T_b_n - T_b_exact, line, label=label)
    # Adjust to plot window
    plt.tight_layout()

    # --------------------------------------------------------------------
    # Print performance metrics
    # --------------------------------------------------------------------

    # Maximum errors in evaluation of borehole wall temperatures
    maxError = np.array([np.max(np.abs(T_b_n-T_b_exact)) for T_b_n in T_b])
    # Print results
    print('Simulation results')
    for label, maxError_n, t_calc_n in zip(loadAgg_labels, maxError, t_calc):
        print()
        print((f' {label} ').center(60, '-'))
        print(f'Maximum absolute error : {maxError_n:.3f} degC')
        print(f'Calculation time : {t_calc_n:.3f} sec')

    return


def synthetic_load(x):
    """
    Synthetic load profile of Bernier et al. (2004).

    Returns load y (in watts) at time x (in hours).
    """
    A = 2000.0
    B = 2190.0
    C = 80.0
    D = 2.0
    E = 0.01
    F = 0.0
    G = 0.95
```

(continues on next page)

(continued from previous page)

```
    func = (168.0-C)/168.0
    for i in [1, 2, 3]:
        func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0) - 1.0) \
                          *(np.sin(pi*i/84.0*(x - B)))
    func = func*A*np.sin(pi/12.0*(x - B)) \
        *np.sin(pi/4380.0*(x - B))

    y = func + (-1.0)**np.floor(D/8760.0*(x - B))*abs(func) \
        + E*(-1.0)**np.floor(D/8760.0*(x - B)) \
        /np.sign(np.cos(D*pi/4380.0*(x - F)) + G)
    return -y


# Main function
if __name__ == '__main__':
    main()
```

References

4.12 Simulation of fluid temperatures in a borehole

This example demonstrates the use of the *pipes* module to predict the fluid temperature variations in a borehole with known heat extraction rates.

The g-function of a single borehole is first calculated. Then, the borehole wall temperature variations are calculated using the load aggregation scheme of Claesson and Javed[1]. The time-variation of heat extraction rates is given by the synthetic load profile of Bernier et al.[2]. Three pipe configurations are compared: (1) a single U-tube, using the model of Eskilson and Claesson[3], (2) a double U-tube in parallel, using the model of Cimmino[4], and (3) a double U-tube in series, using the model of Cimmino[Page 123, 4].

The script is located in: *pygfunction/examples/fluid_temperature.py*

```
# -*- coding: utf-8 -*-
""" Example of simulation of a geothermal system and comparison between
    single and double U-tube pipe configurations.

    The g-function of a single borehole is calculated for boundary condition of
    uniform borehole wall temperature along the borehole. Then, the borehole
    wall temperature variations resulting from a time-varying load profile
    are simulated using the aggregation method of Claesson and Javed (2012).
    Predicted outlet fluid temperatures of three pipe configurations
    (single U-tube, double U-tube in series, double U-tube in parallel) are
    compared.
```

(continues on next page)

[1] Claesson, J., & Javed, S. (2011). A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. ASHRAE Transactions, 118 (1): 530–539.

[2] Bernier, M., Pinel, P., Labib, R. and Paillot, R. (2004). A multiple load aggregation algorithm for annual hourly simulations of GCHP systems. HVAC&R Research 10 (4): 471–487.

[3] Eskilson, P., & Claesson, J. (1988). Simulation model for thermally interacting heat extraction boreholes. Numerical Heat Transfer 13 : 149-165.

[4] Cimmino, M. (2016). Fluid and borehole wall temperature profiles in vertical geothermal boreholes with multiple U-tubes. Renewable Energy 96 : 137-147.

(continued from previous page)

```
12
13  """
14  import matplotlib.pyplot as plt
15  import numpy as np
16  from scipy.constants import pi
17
18  import pygfunction as gt
19
20
21  def main():
22      # -------------------------------------------------------------------------
23      # Simulation parameters
24      # -------------------------------------------------------------------------
25
26      # Borehole dimensions
27      D = 4.0              # Borehole buried depth (m)
28      H = 150.0            # Borehole length (m)
29      r_b = 0.075          # Borehole radius (m)
30
31      # Pipe dimensions
32      rp_out = 0.0211      # Pipe outer radius (m)
33      rp_in = 0.0147       # Pipe inner radius (m)
34      D_s = 0.052          # Shank spacing (m)
35      epsilon = 1.0e-6     # Pipe roughness (m)
36
37      # Pipe positions
38      # Single U-tube [(x_in, y_in), (x_out, y_out)]
39      pos_single = [(-D_s, 0.), (D_s, 0.)]
40      # Double U-tube [(x_in1, y_in1), (x_in2, y_in2),
41      #                (x_out1, y_out1), (x_in2, y_in2)]
42      # Note: in series configuration, fluid enters pipe (in,1), exits (out,1),
43      # then enters (in,2) and finally exits (out,2)
44      pos_double = [(-D_s, 0.), (0., -D_s), (D_s, 0.), (0., D_s)]
45
46      # Ground properties
47      alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)
48      k_s = 2.0            # Ground thermal conductivity (W/m.K)
49      T_g = 10.0           # Undisturbed ground temperature (degC)
50
51      # Grout properties
52      k_g = 1.0            # Grout thermal conductivity (W/m.K)
53
54      # Pipe properties
55      k_p = 0.4            # Pipe thermal conductivity (W/m.K)
56
57      # Fluid properties
58      m_flow = 0.25        # Total fluid mass flow rate (kg/s)
59      # The fluid is propylene-glycol (20 %) at 20 degC
60      fluid = gt.media.Fluid('MPG', 20.)
61      cp_f = fluid.cp      # Fluid specific isobaric heat capacity (J/kg.K)
62      den_f = fluid.rho    # Fluid density (kg/m3)
63      visc_f = fluid.mu    # Fluid dynamic viscosity (kg/m.s)
```

(continues on next page)

(continued from previous page)

```
    k_f = fluid.k       # Fluid thermal conductivity (W/m.K)

    # g-Function calculation options
    options = {'nSegments': 8,
               'disp': True}

    # Simulation parameters
    dt = 3600.                  # Time step (s)
    tmax = 1.*8760. * 3600.     # Maximum time (s)
    Nt = int(np.ceil(tmax/dt))  # Number of time steps
    time = dt * np.arange(1, Nt+1)

    # Evaluate heat extraction rate
    Q = synthetic_load(time/3600.)

    # Load aggregation scheme
    LoadAgg = gt.load_aggregation.ClaessonJaved(dt, tmax)

    # -------------------------------------------------------------------------
    # Calculate g-function
    # -------------------------------------------------------------------------

    # The field contains only one borehole
    borehole = gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)
    boreField = [borehole]
    # Get time values needed for g-function evaluation
    time_req = LoadAgg.get_times_for_simulation()
    # Calculate g-function
    gFunc = gt.gfunction.gFunction(
        boreField, alpha, time=time_req, options=options)
    # gt.gfunction.uniform_temperature(boreField, time_req, alpha,
    #                                  nSegments=nSegments)
    # Initialize load aggregation scheme
    LoadAgg.initialize(gFunc.gFunc/(2*pi*k_s))

    # -------------------------------------------------------------------------
    # Initialize pipe models
    # -------------------------------------------------------------------------

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
        rp_in, rp_out, k_p)
    # Fluid to inner pipe wall thermal resistance (Single U-tube and double
    # U-tube in series)
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow, rp_in, visc_f, den_f, k_f, cp_f, epsilon)
    R_f_ser = 1.0/(h_f*2*pi*rp_in)
    # Fluid to inner pipe wall thermal resistance (Double U-tube in parallel)
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow/2, rp_in, visc_f, den_f, k_f, cp_f, epsilon)
    R_f_par = 1.0/(h_f*2*pi*rp_in)

```

(continues on next page)

(continued from previous page)

```
    # Single U-tube
    SingleUTube = gt.pipes.SingleUTube(pos_single, rp_in, rp_out,
                                       borehole, k_s, k_g, R_f_ser + R_p)
    # Double U-tube (parallel)
    DoubleUTube_par = gt.pipes.MultipleUTube(pos_double, rp_in, rp_out,
                                             borehole, k_s, k_g, R_f_par + R_p,
                                             nPipes=2, config='parallel')
    # Double U-tube (series)
    DoubleUTube_ser = gt.pipes.MultipleUTube(pos_double, rp_in, rp_out,
                                             borehole, k_s, k_g, R_f_ser + R_p,
                                             nPipes=2, config='series')

    # -------------------------------------------------------------------------
    # Simulation
    # -------------------------------------------------------------------------

    T_b = np.zeros(Nt)
    T_f_in_single = np.zeros(Nt)
    T_f_in_double_par = np.zeros(Nt)
    T_f_in_double_ser = np.zeros(Nt)
    T_f_out_single = np.zeros(Nt)
    T_f_out_double_par = np.zeros(Nt)
    T_f_out_double_ser = np.zeros(Nt)
    for i, (t, Q_b_i) in enumerate(zip(time, Q)):
        # Increment time step by (1)
        LoadAgg.next_time_step(t)

        # Apply current load
        LoadAgg.set_current_load(Q_b_i/H)

        # Evaluate borehole wall temperature
        deltaT_b = LoadAgg.temporal_superposition()
        T_b[i] = T_g - deltaT_b

        # Evaluate inlet fluid temperature
        T_f_in_single[i] = SingleUTube.get_inlet_temperature(
                Q[i], T_b[i], m_flow, cp_f)
        T_f_in_double_par[i] = DoubleUTube_par.get_inlet_temperature(
                Q[i], T_b[i], m_flow, cp_f)
        T_f_in_double_ser[i] = DoubleUTube_ser.get_inlet_temperature(
                Q[i], T_b[i], m_flow, cp_f)

        # Evaluate outlet fluid temperature
        T_f_out_single[i] = SingleUTube.get_outlet_temperature(
                T_f_in_single[i],  T_b[i], m_flow, cp_f)
        T_f_out_double_par[i] = DoubleUTube_par.get_outlet_temperature(
                T_f_in_double_par[i],  T_b[i], m_flow, cp_f)
        T_f_out_double_ser[i] = DoubleUTube_ser.get_outlet_temperature(
                T_f_in_double_ser[i],  T_b[i], m_flow, cp_f)

    # -------------------------------------------------------------------------
    # Plot hourly heat extraction rates and temperatures
```

(continues on next page)

(continued from previous page)

```
    # -------------------------------------------------------------------------

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(211)
    # Axis labels
    ax1.set_xlabel(r'Time [hours]')
    ax1.set_ylabel(r'Total heat extraction rate [W]')
    gt.utilities._format_axes(ax1)

    # Plot heat extraction rates
    hours = np.arange(1, Nt+1) * dt / 3600.
    ax1.plot(hours, Q)

    ax2 = fig.add_subplot(212)
    # Axis labels
    ax2.set_xlabel(r'Time [hours]')
    ax2.set_ylabel(r'Temperature [degC]')
    gt.utilities._format_axes(ax2)

    # Plot temperatures
    ax2.plot(hours, T_b, 'k-', lw=1.5, label='Borehole wall')
    ax2.plot(hours, T_f_out_single, '--',
             label='Outlet, single U-tube')
    ax2.plot(hours, T_f_out_double_par, '-.',
             label='Outlet, double U-tube (parallel)')
    ax2.plot(hours, T_f_out_double_ser, ':',
             label='Outlet, double U-tube (series)')
    ax2.legend()

    # Adjust to plot window
    plt.tight_layout()

    # -------------------------------------------------------------------------
    # Plot fluid temperature profiles
    # -------------------------------------------------------------------------

    # Evaluate temperatures at nz evenly spaced depths along the borehole
    # at the (it+1)-th time step
    nz = 20
    it = 8724
    z = np.linspace(0., H, num=nz)
    T_f_single = SingleUTube.get_temperature(z,
                                             T_f_in_single[it],
                                             T_b[it],
                                             m_flow,
                                             cp_f)
    T_f_double_par = DoubleUTube_par.get_temperature(z,
                                                     T_f_in_double_par[it],
                                                     T_b[it],
                                                     m_flow,
```

(continues on next page)

(continued from previous page)

```
                                                    cp_f)
    T_f_double_ser = DoubleUTube_ser.get_temperature(z,
                                                    T_f_in_double_ser[it],
                                                    T_b[it],
                                                    m_flow,
                                                    cp_f)

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax3 = fig.add_subplot(131)
    # Axis labels
    ax3.set_xlabel(r'Temperature [degC]')
    ax3.set_ylabel(r'Depth from borehole head [m]')
    gt.utilities._format_axes(ax3)

    # Plot temperatures
    ax3.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]), 'k--')
    ax3.plot(T_f_single, z, 'b-')
    ax3.legend(['Borehole wall', 'Fluid'])

    ax4 = fig.add_subplot(132)
    # Axis labels
    ax4.set_xlabel(r'Temperature [degC]')
    ax4.set_ylabel(r'Depth from borehole head [m]')
    gt.utilities._format_axes(ax4)

    # Plot temperatures
    ax4.plot(T_f_double_par, z, 'b-')
    ax4.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]), 'k--')

    ax5 = fig.add_subplot(133)
    # Axis labels
    ax5.set_xlabel(r'Temperature [degC]')
    ax5.set_ylabel(r'Depth from borehole head [m]')
    gt.utilities._format_axes(ax5)

    # Plot temperatures
    ax5.plot(T_f_double_ser, z, 'b-')
    ax5.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]), 'k--')

    # Reverse y-axes
    ax3.set_ylim(ax3.get_ylim()[::-1])
    ax4.set_ylim(ax4.get_ylim()[::-1])
    ax5.set_ylim(ax5.get_ylim()[::-1])
    # Adjust to plot window
    plt.tight_layout()

    return


def synthetic_load(x):
```

(continues on next page)

(continued from previous page)

```
    """
    Synthetic load profile of Bernier et al. (2004).

    Returns load y (in watts) at time x (in hours).
    """
    A = 2000.0
    B = 2190.0
    C = 80.0
    D = 2.0
    E = 0.01
    F = 0.0
    G = 0.95

    func = (168.0-C)/168.0
    for i in [1, 2, 3]:
        func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0)-1.0) \
                          *(np.sin(pi*i/84.0*(x-B)))
    func = func*A*np.sin(pi/12.0*(x-B)) \
           *np.sin(pi/4380.0*(x-B))

    y = func + (-1.0)**np.floor(D/8760.0*(x-B))*abs(func) \
      + E*(-1.0)**np.floor(D/8760.0*(x-B))/np.sign(np.cos(D*pi/4380.0*(x-F))+G)
    return -y


# Main function
if __name__ == '__main__':
    main()
```

References

4.13 Calculation of g-functions with equal inlet fluid temperature

This example demonstrates the use of the *g-function* module and the *pipes* module to calculate *g*-functions using a boundary condition of equal inlet fluid temperature into all boreholes, based on the method of Cimmino[1]. The total rate of heat extraction in the bore field is constant.

The following script generates the *g*-functions of a rectangular field of 6 x 4 boreholes. The *g*-function using a boundary condition of equal inlet fluid temperature is compared to the *g*-functions obtained using boundary conditions of uniform heat extraction rate and of uniform borehole wall temperature.

The script is located in: *pygfunction/examples/equal_inlet_temperature.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of g-functions using equal inlet temperatures.

    The g-functions of a field of 6x4 boreholes are calculated for boundary
    conditions of (a) uniform heat extraction rate, equal for all boreholes,
    (b) uniform borehole wall temperature along the boreholes, equal for all
```

(continues on next page)

[1] Cimmino, M. (2015). The effects of borehole thermal resistances and fluid flow rate on the g-functions of geothermal bore fields. International Journal of Heat and Mass Transfer, 91, 1119-1127.

(continued from previous page)

```
    boreholes, and (c) equal inlet fluid temperature into all boreholes.

"""
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import AutoMinorLocator
from scipy import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)
    B = 7.5              # Borehole spacing (m)

    # Pipe dimensions
    r_out = 0.0211       # Pipe outer radius (m)
    r_in = 0.0147        # Pipe inner radius (m)
    D_s = 0.052          # Shank spacing (m)
    epsilon = 1.0e-6     # Pipe roughness (m)

    # Pipe positions
    # Single U-tube [(x_in, y_in), (x_out, y_out)]
    pos_pipes = [(-D_s, 0.), (D_s, 0.)]

    # Ground properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)
    k_s = 2.0            # Ground thermal conductivity (W/m.K)

    # Grout properties
    k_g = 1.0            # Grout thermal conductivity (W/m.K)

    # Pipe properties
    k_p = 0.4            # Pipe thermal conductivity (W/m.K)

    # Fluid properties
    m_flow_borehole = 0.25  # Total fluid mass flow rate per borehole (kg/s)
    # The fluid is propylene-glycol (20 %) at 20 degC
    fluid = gt.media.Fluid('MPG', 20.)
    cp_f = fluid.cp      # Fluid specific isobaric heat capacity (J/kg.K)
    rho_f = fluid.rho    # Fluid density (kg/m3)
    mu_f = fluid.mu      # Fluid dynamic viscosity (kg/m.s)
    k_f = fluid.k        # Fluid thermal conductivity (W/m.K)

    # g-Function calculation options
```

(continues on next page)

(continued from previous page)

```
    nSegments = 8
    options = {'nSegments': nSegments,
               'disp': True}

    # Geometrically expanding time vector.
    dt = 100*3600.                  # Time step
    tmax = 3000. * 8760. * 3600.    # Maximum time
    Nt = 25                         # Number of time steps
    ts = H**2/(9.*alpha)            # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)

    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    # Field of 6x4 (n=24) boreholes
    N_1 = 6
    N_2 = 4
    boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
    nBoreholes = len(boreField)

    # -------------------------------------------------------------------------
    # Initialize pipe model
    # -------------------------------------------------------------------------

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
        r_in, r_out, k_p)
    # Fluid to inner pipe wall thermal resistance (Single U-tube)
    m_flow_pipe = m_flow_borehole
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f = 1.0/(h_f*2*pi*r_in)

    # Single U-tube, same for all boreholes in the bore field
    UTubes = []
    for borehole in boreField:
        SingleUTube = gt.pipes.SingleUTube(pos_pipes, r_in, r_out,
                                           borehole, k_s, k_g, R_f + R_p)
        UTubes.append(SingleUTube)
    m_flow_network = m_flow_borehole*nBoreholes
    network = gt.networks.Network(
        boreField, UTubes, m_flow_network=m_flow_network, cp_f=cp_f,
        nSegments=nSegments)

    # -------------------------------------------------------------------------
    # Evaluate the g-functions for the borefield
    # -------------------------------------------------------------------------

    # Calculate the g-function for uniform heat extraction rate
    gfunc_uniform_Q = gt.gfunction.gFunction(
        boreField, alpha, time=time, boundary_condition='UHTR', options=options)
```

(continues on next page)

(continued from previous page)

```
    # Calculate the g-function for uniform borehole wall temperature
    gfunc_uniform_T = gt.gfunction.gFunction(
        boreField, alpha, time=time, boundary_condition='UBWT', options=options)

    # Calculate the g-function for equal inlet fluid temperature
    gfunc_equal_Tf_in = gt.gfunction.gFunction(
        network, alpha, time=time, boundary_condition='MIFT', options=options)

    # -------------------------------------------------------------------------
    # Plot g-functions
    # -------------------------------------------------------------------------

    ax = gfunc_uniform_Q.visualize_g_function().axes[0]
    ax.plot(np.log(time/ts), gfunc_uniform_T.gFunc, 'k--')
    ax.plot(np.log(time/ts), gfunc_equal_Tf_in.gFunc, 'r-.')
    ax.legend(['Uniform heat extraction rate',
               'Uniform borehole wall temperature',
               'Equal inlet temperature'])
    plt.tight_layout()

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.14 Simulation of fluid temperatures in a field of multiple boreholes

This example demonstrates the use of the *networks* module to predict the fluid temperature variations in a bore field with known heat extraction rates.

The g-function of a bore field is first calculated using the equal inlet fluid temperature boundary condition[1]. Then, the borehole wall temperature variations are calculated using the load aggregation scheme of Claesson and Javed[2]. The time-variation of heat extraction rates is given by the synthetic load profile of Bernier et al.[3]. Predicted inlet and outlet fluid temperatures of double U-tube boreholes are calculated using the model of Cimmino[4].

The script is located in: *pygfunction/examples/fluid_temperature_multiple_boreholes.py*

```
# -*- coding: utf-8 -*-
""" Example of simulation of a geothermal system with multiple boreholes.
```

(continues on next page)

[1] Cimmino, M. (2015). The effects of borehole thermal resistances and fluid flow rate on the g-functions of geothermal bore fields. International Journal of Heat and Mass Transfer, 91, 1119-1127.

[2] Claesson, J., & Javed, S. (2011). A load-aggregation method to calculate extraction temperatures of borehole heat exchangers. ASHRAE Transactions, 118 (1): 530–539.

[3] Bernier, M., Pinel, P., Labib, R. and Paillot, R. (2004). A multiple load aggregation algorithm for annual hourly simulations of GCHP systems. HVAC&R Research 10 (4): 471–487.

[4] Cimmino, M. (2016). Fluid and borehole wall temperature profiles in vertical geothermal boreholes with multiple U-tubes. Renewable Energy 96 : 137-147.

(continued from previous page)

```
    The g-function of a bore field is calculated for boundary condition of
    mixed inlet fluid temperature into the boreholes. Then, the borehole
    wall temperature variations resulting from a time-varying load profile
    are simulated using the aggregation method of Claesson and Javed (2012).
    Predicted outlet fluid temperatures of double U-tube borehole are
    evaluated.

"""
import matplotlib.pyplot as plt
import numpy as np
from scipy.constants import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    H = 150.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)

    # Bore field geometry (rectangular array)
    N_1 = 6              # Number of boreholes in the x-direction (columns)
    N_2 = 4              # Number of boreholes in the y-direction (rows)
    B = 7.5              # Borehole spacing, in both directions (m)

    # Pipe dimensions
    r_out = 0.0211       # Pipe outer radius (m)
    r_in = 0.0147        # Pipe inner radius (m)
    D_s = 0.052          # Shank spacing (m)
    epsilon = 1.0e-6     # Pipe roughness (m)

    # Pipe positions
    # Double U-tube [(x_in1, y_in1), (x_in2, y_in2),
    #                (x_out1, y_out1), (x_out2, y_out2)]
    pos = [(-D_s, 0.), (0., -D_s), (D_s, 0.), (0., D_s)]

    # Ground properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)
    k_s = 2.0            # Ground thermal conductivity (W/m.K)
    T_g = 10.0           # Undisturbed ground temperature (degC)

    # Grout properties
    k_g = 1.0            # Grout thermal conductivity (W/m.K)

    # Pipe properties
    k_p = 0.4            # Pipe thermal conductivity (W/m.K)
```

(continues on next page)

(continued from previous page)

```
    # Fluid properties
    m_flow_borehole = 0.25      # Total fluid mass flow rate per borehole (kg/s)
    m_flow_network = m_flow_borehole*N_1*N_2    # Total fluid mass flow rate (kg/s)
    # The fluid is propylene-glycol (20 %) at 20 degC
    fluid = gt.media.Fluid('MPG', 20.)
    cp_f = fluid.cp     # Fluid specific isobaric heat capacity (J/kg.K)
    rho_f = fluid.rho   # Fluid density (kg/m3)
    mu_f = fluid.mu     # Fluid dynamic viscosity (kg/m.s)
    k_f = fluid.k       # Fluid thermal conductivity (W/m.K)

    # g-Function calculation options
    options = {'nSegments': 8,
               'disp': True}

    # Simulation parameters
    dt = 3600.                  # Time step (s)
    tmax = 1.*8760. * 3600.     # Maximum time (s)
    Nt = int(np.ceil(tmax/dt))  # Number of time steps
    time = dt * np.arange(1, Nt+1)

    # Load aggregation scheme
    LoadAgg = gt.load_aggregation.ClaessonJaved(dt, tmax)

    # -------------------------------------------------------------------------
    # Initialize bore field and pipe models
    # -------------------------------------------------------------------------

    # The field is a retangular array
    boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
    nBoreholes = len(boreField)

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
            r_in, r_out, k_p)

    # Fluid to inner pipe wall thermal resistance (Double U-tube in parallel)
    m_flow_pipe = m_flow_borehole/2
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
            m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f = 1.0/(h_f*2*pi*r_in)

    # Double U-tube (parallel), same for all boreholes in the bore field
    UTubes = []
    for borehole in boreField:
        UTube = gt.pipes.MultipleUTube(
            pos, r_in, r_out, borehole, k_s, k_g, R_f + R_p,
            nPipes=2, config='parallel')
        UTubes.append(UTube)
    # Build a network object from the list of UTubes
    network = gt.networks.Network(
        boreField, UTubes, m_flow_network=m_flow_network, cp_f=cp_f)
```

(continues on next page)

(continued from previous page)

```
    # --------------------------------------------------------------------------
    # Calculate g-function
    # --------------------------------------------------------------------------

    # Get time values needed for g-function evaluation
    time_req = LoadAgg.get_times_for_simulation()
    # Calculate g-function
    gFunc = gt.gfunction.gFunction(
        network, alpha, time=time_req, boundary_condition='MIFT',
        options=options)
    # Initialize load aggregation scheme
    LoadAgg.initialize(gFunc.gFunc/(2*pi*k_s))

    # --------------------------------------------------------------------------
    # Simulation
    # --------------------------------------------------------------------------

    # Evaluate heat extraction rate
    Q_tot = nBoreholes*synthetic_load(time/3600.)

    T_b = np.zeros(Nt)
    T_f_in = np.zeros(Nt)
    T_f_out = np.zeros(Nt)
    for i, (t, Q_i) in enumerate(zip(time, Q_tot)):
        # Increment time step by (1)
        LoadAgg.next_time_step(t)

        # Apply current load (in watts per meter of borehole)
        Q_b = Q_i/nBoreholes
        LoadAgg.set_current_load(Q_b/H)

        # Evaluate borehole wall temperature
        deltaT_b = LoadAgg.temporal_superposition()
        T_b[i] = T_g - deltaT_b

        # Evaluate inlet fluid temperature (all boreholes are the same)
        T_f_in[i] = network.get_network_inlet_temperature(
                Q_tot[i], T_b[i], m_flow_network, cp_f, nSegments=1)

        # Evaluate outlet fluid temperature
        T_f_out[i] = network.get_network_outlet_temperature(
                T_f_in[i],  T_b[i], m_flow_network, cp_f, nSegments=1)

    # --------------------------------------------------------------------------
    # Plot hourly heat extraction rates and temperatures
    # --------------------------------------------------------------------------

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(211)
```

(continues on next page)

(continued from previous page)

```
    # Axis labels
    ax1.set_xlabel(r'Time [hours]')
    ax1.set_ylabel(r'Total heat extraction rate [W]')
    gt.utilities._format_axes(ax1)

    # Plot heat extraction rates
    hours = np.arange(1, Nt+1) * dt / 3600.
    ax1.plot(hours, Q_tot)

    ax2 = fig.add_subplot(212)
    # Axis labels
    ax2.set_xlabel(r'Time [hours]')
    ax2.set_ylabel(r'Temperature [degC]')
    gt.utilities._format_axes(ax2)

    # Plot temperatures
    ax2.plot(hours, T_b, label='Borehole wall')
    ax2.plot(hours, T_f_out, '-.',
             label='Outlet, double U-tube (parallel)')
    ax2.legend()

    # Adjust to plot window
    plt.tight_layout()

    # -------------------------------------------------------------------------
    # Plot fluid temperature profiles
    # -------------------------------------------------------------------------

    # Evaluate temperatures at nz evenly spaced depths along the borehole
    # at the (it+1)-th time step
    nz = 20
    it = 8724
    z = np.linspace(0., H, num=nz)
    T_f = UTubes[0].get_temperature(
        z, T_f_in[it], T_b[it], m_flow_borehole, cp_f)


    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax3 = fig.add_subplot(111)
    # Axis labels
    ax3.set_xlabel(r'Temperature [degC]')
    ax3.set_ylabel(r'Depth from borehole head [m]')
    gt.utilities._format_axes(ax3)

    # Plot temperatures
    pltFlu = ax3.plot(T_f, z, 'b-', label='Fluid')
    pltWal = ax3.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]),
                      'k--', label='Borehole wall')
    ax3.legend(handles=[pltFlu[0]]+pltWal)

```

(continues on next page)

(continued from previous page)

```
    # Reverse y-axes
    ax3.set_ylim(ax3.get_ylim()[::-1])
    # Adjust to plot window
    plt.tight_layout()

    return


def synthetic_load(x):
    """
    Synthetic load profile of Bernier et al. (2004).

    Returns load y (in watts) at time x (in hours).
    """
    A = 2000.0
    B = 2190.0
    C = 80.0
    D = 2.0
    E = 0.01
    F = 0.0
    G = 0.95

    func = (168.0-C)/168.0
    for i in [1, 2, 3]:
        func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0)-1.0) \
                          *(np.sin(pi*i/84.0*(x-B)))
    func = func*A*np.sin(pi/12.0*(x-B)) \
           *np.sin(pi/4380.0*(x-B))

    y = func + (-1.0)**np.floor(D/8760.0*(x-B))*abs(func) \
      + E*(-1.0)**np.floor(D/8760.0*(x-B))/np.sign(np.cos(D*pi/4380.0*(x-F))+G)
    return -y


# Main function
if __name__ == '__main__':
    main()
```

References

4.15 Calculation of fluid temperature profiles in a borehole with independent U-tubes

This example demonstrates the use of the *pipes* module to calculate the fluid temperature profiles in a borehole with independent U-tubes, based on the method of Cimmino[1]. The borehole wall temperature is uniform in this example.

The following script evaluates the fluid temperatures in a borehole with 4 independent U-tubes with different inlet fluid temperatures and different inlet fluid mass flow rates. The resulting fluid temperature profiles are verified against the

[1] Cimmino, M. (2016). Fluid and borehole wall temperature profiles in vertical geothermal boreholes with multiple U-tubes. Renewable Energy 96 : 137-147.

fluid temeprature profiles presented by Cimmino[Page 137, 1].

The script is located in: *pygfunction/examples/multiple_independent_Utubes.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of fluid temperature profiles in a borehole with
    independent U-tubes.

    The fluid temperature profiles in a borehole with 4 independent U-tubes are
    calculated. The borehole has 4 U-tubes, each with different inlet fluid
    temperatures and different inlet fluid mass flow rates. The borehole wall
    temperature is uniform. Results are verified against the results of
    Cimmino (2016).

"""
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import AutoMinorLocator
from scipy import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 2.5              # Borehole buried depth (m)
    H = 100.0            # Borehole length (m)
    r_b = 0.075          # Borehole radius (m)

    # Pipe dimensions
    r_out = 0.010        # Pipe outer radius (m)
    r_in = 0.008         # Pipe inner radius (m)
    D_s = 0.060          # Shank spacing (m)

    # Pipe positions
    nPipes = 4           # Number of U-tube pipes (-)
    pos_pipes = _pipePositions(D_s, nPipes)

    # Ground properties
    k_s = 2.0            # Ground thermal conductivity (W/m.K)

    # Grout properties
    k_g = 1.0            # Grout thermal conductivity (W/m.K)

    # Fluid properties
    R_fp = 1e-30         # Fluid to outer pipe wall thermal resistance (m.K/W)
    # Fluid specific isobaric heat capacity per U-tube (J/kg.K)
    cp_f = 4000.*np.ones(nPipes)

```

(continues on next page)

(continued from previous page)

```
    # Borehole wall temperature (degC)
    T_b = 2.0
    # Total fluid mass flow rate per U-tube (kg/s)
    m_flow_borehole = np.array([0.40, 0.35, 0.30, 0.25])
    # Inlet fluid temperatures per U-tube (degC)
    T_f_in = np.array([6.0, -6.0, 5.0, -5.0])

    # Path to validation data
    filePath = './data/Cimmi16_multiple_independent_Utubes.txt'

    # -------------------------------------------------------------------------
    # Initialize pipe model
    # -------------------------------------------------------------------------

    # Borehole object
    borehole = gt.boreholes.Borehole(H, D, r_b, 0., 0.)
    # Multiple independent U-tubes
    MultipleUTube = gt.pipes.IndependentMultipleUTube(
            pos_pipes, r_in, r_out, borehole, k_s, k_g, R_fp, nPipes, J=0)

    # -------------------------------------------------------------------------
    # Evaluate the outlet fluid temperatures and fluid temperature profiles
    # -------------------------------------------------------------------------

    # Calculate the outlet fluid temperatures
    T_f_out = MultipleUTube.get_outlet_temperature(
        T_f_in, T_b, m_flow_borehole, cp_f)

    # Evaluate temperatures at nz evenly spaced depths along the borehole
    nz = 20
    z = np.linspace(0., H, num=nz)
    T_f = MultipleUTube.get_temperature(z, T_f_in, T_b, m_flow_borehole, cp_f)

    # -------------------------------------------------------------------------
    # Plot fluid temperature profiles
    # -------------------------------------------------------------------------

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(111)
    # Axis labels
    ax1.set_xlabel(r'Temperature [degC]')
    ax1.set_ylabel(r'Depth from borehole head [m]')
    gt.utilities._format_axes(ax1)

    # Plot temperatures
    ax1.plot(T_f, z, 'k.')
    ax1.plot(np.array([T_b, T_b]), np.array([0., H]), 'k--')
    # Labels
    calculated = mlines.Line2D([], [],
                               color='black',
```

(continues on next page)

(continued from previous page)

```
                                ls='None',
                                marker='.',
                                label='Fluid')
    borehole_temp = mlines.Line2D([], [],
                                  color='black',
                                  ls='--',
                                  marker='None',
                                  label='Borehole wall')
    plt.tight_layout()

    # -------------------------------------------------------------------------
    # Load data from Cimmino (2016)
    # -------------------------------------------------------------------------
    data = np.loadtxt(filePath, skiprows=1)
    ax1.plot(data[:,2:], data[:,0], 'b-',)
    reference = mlines.Line2D([], [],
                              color='blue',
                              ls='-',
                              lw=1.5,
                              marker='None',
                              label='Cimmino (2016)')
    ax1.legend(handles=[borehole_temp, calculated, reference],
               loc='upper left')

    # Reverse y-axis
    ax1.set_ylim(ax1.get_ylim()[::-1])
    # Adjust to plot window

    return


def _pipePositions(Ds, nPipes):
    """ Positions pipes in an axisymetric configuration.
    """
    dt = pi / float(nPipes)
    pos = [(0., 0.) for i in range(2*nPipes)]
    for i in range(nPipes):
        pos[i] = (Ds*np.cos(2.0*i*dt+pi), Ds*np.sin(2.0*i*dt+pi))
        pos[i+nPipes] = (Ds*np.cos(2.0*i*dt+pi+dt), Ds*np.sin(2.0*i*dt+pi+dt))
    return pos


# Main function
if __name__ == '__main__':
    main()
```

References

4.16 Evaluation of thermal resistances using the multipole method

This example demonstrates the use of the *pipes.thermal_resistances()* function to evaluate internal thermal resistances in a borehole. The example also covers the use of the *pipes.multipole()* function to evaluate the 2D temperature field in and around a borehole.

The thermal resistances of a borehole with two pipes are evaluated using the multipole method of Claesson and Hellstrom[1]. Based on the calculated thermal resistances, the heat flows from the pipes required to obtain pipe temperatures of 1 degC are evaluated. The temperatures in and around the borehole with 2 pipes are then calculated. Results are verified against the results of Claesson and Hellstrom[Page 141, 1].

The script is located in: *pygfunction/examples/multipole_temperature.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of grout and ground temperatures using the multipole
    method.

    The thermal resistances of a borehole with two pipes are evaluated using
    the multipole method of Claesson and Hellstrom (2011). Based on the
    calculated thermal resistances, the heat flows from the pipes required to
    obtain pipe temperatures of 1 degC are evaluated. The temperatures in and
    around the borehole with 2 pipes are then calculated. Results are verified
    against the results of Claesson and Hellstrom (2011).

"""
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import AutoMinorLocator
from scipy import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    r_b = 0.070          # Borehole radius (m)

    # Pipe dimensions
    n_p = 2              # Number of pipes
    # Pipe outer radius (m)
    rp_out = 0.02*np.ones(n_p)

    # Pipe positions
    # Single U-tube [(x_1, y_1), (x_2, y_2)]
    pos_pipes = [(0.03, 0.00), (-0.03, 0.02)]
```

(continues on next page)

[1] Claesson, J., & Hellstrom, G. (2011). Multipole method to calculate borehole thermal resistances in a borehole heat exchanger. HVAC&R Research, 17(6), 895-911.

(continued from previous page)

```
    # Ground properties
    k_s = 2.5           # Ground thermal conductivity (W/m.K)

    # Grout properties
    k_g = 1.5           # Grout thermal conductivity (W/m.K)

    # Fluid properties
    # Fluid to outer pipe wall thermal resistance (m.K/W)
    R_fp = 1.2/(2*pi*k_g)*np.ones(n_p)

    # Borehole wall temperature (degC)
    T_b = 0.0

    # Fluid temperatures (degC)
    T_f = np.array([1., 1.])

    # Path to validation data
    filePath = './data/ClaHel11_multipole_temperature.txt'

    # Thermal resistances for J=3
    R_Claesson = 0.01*np.array([25.592, 1.561, 25.311])

    # Number of multipoles per pipe
    J = 3

    # -------------------------------------------------------------------------
    # Evaluate the internal thermal resistances
    # -------------------------------------------------------------------------

    # Thermal resistances
    (R, Rd) = gt.pipes.thermal_resistances(pos_pipes, rp_out, r_b, k_s, k_g,
                                           R_fp, J=3)
    print(50*'-')
    print('Thermal resistance:\t\t100*R11\t100*R12\t100*R22')
    print(f'Claesson and Hellstrom:\t{100*R_Claesson[0]:.3f}'
          f'\t{100*R_Claesson[1]:.3f}\t{100*R_Claesson[2]:.3f}')
    print(f'Present:\t\t\t\t{100*R[0,0]:.3f}\t{100*R[0,1]:.3f}'
          f'\t{100*R[1,1]:.3f}')
    print(50*'-')

    # Heat flows
    Q = np.linalg.solve(R, T_f - T_b)

    # -------------------------------------------------------------------------
    # Temperatures along y=0.
    # -------------------------------------------------------------------------

    # Grid points to evaluate temperatures
    x = np.linspace(-0.1, 0.1, num=200)
    y = np.zeros_like(x)

```

(continues on next page)

(continued from previous page)

```
    # Evaluate temperatures using multipole method
    (T_f, T, it, eps_max) = gt.pipes.multipole(pos_pipes, rp_out, r_b, k_s,
                                               k_g, R_fp, T_b, Q, J,
                                               x_T=x, y_T=y)

    # Load validation data
    data = np.loadtxt(filePath, skiprows=1)

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(111)
    # Axis labels
    ax1.set_xlabel(r'x (m)')
    ax1.set_ylabel(r'$T(x,0)$')
    # Axis limits
    ax1.set_xlim([-0.1, 0.1])
    ax1.set_ylim([-0.2, 1.2])
    # Show grid
    ax1.grid()
    gt.utilities._format_axes(ax1)

    ax1.plot(x, T, label='pygfunction')
    ax1.plot(data[:,0], data[:,1], 'ko',
             label='Claesson and Hellstrom (2011)')
    ax1.legend(loc='upper left')

    # Adjust to plot window
    plt.tight_layout()

    # -------------------------------------------------------------------------
    # Temperatures in -0.1 < x < 0.1, -0.1 < y < 0.1
    # -------------------------------------------------------------------------

    # Grid points to evaluate temperatures
    N_xy = 200
    x = np.linspace(-0.1, 0.1, num=N_xy)
    y = np.linspace(-0.1, 0.1, num=N_xy)
    X, Y = np.meshgrid(x, y)

    # Evaluate temperatures using multipole method
    (T_f, T, it, eps_max) = gt.pipes.multipole(pos_pipes, rp_out, r_b, k_s,
                                               k_g, R_fp, T_b, Q, J,
                                               x_T=X.flatten(),
                                               y_T=Y.flatten())

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(111)
    # Axis labels
    ax1.set_xlabel('x (m)')
```

(continues on next page)

(continued from previous page)

```
    ax1.set_ylabel('y (m)')
    # Axis limits
    plt.axis([-0.1, 0.1, -0.1, 0.1])
    plt.gca().set_aspect('equal', adjustable='box')
    gt.utilities._format_axes(ax1)

    # Borehole wall outline
    borewall = plt.Circle((0., 0.), radius=r_b,
                          fill=False, linestyle='--', linewidth=2.)
    ax1.add_patch(borewall)
    # Pipe outlines
    for pos, r_out_n in zip(pos_pipes, rp_out):
        pipe = plt.Circle(pos, radius=r_out_n,
                          fill=False, linestyle='-', linewidth=4.)
        ax1.add_patch(pipe)
    # Temperature contours
    CS = ax1.contour(X, Y, T.reshape((N_xy, N_xy)),
                     np.linspace(-0.2, 1.0, num=7))
    plt.clabel(CS, inline=1, fontsize=10)

    # Adjust to plot window
    plt.tight_layout()

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.17 Calculation of effective bore field thermal resistance

This example demonstrates the use of the `pipes.field_thermal_resistance()` function to evaluate the bore field thermal resistance. The concept effective bore field thermal is detailed by Cimmino[1]

The following script evaluates the effective bore field thermal resistance for fields of 1 to 5 series-connected boreholes with fluid flow rates ranging from 0.01 kg/s ti 1.00 kg/s.

The script is located in: *pygfunction/examples/bore_field_thermal_resistance.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of effective bore field thermal resistance.

    The effective bore field thermal resistance of fields of up to 5 boreholes
    of equal lengths connected in series is calculated for various fluid flow
    rates.

```

(continues on next page)

[1] Cimmino, M. (2018). g-Functions for bore fields with mixed parallel and series connections considering the axial fluid temperature variations. Proceedings of the IGSHPA Sweden Research Track 2018. Stockholm, Sweden. pp. 262-270.

(continued from previous page)

```
"""
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import AutoMinorLocator
from scipy import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Number of boreholes
    nBoreholes = 5
    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    # Borehole length (m)
    H = 150.
    r_b = 0.075          # Borehole radius (m)
    B = 7.5              # Borehole spacing (m)

    # Pipe dimensions
    r_out = 0.02         # Pipe outer radius (m)
    r_in = 0.015         # Pipe inner radius (m)
    D_s = 0.05           # Shank spacing (m)
    epsilon = 1.0e-6     # Pipe roughness (m)

    # Pipe positions
    # Single U-tube [(x_in, y_in), (x_out, y_out)]
    pos_pipes = [(-D_s, 0.), (D_s, 0.)]

    # Ground properties
    k_s = 2.0            # Ground thermal conductivity (W/m.K)

    # Grout properties
    k_g = 1.0            # Grout thermal conductivity (W/m.K)

    # Pipe properties
    k_p = 0.4            # Pipe thermal conductivity (W/m.K)

    # Fluid properties
    # Total fluid mass flow rate per borehole (kg/s), from 0.01 kg/s to 1 kg/s
    m_flow_network = 10**np.arange(-2, 0.001, 0.05)
    # The fluid is propylene-glycol (20 %) at 20 degC
    fluid = gt.media.Fluid('MPG', 20.)
    cp_f = fluid.cp      # Fluid specific isobaric heat capacity (J/kg.K)
    rho_f = fluid.rho    # Fluid density (kg/m3)
    mu_f = fluid.mu      # Fluid dynamic viscosity (kg/m.s)
    k_f = fluid.k        # Fluid thermal conductivity (W/m.K)

```

(continues on next page)

(continued from previous page)

```
    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    boreField = []
    bore_connectivity = []
    for i in range(nBoreholes):
        x = i*B
        borehole = gt.boreholes.Borehole(H, D, r_b, x, 0.)
        boreField.append(borehole)
        # Boreholes are connected in series: The index of the upstream
        # borehole is that of the previous borehole
        bore_connectivity.append(i - 1)

    # -------------------------------------------------------------------------
    # Evaluate the effective bore field thermal resistance
    # -------------------------------------------------------------------------

    # Initialize result array
    R = np.zeros((nBoreholes, len(m_flow_network)))
    for i in range(nBoreholes):
        for j, m_flow_network_j in enumerate(m_flow_network):
            nBoreholes = i + 1
            # Boreholes are connected in series
            m_flow_borehole = m_flow_network_j
            # Boreholes are single U-tube
            m_flow_pipe = m_flow_borehole

            # Pipe thermal resistance
            R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
                    r_in, r_out, k_p)
            # Fluid to inner pipe wall thermal resistance (Single U-tube)
            h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
                    m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
            R_f = 1.0/(h_f*2*pi*r_in)

            # Single U-tube, same for all boreholes in the bore field
            UTubes = []
            for borehole in boreField:
                SingleUTube = gt.pipes.SingleUTube(
                    pos_pipes, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
                UTubes.append(SingleUTube)
            network = gt.networks.Network(
                boreField[:nBoreholes],
                UTubes[:nBoreholes],
                bore_connectivity=bore_connectivity[:nBoreholes])

            # Effective bore field thermal resistance
            R_field = gt.networks.network_thermal_resistance(
                network, m_flow_network_j, cp_f)
            # Add to result array
            R[i,j] = R_field
```

(continues on next page)

(continued from previous page)

```
    # -------------------------------------------------------------------------
    # Plot bore field thermal resistances
    # -------------------------------------------------------------------------

    # Configure figure and axes
    fig = gt.utilities._initialize_figure()

    ax1 = fig.add_subplot(111)
    # Axis labels
    ax1.set_xlabel(r'$\dot{m}$ [kg/s]')
    ax1.set_ylabel(r'$R^*_{field}$ [m.K/W]')
    # Axis limits
    ax1.set_xlim([0., 1.])
    ax1.set_ylim([0., 1.])

    gt.utilities._format_axes(ax1)

    # Bore field thermal resistances
    ax1.plot(m_flow_network, R[0,:], '-', label='1 borehole')
    ax1.plot(m_flow_network, R[2,:], '--', label='3 boreholes')
    ax1.plot(m_flow_network, R[4,:], '-.', label='5 boreholes')
    ax1.legend()
    # Adjust to plot window
    plt.tight_layout()

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.18 Calculation of g-functions with mixed parallel and series connections

This example demonstrates the use of the *g-function* module and the *networks* module to calculate *g*-functions considering the piping connections between the boreholes, based on the method of Cimmino[1]. For boreholes connected in series, it is considered the outlet fluid temperature of the upstream borehole is equal to the inlet fluid temperature of the downstream borehole. The total rate of heat extraction in the bore field is constant.

The following script generates the *g*-functions of a field of 5 equally spaced borehole on a straight line and connected in series. The boreholes have different lengths. The *g*-function considering piping conections is compared to the *g*-function obtained using a boundary condition of uniform borehole wall temperature.

The script is located in: *pygfunction/examples/mixed_inlet_conditions.py*

[1] Cimmino, M. (2018). g-Functions for bore fields with mixed parallel and series connections considering the axial fluid temperature variations. Proceedings of the IGSHPA Sweden Research Track 2018. Stockholm, Sweden. pp. 262-270.

```python
# -*- coding: utf-8 -*-
""" Example of calculation of g-functions using mixed inlet temperatures.

    The g-functions of a field of 5 boreholes of different lengths connected
    in series are calculated for 2 boundary conditions: (a) uniform borehole
    wall temperature, and (b) series connections between boreholes. The
    g-function for case (b) is based on the effective borehole wall
    temperature, rather than the average borehole wall temperature.

"""
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import AutoMinorLocator
from scipy import pi

import pygfunction as gt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0              # Borehole buried depth (m)
    # Borehole length (m)
    H_boreholes = np.array([75.0, 100.0, 125.0, 150.0, 75.0])
    H_mean = np.mean(H_boreholes)
    r_b = 0.075          # Borehole radius (m)
    B = 7.5              # Borehole spacing (m)

    # Pipe dimensions
    r_out = 0.02         # Pipe outer radius (m)
    r_in = 0.015         # Pipe inner radius (m)
    D_s = 0.05           # Shank spacing (m)
    epsilon = 1.0e-6     # Pipe roughness (m)

    # Pipe positions
    # Single U-tube [(x_in, y_in), (x_out, y_out)]
    pos_pipes = [(-D_s, 0.), (D_s, 0.)]

    # Ground properties
    alpha = 1.0e-6       # Ground thermal diffusivity (m2/s)
    k_s = 2.0            # Ground thermal conductivity (W/m.K)

    # Grout properties
    k_g = 1.0            # Grout thermal conductivity (W/m.K)

    # Pipe properties
    k_p = 0.4            # Pipe thermal conductivity (W/m.K)

    # Fluid properties
    m_flow_network = 0.25   # Total fluid mass flow rate in network (kg/s)
```

(continues on next page)

(continued from previous page)

```
    # All boreholes are in series
    m_flow_borehole = m_flow_network
    # The fluid is propylene-glycol (20 %) at 20 degC
    fluid = gt.media.Fluid('MPG', 20.)
    cp_f = fluid.cp     # Fluid specific isobaric heat capacity (J/kg.K)
    rho_f = fluid.rho   # Fluid density (kg/m3)
    mu_f = fluid.mu     # Fluid dynamic viscosity (kg/m.s)
    k_f = fluid.k       # Fluid thermal conductivity (W/m.K)

    # g-Function calculation options
    nSegments = 8
    options = {'nSegments': nSegments,
               'disp': True,
               'profiles': True}
    # The similarities method is used since the 'equivalent' method does not
    # apply if boreholes are connected in series
    method = 'similarities'

    # Geometrically expanding time vector.
    dt = 100*3600.                  # Time step
    tmax = 3000. * 8760. * 3600.    # Maximum time
    Nt = 25                         # Number of time steps
    ts = H_mean**2/(9.*alpha)   # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)

    # -------------------------------------------------------------------------
    # Borehole field
    # -------------------------------------------------------------------------

    boreField = []
    bore_connectivity = []
    for i, H in enumerate(H_boreholes):
        x = i*B
        borehole = gt.boreholes.Borehole(H, D, r_b, x, 0.)
        boreField.append(borehole)
        # Boreholes are connected in series: The index of the upstream
        # borehole is that of the previous borehole
        bore_connectivity.append(i - 1)

    # -------------------------------------------------------------------------
    # Initialize pipe model
    # -------------------------------------------------------------------------

    # Pipe thermal resistance
    R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
        r_in, r_out, k_p)
    # Fluid to inner pipe wall thermal resistance (Single U-tube)
    m_flow_pipe = m_flow_borehole
    h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
        m_flow_pipe,  r_in, mu_f, rho_f, k_f, cp_f, epsilon)
    R_f = 1.0/(h_f*2*pi*r_in)

```

(continues on next page)

(continued from previous page)

```
    # Single U-tube, same for all boreholes in the bore field
    UTubes = []
    for borehole in boreField:
        SingleUTube = gt.pipes.SingleUTube(
            pos_pipes, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
        UTubes.append(SingleUTube)
    network = gt.networks.Network(
        boreField, UTubes, bore_connectivity=bore_connectivity,
        m_flow_network=m_flow_network, cp_f=cp_f, nSegments=nSegments)

    # -------------------------------------------------------------------------
    # Evaluate the g-functions for the borefield
    # -------------------------------------------------------------------------

    # Calculate the g-function for uniform temperature
    gfunc_Tb = gt.gfunction.gFunction(
        boreField, alpha, time=time, boundary_condition='UBWT',
        options=options, method=method)

    # Calculate the g-function for mixed inlet fluid conditions
    gfunc_equal_Tf_mixed = gt.gfunction.gFunction(
        network, alpha, time=time, boundary_condition='MIFT', options=options,
        method=method)

    # -------------------------------------------------------------------------
    # Plot g-functions
    # -------------------------------------------------------------------------

    ax = gfunc_Tb.visualize_g_function().axes[0]
    ax.plot(np.log(time/ts), gfunc_equal_Tf_mixed.gFunc, 'r-.')
    ax.legend(['Uniform temperature', 'Mixed inlet temperature'])
    plt.tight_layout()

    # For the mixed inlet fluid temperature condition, draw the temperatures
    # and heat extraction rates
    gfunc_equal_Tf_mixed.visualize_temperatures()
    gfunc_equal_Tf_mixed.visualize_temperature_profiles()
    gfunc_equal_Tf_mixed.visualize_heat_extraction_rates()
    gfunc_equal_Tf_mixed.visualize_heat_extraction_rate_profiles()

    return


# Main function
if __name__ == '__main__':
    main()
```

References

4.19 Calculation of g-functions with inclined boreholes

This example demonstrates the use of the *g-function* module to calculate g-functions of fields of inclined boreholes using a boundary condition of uniform and equal borehole wall temperature for all boreholes. The total rate of heat extraction in the bore field is constant.

The following script generates the g-functions of two bore fields. The first field corresponds to the optimal configuration presented by Claesson and Eskilson[1]. The second field corresponds to the configuration comprised of 8 boreholes in a circle presented by the same authors.

The script is located in: *pygfunction/examples/inclined_boreholes.py*

```
# -*- coding: utf-8 -*-
""" Example of calculation of g-functions with inclined boreholesusing uniform
    and equal borehole wall temperatures.

    The g-functions of two fields of 8 boreholes are calculated for boundary
    condition of uniform borehole wall temperature along the boreholes, equal
    for all boreholes. The first field corresponds to the "optimum"
    configuration presented by Claesson and Eskilson (1987) and the second
    field corresponds to the configuration comprised of 8 boreholes in a
    circle.

    Claesson J, and Eskilson, P. (1987). Conductive heat extraction by
    thermally interacting deep boreholes, in "Thermal analysis of heat
    extraction boreholes". Ph.D. Thesis, University of Lund, Lund, Sweden.
"""

import pygfunction as gt
import numpy as np
import matplotlib.pyplot as plt


def main():
    # -------------------------------------------------------------------------
    # Simulation parameters
    # -------------------------------------------------------------------------

    # Borehole dimensions
    D = 4.0             # Borehole buried depth (m)
    H = 150.0           # Borehole length (m)
    r_b = 0.075         # Borehole radius (m)
    tilt = np.radians(20.)  # Borehole inclination (rad)

    # Thermal properties
    alpha = 1.0e-6      # Ground thermal diffusivity (m2/s)

    # g-Function calculation options
    options = {'disp': True}
```

(continues on next page)

[1] Claesson J, and Eskilson, P. (1987). Conductive heat extraction by thermally interacting deep boreholes, in "Thermal analysis of heat extraction boreholes". Ph.D. Thesis, University of Lund, Lund, Sweden.

(continued from previous page)

```
    # Geometrically expanding time vector.
    dt = 100*3600.                  # Time step
    tmax = 3000. * 8760. * 3600.    # Maximum time
    Nt = 25                         # Number of time steps
    ts = H**2/(9.*alpha)            # Bore field characteristic time
    time = gt.utilities.time_geometric(dt, tmax, Nt)
    lntts = np.log(time/ts)

    # -------------------------------------------------------------------------
    # Borehole fields
    # -------------------------------------------------------------------------
    """
    Bore field #1

    This field corresponds to the optimal configuration presented by
    Claesson and Eskilson (1987). The field is built using the `cardinal_point`
    function to define the orientation of each borehole, individually.
    """
    B = 7.5 # Borehole spacing (m)
    # Orientation of the boreholes
    borehole_orientations = [
        gt.utilities.cardinal_point('W'),
        gt.utilities.cardinal_point('NW'),
        gt.utilities.cardinal_point('SW'),
        gt.utilities.cardinal_point('N'),
        gt.utilities.cardinal_point('S'),
        gt.utilities.cardinal_point('NE'),
        gt.utilities.cardinal_point('SE'),
        gt.utilities.cardinal_point('E')]

    # "Optimal" field of 8 boreholes
    boreField1 = []
    for i, orientation in enumerate(borehole_orientations):
        borehole = gt.boreholes.Borehole(
            H, D, r_b, i * B, 0., tilt=tilt, orientation=orientation)
        boreField1.append(borehole)

    # Visualize the borehole field
    fig1 = gt.boreholes.visualize_field(boreField1)

    """
    Bore field #2

    This field corresponds to the configuration comprised of 8 boreholes in
    a circle presented by Claesson and Eskilson (1987). The field is built
    using the `circle_field` function.
    """
    N = 8   # Number of boreholes
    R = 3.  # Borehole spacing from the center of the field (m)

    # Field of 6 boreholes in a circle
```

(continues on next page)

(continued from previous page)

```
    boreField2 = gt.boreholes.circle_field(N, R, H, D, r_b, tilt=tilt)

    # Visualize the borehole field
    fig2 = gt.boreholes.visualize_field(boreField2)

    # -------------------------------------------------------------------------
    # Evaluate g-functions for all fields
    # -------------------------------------------------------------------------
    # Bore field #1
    gfunc1 = gt.gfunction.gFunction(
        boreField1, alpha, time=time, options=options, method='similarities')
    fig3 = gfunc1.visualize_g_function()
    fig3.suptitle('"Optimal" field of 8 boreholes')
    fig3.tight_layout()
    # Bore field #2
    gfunc2 = gt.gfunction.gFunction(
        boreField2, alpha, time=time, options=options, method='similarities')
    fig4 = gfunc2.visualize_g_function()
    fig4.suptitle(f'Field of {N} boreholes in a circle')
    fig4.tight_layout()


# Main function
if __name__ == '__main__':
    main()
```

References

CHAPTER

FIVE

NOMENCLATURE

5.1 Geometry

Parameters in the table below are related to the definition of borehole and pipe geometries.

Parameter	Units	Description
H	m	Borehole length
D	m	Borehole burial depth
r_b	m	Borehole radius
x	m	Position of borehole in x-axis
y	m	Position of borehole in y-axis
r_in	m	Inner radius of pipe
r_out	m	Outer radius of pipe

5.2 Fluid properties

Parameters in the table below are related to the evaluation of fluid thermal properties in the *media* module.

Parameter	Units	Description
T	degC	Temperature of the fluid
P	Pa	Pressure of the fluid
rho	kg/m3	Density of the fluid
mu	Pa.s or N.s/m2	Dynamic viscosity
nu	m2/s	Kinematic viscosity
cp	J/kg.K	Specific isobaric heat capacity
k	W/m.K	Thermal conductivity
rhoCp	J/m3.K	Volumetric heat capacity

5.3 Heat transfer

Parameters in the table below are used throughout **pygfunction** for the calculation of heat transfer in geothermal bore fields.

Parameter	Units	Description
time	s	The time in seconds
alpha	m2/s	Soil thermal diffusivity
h	•	Value of the dimensionless FLS solution
k_s	W/m.K	Soil thermal conductivity
k_g	W/m.K	Grout thermal conductivity
k_p	W/m.K	Pipe thermal conductivity
m_flow_borehole	kg/s	Mass flow rate(s) into a borehole
m_flow_network	kg/s	Mass flow rate(s) into a network (i.e.a borefield)
m_flow_pipe	kg/s	Mass flow rate through a pipe
Q_b	Watts	Total net heat extraction rate along borehole segment(s)
Q_f	Watts	Total net heat extraction rate of a fluid circuit (i.e. from inlet to outlet of a borehole pipe or network)
Q_t	Watts	Total net heat extraction rate of a borehole or network
R_b	m.K/W	Effective borehole thermal resistance
R_fp	m.K/W	Fluid to outer pipe wall thermal resistance
R_p	m.K/W	Conduction thermal resistance of a pipe wall
T_b	degC	Effective borehole wall temprature
T_f	degC	Fluid temperature

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

I

K

L

M

N

P

R

S

T

U

V