
pygfunction Documentation

Release 2.2.2

Massimo Cimmino

Jan 09, 2023

CONTENTS

1	Setting up pygfunction	3
2	Citing pygfunction in scientific publications	5
2.1	pygfunction v2.2	5
2.2	pygfunction v1.0	5
3	Modules	7
3.1	Borehole Module	7
3.2	g-Function Module	15
3.3	Heat Transfer Module	24
3.4	Load Aggregation Module	34
3.5	Media Module	39
3.6	Networks Module	41
3.7	Pipes Module	51
3.8	Utilities Module	87
4	Examples	91
4.1	Definition of a bore field using pre-defined configurations	91
4.2	Definition of a bore field using custom borehole positions	92
4.3	Definition of a bore field using external text file	94
4.4	Definition and visualization of a borehole	95
4.5	Calculation of g-functions with uniform borehole heat extraction rates	99
4.6	Calculation of g-functions with uniform borehole wall temperature	101
4.7	Compare the accuracy and speed of different g-function solvers	104
4.8	Calculation of g-functions with unequal numbers of segments	108
4.9	Calculation of g-Functions computed with non-uniform segment lengths	111
4.10	Simulation of a borehole using load aggregation	115
4.11	Compare the accuracy and speed of different load aggregation algorithms	119
4.12	Simulation of fluid temperatures in a borehole	123
4.13	Calculation of g-functions with equal inlet fluid temperature	129
4.14	Simulation of fluid temperatures in a field of multiple boreholes	132
4.15	Calculation of fluid temperature profiles in a borehole with independent U-tubes	137
4.16	Evaluation of thermal resistances using the multipole method	141
4.17	Calculation of effective bore field thermal resistance	144
4.18	Calculation of g-functions with mixed parallel and series connections	147
4.19	Calculation of g-functions with inclined boreholes	151
5	Nomenclature	155
5.1	Geometry	155
5.2	Fluid properties	155

5.3 Heat transfer	156
6 Indices and tables	157
Python Module Index	159
Index	161

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

SETTING UP PYGFUNCTION

pygfunction uses Python 3.7, along with the following packages:

- matplotlib ($\geq 3.5.1$),
- numpy ($\geq 1.21.5$)
- scipy ($\geq 1.7.3$)
- SecondaryCoolantProps (≥ 1.1)

pygfunction's- documentation is built using:

- sphinx ($\geq 4.4.0$)
- numpydoc ($\geq 1.2.0$)

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/`

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},  
}
```

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:



`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:

#	<i>x</i>	<i>y</i>	<i>H</i>	<i>D</i>	<i>r_b</i>	<i>tilt</i>	<i>orientation</i>
0.	0.	0.	100.	2.5	0.075	0.	0.
5.	0.	0.	100.	2.5	0.075	0.	0.
0.	5.	0.	100.	2.5	0.075	0.	0.
0.	10.	0.	100.	2.5	0.075	0.	0.
0.	20.	0.	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 in¹.

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 ($\text{abs}(d1 - d2)$) 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 $\text{abs}(H1 - H2)/H2 < \text{tol}$. Default is $1.0\text{e-}6$.

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 in². 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 $t < r_b^2 / (25 * \text{self.alpha})$. 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 m^2/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 corresponds to boundary condition *BC-I* as defined by Cimmino and Bernier (2014)^{Page 16, 2}.

- **‘UBWT’** :

Uniform borehole wall temperature. This 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 ($\text{abs}(d1-d2)$) 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 $\text{abs}(H1 - H2)/H2 < \text{tol}$. Default is $1.0\text{e-}6$.

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 in⁸. 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 ($\text{abs}(d1 - d2)$) 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 $\text{abs}(H1 - H2)/H2 < \text{tol}$. 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 corresponds to *BC-I*, as defined by⁹.

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 ($\text{abs}(d1 - d2)$) 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 $\text{abs}(H1 - H2)/H2 < \text{tol}$. 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 corresponds to *BC-III*, as defined by¹⁰.

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 ($\text{abs}(d1 - d2)$) 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 $\text{abs}(H1 - H2)/H2 < \text{tol}$. Default is $1.0\text{e-}6$.

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 Javed¹ 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 Bernier². The FLS solution is given by:

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

For inclined boreholes, the FLS solution was proposed by Lazzarotto⁴ and Lazzarotto and Björk⁵. The FLS solution is given by:

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

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 $h_{2 \rightarrow 1}(t)$ can be conveniently calculated by:

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

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 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, rea-Source=True, img-Source=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 Cimmino⁷. The equivalent FLS solution is given by:

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

Note: The reciprocal thermal response factor $h_{2 \rightarrow 1}(t)$ can be conveniently calculated by:

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

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 Lazzarotto⁹. The inclined FLS solution is given by:

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

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 $h_{2 \rightarrow 1}(t)$ can be conveniently calculated by:

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

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 Javed¹⁰ and extended to boreholes with different vertical positions by Cimmino and Bernier¹¹. The FLS solution is given by:

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

Note: The reciprocal thermal response factor $h_{2 \rightarrow 1}(t)$ can be conveniently calculated by:

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

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 Javed¹.

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 Liu².

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] Prandtl 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:

$$Q_b = a_{in}T_{f, network, in} + a_bT_b$$

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:

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

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**

[(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 inlet fluid temperatures of all boreholes.

Returns coefficients for the relation:

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

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**

[(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:

$$Q_{\text{network}} = a_{\text{in}} T_{f, \text{network}, \text{in}} + a_{\text{b}} T_{\text{b}}$$

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 inlet fluid temperature of the network.

Returns coefficients for the relation:

$$\mathbf{T}_{f, \text{network}, \text{in}} = \mathbf{a}_{q, f} Q_f + \mathbf{a}_b \mathbf{T}_b$$

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:

$$\mathbf{T}_{f, \text{network}, \text{out}} = \mathbf{a}_{in} T_{f, \text{network}, \text{in}} + \mathbf{a}_b \mathbf{T}_b$$

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:

$$T_{f,borehole,out} = a_{in}T_{f,network,in} + a_bT_b$$

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 *T_{in}*.

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 *Q_t*.

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 Hellstrom¹. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom².

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 Cimmino³. 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:

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

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:

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

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:

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

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:

$$T_{f,out} = a_{in}T_{f,in} + a_bT_b$$

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:

$$T_f(z) = a_{in}T_{f,in} + a_bT_b$$

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 Cimmino⁴ for boreholes with any number of U-tubes. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom⁵.

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:

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

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:

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

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 Cimmino⁶ for boreholes with any number of U-tubes. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom⁷.

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 Cimmino⁸. 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:

$$Q_b = a_{in}T_{f,in} + a_bT_b$$

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:

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

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:

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

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:

$$T_f(z) = a_{in}T_{f,in} + a_bT_b$$

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 Hellstrom⁹. Internal borehole thermal resistances are evaluated using the multipole method of Claesson and Hellstrom¹⁰.

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 Cimmino¹¹. 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:

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

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:

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

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:

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

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:

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

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:

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

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 Cimmino¹². 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 discontinuities 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.

$$\text{Nu}_{a,in} = 3.66 + 1.2(r^*)^{-0.8}$$

$$\text{Nu}_{a,out} = 3.66 + 1.2(r^*)^{0.5}$$

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 $2300 < \text{Re} < 4000$ 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 Hellstrom¹⁶.

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 method¹⁸. Thermal resistances are defined by:

$$\mathbf{T_f} - T_b = \mathbf{R} \cdot \mathbf{Q_p}$$

Delta-circuit thermal resistances are defined by:

$$q_{p,i,j} = \frac{T_{f,i} - T_{f,j}}{R_{i,j}^{\Delta}}$$
$$q_{p,i,i} = \frac{T_{f,i} - T_b}{R_{i,i}^{\Delta}}$$

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 Javed³.

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 Pasquier⁴.

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.])
```

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*

```
1  # -*- coding: utf-8 -*-
2  """ Example of definition of a bore field using pre-defined configurations.
3
4  """
5  import pygfunction as gt
6
7
8  def main():
9      # -----
10     # Parameters
11     # -----
12
13     # Borehole dimensions
14     D = 4.0          # Borehole buried depth (m)
15     H = 150.0        # Borehole length (m)
16     r_b = 0.075      # Borehole radius (m)
17     B = 7.5          # Borehole spacing (m)
18     N_1 = 4          # Number of boreholes in the x-direction (columns)
19     N_2 = 3          # Number of boreholes in the y-direction (rows)
20
21     # Circular field
22     N_b = 8          # Number of boreholes
23     R = 5.0          # Distance of the boreholes from the center of the field (m)
24
25     # -----
26     # Borehole fields
27     # -----
28
29     # Rectangular field of 4 x 3 boreholes
30     rectangularField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
```

(continues on next page)

(continued from previous page)

```

31
32     # Box-shaped field of 4 x 3 boreholes
33     boxField = gt.boreholes.box_shaped_field(N_1, N_2, B, B, H, D, r_b)
34
35     # U-shaped field of 4 x 3 boreholes
36     UField = gt.boreholes.U_shaped_field(N_1, N_2, B, B, H, D, r_b)
37
38     # L-shaped field of 4 x 3 boreholes
39     LField = gt.boreholes.L_shaped_field(N_1, N_2, B, B, H, D, r_b)
40
41     # Circular field of 8 boreholes
42     circleField = gt.boreholes.circle_field(N_b, R, H, D, r_b)
43
44     # -----
45     # Draw bore fields
46     # -----
47     for field in [rectangularField, boxField, UField, LField, circleField]:
48         gt.boreholes.visualize_field(field)
49
50     return
51
52
53 # Main function
54 if __name__ == '__main__':
55     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*

```

1  # -*- coding: utf-8 -*-
2  """ Example of definition of a bore field using custom borehole positions.
3
4  """
5  import pygfunction as gt
6
7
8  def main():
9      # -----
10     # Parameters
11     # -----
12
13     # Borehole dimensions

```

(continues on next page)

(continued from previous page)

```

14 D = 4.0          # Borehole buried depth (m)
15 H = 150.0       # Borehole length (m)
16 r_b = 0.075     # Borehole radius (m)
17
18 # Borehole positions
19 # Note: Two duplicate boreholes have been added to this list of positions.
20 #       Position 1 has a borehole that is directly on top of another bore
21 #       Position 2 has a borehole with radius inside of another bore
22 #       The duplicates will be removed with the remove_duplicates function
23 pos = [(0.0, 0.0),
24        (0.0, 0.0), # Duplicate (for example purposes)
25        (0.03, 0.0), # Duplicate (for example purposes)
26        (5.0, 0.),
27        (3.5, 4.0),
28        (1.0, 7.0),
29        (5.5, 5.5)]
30
31 # -----
32 # Borehole field
33 # -----
34
35 # Build list of boreholes
36 field = [gt.boreholes.Borehole(H, D, r_b, x, y) for (x, y) in pos]
37
38 # -----
39 # Find and remove duplicates from borehole field
40 # -----
41
42 field = gt.boreholes.remove_duplicates(field, disp=True)
43
44 # -----
45 # Draw bore field
46 # -----
47
48 gt.boreholes.visualize_field(field)
49
50 return
51
52
53 # Main function
54 if __name__ == '__main__':
55     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*

```
1  # -*- coding: utf-8 -*-
2  """ Example of definition of a bore field using custom borehole positions.
3
4  """
5  import pygfunction as gt
6
7
8  def main():
9      # -----
10     # Parameters
11     # -----
12
13     # Filepath to bore field text file
14     filename = './data/custom_field_32_boreholes.txt'
15
16     # -----
17     # Borehole field
18     # -----
19
20     # Build list of boreholes
21     field = gt.boreholes.field_from_file(filename)
22
23     # -----
24     # Draw bore field
25     # -----
26
27     gt.boreholes.visualize_field(field)
28
29     return
30
31
32 # Main function
33 if __name__ == '__main__':
34     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*

```

1  # -*- coding: utf-8 -*-
2  """ Example definition of a borehole. A top-view plot of the borehole is
3      created and the borehole resistance is computed.
4
5      """
6  import numpy as np
7  from scipy.constants import pi
8
9  import pygfunction as gt
10
11
12  def main():
13      # -----
14      # Simulation parameters
15      # -----
16
17      # Borehole dimensions
18      D = 5.          # Borehole buried depth (m)
19      H = 400.        # Borehole length (m)
20      r_b = 0.0875    # Borehole radius (m)
21
22      # Pipe dimensions (all configurations)
23      epsilon = 1.0e-6 # Pipe roughness (m)
24
25      # Pipe dimensions (single U-tube and double U-tube)
26      r_out = 0.0211   # Pipe outer radius (m)
27      r_in = 0.0147    # Pipe inner radius (m)
28      D_s = 0.052      # Shank spacing (m)
29
30      # Pipe dimensions (coaxial)
31      r_in_in = 0.0221 # Inside pipe inner radius (m)
32      r_in_out = 0.025 # Inside pipe outer radius (m)
33      r_out_in = 0.0487 # Outer pipe inside radius (m)
34      r_out_out = 0.055 # Outer pipe outside radius (m)
35      # Vectors of inner and outer pipe radii
36      # Note : The dimensions of the inlet pipe are the first elements of
37      #         the vectors. In this example, the inlet pipe is the inside pipe.
38      r_inner = np.array([r_in_in, r_out_in]) # Inner pipe radii (m)
39      r_outer = np.array([r_in_out, r_out_out]) # Outer pip radii (m)
40
41      # Ground properties
42      k_s = 2.0        # Ground thermal conductivity (W/m.K)
43
44      # Grout properties

```

(continues on next page)

(continued from previous page)

```

45 k_g = 1.0          # Grout thermal conductivity (W/m.K)
46
47 # Pipe properties
48 k_p = 0.4          # Pipe thermal conductivity (W/m.K)
49
50 # Fluid properties
51 # Total fluid mass flow rate per borehole (kg/s)
52 m_flow_borehole = 1.0
53 # The fluid is propylene-glycol (20 %) at 20 degC
54 fluid = gt.media.Fluid('MPG', 20.)
55 cp_f = fluid.cp     # Fluid specific isobaric heat capacity (J/kg.K)
56 rho_f = fluid.rho   # Fluid density (kg/m3)
57 mu_f = fluid.mu     # Fluid dynamic viscosity (kg/m.s)
58 k_f = fluid.k       # Fluid thermal conductivity (W/m.K)
59
60 # -----
61 # Initialize borehole model
62 # -----
63
64 borehole = gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)
65
66 # -----
67 # Define a single U-tube borehole
68 # -----
69
70 # Pipe positions
71 # Single U-tube [(x_in, y_in), (x_out, y_out)]
72 pos_single = [(-D_s, 0.), (D_s, 0.)]
73
74 # Pipe thermal resistance
75 R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
76     r_in, r_out, k_p)
77
78 # Fluid to inner pipe wall thermal resistance
79 m_flow_pipe = m_flow_borehole
80 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
81     m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
82 R_f = 1.0 / (h_f * 2 * pi * r_in)
83
84 # Single U-tube GHE in borehole
85 SingleUTube = gt.pipes.SingleUTube(
86     pos_single, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
87
88 # Check the geometry to make sure it is physically possible
89
90 # This class method is automatically called at the instantiation of the
91 # pipe object and raises an error if the pipe geometry is invalid. It is
92 # manually called here for demonstration.
93 check_single = SingleUTube._check_geometry()
94 print(f'The geometry of the borehole is valid (realistic/possible): '
95       f'{check_single!s}.')
96

```

(continues on next page)

(continued from previous page)

```

97  # Evaluate and print the effective borehole thermal resistance
98  R_b = SingleUTube.effective_borehole_thermal_resistance(
99      m_flow_borehole, fluid.cp)
100  print(f'Single U-tube Borehole thermal resistance: '
101        f'{R_b:.4f} m.K/W')
102
103  # Visualize the borehole geometry and save the figure
104  fig_single = SingleUTube.visualize_pipes()
105  fig_single.savefig('single-u-tube-borehole.png')
106
107  # -----
108  # Define a double U-tube borehole
109  # -----
110
111  # Pipe positions
112  # Double U-tube [(x_in1, y_in1), (x_in2, y_in2),
113  #               (x_out1, y_out1), (x_out2, y_out2)]
114  # Note: in series configuration, fluid enters pipe (in,1), exits (out,1),
115  # then enters (in,2) and finally exits (out,2)
116  # (if you view visualize_pipe, series is 1->3->2->4)
117  pos_double = [(-D_s, 0.), (0., -D_s), (D_s, 0.), (0., D_s)]
118
119  # Pipe thermal resistance
120  R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
121      r_in, r_out, k_p)
122
123  # Fluid to inner pipe wall thermal resistance
124  # Double U-tube in series
125  m_flow_pipe_series = m_flow_borehole
126  h_f_series = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
127      m_flow_pipe_series, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
128  R_f_series = 1.0 / (h_f_series * 2 * pi * r_in)
129  # Double U-tube in parallel
130  m_flow_pipe_parallel = m_flow_borehole / 2
131  h_f_parallel = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
132      m_flow_pipe_parallel, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
133  R_f_parallel = 1.0 / (h_f_parallel * 2 * pi * r_in)
134
135  # Double U-tube GHE in borehole
136  # Double U-tube in series
137  DoubleUTube_series = gt.pipes.MultipleUTube(
138      pos_double, r_in, r_out, borehole, k_s, k_g, R_p + R_f_series, 2,
139      config='series')
140  # Double U-tube in parallel
141  DoubleUTube_parallel = gt.pipes.MultipleUTube(
142      pos_double, r_in, r_out, borehole, k_s, k_g, R_p + R_f_parallel, 2,
143      config='parallel')
144
145  # Evaluate and print the effective borehole thermal resistance
146  R_b_series = DoubleUTube_series.effective_borehole_thermal_resistance(
147      m_flow_borehole, fluid.cp)
148  print(f'Double U-tube (series) Borehole thermal resistance: '

```

(continues on next page)

(continued from previous page)

```

149         f'{R_b_series:.4f} m.K/W')
150     R_b_parallel = DoubleUTube_parallel.effective_borehole_thermal_resistance(
151         m_flow_borehole, fluid.cp)
152     print(f'Double U-tube (parallel) Borehole thermal resistance: '
153           f'{R_b_parallel:.4f} m.K/W')
154
155     # Visualize the borehole geometry and save the figure
156     fig_double = DoubleUTube_series.visualize_pipes()
157     fig_double.savefig('double-u-tube-borehole.png')
158
159     # -----
160     # Define a coaxial borehole
161     # -----
162
163     # Pipe positions
164     # Coaxial pipe (x, y)
165     pos = (0., 0.)
166
167     # Pipe thermal resistance
168     # (the two pipes have the same thermal conductivity, k_p)
169     # Inner pipe
170     R_p_in = gt.pipes.conduction_thermal_resistance_circular_pipe(
171         r_in_in, r_in_out, k_p)
172     # Outer pipe
173     R_p_out = gt.pipes.conduction_thermal_resistance_circular_pipe(
174         r_out_in, r_out_out, k_p)
175
176     # Fluid-to-fluid thermal resistance
177     # Inner pipe
178     h_f_in = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
179         m_flow_borehole, r_in_in, mu_f, rho_f, k_f, cp_f, epsilon)
180     R_f_in = 1.0 / (h_f_in * 2 * pi * r_in_in)
181     # Outer pipe
182     h_f_a_in, h_f_a_out = \
183         gt.pipes.convective_heat_transfer_coefficient_concentric_annulus(
184             m_flow_borehole, r_in_out, r_out_in, mu_f, rho_f, k_f, cp_f,
185             epsilon)
186     R_f_out_in = 1.0 / (h_f_a_in * 2 * pi * r_in_out)
187     R_ff = R_f_in + R_p_in + R_f_out_in
188
189     # Coaxial GHE in borehole
190     R_f_out_out = 1.0 / (h_f_a_out * 2 * pi * r_out_in)
191     R_fp = R_p_out + R_f_out_out
192     Coaxial = gt.pipes.Coaxial(
193         pos, r_inner, r_outer, borehole, k_s, k_g, R_ff, R_fp, J=2)
194
195     # Evaluate and print the effective borehole thermal resistance
196     R_b = Coaxial.effective_borehole_thermal_resistance(
197         m_flow_borehole, fluid.cp)
198     print(f'Coaxial tube Borehole thermal resistance: {R_b:.4f} m.K/W')
199
200     # Visualize the borehole geometry and save the figure

```

(continues on next page)

(continued from previous page)

```

201 fig_coaxial = Coaxial.visualize_pipes()
202 fig_coaxial.savefig('coaxial-borehole.png')
203
204
205 if __name__ == '__main__':
206     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 Bernier¹.

The script is located in: *pygfunction/examples/uniform_heat_extraction_rate.py*

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of g-functions using uniform heat extraction rates.
3
4      The g-functions of fields of 3x2, 6x4 and 10x10 boreholes are calculated
5      for boundary condition of uniform heat extraction rate along the boreholes,
6      equal for all boreholes.
7
8  """
9  import matplotlib.lines as mlines
10 import matplotlib.pyplot as plt
11 import numpy as np
12 from matplotlib.ticker import AutoMinorLocator
13
14 import pygfunction as gt
15
16
17 def main():
18     # -----
19     # Simulation parameters
20     # -----
21
22     # Borehole dimensions
23     D = 4.0          # Borehole buried depth (m)
24     H = 150.0        # Borehole length (m)
25     r_b = 0.075      # Borehole radius (m)
26     B = 7.5          # Borehole spacing (m)
27
28     # Thermal properties
29     alpha = 1.0e-6    # Ground thermal diffusivity (m2/s)
30

```

(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)

```

31 # Path to validation data
32 filePath = './data/CiBel4_uniform_heat_extraction_rate.txt'
33
34 # g-Function calculation options
35 # The second field is evaluated with more segments to draw the
36 # temperature profiles. A uniform discretization is used to compare results
37 # with Cimmino and Bernier (2014).
38 options = [{'nSegments': 1,
39             'segment_ratios': None,
40             'disp': True,
41             'profiles': True},
42            {'nSegments': 12,
43             'segment_ratios': None,
44             'disp': True,
45             'profiles': True},
46            {'nSegments': 1,
47             'segment_ratios': None,
48             'disp': True,
49             'profiles': True}]
50
51 # The 'similarities' method is used to consider unequal numbers of segments
52 # per borehole and to plot heat extraction rate profiles along
53 # individual boreholes
54 method = 'similarities'
55
56 # Geometrically expanding time vector.
57 dt = 100*3600. # Time step
58 tmax = 3000. * 8760. * 3600. # Maximum time
59 Nt = 25 # Number of time steps
60 ts = H**2/(9.*alpha) # Bore field characteristic time
61 time = gt.utilities.time_geometric(dt, tmax, Nt)
62
63 # -----
64 # Borehole fields
65 # -----
66
67 # Field of 3x2 (n=6) boreholes
68 N_1 = 3
69 N_2 = 2
70 boreField1 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
71
72 # Field of 6x4 (n=24) boreholes
73 N_1 = 6
74 N_2 = 4
75 boreField2 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
76
77 # Field of 10x10 (n=100) boreholes
78 N_1 = 10
79 N_2 = 10
80 boreField3 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
81
82 # -----

```

(continues on next page)

(continued from previous page)

```

83  # Load data from Cimmino and Bernier (2014)
84  # -----
85  data = np.loadtxt(filePath, skiprows=55)
86
87  # -----
88  # Evaluate g-functions for all fields
89  # -----
90  for i, field in enumerate([boreField1, boreField2, boreField3]):
91      gfunc = gt.gfunction.gFunction(
92          field, alpha, time=time, boundary_condition='UHTR',
93          options=options[i], method=method)
94      # Draw g-function
95      ax = gfunc.visualize_g_function().axes[0]
96      # Draw reference g-function
97      ax.plot(data[:,0], data[:,i+1], 'bx')
98      ax.legend(['pygfunction', 'Cimmino and Bernier (2014)'])
99      ax.set_title('Field of {} boreholes'.format(len(field)))
100     plt.tight_layout()
101
102     # For the second borefield, draw the evolution of heat extraction rates
103     if i == 1:
104         gfunc.visualize_temperatures(iBoreholes=[18, 12, 14])
105         gfunc.visualize_temperature_profiles(iBoreholes=[14])
106
107     return
108
109
110 # Main function
111 if __name__ == '__main__':
112     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 Bernier¹.

The script is located in: *pygfunction/examples/uniform_temperature.py*

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of g-functions using uniform and equal borehole
3      wall temperatures.
4
5      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)

```

6     for boundary condition of uniform borehole wall temperature along the
7     boreholes, equal for all boreholes.
8
9     """
10    import matplotlib.pyplot as plt
11    import numpy as np
12    from time import perf_counter
13
14    import pygfunction as gt
15
16
17    def main():
18        # -----
19        # Simulation parameters
20        # -----
21
22        # Borehole dimensions
23        D = 4.0          # Borehole buried depth (m)
24        H = 150.0        # Borehole length (m)
25        r_b = 0.075      # Borehole radius (m)
26        B = 7.5          # Borehole spacing (m)
27
28        # Thermal properties
29        alpha = 1.0e-6   # Ground thermal diffusivity (m2/s)
30
31        # Path to validation data
32        filePath = './data/CiBe14_uniform_temperature.txt'
33
34        # g-Function calculation options
35        # A uniform discretization is used to compare results with Cimmino and
36        # Bernier (2014).
37        options = {'nSegments': 12,
38                  'segment_ratios': None,
39                  'disp': True,
40                  'profiles': True}
41
42        # Geometrically expanding time vector.
43        dt = 100*3600.    # Time step
44        tmax = 3000. * 8760. * 3600. # Maximum time
45        Nt = 25           # Number of time steps
46        ts = H**2/(9.*alpha) # Bore field characteristic time
47        time = gt.utilities.time_geometric(dt, tmax, Nt)
48        lntts = np.log(time/ts)
49
50        # -----
51        # Borehole fields
52        # -----
53
54        # Field of 3x2 (n=6) boreholes
55        N_1 = 3
56        N_2 = 2
57        boreField1 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)

```

(continues on next page)

(continued from previous page)

```

58
59 # Field of 6x4 (n=24) boreholes
60 N_1 = 6
61 N_2 = 4
62 boreField2 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
63
64 # Field of 10x10 (n=100) boreholes
65 N_1 = 10
66 N_2 = 10
67 boreField3 = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
68
69 # -----
70 # Load data from Cimmino and Bernier (2014)
71 # -----
72 data = np.loadtxt(filePath, skiprows=55)
73
74 # -----
75 # Evaluate g-functions for all fields
76 # -----
77 for i, field in enumerate([boreField1, boreField2, boreField3]):
78     nBoreholes = len(field)
79     # Compare 'similarities' and 'equivalent' solvers
80     t0 = perf_counter()
81     gfunc_similarities = gt.gfunction.gFunction(
82         field, alpha, time=time, options=options, method='similarities')
83     t1 = perf_counter()
84     t_similarities = t1 - t0
85     gfunc_equivalent = gt.gfunction.gFunction(
86         field, alpha, time=time, options=options, method='equivalent')
87     t2 = perf_counter()
88     t_equivalent = t2 - t1
89     # Draw g-function
90     ax = gfunc_similarities.visualize_g_function().axes[0]
91     ax.plot(lntts, gfunc_equivalent.gFunc)
92     # Draw reference g-function
93     ax.plot(data[:,0], data[:,i+1], 'o')
94     ax.legend([f'similarities (t = {t_similarities:.3f} sec)',
95              f'equivalent (t = {t_equivalent:.3f} sec)',
96              'Cimmino and Bernier (2014)'])
97     ax.set_title(f'Field of {nBoreholes} boreholes')
98     plt.tight_layout()
99
100 # For the second borefield, draw the evolution of heat extraction rates
101 if i == 1:
102     fig = gfunc_similarities.visualize_heat_extraction_rates(
103         iBoreholes=[18, 12, 14])
104     fig.suptitle(f'Field of {nBoreholes} boreholes: 'similarities' "
105                f"solver")
106     fig.tight_layout()
107
108     fig = gfunc_equivalent.visualize_heat_extraction_rates()
109     fig.suptitle(f'Field of {nBoreholes} boreholes: 'equivalent' "

```

(continues on next page)

(continued from previous page)

```

110         f"solver")
111     fig.tight_layout()
112
113     fig = gfunc_similarities.visualize_heat_extraction_rate_profiles(
114         iBoreholes=[18, 12, 14])
115     fig.suptitle(f"Field of {nBoreholes} boreholes: 'similarities' "
116                f"solver")
117     fig.tight_layout()
118
119     fig = gfunc_equivalent.visualize_heat_extraction_rate_profiles()
120     fig.suptitle(f"Field of {nBoreholes} boreholes: 'equivalent' "
121                f"solver")
122     fig.tight_layout()
123
124     return
125
126
127 # Main function
128 if __name__ == '__main__':
129     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 by 4 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`

```

1 # -*- coding: utf-8 -*-
2 """ Comparison of solvers for the evaluation of g-functions using uniform and
3     equal borehole wall temperatures.
4
5     The g-function of a field of 6x4 boreholes is calculated for a boundary
6     condition of uniform borehole wall temperature along the boreholes, equal
7     for all boreholes. Three different solvers are compared : 'detailed',
8     '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)
    lnnts = 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)

```

61 field = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
62
63 # -----
64 # Evaluate g-functions
65 # -----
66 t0 = perf_counter()
67 gfunc_detailed = gt.gfunction.gFunction(
68     field, alpha, time=time, options=options, method='detailed')
69 t1 = perf_counter()
70 t_detailed = t1 - t0
71 gfunc_similarities = gt.gfunction.gFunction(
72     field, alpha, time=time, options=options, method='similarities')
73 t2 = perf_counter()
74 t_similarities = t2 - t1
75 gfunc_equivalent = gt.gfunction.gFunction(
76     field, alpha, time=time, options=options, method='equivalent')
77 t3 = perf_counter()
78 t_equivalent = t3 - t2
79
80 # -----
81 # Plot results
82 # -----
83 # Draw g-functions
84 ax = gfunc_detailed.visualize_g_function().axes[0]
85 ax.plot(lntts, gfunc_similarities.gFunc, 'bx')
86 ax.plot(lntts, gfunc_equivalent.gFunc, 'ro')
87 ax.legend([f'detailed (t = {t_detailed:.3f} sec)',
88           f'similarities (t = {t_similarities:.3f} sec)',
89           f'equivalent (t = {t_equivalent:.3f} sec)'])
90 ax.set_title(f'Field of {N_1} by {N_2} boreholes')
91 plt.tight_layout()
92
93 # Draw absolute error
94 # Configure figure and axes
95 fig = gt.utilities._initialize_figure()
96 ax = fig.add_subplot(111)
97 # Axis labels
98 ax.set_xlabel(r'ln$(t/t_s)$')
99 ax.set_ylabel(r'Absolute error')
100 gt.utilities._format_axes(ax)
101 # Absolute error
102 ax.plot(lntts, np.abs(gfunc_similarities.gFunc - gfunc_detailed.gFunc),
103         '-', label='similarities')
104 ax.plot(lntts, np.abs(gfunc_equivalent.gFunc - gfunc_detailed.gFunc),
105         '--', label='equivalent')
106 ax.legend()
107 ax.set_title(f"Absolute error relative to the 'detailed' solver "
108             f"(Field of {N_1} by {N_2} boreholes)")
109 # Adjust to plot window
110 fig.tight_layout()
111
112 # Draw relative error

```

(continues on next page)

(continued from previous page)

```

113 # Configure figure and axes
114 fig = gt.utilities._initialize_figure()
115 ax = fig.add_subplot(111)
116 # Axis labels
117 ax.set_xlabel(r'\ln$(t/t_s)$')
118 ax.set_ylabel(r'Relative error')
119 gt.utilities._format_axes(ax)
120 # Relative error
121 gFunc_ref = gfunc_detailed.gFunc # reference g-function
122 ax.plot(lntts, (gfunc_similarities.gFunc - gFunc_ref) / gFunc_ref,
123         '-', label='similarities')
124 ax.plot(lntts, (gfunc_equivalent.gFunc - gFunc_ref) / gFunc_ref,
125         '--', label='equivalent')
126 ax.legend()
127 ax.set_title(f"Relative error relative to the 'detailed' solver "
128             f"(Field of {N_1} by {N_2} boreholes)")
129 # Adjust to plot window
130 fig.tight_layout()
131
132 # -----
133 # Borehole field (Second bore field)
134 # -----
135
136 # Field of 6x4 (n=24) boreholes
137 N_1 = 12
138 N_2 = 10
139 field = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
140
141 # -----
142 # Evaluate g-functions
143 # -----
144 gfunc_similarities = gt.gfunction.gFunction(
145     field, alpha, time=time, options=options, method='similarities')
146 t2 = perf_counter()
147 t_similarities = t2 - t1
148 gfunc_equivalent = gt.gfunction.gFunction(
149     field, alpha, time=time, options=options, method='equivalent')
150 t3 = perf_counter()
151 t_equivalent = t3 - t2
152
153 # -----
154 # Plot results
155 # -----
156 # Draw g-functions
157 ax = gfunc_similarities.visualize_g_function().axes[0]
158 ax.plot(lntts, gfunc_equivalent.gFunc, 'ro')
159 ax.legend([f'similarities (t = {t_similarities:.3f} sec)',
160           f'equivalent (t = {t_equivalent:.3f} sec)'])
161 ax.set_title(f'Field of {N_1} by {N_2} boreholes')
162 plt.tight_layout()
163
164 # Draw absolute error

```

(continues on next page)

(continued from previous page)

```

165     # Configure figure and axes
166     fig = gt.utilities._initialize_figure()
167     ax = fig.add_subplot(111)
168     # Axis labels
169     ax.set_xlabel(r'ln$(t/t_s)$')
170     ax.set_ylabel(r'Absolute error')
171     gt.utilities._format_axes(ax)
172     # Absolute error
173     ax.plot(lntts, np.abs(gfunc_equivalent.gFunc - gfunc_similarities.gFunc),
174             label='equivalent')
175     ax.legend()
176     ax.set_title(f"Absolute error relative to the 'similarities' solver "
177                 f"(Field of {N_1} by {N_2} boreholes)")
178     # Adjust to plot window
179     fig.tight_layout()
180
181     # Draw relative error
182     # Configure figure and axes
183     fig = gt.utilities._initialize_figure()
184     ax = fig.add_subplot(111)
185     # Axis labels
186     ax.set_xlabel(r'ln$(t/t_s)$')
187     ax.set_ylabel(r'Relative error')
188     gt.utilities._format_axes(ax)
189     # Relative error
190     ax.plot(lntts, (gfunc_equivalent.gFunc - gfunc_similarities.gFunc) / gfunc_
191     ↪ similarities.gFunc,
192             label='equivalent')
193     ax.legend()
194     ax.set_title(f"Relative error relative to the 'similarities' solver "
195                 f"(Field of {N_1} by {N_2} boreholes)")
196     # Adjust to plot window
197     fig.tight_layout()
198
199     return
200
201 # Main function
202 if __name__ == '__main__':
203     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`

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of g-functions with varied declaration of segments
3      using uniform borehole wall temperature.
4
5      The g-functions of a field of 6x4 boreholes is calculated with unequal
6      number of segments.
7  """
8  import matplotlib.pyplot as plt
9  from matplotlib.ticker import AutoMinorLocator
10 import numpy as np
11
12 import pygfunction as gt
13
14
15 def main():
16     # -----
17     # Simulation parameters
18     # -----
19
20     # Borehole dimensions
21     D = 4.0          # Borehole buried depth (m)
22     H = 150.0        # Borehole length (m)
23     r_b = 0.075      # Borehole radius (m)
24     B = 7.5          # Borehole spacing (m)
25
26     # Thermal properties
27     alpha = 1.0e-6    # Ground thermal diffusivity (m2/s)
28
29     # Geometrically expanding time vector.
30     dt = 100*3600.    # Time step
31     tmax = 3000. * 8760. * 3600. # Maximum time
32     Nt = 25           # Number of time steps
33     ts = H**2/(9.*alpha) # Bore field characteristic time
34     time = gt.utilities.time_geometric(dt, tmax, Nt)
35
36     # -----
37     # Borehole field
38     # -----
39
40     # Field of 6x4 (n=24) boreholes
41     N_1 = 6
42     N_2 = 4
43     boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
44     gt.boreholes.visualize_field(boreField)
45
46     # -----
47     # Evaluate g-functions with different segment options
48     # -----
49
50     # The 'similarities' method is used to consider unequal numbers of segments

```

(continues on next page)

(continued from previous page)

```

51  # per borehole and to plot heat extraction rate profiles along
52  # individual boreholes
53  method = 'similarities'
54
55  # Calculate g-function with equal number of segments for all boreholes and
56  # uniform segment lengths
57  nSegments = 24
58  options = {'nSegments': nSegments,
59            'segment_ratios': None,
60            'disp': True}
61
62  gfunc_equal = gt.gfunction.gFunction(
63      boreField, alpha, time=time, options=options, method=method)
64
65  # Calculate g-function with predefined number of segments for each
66  # borehole, the segment lengths will be uniform along each borehole, but
67  # the number of segments for the boreholes can be unequal.
68
69  # Boreholes 12, 14 and 18 have more segments than the others and their
70  # heat extraction rate profiles are plotted.
71  nSegments = [12] * len(boreField)
72  nSegments[12] = 24
73  nSegments[14] = 24
74  nSegments[18] = 24
75  options = {'nSegments': nSegments,
76            'segment_ratios': None,
77            'disp': True,
78            'profiles': True}
79
80  gfunc_unequal = gt.gfunction.gFunction(
81      boreField, alpha, time=time, options=options, method=method)
82
83  # Calculate g-function with equal number of segments for each borehole,
84  # unequal segment lengths along the length of the borehole defined by
85  # segment ratios. The segment ratios for each borehole are the same.
86
87  nSegments = 8
88  # Define the segment ratios for each borehole in each segment
89  # the segment lengths are defined top to bottom left to right
90  segment_ratios = np.array([0.05, 0.10, 0.10, 0.25, 0.25, 0.10, 0.10, 0.05])
91  options = {'nSegments': nSegments,
92            'segment_ratios': segment_ratios,
93            'disp': True,
94            'profiles': True}
95
96  g_func_predefined = gt.gfunction.gFunction(
97      boreField, alpha, time=time, options=options, method=method)
98
99  # -----
100  # Plot g-functions
101  # -----

```

(continues on next page)

(continued from previous page)

```

103 ax = gfunc_equal.visualize_g_function().axes[0]
104 ax.plot(np.log(time/ts), gfunc_unequal.gFunc, 'r-.')
105 ax.plot(np.log(time/ts), g_func_predefined.gFunc, 'k-.')
106 ax.legend(['Equal number of segments',
107           'Unequal number of segments',
108           'Unequal segment lengths'])
109 plt.tight_layout()
110
111 # Heat extraction rate profiles
112 fig = gfunc_unequal.visualize_heat_extraction_rates(
113     iBoreholes=[18, 12, 14])
114 fig.suptitle('Heat extraction rates (unequal number of segments)')
115 fig.tight_layout()
116 fig = g_func_predefined.visualize_heat_extraction_rates(
117     iBoreholes=[18, 12, 14])
118 fig.suptitle('Heat extraction rates (unequal segment lengths)')
119 fig.tight_layout()
120 fig = gfunc_unequal.visualize_heat_extraction_rate_profiles(
121     iBoreholes=[18, 12, 14])
122 fig.suptitle('Heat extraction rate profiles (unequal number of segments)')
123 fig.tight_layout()
124 fig = g_func_predefined.visualize_heat_extraction_rate_profiles(
125     iBoreholes=[18, 12, 14])
126 fig.suptitle('Heat extraction rate profiles (unequal segment lengths)')
127 fig.tight_layout()
128
129 return
130
131
132 # Main function
133 if __name__ == '__main__':
134     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`

```

1 # -*- coding: utf-8 -*-
2 """ Example of g-function calculation using non-uniform segment lengths along
3     the boreholes.
4
5     The g-functions of a field of 6x4 boreholes are calculated for two

```

(continues on next page)

(continued from previous page)

```

6     boundary conditions : (1) a uniform borehole wall temperature along the
7     boreholes equal for all boreholes, and (2) an equal inlet fluid
8     temperature into the boreholes. g-Functions using 8 segments in a
9     non-uniform discretization are compared to reference g-functions
10    calculated using 48 segments of equal lengths. It is shown that g-functions
11    can be calculated accurately using a small number of segments.
12    """
13
14    import pygfunction as gt
15    from numpy import pi
16    import matplotlib.pyplot as plt
17    import numpy as np
18
19
20    def main():
21        # -----
22        # Simulation parameters
23        # -----
24
25        # Borehole dimensions
26        D = 4.0          # Borehole buried depth (m)
27        H = 150.0        # Borehole length (m)
28        r_b = 0.075      # Borehole radius (m)
29        B = 7.5          # Borehole spacing (m)
30
31        # Pipe dimensions
32        r_out = 0.0211    # Pipe outer radius (m)
33        r_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_pipes = [(-D_s, 0.), (D_s, 0.)]
40
41        # Ground properties
42        alpha = 1.0e-6    # Ground thermal diffusivity (m2/s)
43        k_s = 2.0         # Ground thermal conductivity (W/m.K)
44
45        # Grout properties
46        k_g = 1.0         # Grout thermal conductivity (W/m.K)
47
48        # Pipe properties
49        k_p = 0.4         # Pipe thermal conductivity (W/m.K)
50
51        # Fluid properties
52        m_flow_borehole = 0.25 # Total fluid mass flow rate per borehole (kg/s)
53        # The fluid is propylene-glycol (20 %) at 20 degC
54        fluid = gt.media.Fluid('MPG', 20.)
55        cp_f = fluid.cp    # Fluid specific isobaric heat capacity (J/kg.K)
56        rho_f = fluid.rho  # Fluid density (kg/m3)
57        mu_f = fluid.mu    # Fluid dynamic viscosity (kg/m.s)

```

(continues on next page)

(continued from previous page)

```

58 k_f = fluid.k          # Fluid thermal conductivity (W/m.K)
59
60 # g-Function calculation options
61
62 # Number of segments used in the reference calculation with uniform
63 # discretization
64 nSegments_uniform = 48
65 options_uniform = {'nSegments': nSegments_uniform,
66                   'segment_ratios': None,
67                   'disp': True}
68 # Number of segments used in the calculation with non-uniform
69 # discretization
70 nSegments_unequal = 8
71 segment_ratios = gt.utilities.segment_ratios(
72     nSegments_unequal, end_length_ratio=0.02)
73 options_unequal = {'nSegments': nSegments_unequal,
74                   'segment_ratios': segment_ratios,
75                   'disp': True}
76
77 # Geometrically expanding time vector.
78 dt = 100*3600.          # Time step
79 tmax = 3000. * 8760. * 3600. # Maximum time
80 Nt = 25                 # Number of time steps
81 ts = H**2/(9.*alpha)    # Bore field characteristic time
82 time = gt.utilities.time_geometric(dt, tmax, Nt)
83
84 # -----
85 # Borehole field
86 # -----
87
88 # Field of 6x4 (n=24) boreholes
89 N_1 = 6
90 N_2 = 4
91 boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
92 gt.boreholes.visualize_field(boreField)
93 nBoreholes = len(boreField)
94
95 # -----
96 # Initialize pipe model
97 # -----
98
99 # Pipe thermal resistance
100 R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
101     r_in, r_out, k_p)
102 # Fluid to inner pipe wall thermal resistance (Single U-tube)
103 m_flow_pipe = m_flow_borehole
104 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
105     m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
106 R_f = 1.0/(h_f*2*pi*r_in)
107
108 # Single U-tube, same for all boreholes in the bore field
109 UTubes = []

```

(continues on next page)

(continued from previous page)

```

110 for borehole in boreField:
111     SingleUTube = gt.pipes.SingleUTube(
112         pos_pipes, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
113     UTubes.append(SingleUTube)
114 m_flow_network = m_flow_borehole*nBoreholes
115
116 # Network of boreholes connected in parallel
117 network = gt.networks.Network(
118     boreField, UTubes, m_flow_network=m_flow_network, cp_f=cp_f)
119
120 # -----
121 # Evaluate the g-functions for the borefield
122 # -----
123
124 # Compute g-function for the converged MIFT case with equal number of
125 # segments per borehole, and equal segment lengths along the boreholes
126 gfunc_MIFT_uniform = gt.gfunction.gFunction(
127     network, alpha, time=time, boundary_condition='MIFT',
128     options=options_uniform)
129
130 # Calculate the g-function for uniform borehole wall temperature
131 gfunc_UBWT_uniform = gt.gfunction.gFunction(
132     boreField, alpha, time=time, boundary_condition='UBWT',
133     options=options_uniform)
134
135 # Compute g-function for the MIFT case with equal number of segments per
136 # borehole, and non-uniform segment lengths along the boreholes
137 gfunc_MIFT_unequal = gt.gfunction.gFunction(
138     network, alpha, time=time, boundary_condition='MIFT',
139     options=options_unequal)
140
141 # Calculate the g-function for uniform borehole wall temperature
142 gfunc_UBWT_unequal = gt.gfunction.gFunction(
143     boreField, alpha, time=time, boundary_condition='UBWT',
144     options=options_unequal)
145
146 # Compute the rmse between the reference cases and the discretized
147 # (predicted) cases
148 RMSE_MIFT = RMSE(gfunc_MIFT_uniform.gFunc, gfunc_MIFT_unequal.gFunc)
149 print(f'RMSE (MIFT) = {RMSE_MIFT:.5f}')
150 RMSE_UBWT = RMSE(gfunc_UBWT_uniform.gFunc, gfunc_UBWT_unequal.gFunc)
151 print(f'RMSE (UBWT) = {RMSE_UBWT:.5f}')
152
153 # -----
154 # Plot g-functions
155 # -----
156
157 ax = gfunc_MIFT_uniform.visualize_g_function().axes[0]
158 ax.plot(np.log(time / ts), gfunc_UBWT_uniform.gFunc)
159 ax.plot(np.log(time / ts), gfunc_MIFT_unequal.gFunc, 'o')
160 ax.plot(np.log(time / ts), gfunc_UBWT_unequal.gFunc, 'o')
161 ax.legend(

```

(continues on next page)

(continued from previous page)

```

162     ['Equal inlet temperature (uniform segments)',
163      'Uniform borehole wall temperature (uniform segments)',
164      'Equal inlet temperature (non-uniform segments)',
165      'Uniform borehole wall temperature (non-uniform segments)'])
166 plt.tight_layout()
167
168     return
169
170
171 def RMSE(reference, predicted):
172     rmse = np.linalg.norm(predicted - reference) / len(reference)
173     return rmse
174
175
176 # Main function
177 if __name__ == '__main__':
178     main()

```

4.10 Simulation of a borehole using load aggregation

This example demonstrates the use of the *load aggregation* 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 Javed¹. 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*

```

1  # -*- coding: utf-8 -*-
2  """ Example of simulation of a geothermal system.
3
4      The g-function of a single borehole is calculated for boundary condition of
5      uniform borehole wall temperature along the borehole. Then, the borehole
6      wall temperature variations resulting from a time-varying load profile
7      are simulated using the aggregation method of Claesson and Javed (2012).
8
9  """
10 import matplotlib.pyplot as plt
11 import numpy as np
12 from scipy.constants import pi
13 from scipy.interpolate import interp1d
14 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)

```

15
16 import pygfunction as gt
17
18
19 def main():
20     # -----
21     # Simulation parameters
22     # -----
23
24     # Borehole dimensions
25     D = 4.0          # Borehole buried depth (m)
26     H = 150.0        # Borehole length (m)
27     r_b = 0.075      # Borehole radius (m)
28
29     # Ground properties
30     alpha = 1.0e-6    # Ground thermal diffusivity (m2/s)
31     k_s = 2.0         # Ground thermal conductivity (W/m.K)
32     T_g = 10.0        # Undisturbed ground temperature (degC)
33
34     # g-Function calculation options
35     options = {'nSegments': 8,
36               'disp': True}
37
38     # Simulation parameters
39     dt = 3600.        # Time step (s)
40     tmax = 20.*8760. * 3600. # Maximum time (s)
41     Nt = int(np.ceil(tmax/dt)) # Number of time steps
42     time = dt * np.arange(1, Nt+1)
43
44     # Evaluate heat extraction rate
45     Q_b = synthetic_load(time/3600.)
46
47     # Load aggregation scheme
48     LoadAgg = gt.load_aggregation.ClaessonJaved(dt, tmax)
49
50     # -----
51     # Calculate g-function
52     # -----
53
54     # The field contains only one borehole
55     boreField = [gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)]
56     # Get time values needed for g-function evaluation
57     time_req = LoadAgg.get_times_for_simulation()
58     # Calculate g-function
59     gFunc = gt.gfunction.gFunction(
60         boreField, alpha, time=time_req, options=options)
61     # Initialize load aggregation scheme
62     LoadAgg.initialize(gFunc.gFunc/(2*pi*k_s))
63
64     # -----
65     # Simulation
66     # -----

```

(continues on next page)

(continued from previous page)

```

67 T_b = np.zeros(Nt)
68 for i, (t, Q_b_i) in enumerate(zip(time, Q_b)):
69     # Increment time step by (1)
70     LoadAgg.next_time_step(t)
71
72     # Apply current load
73     LoadAgg.set_current_load(Q_b_i/H)
74
75     # Evaluate borehole wall temeprature
76     deltaT_b = LoadAgg.temporal_superposition()
77     T_b[i] = T_g - deltaT_b
78
79
80 # -----
81 # Calculate exact solution from convolution in the Fourier domain
82 # -----
83
84 # Heat extraction rate increment
85 dQ = np.zeros(Nt)
86 dQ[0] = Q_b[0]
87 dQ[1:] = Q_b[1:] - Q_b[:-1]
88 # Interpolated g-function
89 g = interp1d(time_req, gFunc.gFunc)(time)
90
91 # Convolution in Fourier domain
92 T_b_exact = T_g - fftconvolve(dQ, g/(2.0*pi*k_s*H), mode='full')[0:Nt]
93
94 # -----
95 # plot results
96 # -----
97
98 # Configure figure and axes
99 fig = gt.utilities._initialize_figure()
100
101 ax1 = fig.add_subplot(311)
102 # Axis labels
103 ax1.set_xlabel(r'$t$ [hours]')
104 ax1.set_ylabel(r'$Q_b$ [W]')
105 gt.utilities._format_axes(ax1)
106
107 hours = np.arange(1, Nt+1) * dt / 3600.
108 ax1.plot(hours, Q_b)
109
110 ax2 = fig.add_subplot(312)
111 # Axis labels
112 ax2.set_xlabel(r'$t$ [hours]')
113 ax2.set_ylabel(r'$T_b$ [degC]')
114 gt.utilities._format_axes(ax2)
115
116 ax2.plot(hours, T_b)
117 ax2.plot(hours, T_b_exact, 'k.')
118

```

(continues on next page)

(continued from previous page)

```

119     ax3 = fig.add_subplot(313)
120     # Axis labels
121     ax3.set_xlabel(r'$t$ [hours]')
122     ax3.set_ylabel(r'Error [degC]')
123     gt.utilities._format_axes(ax3)
124
125     ax3.plot(hours, T_b - T_b_exact)
126
127     # Adjust to plot window
128     plt.tight_layout()
129
130     return
131
132
133 def synthetic_load(x):
134     """
135     Synthetic load profile of Bernier et al. (2004).
136
137     Returns load y (in watts) at time x (in hours).
138     """
139     A = 2000.0
140     B = 2190.0
141     C = 80.0
142     D = 2.0
143     E = 0.01
144     F = 0.0
145     G = 0.95
146
147     func = (168.0-C)/168.0
148     for i in [1,2,3]:
149         func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0)-1.0) \
150                 *(np.sin(pi*i/84.0*(x-B)))
151     func = func*A*np.sin(pi/12.0*(x-B)) \
152           *np.sin(pi/4380.0*(x-B))
153
154     y = func + (-1.0)**np.floor(D/8760.0*(x-B))*abs(func) \
155         + E*(-1.0)**np.floor(D/8760.0*(x-B))/np.sign(np.cos(D*pi/4380.0*(x-F)))+G
156     return -y
157
158
159 # Main function
160 if __name__ == '__main__':
161     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 aggregation* 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, Liu², and Claesson and Javed³. 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*

```

1  # -*- coding: utf-8 -*-
2  """ Comparison of the accuracy and computational speed of different load
3      aggregation algorithms.
4
5      The g-function of a single borehole is calculated for boundary condition of
6      uniform borehole wall temperature along the borehole. Then, the borehole
7      wall temperature variations resulting from a time-varying load profile
8      are simulated using the aggregation methods of Bernier et al. (2004),
9      Liu (2005), and Claesson and Javed (2012). Results are compared to the
10     exact solution obtained by convolution in the Fourier domain.
11
12     Default parameters are used for each of the aggregation schemes.
13
14     """
15  from time import perf_counter
16
17  import matplotlib.pyplot as plt
18  import numpy as np
19  from scipy.constants import pi
20  from scipy.interpolate import interp1d
21  from scipy.signal import fftconvolve
22
23  import pygfunction as gt
24
25
26  def main():
27      # -----
28      # Simulation parameters
29      # -----

```

(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)

```

30
31 # Borehole dimensions
32 D = 4.0          # Borehole buried depth (m)
33 H = 150.0        # Borehole length (m)
34 r_b = 0.075      # Borehole radius (m)
35
36 # Ground properties
37 alpha = 1.0e-6   # Ground thermal diffusivity (m2/s)
38 k_s = 2.0         # Ground thermal conductivity (W/m.K)
39 T_g = 10.0        # Undisturbed ground temperature (degC)
40
41 # g-Function calculation options
42 options = {'nSegments': 8,
43           'disp': True}
44
45 # Simulation parameters
46 dt = 3600.        # Time step (s)
47 tmax = 20.*8760. * 3600. # Maximum time (s)
48 Nt = int(np.ceil(tmax/dt)) # Number of time steps
49 time = dt * np.arange(1, Nt+1)
50
51 # Evaluate heat extraction rate
52 Q_b = synthetic_load(time/3600.)
53
54 # Load aggregation schemes
55 ClaessonJaved = gt.load_aggregation.ClaessonJaved(dt, tmax)
56 MLAA = gt.load_aggregation.MLAA(dt, tmax)
57 Liu = gt.load_aggregation.Liu(dt, tmax)
58 LoadAggSchemes = [ClaessonJaved, MLAA, Liu]
59 loadAgg_labels = ['Claesson and Javed', 'MLAA', 'Liu']
60 loadAgg_lines = ['b-', 'k--', 'r-.']
61
62 # -----
63 # Calculate g-function
64 # -----
65
66 # The field contains only one borehole
67 boreField = [gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)]
68 # Evaluate the g-function on a geometrically expanding time grid
69 time_gFunc = gt.utilities.time_geometric(dt, tmax, 50)
70 # Calculate g-function
71 gFunc = gt.gfunction.gFunction(
72     boreField, alpha, time=time_gFunc, options=options)
73
74 # -----
75 # Simulation
76 # -----
77 nLoadAgg = len(LoadAggSchemes)
78 T_b = np.zeros((nLoadAgg, Nt))
79
80 t_calc = np.zeros(nLoadAgg)
81 for n, (LoadAgg, label) in enumerate(zip(LoadAggSchemes, loadAgg_labels)):

```

(continues on next page)

(continued from previous page)

```

82     print(f'Simulation using {label} ...')
83     # Interpolate g-function at required times
84     time_req = LoadAgg.get_times_for_simulation()
85     gFunc_int = interp1d(np.hstack([0., time_gFunc]),
86                          np.hstack([0., gFunc.gFunc]),
87                          kind='cubic',
88                          bounds_error=False,
89                          fill_value=(0., gFunc.gFunc[-1]))(time_req)
90     # Initialize load aggregation scheme
91     LoadAgg.initialize(gFunc_int/(2*pi*k_s))
92
93     tic = perf_counter()
94     for i in range(Nt):
95         # Increment time step by (1)
96         LoadAgg.next_time_step(time[i])
97
98         # Apply current load
99         LoadAgg.set_current_load(Q_b[i]/H)
100
101         # Evaluate borehole wall temperature
102         deltaT_b = LoadAgg.temporal_superposition()
103         T_b[n,i] = T_g - deltaT_b
104     toc = perf_counter()
105     t_calc[n] = toc - tic
106
107     # -----
108     # Calculate exact solution from convolution in the Fourier domain
109     # -----
110
111     # Heat extraction rate increment
112     dQ = np.zeros(Nt)
113     dQ[0] = Q_b[0]
114     dQ[1:] = Q_b[1:] - Q_b[:-1]
115     # Interpolated g-function
116     g = interp1d(time_gFunc, gFunc.gFunc)(time)
117
118     # Convolution in Fourier domain
119     T_b_exact = T_g - fftconvolve(dQ, g/(2.0*pi*k_s*H), mode='full')[0:Nt]
120
121     # -----
122     # plot results
123     # -----
124
125     # Configure figure and axes
126     fig = gt.utilities._initialize_figure()
127
128     ax1 = fig.add_subplot(311)
129     # Axis labels
130     ax1.set_xlabel(r'$t$ [hours]')
131     ax1.set_ylabel(r'$Q_b$ [W]')
132     gt.utilities._format_axes(ax1)
133     hours = np.array([(j+1)*dt/3600. for j in range(Nt)])

```

(continues on next page)

(continued from previous page)

```

134 ax1.plot(hours, Q_b)
135
136 ax2 = fig.add_subplot(312)
137 # Axis labels
138 ax2.set_xlabel(r'$t$ [hours]')
139 ax2.set_ylabel(r'$T_b$ [degC]')
140 gt.utilities._format_axes(ax2)
141 for T_b_n, line, label in zip(T_b, loadAgg_lines, loadAgg_labels):
142     ax2.plot(hours, T_b_n, line, label=label)
143 ax2.plot(hours, T_b_exact, 'k.', label='exact')
144 ax2.legend()
145
146 ax3 = fig.add_subplot(313)
147 # Axis labels
148 ax3.set_xlabel(r'$t$ [hours]')
149 ax3.set_ylabel(r'Error [degC]')
150 gt.utilities._format_axes(ax3)
151 for T_b_n, line, label in zip(T_b, loadAgg_lines, loadAgg_labels):
152     ax3.plot(hours, T_b_n - T_b_exact, line, label=label)
153 # Adjust to plot window
154 plt.tight_layout()
155
156 # -----
157 # Print performance metrics
158 # -----
159
160 # Maximum errors in evaluation of borehole wall temperatures
161 maxError = np.array([np.max(np.abs(T_b_n-T_b_exact)) for T_b_n in T_b])
162 # Print results
163 print('Simulation results')
164 for label, maxError_n, t_calc_n in zip(loadAgg_labels, maxError, t_calc):
165     print()
166     print((f' {label} ').center(60, '-'))
167     print(f'Maximum absolute error : {maxError_n:.3f} degC')
168     print(f'Calculation time : {t_calc_n:.3f} sec')
169
170 return
171
172
173 def synthetic_load(x):
174     """
175     Synthetic load profile of Bernier et al. (2004).
176
177     Returns load y (in watts) at time x (in hours).
178     """
179     A = 2000.0
180     B = 2190.0
181     C = 80.0
182     D = 2.0
183     E = 0.01
184     F = 0.0
185     G = 0.95

```

(continues on next page)

(continued from previous page)

```

186
187 func = (168.0-C)/168.0
188 for i in [1, 2, 3]:
189     func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0) - 1.0) \
190             *(np.sin(pi*i/84.0*(x - B)))
191 func = func*A*np.sin(pi/12.0*(x - B)) \
192       *np.sin(pi/4380.0*(x - B))
193
194 y = func + (-1.0)**np.floor(D/8760.0*(x - B))*abs(func) \
195     + E*(-1.0)**np.floor(D/8760.0*(x - B)) \
196     /np.sign(np.cos(D*pi/4380.0*(x - F)) + G)
197 return -y
198
199
200 # Main function
201 if __name__ == '__main__':
202     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 Javed¹. 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 Claesson³, (2) a double U-tube in parallel, using the model of Cimmino⁴, 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*

```

1  # -*- coding: utf-8 -*-
2  """ Example of simulation of a geothermal system and comparison between
3      single and double U-tube pipe configurations.
4
5      The g-function of a single borehole is calculated for boundary condition of
6      uniform borehole wall temperature along the borehole. Then, the borehole
7      wall temperature variations resulting from a time-varying load profile
8      are simulated using the aggregation method of Claesson and Javed (2012).
9      Predicted outlet fluid temperatures of three pipe configurations
10     (single U-tube, double U-tube in series, double U-tube in parallel) are
11     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)

```

64 k_f = fluid.k          # Fluid thermal conductivity (W/m.K)
65
66 # g-Function calculation options
67 options = {'nSegments': 8,
68            'disp': True}
69
70 # Simulation parameters
71 dt = 3600.              # Time step (s)
72 tmax = 1.*8760. * 3600. # Maximum time (s)
73 Nt = int(np.ceil(tmax/dt)) # Number of time steps
74 time = dt * np.arange(1, Nt+1)
75
76 # Evaluate heat extraction rate
77 Q = synthetic_load(time/3600.)
78
79 # Load aggregation scheme
80 LoadAgg = gt.load_aggregation.ClaessonJaved(dt, tmax)
81
82 # -----
83 # Calculate g-function
84 # -----
85
86 # The field contains only one borehole
87 borehole = gt.boreholes.Borehole(H, D, r_b, x=0., y=0.)
88 boreField = [borehole]
89 # Get time values needed for g-function evaluation
90 time_req = LoadAgg.get_times_for_simulation()
91 # Calculate g-function
92 gFunc = gt.gfunction.gFunction(
93     boreField, alpha, time=time_req, options=options)
94 # gt.gfunction.uniform_temperature(boreField, time_req, alpha,
95 #                                   nSegments=nSegments)
96 # Initialize load aggregation scheme
97 LoadAgg.initialize(gFunc.gFunc/(2*pi*k_s))
98
99 # -----
100 # Initialize pipe models
101 # -----
102
103 # Pipe thermal resistance
104 R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
105     rp_in, rp_out, k_p)
106 # Fluid to inner pipe wall thermal resistance (Single U-tube and double
107 # U-tube in series)
108 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
109     m_flow, rp_in, visc_f, den_f, k_f, cp_f, epsilon)
110 R_f_ser = 1.0/(h_f*2*pi*rp_in)
111 # Fluid to inner pipe wall thermal resistance (Double U-tube in parallel)
112 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
113     m_flow/2, rp_in, visc_f, den_f, k_f, cp_f, epsilon)
114 R_f_par = 1.0/(h_f*2*pi*rp_in)
115

```

(continues on next page)

(continued from previous page)

```

116 # Single U-tube
117 SingleUTube = gt.pipes.SingleUTube(pos_single, rp_in, rp_out,
118                                   borehole, k_s, k_g, R_f_ser + R_p)
119 # Double U-tube (parallel)
120 DoubleUTube_par = gt.pipes.MultipleUTube(pos_double, rp_in, rp_out,
121                                           borehole, k_s, k_g, R_f_par + R_p,
122                                           nPipes=2, config='parallel')
123 # Double U-tube (series)
124 DoubleUTube_ser = gt.pipes.MultipleUTube(pos_double, rp_in, rp_out,
125                                           borehole, k_s, k_g, R_f_ser + R_p,
126                                           nPipes=2, config='series')
127
128 # -----
129 # Simulation
130 # -----
131
132 T_b = np.zeros(Nt)
133 T_f_in_single = np.zeros(Nt)
134 T_f_in_double_par = np.zeros(Nt)
135 T_f_in_double_ser = np.zeros(Nt)
136 T_f_out_single = np.zeros(Nt)
137 T_f_out_double_par = np.zeros(Nt)
138 T_f_out_double_ser = np.zeros(Nt)
139 for i, (t, Q_b_i) in enumerate(zip(time, Q)):
140     # Increment time step by (1)
141     LoadAgg.next_time_step(t)
142
143     # Apply current load
144     LoadAgg.set_current_load(Q_b_i/H)
145
146     # Evaluate borehole wall temperature
147     deltaT_b = LoadAgg.temporal_superposition()
148     T_b[i] = T_g - deltaT_b
149
150     # Evaluate inlet fluid temperature
151     T_f_in_single[i] = SingleUTube.get_inlet_temperature(
152         Q[i], T_b[i], m_flow, cp_f)
153     T_f_in_double_par[i] = DoubleUTube_par.get_inlet_temperature(
154         Q[i], T_b[i], m_flow, cp_f)
155     T_f_in_double_ser[i] = DoubleUTube_ser.get_inlet_temperature(
156         Q[i], T_b[i], m_flow, cp_f)
157
158     # Evaluate outlet fluid temperature
159     T_f_out_single[i] = SingleUTube.get_outlet_temperature(
160         T_f_in_single[i], T_b[i], m_flow, cp_f)
161     T_f_out_double_par[i] = DoubleUTube_par.get_outlet_temperature(
162         T_f_in_double_par[i], T_b[i], m_flow, cp_f)
163     T_f_out_double_ser[i] = DoubleUTube_ser.get_outlet_temperature(
164         T_f_in_double_ser[i], T_b[i], m_flow, cp_f)
165
166 # -----
167 # Plot hourly heat extraction rates and temperatures

```

(continues on next page)

(continued from previous page)

```

168 # -----
169
170 # Configure figure and axes
171 fig = gt.utilities._initialize_figure()
172
173 ax1 = fig.add_subplot(211)
174 # Axis labels
175 ax1.set_xlabel(r'Time [hours]')
176 ax1.set_ylabel(r'Total heat extraction rate [W]')
177 gt.utilities._format_axes(ax1)
178
179 # Plot heat extraction rates
180 hours = np.arange(1, Nt+1) * dt / 3600.
181 ax1.plot(hours, Q)
182
183 ax2 = fig.add_subplot(212)
184 # Axis labels
185 ax2.set_xlabel(r'Time [hours]')
186 ax2.set_ylabel(r'Temperature [degC]')
187 gt.utilities._format_axes(ax2)
188
189 # Plot temperatures
190 ax2.plot(hours, T_b, 'k-', lw=1.5, label='Borehole wall')
191 ax2.plot(hours, T_f_out_single, '--',
192          label='Outlet, single U-tube')
193 ax2.plot(hours, T_f_out_double_par, '-.',
194          label='Outlet, double U-tube (parallel)')
195 ax2.plot(hours, T_f_out_double_ser, ':',
196          label='Outlet, double U-tube (series)')
197 ax2.legend()
198
199 # Adjust to plot window
200 plt.tight_layout()
201
202 # -----
203 # Plot fluid temperature profiles
204 # -----
205
206 # Evaluate temperatures at nz evenly spaced depths along the borehole
207 # at the (it+1)-th time step
208 nz = 20
209 it = 8724
210 z = np.linspace(0., H, num=nz)
211 T_f_single = SingleUTube.get_temperature(z,
212                                         T_f_in_single[it],
213                                         T_b[it],
214                                         m_flow,
215                                         cp_f)
216 T_f_double_par = DoubleUTube_par.get_temperature(z,
217                                                  T_f_in_double_par[it],
218                                                  T_b[it],
219                                                  m_flow,

```

(continues on next page)

(continued from previous page)

```

220                                     cp_f)
221     T_f_double_ser = DoubleUTube_ser.get_temperature(z,
222                                                     T_f_in_double_ser[it],
223                                                     T_b[it],
224                                                     m_flow,
225                                                     cp_f)
226
227     # Configure figure and axes
228     fig = gt.utilities._initialize_figure()
229
230     ax3 = fig.add_subplot(131)
231     # Axis labels
232     ax3.set_xlabel(r'Temperature [degC]')
233     ax3.set_ylabel(r'Depth from borehole head [m]')
234     gt.utilities._format_axes(ax3)
235
236     # Plot temperatures
237     ax3.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]), 'k--')
238     ax3.plot(T_f_single, z, 'b-')
239     ax3.legend(['Borehole wall', 'Fluid'])
240
241     ax4 = fig.add_subplot(132)
242     # Axis labels
243     ax4.set_xlabel(r'Temperature [degC]')
244     ax4.set_ylabel(r'Depth from borehole head [m]')
245     gt.utilities._format_axes(ax4)
246
247     # Plot temperatures
248     ax4.plot(T_f_double_par, z, 'b-')
249     ax4.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]), 'k--')
250
251     ax5 = fig.add_subplot(133)
252     # Axis labels
253     ax5.set_xlabel(r'Temperature [degC]')
254     ax5.set_ylabel(r'Depth from borehole head [m]')
255     gt.utilities._format_axes(ax5)
256
257     # Plot temperatures
258     ax5.plot(T_f_double_ser, z, 'b-')
259     ax5.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]), 'k--')
260
261     # Reverse y-axes
262     ax3.set_ylim(ax3.get_ylim()[::-1])
263     ax4.set_ylim(ax4.get_ylim()[::-1])
264     ax5.set_ylim(ax5.get_ylim()[::-1])
265     # Adjust to plot window
266     plt.tight_layout()
267
268     return
269
270
271 def synthetic_load(x):

```

(continues on next page)

(continued from previous page)

```

272 """
273 Synthetic load profile of Bernier et al. (2004).
274
275 Returns load y (in watts) at time x (in hours).
276 """
277 A = 2000.0
278 B = 2190.0
279 C = 80.0
280 D = 2.0
281 E = 0.01
282 F = 0.0
283 G = 0.95
284
285 func = (168.0-C)/168.0
286 for i in [1, 2, 3]:
287     func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0)-1.0) \
288             *(np.sin(pi*i/84.0*(x-B)))
289 func = func*A*np.sin(pi/12.0*(x-B)) \
290     *np.sin(pi/4380.0*(x-B))
291
292 y = func + (-1.0)**np.floor(D/8760.0*(x-B))*abs(func) \
293     + E*(-1.0)**np.floor(D/8760.0*(x-B))/np.sign(np.cos(D*pi/4380.0*(x-F)))+G
294 return -y
295
296
297 # Main function
298 if __name__ == '__main__':
299     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 Cimmino¹. 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*

```

1 # -*- coding: utf-8 -*-
2 """ Example of calculation of g-functions using equal inlet temperatures.
3
4 The g-functions of a field of 6x4 boreholes are calculated for boundary
5 conditions of (a) uniform heat extraction rate, equal for all boreholes,
6 (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)

```

7     boreholes, and (c) equal inlet fluid temperature into all boreholes.
8
9     """
10    import matplotlib.pyplot as plt
11    import numpy as np
12    from matplotlib.ticker import AutoMinorLocator
13    from scipy import pi
14
15    import pygfunction as gt
16
17
18    def main():
19        # -----
20        # Simulation parameters
21        # -----
22
23        # Borehole dimensions
24        D = 4.0          # Borehole buried depth (m)
25        H = 150.0        # Borehole length (m)
26        r_b = 0.075      # Borehole radius (m)
27        B = 7.5          # Borehole spacing (m)
28
29        # Pipe dimensions
30        r_out = 0.0211   # Pipe outer radius (m)
31        r_in = 0.0147    # Pipe inner radius (m)
32        D_s = 0.052      # Shank spacing (m)
33        epsilon = 1.0e-6 # Pipe roughness (m)
34
35        # Pipe positions
36        # Single U-tube [(x_in, y_in), (x_out, y_out)]
37        pos_pipes = [(-D_s, 0.), (D_s, 0.)]
38
39        # Ground properties
40        alpha = 1.0e-6   # Ground thermal diffusivity (m2/s)
41        k_s = 2.0        # Ground thermal conductivity (W/m.K)
42
43        # Grout properties
44        k_g = 1.0        # Grout thermal conductivity (W/m.K)
45
46        # Pipe properties
47        k_p = 0.4        # Pipe thermal conductivity (W/m.K)
48
49        # Fluid properties
50        m_flow_borehole = 0.25 # Total fluid mass flow rate per borehole (kg/s)
51        # The fluid is propylene-glycol (20 %) at 20 degC
52        fluid = gt.media.Fluid('MPG', 20.)
53        cp_f = fluid.cp   # Fluid specific isobaric heat capacity (J/kg.K)
54        rho_f = fluid.rho # Fluid density (kg/m3)
55        mu_f = fluid.mu   # Fluid dynamic viscosity (kg/m.s)
56        k_f = fluid.k     # Fluid thermal conductivity (W/m.K)
57
58        # g-Function calculation options

```

(continues on next page)

(continued from previous page)

```

59 nSegments = 8
60 options = {'nSegments': nSegments,
61           'disp': True}
62
63 # Geometrically expanding time vector.
64 dt = 100*3600.          # Time step
65 tmax = 3000. * 8760. * 3600. # Maximum time
66 Nt = 25                # Number of time steps
67 ts = H**2/(9.*alpha)    # Bore field characteristic time
68 time = gt.utilities.time_geometric(dt, tmax, Nt)
69
70 # -----
71 # Borehole field
72 # -----
73
74 # Field of 6x4 (n=24) boreholes
75 N_1 = 6
76 N_2 = 4
77 boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
78 nBoreholes = len(boreField)
79
80 # -----
81 # Initialize pipe model
82 # -----
83
84 # Pipe thermal resistance
85 R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
86     r_in, r_out, k_p)
87 # Fluid to inner pipe wall thermal resistance (Single U-tube)
88 m_flow_pipe = m_flow_borehole
89 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
90     m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
91 R_f = 1.0/(h_f*2*pi*r_in)
92
93 # Single U-tube, same for all boreholes in the bore field
94 UTubes = []
95 for borehole in boreField:
96     SingleUTube = gt.pipes.SingleUTube(pos_pipes, r_in, r_out,
97                                         borehole, k_s, k_g, R_f + R_p)
98     UTubes.append(SingleUTube)
99 m_flow_network = m_flow_borehole*nBoreholes
100 network = gt.networks.Network(
101     boreField, UTubes, m_flow_network=m_flow_network, cp_f=cp_f,
102     nSegments=nSegments)
103
104 # -----
105 # Evaluate the g-functions for the borefield
106 # -----
107
108 # Calculate the g-function for uniform heat extraction rate
109 gfunc_uniform_Q = gt.gfunction.gFunction(
110     boreField, alpha, time=time, boundary_condition='UHTR', options=options)

```

(continues on next page)

(continued from previous page)

```

111
112     # Calculate the g-function for uniform borehole wall temperature
113     gfunc_uniform_T = gt.gfunction.gFunction(
114         boreField, alpha, time=time, boundary_condition='UBWT', options=options)
115
116     # Calculate the g-function for equal inlet fluid temperature
117     gfunc_equal_Tf_in = gt.gfunction.gFunction(
118         network, alpha, time=time, boundary_condition='MIFT', options=options)
119
120     # -----
121     # Plot g-functions
122     # -----
123
124     ax = gfunc_uniform_Q.visualize_g_function().axes[0]
125     ax.plot(np.log(time/ts), gfunc_uniform_T.gFunc, 'k--')
126     ax.plot(np.log(time/ts), gfunc_equal_Tf_in.gFunc, 'r-.')
127     ax.legend(['Uniform heat extraction rate',
128               'Uniform borehole wall temperature',
129               'Equal inlet temperature'])
130     plt.tight_layout()
131
132     return
133
134
135 # Main function
136 if __name__ == '__main__':
137     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 condition¹. Then, the borehole wall temperature variations are calculated using the load aggregation scheme of Claesson and Javed². 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 Cimmino⁴.

The script is located in: *pygfunction/examples/fluid_temperature_multiple_boreholes.py*

```

1  # -*- coding: utf-8 -*-
2  """ 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)

```

55
56 # Fluid properties
57 m_flow_borehole = 0.25      # Total fluid mass flow rate per borehole (kg/s)
58 m_flow_network = m_flow_borehole*N_1*N_2    # 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 rho_f = fluid.rho          # Fluid density (kg/m3)
63 mu_f = fluid.mu            # Fluid dynamic viscosity (kg/m.s)
64 k_f = fluid.k              # Fluid thermal conductivity (W/m.K)
65
66 # g-Function calculation options
67 options = {'nSegments': 8,
68            'disp': True}
69
70 # Simulation parameters
71 dt = 3600.                  # Time step (s)
72 tmax = 1.*8760. * 3600.     # Maximum time (s)
73 Nt = int(np.ceil(tmax/dt))  # Number of time steps
74 time = dt * np.arange(1, Nt+1)
75
76 # Load aggregation scheme
77 LoadAgg = gt.load_aggregation.ClaessonJaved(dt, tmax)
78
79 # -----
80 # Initialize bore field and pipe models
81 # -----
82
83 # The field is a rectangular array
84 boreField = gt.boreholes.rectangle_field(N_1, N_2, B, B, H, D, r_b)
85 nBoreholes = len(boreField)
86
87 # Pipe thermal resistance
88 R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
89     r_in, r_out, k_p)
90
91 # Fluid to inner pipe wall thermal resistance (Double U-tube in parallel)
92 m_flow_pipe = m_flow_borehole/2
93 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
94     m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
95 R_f = 1.0/(h_f*2*pi*r_in)
96
97 # Double U-tube (parallel), same for all boreholes in the bore field
98 UTubes = []
99 for borehole in boreField:
100     UTube = gt.pipes.MultipleUTube(
101         pos, r_in, r_out, borehole, k_s, k_g, R_f + R_p,
102         nPipes=2, config='parallel')
103     UTubes.append(UTube)
104 # Build a network object from the list of UTubes
105 network = gt.networks.Network(
106     boreField, UTubes, m_flow_network=m_flow_network, cp_f=cp_f)

```

(continues on next page)

(continued from previous page)

```

107
108 # -----
109 # Calculate g-function
110 # -----
111
112 # Get time values needed for g-function evaluation
113 time_req = LoadAgg.get_times_for_simulation()
114 # Calculate g-function
115 gFunc = gt.gfunction.gFunction(
116     network, alpha, time=time_req, boundary_condition='MIFT',
117     options=options)
118 # Initialize load aggregation scheme
119 LoadAgg.initialize(gFunc.gFunc/(2*pi*k_s))
120
121 # -----
122 # Simulation
123 # -----
124
125 # Evaluate heat extraction rate
126 Q_tot = nBoreholes*synthetic_load(time/3600.)
127
128 T_b = np.zeros(Nt)
129 T_f_in = np.zeros(Nt)
130 T_f_out = np.zeros(Nt)
131 for i, (t, Q_i) in enumerate(zip(time, Q_tot)):
132     # Increment time step by (1)
133     LoadAgg.next_time_step(t)
134
135     # Apply current load (in watts per meter of borehole)
136     Q_b = Q_i/nBoreholes
137     LoadAgg.set_current_load(Q_b/H)
138
139     # Evaluate borehole wall temperature
140     deltaT_b = LoadAgg.temporal_superposition()
141     T_b[i] = T_g - deltaT_b
142
143     # Evaluate inlet fluid temperature (all boreholes are the same)
144     T_f_in[i] = network.get_network_inlet_temperature(
145         Q_tot[i], T_b[i], m_flow_network, cp_f, nSegments=1)
146
147     # Evaluate outlet fluid temperature
148     T_f_out[i] = network.get_network_outlet_temperature(
149         T_f_in[i], T_b[i], m_flow_network, cp_f, nSegments=1)
150
151 # -----
152 # Plot hourly heat extraction rates and temperatures
153 # -----
154
155 # Configure figure and axes
156 fig = gt.utilities._initialize_figure()
157
158 ax1 = fig.add_subplot(211)

```

(continues on next page)

(continued from previous page)

```

159 # Axis labels
160 ax1.set_xlabel(r'Time [hours]')
161 ax1.set_ylabel(r'Total heat extraction rate [W]')
162 gt.utilities._format_axes(ax1)
163
164 # Plot heat extraction rates
165 hours = np.arange(1, Nt+1) * dt / 3600.
166 ax1.plot(hours, Q_tot)
167
168 ax2 = fig.add_subplot(212)
169 # Axis labels
170 ax2.set_xlabel(r'Time [hours]')
171 ax2.set_ylabel(r'Temperature [degC]')
172 gt.utilities._format_axes(ax2)
173
174 # Plot temperatures
175 ax2.plot(hours, T_b, label='Borehole wall')
176 ax2.plot(hours, T_f_out, '-',
177          label='Outlet, double U-tube (parallel)')
178 ax2.legend()
179
180 # Adjust to plot window
181 plt.tight_layout()
182
183 # -----
184 # Plot fluid temperature profiles
185 # -----
186
187 # Evaluate temperatures at nz evenly spaced depths along the borehole
188 # at the (it+1)-th time step
189 nz = 20
190 it = 8724
191 z = np.linspace(0., H, num=nz)
192 T_f = UTubes[0].get_temperature(
193     z, T_f_in[it], T_b[it], m_flow_borehole, cp_f)
194
195
196 # Configure figure and axes
197 fig = gt.utilities._initialize_figure()
198
199 ax3 = fig.add_subplot(111)
200 # Axis labels
201 ax3.set_xlabel(r'Temperature [degC]')
202 ax3.set_ylabel(r'Depth from borehole head [m]')
203 gt.utilities._format_axes(ax3)
204
205 # Plot temperatures
206 pltFlu = ax3.plot(T_f, z, 'b-', label='Fluid')
207 pltWal = ax3.plot(np.array([T_b[it], T_b[it]]), np.array([0., H]),
208                  'k--', label='Borehole wall')
209 ax3.legend(handles=[pltFlu[0]]+pltWal)
210

```

(continues on next page)

(continued from previous page)

```

211     # Reverse y-axes
212     ax3.set_ylim(ax3.get_ylim()[::-1])
213     # Adjust to plot window
214     plt.tight_layout()
215
216     return
217
218
219 def synthetic_load(x):
220     """
221     Synthetic load profile of Bernier et al. (2004).
222
223     Returns load y (in watts) at time x (in hours).
224     """
225     A = 2000.0
226     B = 2190.0
227     C = 80.0
228     D = 2.0
229     E = 0.01
230     F = 0.0
231     G = 0.95
232
233     func = (168.0-C)/168.0
234     for i in [1, 2, 3]:
235         func += 1.0/(i*pi)*(np.cos(C*pi*i/84.0)-1.0) \
236                 *(np.sin(pi*i/84.0*(x-B)))
237     func = func*A*np.sin(pi/12.0*(x-B)) \
238           *np.sin(pi/4380.0*(x-B))
239
240     y = func + (-1.0)**np.floor(D/8760.0*(x-B))*abs(func) \
241         + E*(-1.0)**np.floor(D/8760.0*(x-B))/np.sign(np.cos(D*pi/4380.0*(x-F)))+G
242     return -y
243
244
245 # Main function
246 if __name__ == '__main__':
247     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 Cimmino¹. 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 temperature profiles presented by Cimmino^{Page 137, 1}.

The script is located in: `pygfunction/examples/multiple_independent_Utubes.py`

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of fluid temperature profiles in a borehole with
3      independent U-tubes.
4
5      The fluid temperature profiles in a borehole with 4 independent U-tubes are
6      calculated. The borehole has 4 U-tubes, each with different inlet fluid
7      temperatures and different inlet fluid mass flow rates. The borehole wall
8      temperature is uniform. Results are verified against the results of
9      Cimmino (2016).
10
11  """
12  import matplotlib.lines as mlines
13  import matplotlib.pyplot as plt
14  import numpy as np
15  from matplotlib.ticker import AutoMinorLocator
16  from scipy import pi
17
18  import pygfunction as gt
19
20
21  def main():
22      # -----
23      # Simulation parameters
24      # -----
25
26      # Borehole dimensions
27      D = 2.5          # Borehole buried depth (m)
28      H = 100.0        # Borehole length (m)
29      r_b = 0.075      # Borehole radius (m)
30
31      # Pipe dimensions
32      r_out = 0.010     # Pipe outer radius (m)
33      r_in = 0.008      # Pipe inner radius (m)
34      D_s = 0.060      # Shank spacing (m)
35
36      # Pipe positions
37      nPipes = 4        # Number of U-tube pipes (-)
38      pos_pipes = _pipePositions(D_s, nPipes)
39
40      # Ground properties
41      k_s = 2.0         # Ground thermal conductivity (W/m.K)
42
43      # Grout properties
44      k_g = 1.0         # Grout thermal conductivity (W/m.K)
45
46      # Fluid properties
47      R_fp = 1e-30      # Fluid to outer pipe wall thermal resistance (m.K/W)
48      # Fluid specific isobaric heat capacity per U-tube (J/kg.K)
49      cp_f = 4000.*np.ones(nPipes)
50

```

(continues on next page)

(continued from previous page)

```

51  # Borehole wall temperature (degC)
52  T_b = 2.0
53  # Total fluid mass flow rate per U-tube (kg/s)
54  m_flow_borehole = np.array([0.40, 0.35, 0.30, 0.25])
55  # Inlet fluid temperatures per U-tube (degC)
56  T_f_in = np.array([6.0, -6.0, 5.0, -5.0])
57
58  # Path to validation data
59  filePath = './data/Cimmi16_multiple_independent_Utubes.txt'
60
61  # -----
62  # Initialize pipe model
63  # -----
64
65  # Borehole object
66  borehole = gt.boreholes.Borehole(H, D, r_b, 0., 0.)
67  # Multiple independent U-tubes
68  MultipleUTube = gt.pipes.IndependentMultipleUTube(
69      pos_pipes, r_in, r_out, borehole, k_s, k_g, R_fp, nPipes, J=0)
70
71  # -----
72  # Evaluate the outlet fluid temperatures and fluid temperature profiles
73  # -----
74
75  # Calculate the outlet fluid temperatures
76  T_f_out = MultipleUTube.get_outlet_temperature(
77      T_f_in, T_b, m_flow_borehole, cp_f)
78
79  # Evaluate temperatures at nz evenly spaced depths along the borehole
80  nz = 20
81  z = np.linspace(0., H, num=nz)
82  T_f = MultipleUTube.get_temperature(z, T_f_in, T_b, m_flow_borehole, cp_f)
83
84  # -----
85  # Plot fluid temperature profiles
86  # -----
87
88  # Configure figure and axes
89  fig = gt.utilities._initialize_figure()
90
91  ax1 = fig.add_subplot(111)
92  # Axis labels
93  ax1.set_xlabel(r'Temperature [degC]')
94  ax1.set_ylabel(r'Depth from borehole head [m]')
95  gt.utilities._format_axes(ax1)
96
97  # Plot temperatures
98  ax1.plot(T_f, z, 'k.')
99  ax1.plot(np.array([T_b, T_b]), np.array([0., H]), 'k--')
100  # Labels
101  calculated = mlines.Line2D([], [],
102                          color='black',

```

(continues on next page)

(continued from previous page)

```

103         ls='None',
104         marker='.',
105         label='Fluid')
106     borehole_temp = mlines.Line2D([], [],
107                                   color='black',
108                                   ls='--',
109                                   marker='None',
110                                   label='Borehole wall')
111     plt.tight_layout()
112
113     # -----
114     # Load data from Cimmino (2016)
115     # -----
116     data = np.loadtxt(filePath, skiprows=1)
117     ax1.plot(data[:,2:], data[:,0], 'b-',)
118     reference = mlines.Line2D([], [],
119                               color='blue',
120                               ls='-',
121                               lw=1.5,
122                               marker='None',
123                               label='Cimmino (2016)')
124     ax1.legend(handles=[borehole_temp, calculated, reference],
125               loc='upper left')
126
127     # Reverse y-axis
128     ax1.set_ylim(ax1.get_ylim()[::-1])
129     # Adjust to plot window
130
131     return
132
133
134 def _pipePositions(Ds, nPipes):
135     """ Positions pipes in an axisymmetric configuration.
136     """
137     dt = pi / float(nPipes)
138     pos = [(0., 0.) for i in range(2*nPipes)]
139     for i in range(nPipes):
140         pos[i] = (Ds*np.cos(2.0*i*dt+pi), Ds*np.sin(2.0*i*dt+pi))
141         pos[i+nPipes] = (Ds*np.cos(2.0*i*dt+pi+dt), Ds*np.sin(2.0*i*dt+pi+dt))
142     return pos
143
144
145 # Main function
146 if __name__ == '__main__':
147     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 Hellstrom¹. 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`

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of grout and ground temperatures using the multipole
3      method.
4
5      The thermal resistances of a borehole with two pipes are evaluated using
6      the multipole method of Claesson and Hellstrom (2011). Based on the
7      calculated thermal resistances, the heat flows from the pipes required to
8      obtain pipe temperatures of 1 degC are evaluated. The temperatures in and
9      around the borehole with 2 pipes are then calculated. Results are verified
10     against the results of Claesson and Hellstrom (2011).
11
12     """
13     import matplotlib.pyplot as plt
14     import numpy as np
15     from matplotlib.ticker import AutoMinorLocator
16     from scipy import pi
17
18     import pygfunction as gt
19
20
21     def main():
22         # -----
23         # Simulation parameters
24         # -----
25
26         # Borehole dimensions
27         r_b = 0.070          # Borehole radius (m)
28
29         # Pipe dimensions
30         n_p = 2              # Number of pipes
31         # Pipe outer radius (m)
32         rp_out = 0.02*np.ones(n_p)
33
34         # Pipe positions
35         # Single U-tube [(x_1, y_1), (x_2, y_2)]
36         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)

```

37
38 # Ground properties
39 k_s = 2.5          # Ground thermal conductivity (W/m.K)
40
41 # Grout properties
42 k_g = 1.5          # Grout thermal conductivity (W/m.K)
43
44 # Fluid properties
45 # Fluid to outer pipe wall thermal resistance (m.K/W)
46 R_fp = 1.2/(2*pi*k_g)*np.ones(n_p)
47
48 # Borehole wall temperature (degC)
49 T_b = 0.0
50
51 # Fluid temperatures (degC)
52 T_f = np.array([1., 1.])
53
54 # Path to validation data
55 filePath = './data/ClaHell11_multipole_temperature.txt'
56
57 # Thermal resistances for J=3
58 R_Claesson = 0.01*np.array([25.592, 1.561, 25.311])
59
60 # Number of multipoles per pipe
61 J = 3
62
63 # -----
64 # Evaluate the internal thermal resistances
65 # -----
66
67 # Thermal resistances
68 (R, Rd) = gt.pipes.thermal_resistances(pos_pipes, rp_out, r_b, k_s, k_g,
69                                         R_fp, J=3)
70
71 print(50*' - ')
72 print('Thermal resistance:\t\t100*R11\t100*R12\t100*R22')
73 print(f'Claesson and Hellstrom:\t{100*R_Claesson[0]:.3f}'
74       f'\t{100*R_Claesson[1]:.3f}\t{100*R_Claesson[2]:.3f}')
75 print(f'Present:\t\t\t{100*R[0,0]:.3f}\t{100*R[0,1]:.3f}'
76       f'\t{100*R[1,1]:.3f}')
77 print(50*' - ')
78
79 # Heat flows
80 Q = np.linalg.solve(R, T_f - T_b)
81
82 # -----
83 # Temperatures along y=0.
84 # -----
85
86 # Grid points to evaluate temperatures
87 x = np.linspace(-0.1, 0.1, num=200)
88 y = np.zeros_like(x)

```

(continues on next page)

(continued from previous page)

```

89  # Evaluate temperatures using multipole method
90  (T_f, T, it, eps_max) = gt.pipes.multipole(pos_pipes, rp_out, r_b, k_s,
91                                             k_g, R_fp, T_b, Q, J,
92                                             x_T=x, y_T=y)
93
94  # Load validation data
95  data = np.loadtxt(filePath, skiprows=1)
96
97  # Configure figure and axes
98  fig = gt.utilities._initialize_figure()
99
100  ax1 = fig.add_subplot(111)
101  # Axis labels
102  ax1.set_xlabel(r' $x$  (m)')
103  ax1.set_ylabel(r' $T(x,0)$ ')
104  # Axis limits
105  ax1.set_xlim([-0.1, 0.1])
106  ax1.set_ylim([-0.2, 1.2])
107  # Show grid
108  ax1.grid()
109  gt.utilities._format_axes(ax1)
110
111  ax1.plot(x, T, label='pygfunction')
112  ax1.plot(data[:,0], data[:,1], 'ko',
113          label='Claesson and Hellstrom (2011)')
114  ax1.legend(loc='upper left')
115
116  # Adjust to plot window
117  plt.tight_layout()
118
119  # -----
120  # Temperatures in  $-0.1 < x < 0.1$ ,  $-0.1 < y < 0.1$ 
121  # -----
122
123  # Grid points to evaluate temperatures
124  N_xy = 200
125  x = np.linspace(-0.1, 0.1, num=N_xy)
126  y = np.linspace(-0.1, 0.1, num=N_xy)
127  X, Y = np.meshgrid(x, y)
128
129  # Evaluate temperatures using multipole method
130  (T_f, T, it, eps_max) = gt.pipes.multipole(pos_pipes, rp_out, r_b, k_s,
131                                             k_g, R_fp, T_b, Q, J,
132                                             x_T=X.flatten(),
133                                             y_T=Y.flatten())
134
135  # Configure figure and axes
136  fig = gt.utilities._initialize_figure()
137
138  ax1 = fig.add_subplot(111)
139  # Axis labels
140  ax1.set_xlabel('x (m)')

```

(continues on next page)

(continued from previous page)

```

141     ax1.set_ylabel('y (m)')
142     # Axis limits
143     plt.axis([-0.1, 0.1, -0.1, 0.1])
144     plt.gca().set_aspect('equal', adjustable='box')
145     gt.utilities._format_axes(ax1)
146
147     # Borehole wall outline
148     borewall = plt.Circle((0., 0.), radius=r_b,
149                           fill=False, linestyle='--', linewidth=2.)
150     ax1.add_patch(borewall)
151     # Pipe outlines
152     for pos, r_out_n in zip(pos_pipes, rp_out):
153         pipe = plt.Circle(pos, radius=r_out_n,
154                           fill=False, linestyle='-', linewidth=4.)
155         ax1.add_patch(pipe)
156     # Temperature contours
157     CS = ax1.contour(X, Y, T.reshape((N_xy, N_xy)),
158                     np.linspace(-0.2, 1.0, num=7))
159     plt.clabel(CS, inline=1, fontsize=10)
160
161     # Adjust to plot window
162     plt.tight_layout()
163
164     return
165
166 # Main function
167 if __name__ == '__main__':
168     main()
169 
```

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 Cimmino¹

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 to 1.00 kg/s.

The script is located in: `pygfunction/examples/bore_field_thermal_resistance.py`

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of effective bore field thermal resistance.
3
4      The effective bore field thermal resistance of fields of up to 5 boreholes
5      of equal lengths connected in series is calculated for various fluid flow
6      rates.
7 
```

(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)

```

8  """
9  import matplotlib.pyplot as plt
10 import numpy as np
11 from matplotlib.ticker import AutoMinorLocator
12 from scipy import pi
13
14 import pygfunction as gt
15
16
17 def main():
18     # -----
19     # Simulation parameters
20     # -----
21
22     # Number of boreholes
23     nBoreholes = 5
24     # Borehole dimensions
25     D = 4.0          # Borehole buried depth (m)
26     # Borehole length (m)
27     H = 150.
28     r_b = 0.075      # Borehole radius (m)
29     B = 7.5          # Borehole spacing (m)
30
31     # Pipe dimensions
32     r_out = 0.02      # Pipe outer radius (m)
33     r_in = 0.015      # Pipe inner radius (m)
34     D_s = 0.05        # 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_pipes = [(-D_s, 0.), (D_s, 0.)]
40
41     # Ground properties
42     k_s = 2.0          # Ground thermal conductivity (W/m.K)
43
44     # Grout properties
45     k_g = 1.0          # Grout thermal conductivity (W/m.K)
46
47     # Pipe properties
48     k_p = 0.4          # Pipe thermal conductivity (W/m.K)
49
50     # Fluid properties
51     # Total fluid mass flow rate per borehole (kg/s), from 0.01 kg/s to 1 kg/s
52     m_flow_network = 10**np.arange(-2, 0.001, 0.05)
53     # The fluid is propylene-glycol (20 %) at 20 degC
54     fluid = gt.media.Fluid('MPG', 20.)
55     cp_f = fluid.cp    # Fluid specific isobaric heat capacity (J/kg.K)
56     rho_f = fluid.rho  # Fluid density (kg/m3)
57     mu_f = fluid.mu    # Fluid dynamic viscosity (kg/m.s)
58     k_f = fluid.k      # Fluid thermal conductivity (W/m.K)
59

```

(continues on next page)

(continued from previous page)

```

60 # -----
61 # Borehole field
62 # -----
63
64 boreField = []
65 bore_connectivity = []
66 for i in range(nBoreholes):
67     x = i*B
68     borehole = gt.boreholes.Borehole(H, D, r_b, x, 0.)
69     boreField.append(borehole)
70     # Boreholes are connected in series: The index of the upstream
71     # borehole is that of the previous borehole
72     bore_connectivity.append(i - 1)
73
74 # -----
75 # Evaluate the effective bore field thermal resistance
76 # -----
77
78 # Initialize result array
79 R = np.zeros((nBoreholes, len(m_flow_network)))
80 for i in range(nBoreholes):
81     for j, m_flow_network_j in enumerate(m_flow_network):
82         nBoreholes = i + 1
83         # Boreholes are connected in series
84         m_flow_borehole = m_flow_network_j
85         # Boreholes are single U-tube
86         m_flow_pipe = m_flow_borehole
87
88         # Pipe thermal resistance
89         R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
90             r_in, r_out, k_p)
91         # Fluid to inner pipe wall thermal resistance (Single U-tube)
92         h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
93             m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
94         R_f = 1.0/(h_f*2*pi*r_in)
95
96         # Single U-tube, same for all boreholes in the bore field
97         UTubes = []
98         for borehole in boreField:
99             SingleUTube = gt.pipes.SingleUTube(
100                 pos_pipes, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
101             UTubes.append(SingleUTube)
102         network = gt.networks.Network(
103             boreField[:nBoreholes],
104             UTubes[:nBoreholes],
105             bore_connectivity=bore_connectivity[:nBoreholes])
106
107         # Effective bore field thermal resistance
108         R_field = gt.networks.network_thermal_resistance(
109             network, m_flow_network_j, cp_f)
110         # Add to result array
111         R[i,j] = R_field

```

(continues on next page)

(continued from previous page)

```

112 # -----
113 # Plot bore field thermal resistances
114 # -----
115
116
117 # Configure figure and axes
118 fig = gt.utilities._initialize_figure()
119
120 ax1 = fig.add_subplot(111)
121 # Axis labels
122 ax1.set_xlabel(r'$\dot{m}$ [kg/s]')
123 ax1.set_ylabel(r'$R^*_{field}$ [m.K/W]')
124 # Axis limits
125 ax1.set_xlim([0., 1.])
126 ax1.set_ylim([0., 1.])
127
128 gt.utilities._format_axes(ax1)
129
130 # Bore field thermal resistances
131 ax1.plot(m_flow_network, R[0,:], '-', label='1 borehole')
132 ax1.plot(m_flow_network, R[2,:], '--', label='3 boreholes')
133 ax1.plot(m_flow_network, R[4,:], '-.', label='5 boreholes')
134 ax1.legend()
135 # Adjust to plot window
136 plt.tight_layout()
137
138 return
139
140
141 # Main function
142 if __name__ == '__main__':
143     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 Cimmino¹. 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 connections 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.

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of g-functions using mixed inlet temperatures.
3
4      The g-functions of a field of 5 boreholes of different lengths connected
5      in series are calculated for 2 boundary conditions: (a) uniform borehole
6      wall temperature, and (b) series connections between boreholes. The
7      g-function for case (b) is based on the effective borehole wall
8      temperature, rather than the average borehole wall temperature.
9
10 """
11 import matplotlib.pyplot as plt
12 import numpy as np
13 from matplotlib.ticker import AutoMinorLocator
14 from scipy import pi
15
16 import pygfunction as gt
17
18 def main():
19     # -----
20     # Simulation parameters
21     # -----
22
23     # Borehole dimensions
24     D = 4.0          # Borehole buried depth (m)
25     # Borehole length (m)
26     H_boreholes = np.array([75.0, 100.0, 125.0, 150.0, 75.0])
27     H_mean = np.mean(H_boreholes)
28     r_b = 0.075      # Borehole radius (m)
29     B = 7.5          # Borehole spacing (m)
30
31     # Pipe dimensions
32     r_out = 0.02      # Pipe outer radius (m)
33     r_in = 0.015      # Pipe inner radius (m)
34     D_s = 0.05        # 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_pipes = [(-D_s, 0.), (D_s, 0.)]
40
41     # Ground properties
42     alpha = 1.0e-6    # Ground thermal diffusivity (m2/s)
43     k_s = 2.0         # Ground thermal conductivity (W/m.K)
44
45     # Grout properties
46     k_g = 1.0         # Grout thermal conductivity (W/m.K)
47
48     # Pipe properties
49     k_p = 0.4         # Pipe thermal conductivity (W/m.K)
50
51     # Fluid properties
52     m_flow_network = 0.25 # Total fluid mass flow rate in network (kg/s)
53

```

(continues on next page)

(continued from previous page)

```

54 # All boreholes are in series
55 m_flow_borehole = m_flow_network
56 # The fluid is propylene-glycol (20 %) at 20 degC
57 fluid = gt.media.Fluid('MPG', 20.)
58 cp_f = fluid.cp      # Fluid specific isobaric heat capacity (J/kg.K)
59 rho_f = fluid.rho    # Fluid density (kg/m3)
60 mu_f = fluid.mu      # Fluid dynamic viscosity (kg/m.s)
61 k_f = fluid.k        # Fluid thermal conductivity (W/m.K)
62
63 # g-Function calculation options
64 nSegments = 8
65 options = {'nSegments': nSegments,
66           'disp': True,
67           'profiles': True}
68 # The similarities method is used since the 'equivalent' method does not
69 # apply if boreholes are connected in series
70 method = 'similarities'
71
72 # Geometrically expanding time vector.
73 dt = 100*3600.        # Time step
74 tmax = 3000. * 8760. * 3600. # Maximum time
75 Nt = 25               # Number of time steps
76 ts = H_mean**2/(9.*alpha) # Bore field characteristic time
77 time = gt.utilities.time_geometric(dt, tmax, Nt)
78
79 # -----
80 # Borehole field
81 # -----
82
83 boreField = []
84 bore_connectivity = []
85 for i, H in enumerate(H_boreholes):
86     x = i*B
87     borehole = gt.boreholes.Borehole(H, D, r_b, x, 0.)
88     boreField.append(borehole)
89     # Boreholes are connected in series: The index of the upstream
90     # borehole is that of the previous borehole
91     bore_connectivity.append(i - 1)
92
93 # -----
94 # Initialize pipe model
95 # -----
96
97 # Pipe thermal resistance
98 R_p = gt.pipes.conduction_thermal_resistance_circular_pipe(
99     r_in, r_out, k_p)
100 # Fluid to inner pipe wall thermal resistance (Single U-tube)
101 m_flow_pipe = m_flow_borehole
102 h_f = gt.pipes.convective_heat_transfer_coefficient_circular_pipe(
103     m_flow_pipe, r_in, mu_f, rho_f, k_f, cp_f, epsilon)
104 R_f = 1.0/(h_f*2*pi*r_in)
105

```

(continues on next page)

(continued from previous page)

```

106     # Single U-tube, same for all boreholes in the bore field
107     UTubes = []
108     for borehole in boreField:
109         SingleUTube = gt.pipes.SingleUTube(
110             pos_pipes, r_in, r_out, borehole, k_s, k_g, R_f + R_p)
111         UTubes.append(SingleUTube)
112     network = gt.networks.Network(
113         boreField, UTubes, bore_connectivity=bore_connectivity,
114         m_flow_network=m_flow_network, cp_f=cp_f, nSegments=nSegments)
115
116     # -----
117     # Evaluate the g-functions for the borefield
118     # -----
119
120     # Calculate the g-function for uniform temperature
121     gfunc_Tb = gt.gfunction.gFunction(
122         boreField, alpha, time=time, boundary_condition='UBWT',
123         options=options, method=method)
124
125     # Calculate the g-function for mixed inlet fluid conditions
126     gfunc_equal_Tf_mixed = gt.gfunction.gFunction(
127         network, alpha, time=time, boundary_condition='MIFT', options=options,
128         method=method)
129
130     # -----
131     # Plot g-functions
132     # -----
133
134     ax = gfunc_Tb.visualize_g_function().axes[0]
135     ax.plot(np.log(time/ts), gfunc_equal_Tf_mixed.gFunc, 'r-.')
136     ax.legend(['Uniform temperature', 'Mixed inlet temperature'])
137     plt.tight_layout()
138
139     # For the mixed inlet fluid temperature condition, draw the temperatures
140     # and heat extraction rates
141     gfunc_equal_Tf_mixed.visualize_temperatures()
142     gfunc_equal_Tf_mixed.visualize_temperature_profiles()
143     gfunc_equal_Tf_mixed.visualize_heat_extraction_rates()
144     gfunc_equal_Tf_mixed.visualize_heat_extraction_rate_profiles()
145
146     return
147
148
149 # Main function
150 if __name__ == '__main__':
151     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 Eskilson¹. 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*

```

1  # -*- coding: utf-8 -*-
2  """ Example of calculation of g-functions with inclined boreholes using uniform
3      and equal borehole wall temperatures.
4
5      The g-functions of two fields of 8 boreholes are calculated for boundary
6      condition of uniform borehole wall temperature along the boreholes, equal
7      for all boreholes. The first field corresponds to the "optimum"
8      configuration presented by Claesson and Eskilson (1987) and the second
9      field corresponds to the configuration comprised of 8 boreholes in a
10     circle.
11
12     Claesson J, and Eskilson, P. (1987). Conductive heat extraction by
13     thermally interacting deep boreholes, in "Thermal analysis of heat
14     extraction boreholes". Ph.D. Thesis, University of Lund, Lund, Sweden.
15 """
16
17 import pygfunction as gt
18 import numpy as np
19 import matplotlib.pyplot as plt
20
21
22 def main():
23     # -----
24     # Simulation parameters
25     # -----
26
27     # Borehole dimensions
28     D = 4.0          # Borehole buried depth (m)
29     H = 150.0        # Borehole length (m)
30     r_b = 0.075      # Borehole radius (m)
31     tilt = np.radians(20.) # Borehole inclination (rad)
32
33     # Thermal properties
34     alpha = 1.0e-6    # Ground thermal diffusivity (m2/s)
35
36     # g-Function calculation options
37     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)

```

38
39 # Geometrically expanding time vector.
40 dt = 100*3600.          # Time step
41 tmax = 3000. * 8760. * 3600. # Maximum time
42 Nt = 25                 # Number of time steps
43 ts = H**2/(9.*alpha)    # Bore field characteristic time
44 time = gt.utilities.time_geometric(dt, tmax, Nt)
45 lntts = np.log(time/ts)
46
47 # -----
48 # Borehole fields
49 # -----
50 """
51 Bore field #1
52
53 This field corresponds to the optimal configuration presented by
54 Claesson and Eskilson (1987). The field is built using the `cardinal_point`
55 function to define the orientation of each borehole, individually.
56 """
57 B = 7.5 # Borehole spacing (m)
58 # Orientation of the boreholes
59 borehole_orientations = [
60     gt.utilities.cardinal_point('W'),
61     gt.utilities.cardinal_point('NW'),
62     gt.utilities.cardinal_point('SW'),
63     gt.utilities.cardinal_point('N'),
64     gt.utilities.cardinal_point('S'),
65     gt.utilities.cardinal_point('NE'),
66     gt.utilities.cardinal_point('SE'),
67     gt.utilities.cardinal_point('E')]
68
69 # "Optimal" field of 8 boreholes
70 boreField1 = []
71 for i, orientation in enumerate(borehole_orientations):
72     borehole = gt.boreholes.Borehole(
73         H, D, r_b, i * B, 0., tilt=tilt, orientation=orientation)
74     boreField1.append(borehole)
75
76 # Visualize the borehole field
77 fig1 = gt.boreholes.visualize_field(boreField1)
78
79 """
80 Bore field #2
81
82 This field corresponds to the configuration comprised of 8 boreholes in
83 a circle presented by Claesson and Eskilson (1987). The field is built
84 using the `circle_field` function.
85 """
86 N = 8 # Number of boreholes
87 R = 3. # Borehole spacing from the center of the field (m)
88
89 # Field of 6 boreholes in a circle

```

(continues on next page)

(continued from previous page)

```

90 boreField2 = gt.boreholes.circle_field(N, R, H, D, r_b, tilt=tilt)
91
92 # Visualize the borehole field
93 fig2 = gt.boreholes.visualize_field(boreField2)
94
95 # -----
96 # Evaluate g-functions for all fields
97 # -----
98 # Bore field #1
99 gfunc1 = gt.gfunction.gFunction(
100     boreField1, alpha, time=time, options=options, method='similarities')
101 fig3 = gfunc1.visualize_g_function()
102 fig3.suptitle('Optimal" field of 8 boreholes')
103 fig3.tight_layout()
104 # Bore field #2
105 gfunc2 = gt.gfunction.gFunction(
106     boreField2, alpha, time=time, options=options, method='similarities')
107 fig4 = gfunc2.visualize_g_function()
108 fig4.suptitle(f'Field of {N} boreholes in a circle')
109 fig4.tight_layout()
110
111
112 # Main function
113 if __name__ == '__main__':
114     main()

```

References

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/m ³	Density of the fluid
mu	Pa.s or N.s/m ²	Dynamic viscosity
nu	m ² /s	Kinematic viscosity
cp	J/kg.K	Specific isobaric heat capacity
k	W/m.K	Thermal conductivity
rhoCp	J/m ³ .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	m ² /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

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- `pygfunction.boreholes`, 7
- `pygfunction.gfunction`, 15
- `pygfunction.heat_transfer`, 24
- `pygfunction.load_aggregation`, 34
- `pygfunction.media`, 39
- `pygfunction.networks`, 41
- `pygfunction.pipes`, 51
- `pygfunction.utilities`, 87

A

`append_to_dict()` (*pygfunction.media.Fluid* method), 40

B

`Borehole` (class in *pygfunction.boreholes*), 7
`borehole_thermal_resistance()` (in module *pygfunction.pipes*), 81
`box_shaped_field()` (in module *pygfunction.boreholes*), 10

C

`cardinal_point()` (in module *pygfunction.utilities*), 87
`circle_field()` (in module *pygfunction.boreholes*), 11
`ClaessonJaved` (class in *pygfunction.load_aggregation*), 34
`Coaxial` (class in *pygfunction.pipes*), 51
`coefficients_borehole_heat_extraction_rate()` (*pygfunction.networks.Network* method), 42
`coefficients_borehole_heat_extraction_rate()` (*pygfunction.pipes.Coaxial* method), 52
`coefficients_borehole_heat_extraction_rate()` (*pygfunction.pipes.IndependentMultipleUTube* method), 59
`coefficients_borehole_heat_extraction_rate()` (*pygfunction.pipes.MultipleUTube* method), 67
`coefficients_borehole_heat_extraction_rate()` (*pygfunction.pipes.SingleUTube* method), 75
`coefficients_fluid_heat_extraction_rate()` (*pygfunction.networks.Network* method), 43
`coefficients_fluid_heat_extraction_rate()` (*pygfunction.pipes.Coaxial* method), 52
`coefficients_fluid_heat_extraction_rate()` (*pygfunction.pipes.IndependentMultipleUTube* method), 60
`coefficients_fluid_heat_extraction_rate()` (*pygfunction.pipes.MultipleUTube* method), 68
`coefficients_fluid_heat_extraction_rate()` (*pygfunction.pipes.SingleUTube* method), 75
`coefficients_inlet_temperature()` (*pygfunction.networks.Network* method), 43

`coefficients_inlet_temperature()` (*pygfunction.pipes.Coaxial* method), 53
`coefficients_inlet_temperature()` (*pygfunction.pipes.IndependentMultipleUTube* method), 61
`coefficients_inlet_temperature()` (*pygfunction.pipes.MultipleUTube* method), 68
`coefficients_inlet_temperature()` (*pygfunction.pipes.SingleUTube* method), 76
`coefficients_network_heat_extraction_rate()` (*pygfunction.networks.Network* method), 44
`coefficients_network_inlet_temperature()` (*pygfunction.networks.Network* method), 45
`coefficients_network_outlet_temperature()` (*pygfunction.networks.Network* method), 45
`coefficients_outlet_temperature()` (*pygfunction.networks.Network* method), 46
`coefficients_outlet_temperature()` (*pygfunction.pipes.Coaxial* method), 53
`coefficients_outlet_temperature()` (*pygfunction.pipes.IndependentMultipleUTube* method), 61
`coefficients_outlet_temperature()` (*pygfunction.pipes.MultipleUTube* method), 69
`coefficients_outlet_temperature()` (*pygfunction.pipes.SingleUTube* method), 77
`coefficients_temperature()` (*pygfunction.pipes.Coaxial* method), 54
`coefficients_temperature()` (*pygfunction.pipes.IndependentMultipleUTube* method), 62
`coefficients_temperature()` (*pygfunction.pipes.MultipleUTube* method), 69
`coefficients_temperature()` (*pygfunction.pipes.SingleUTube* method), 77
`conduction_thermal_resistance_circular_pipe()` (in module *pygfunction.pipes*), 82
`convective_heat_transfer_coefficient_circular_pipe()` (in module *pygfunction.pipes*), 82
`convective_heat_transfer_coefficient_concentric_annulus()` (in module *pygfunction.pipes*), 83

D

`density()` (*pygfunction.media.Fluid* method), 40
`distance()` (*pygfunction.boreholes.Borehole* method), 7
`dynamic_viscosity()` (*pygfunction.media.Fluid* method), 40

E

`effective_borehole_thermal_resistance()` (*pygfunction.pipes.Coaxial* method), 55
`effective_borehole_thermal_resistance()` (*pygfunction.pipes.IndependentMultipleUTube* method), 62
`effective_borehole_thermal_resistance()` (*pygfunction.pipes.MultipleUTube* method), 70
`effective_borehole_thermal_resistance()` (*pygfunction.pipes.SingleUTube* method), 78
`equal_inlet_temperature()` (in module *pygfunction.gfunction*), 15
`erfint()` (in module *pygfunction.utilities*), 87
`evaluate_g_function()` (*pygfunction.gfunction.gFunction* method), 19
`exp1()` (in module *pygfunction.utilities*), 87

F

`field_from_file()` (in module *pygfunction.boreholes*), 12
`find_duplicates()` (in module *pygfunction.boreholes*), 13
`finite_line_source()` (in module *pygfunction.heat_transfer*), 24
`finite_line_source_approximation()` (in module *pygfunction.heat_transfer*), 27
`finite_line_source_equivalent_boreholes_vectorized()` (in module *pygfunction.heat_transfer*), 28
`finite_line_source_inclined_approximation()` (in module *pygfunction.heat_transfer*), 29
`finite_line_source_inclined_vectorized()` (in module *pygfunction.heat_transfer*), 31
`finite_line_source_vectorized()` (in module *pygfunction.heat_transfer*), 33
`Fluid` (class in *pygfunction.media*), 39
`fluid_friction_factor_circular_pipe()` (in module *pygfunction.pipes*), 84

G

`get_borehole_heat_extraction_rate()` (*pygfunction.networks.Network* method), 46
`get_borehole_heat_extraction_rate()` (*pygfunction.pipes.Coaxial* method), 55
`get_borehole_heat_extraction_rate()` (*pygfunction.pipes.IndependentMultipleUTube* method), 63
`get_borehole_heat_extraction_rate()` (*pygfunction.pipes.MultipleUTube* method), 70

`get_borehole_heat_extraction_rate()` (*pygfunction.pipes.SingleUTube* method), 78
`get_fluid_heat_extraction_rate()` (*pygfunction.networks.Network* method), 47
`get_fluid_heat_extraction_rate()` (*pygfunction.pipes.Coaxial* method), 55
`get_fluid_heat_extraction_rate()` (*pygfunction.pipes.IndependentMultipleUTube* method), 63
`get_fluid_heat_extraction_rate()` (*pygfunction.pipes.MultipleUTube* method), 71
`get_fluid_heat_extraction_rate()` (*pygfunction.pipes.SingleUTube* method), 79
`get_inlet_temperature()` (*pygfunction.networks.Network* method), 47
`get_inlet_temperature()` (*pygfunction.pipes.Coaxial* method), 56
`get_inlet_temperature()` (*pygfunction.pipes.IndependentMultipleUTube* method), 64
`get_inlet_temperature()` (*pygfunction.pipes.MultipleUTube* method), 71
`get_inlet_temperature()` (*pygfunction.pipes.SingleUTube* method), 79
`get_network_heat_extraction_rate()` (*pygfunction.networks.Network* method), 48
`get_network_inlet_temperature()` (*pygfunction.networks.Network* method), 49
`get_network_outlet_temperature()` (*pygfunction.networks.Network* method), 49
`get_outlet_temperature()` (*pygfunction.networks.Network* method), 50
`get_outlet_temperature()` (*pygfunction.pipes.Coaxial* method), 56
`get_outlet_temperature()` (*pygfunction.pipes.IndependentMultipleUTube* method), 64
`get_outlet_temperature()` (*pygfunction.pipes.MultipleUTube* method), 72
`get_outlet_temperature()` (*pygfunction.pipes.SingleUTube* method), 80
`get_temperature()` (*pygfunction.pipes.Coaxial* method), 57
`get_temperature()` (*pygfunction.pipes.IndependentMultipleUTube* method), 65
`get_temperature()` (*pygfunction.pipes.MultipleUTube* method), 72
`get_temperature()` (*pygfunction.pipes.SingleUTube* method), 80
`get_thermal_response_factor_increment()` (*pygfunction.load_aggregation.ClaessonJaved* method), 35
`get_times_for_simulation()` (*pygfunction*

- tion.load_aggregation.ClaessonJaved method*), 35
- get_times_for_simulation()* (*pygfunction.load_aggregation.Liu method*), 36
- get_times_for_simulation()* (*pygfunction.load_aggregation.MLAA method*), 38
- get_total_heat_extraction_rate()* (*pygfunction.pipes.Coaxial method*), 57
- get_total_heat_extraction_rate()* (*pygfunction.pipes.IndependentMultipleUTube method*), 65
- get_total_heat_extraction_rate()* (*pygfunction.pipes.MultipleUTube method*), 73
- get_total_heat_extraction_rate()* (*pygfunction.pipes.SingleUTube method*), 80
- gFunction* (*class in pygfunction.gfunction*), 16
- ## I
- IndependentMultipleUTube* (*class in pygfunction.pipes*), 58
- initialize()* (*pygfunction.load_aggregation.ClaessonJaved method*), 35
- initialize()* (*pygfunction.load_aggregation.Liu method*), 37
- initialize()* (*pygfunction.load_aggregation.MLAA method*), 38
- is_tilted()* (*pygfunction.boreholes.Borehole method*), 8
- is_vertical()* (*pygfunction.boreholes.Borehole method*), 8
- ## K
- kinematic_viscosity()* (*pygfunction.media.Fluid method*), 40
- ## L
- L_shaped_field()* (*in module pygfunction.boreholes*), 9
- Liu* (*class in pygfunction.load_aggregation*), 36
- local_borehole_thermal_resistance()* (*pygfunction.pipes.Coaxial method*), 58
- local_borehole_thermal_resistance()* (*pygfunction.pipes.IndependentMultipleUTube method*), 65
- local_borehole_thermal_resistance()* (*pygfunction.pipes.MultipleUTube method*), 73
- local_borehole_thermal_resistance()* (*pygfunction.pipes.SingleUTube method*), 81
- ## M
- mixed_inlet_temperature()* (*in module pygfunction.gfunction*), 20
- MLAA* (*class in pygfunction.load_aggregation*), 37
- module
- pygfunction.boreholes*, 7
 - pygfunction.gfunction*, 15
 - pygfunction.heat_transfer*, 24
 - pygfunction.load_aggregation*, 34
 - pygfunction.media*, 39
 - pygfunction.networks*, 41
 - pygfunction.pipes*, 51
 - pygfunction.utilities*, 87
- MultipleUTube* (*class in pygfunction.pipes*), 66
- multipole()* (*in module pygfunction.pipes*), 85
- ## N
- Network* (*class in pygfunction.networks*), 41
- network_thermal_resistance()* (*in module pygfunction.networks*), 50
- next_time_step()* (*pygfunction.load_aggregation.ClaessonJaved method*), 35
- next_time_step()* (*pygfunction.load_aggregation.Liu method*), 37
- next_time_step()* (*pygfunction.load_aggregation.MLAA method*), 38
- ## P
- position()* (*pygfunction.boreholes.Borehole method*), 8
- Prandtl_number()* (*pygfunction.media.Fluid method*), 40
- pygfunction.boreholes* module, 7
 - pygfunction.gfunction* module, 15
 - pygfunction.heat_transfer* module, 24
 - pygfunction.load_aggregation* module, 34
 - pygfunction.media* module, 39
 - pygfunction.networks* module, 41
 - pygfunction.pipes* module, 51
 - pygfunction.utilities* module, 87
- ## R
- rectangle_field()* (*in module pygfunction.boreholes*), 13
- remove_duplicates()* (*in module pygfunction.boreholes*), 14
- ## S
- segment_ratios()* (*in module pygfunction.utilities*), 88

`segments()` (*pygfunction.boreholes.Borehole method*), 8
`set_current_load()` (*pygfunction.load_aggregation.ClaessonJaved method*), 35
`set_current_load()` (*pygfunction.load_aggregation.Liu method*), 37
`set_current_load()` (*pygfunction.load_aggregation.MLAA method*), 38
`SingleUTube` (class in *pygfunction.pipes*), 74
`specific_heat_capacity()` (*pygfunction.media.Fluid method*), 40

T

`temporal_superposition()` (*pygfunction.load_aggregation.ClaessonJaved method*), 36
`temporal_superposition()` (*pygfunction.load_aggregation.Liu method*), 37
`temporal_superposition()` (*pygfunction.load_aggregation.MLAA method*), 38
`thermal_conductivity()` (*pygfunction.media.Fluid method*), 41
`thermal_resistances()` (in module *pygfunction.pipes*), 86
`time_ClaessonJaved()` (in module *pygfunction.utilities*), 88
`time_geometric()` (in module *pygfunction.utilities*), 89
`time_MarcottePasquier()` (in module *pygfunction.utilities*), 89
`visualize_heat_extraction_rates()` (*pygfunction.gfunction.gFunction method*), 19
`visualize_pipes()` (*pygfunction.pipes.Coaxial method*), 58
`visualize_pipes()` (*pygfunction.pipes.IndependentMultipleUTube method*), 66
`visualize_pipes()` (*pygfunction.pipes.MultipleUTube method*), 74
`visualize_pipes()` (*pygfunction.pipes.SingleUTube method*), 81
`visualize_temperature_profiles()` (*pygfunction.gfunction.gFunction method*), 20
`visualize_temperatures()` (*pygfunction.gfunction.gFunction method*), 20
`volumetric_heat_capacity()` (*pygfunction.media.Fluid method*), 41

U

`U_shaped_field()` (in module *pygfunction.boreholes*), 9
`uniform_heat_extraction()` (in module *pygfunction.gfunction*), 22
`uniform_temperature()` (in module *pygfunction.gfunction*), 23
`update_thermal_resistances()` (*pygfunction.pipes.Coaxial method*), 58
`update_thermal_resistances()` (*pygfunction.pipes.IndependentMultipleUTube method*), 66
`update_thermal_resistances()` (*pygfunction.pipes.MultipleUTube method*), 73
`update_thermal_resistances()` (*pygfunction.pipes.SingleUTube method*), 81

V

`visualize_field()` (in module *pygfunction.boreholes*), 14
`visualize_g_function()` (*pygfunction.gfunction.gFunction method*), 19
`visualize_heat_extraction_rate_profiles()` (*pygfunction.gfunction.gFunction method*), 19