
GHEtool

Release 2.2.1

Wouter Peere and Tobias Blanke

Apr 11, 2024

GHETOOL

1	Graphical user interface	3
	Python Module Index	115
	Index	117

Dear visitor

Welcome on this ReadTheDocs for the open-source borefield sizing tool GHEtool.

Here one can find (extensive) documentation of GHEtool codebase. Please find below the content of this ReadTheDocs.

GRAPHICAL USER INTERFACE

There are two graphical user interfaces available which are built using GHETool: GHETool Pro and GHETool Community

1.1 GHETool Pro

GHETool Pro is the official and supported version of GHETool which supports drilling companies, engineering firms, architects, government organizations in their geothermal design process. With GHETool Pro they can minimize the environmental and societal impact while maximizing the cost-effective utilization of geothermal projects. Visit our website at <https://ghetool.eu> to learn more about the synergy between this open-source package and GHETool Pro.

The user manual for GHETool Pro, together with instructions for installation, you can find [here](#).



1.2 GHEtool Community

Besides GHEtool Pro, an open-source alternative for the graphical user interface is available in the form of *GHEtool Community*. This version is built and maintained by the community, and **has no official support like GHEtool Pro**. You can read all about this *GHEtool Community* on their [GitHub Repo](#).

1.2.1 Cite GHEtool

General citation

Whenever you work with GHEtool (in general), please reference this tool using the JOSS paper.

Peere, W., Blanke, T.(2022). GHEtool: An open-source tool for borefield sizing in Python. *Journal of Open Source Software*, 7(76), 4406, <https://doi.org/10.21105/joss.04406>

Whenever you use a specific functionality within GHEtool, one can also reference a specific paper/article related to this functionality.

Hybrid sizing methodology (L2) | borefield quadrants

Whenever you use the hybrid sizing methodology (L2 in GHEtool) or make use of borefield quadrants, please use the citations below.

Peere, W., Picard, D., Cupeiro Figueroa, I., Boydens, W., and Helsen, L. (2021). Validated combined first and last year borefield sizing methodology. In *Proceedings of International Building Simulation Conference 2021*. Brugge (Belgium), 1-3 September 2021. <https://doi.org/10.26868/25222708.2021.30180>

Peere, W. (2020). Methode voor economische optimalisatie van geothermische verwarmings- en koelsystemen. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.

Active-passive cooling example

Whenever you use the example of active-passive cooling, please use the citations below.

Coninx, M., De Nies, J., Hermans, L., Peere, W., Boydens, W., Helsen, L. (2024). Cost-efficient cooling of buildings by means of geothermal borefields with active and passive cooling. *Applied Energy*, 355, Art. No. 122261, <https://doi.org/10.1016/j.apenergy.2023.122261>.

1.2.2 GHEtool in literature

Below is a general list of all the articles which use or mention GHEtool in literature. Please let us know if we missed a contribution.

Note that for some of the publications, the code used in that publication, is available in this ReadTheDocs. In this way, we want to contribute to the transparency in the academic world by sharing computer code so it is easier to replicate and verify published results.

The articles for which this is the case, are:

Verleyen et al. (2022)

The code below is used in the article of Verleyen et al. (2022)¹.

```

1  """
2  This document contains all the scripts for the figures of Verleyen et al. (2022).
3  One needs GHEtool version 2.0.6 to run this code.
4  """
5  import math
6
7  import matplotlib.pyplot as plt
8  import numpy as np
9
10 # code for making figures black-and-white ready
11 from cycler import cycler
12
13 from GHEtool import Borefield, GroundData
14
15 color_c = cycler('color', ['k'])
16 style_c = cycler('linestyle', ['-', '--', ':', '-.'])
17 markr_c = cycler('marker', ['', '.', 'o'])
18 c_cms = color_c * markr_c * style_c
19 c_csm = color_c * style_c * markr_c
20 plt.rc('axes', prop_cycle=c_cms)
21 plt.rcParams['lines.markersize'] = 8
22
23
24 def figure_1():
25     """
26     This function generates the first figure (part a/b) of the article.
27     """
28
29     # initiate ground data
30     ground_data = GroundData(2.4, 10, 0.12)
31
32     # initiate borefield model
33     borefield = Borefield()
34     borefield.set_ground_parameters(ground_data)
35     borefield.create_rectangular_borefield(10, 10, 6, 6, 100, 4)
36
37     # initiate depth array
38     depth = 150
39
40     # dimensionless time
41     ts = 150**2 / (9 * ground_data.k_s)
42
43     # time array
44     nb_of_timesteps = 50
45     time_dimensionless = np.linspace(2, 14, nb_of_timesteps)
46
47     # convert to seconds

```

(continues on next page)

¹ Verleyen, L., Peere, W., Michiels, E., Boydens, W., Helsen, L. (2022). The beauty of reason and insight: a story about 30 years old borefield equations. IEA HPT Magazine 40(3), 36-39, <https://doi.org/10.23697/6q4n-3223>

(continued from previous page)

```

48     time_in_seconds = np.exp(time_dimensionless) * ts
49
50     # calculate g-functions
51     result = np.zeros(nb_of_timesteps)
52     result = borefield.gfunction(time_in_seconds, depth)
53
54     # create figure
55     fig, axs = plt.subplots(1, 2, figsize=(10, 3), constrained_layout=True)
56
57     # create figure g-function (lin)
58     # plot g-functions
59     axs[0].plot(time_in_seconds/8760/3600, result / (2 * math.pi * 2.4))
60
61     # layout
62     axs[0].set_title("Step response when applying a constant heat injection")
63     axs[0].set_xlabel("Time (years)")
64     axs[0].set_ylabel("Temperature difference (K)")
65     axs[0].set_ylim(0, 5)
66     axs[0].set_xlim(0, 40)
67
68     # create figure g-function (semi-log)
69     # plot g-functions
70     axs[1].plot(time_dimensionless, result)
71
72     # layout
73     axs[1].set_title("Equivalent g-function for the step response")
74     axs[1].set_xlabel("ln(t/ts)")
75     axs[1].set_ylabel("g-function value")
76     axs[1].set_ylim(-2, 60)
77
78     # plt.legend()
79     plt.show()
80
81
82 def figure_2():
83     """
84     This function generates the second figure of the article.
85     """
86
87     # initiate ground data
88     ground_data = GroundData(2.4, 10, 0.12)
89
90     # initiate borefield model
91     borefield = Borefield()
92     borefield.set_ground_parameters(ground_data)
93     borefield.create_rectangular_borefield(10, 10, 6, 6, 100, 4)
94
95     # initiate depth array
96     depths = np.array([25, 50, 100, 150, 200])
97
98     # dimensionless time
99     ts = 150**2 / (9 * ground_data.k_s)

```

(continues on next page)

(continued from previous page)

```

100
101 # time array
102 nb_of_timesteps = 50
103 time_dimensionless = np.linspace(2, 14, nb_of_timesteps)
104
105 # convert to seconds
106 time_in_seconds = np.exp(time_dimensionless) * ts
107
108 # calculate g-functions
109 results = np.zeros((5, nb_of_timesteps))
110 for i in range(5):
111     results[i] = borefield.gfunction(time_in_seconds, depths[i])
112
113 # create figure
114 plt.figure()
115
116 # plot g-functions
117 plt.plot(time_dimensionless, results[0], label="25m")
118 plt.plot(time_dimensionless, results[1], label="50m")
119 plt.plot(time_dimensionless, results[2], label="100m")
120 plt.plot(time_dimensionless, results[3], label="150m")
121 plt.plot(time_dimensionless, results[4], label="200m")
122
123 # plot lines for Ra
124 line1 = math.log(6*3600/ts)
125 line2 = math.log((20 * 8760 + 730) * 3600 / ts)
126 plt.vlines(line1, ymin=-5, ymax=5, colors="black", lw=0.75, ls="--")
127 plt.vlines(line2, ymin=-5, ymax=50, colors="black", lw=0.75, ls="--")
128
129 plt.annotate('', xy=(line1, -0.4), xytext=(line2, -0.4),
130             arrowprops=dict(arrowstyle='<->', color='black'))
131 plt.text((line1+line2)/2, 0.2, "Ra", horizontalalignment='center')
132
133 # layout
134 # plt.title("G-function values for different borefield depths")
135 plt.xlabel("ln(t/ts)")
136 plt.ylabel("g-function value")
137 plt.ylim(-2, 60)
138
139 plt.legend()
140 plt.show()
141
142
143 def figure_3():
144     """
145     This function creates the third figure of the article.
146     """
147
148     # initiate borefield model
149     borefield = Borefield()
150
151     # initiate depth for evaluations

```

(continues on next page)

(continued from previous page)

```

152     nb_depths = 20
153     depth_array = np.linspace(50, 350, nb_depths)
154
155     # initiate list of borefield configurations
156     configs = [(10, 10), (11, 11), (12, 12), (14, 14),
157               (15, 15), (18, 18), (20, 20)]
158
159     results = []
160
161     for n1, n2 in configs:
162         depths = []
163         for H in depth_array:
164             # set ground data
165             ground_data = GroundData(2.4, 10, 0.12)
166
167             # set ground data
168             borefield.set_ground_parameters(ground_data)
169
170             # set borefield
171             borefield.create_rectangular_borefield(n1, n2, 7, 7, H, 4)
172
173             # calculate gfunction
174             gfunction = borefield.gfunction(borefield.time, H)
175
176             # calculate Ra
177             Ra = (gfunction[2] - gfunction[1]) / (2 * math.pi * ground_data.k_s)
178
179             # add to depths
180             depths.append(Ra)
181
182             # add to results
183             results.append(depths)
184
185     # create figure
186     plt.figure()
187     for i, config in enumerate(configs):
188         plt.plot(depth_array, results[i], label=str(config[0]) + "x" + str(config[1]))
189
190     # plt.title("Ra for different borefield configurations")
191     plt.xlabel("Depth (m)")
192     plt.ylabel("Ra (mK/W)")
193
194     plt.legend()
195     plt.show()
196
197
198 def figure_4():
199     """
200     This code creates the fourth figure of the article
201     """
202     # initiate borefield
203     borefield = Borefield()

```

(continues on next page)

(continued from previous page)

```

204
205     # set the correct sizing method
206     borefield.sizing_setup(L2_sizing=True)
207
208     # initiate array with imbalances
209     imbalance_array = np.linspace(200, 1600, 20)
210
211     # initiate list of borefield configurations
212     configs = [(10, 10), (11, 11), (12, 12), (14, 14),
213               (15, 15), (18, 18), (20, 20)]
214
215     # initiate loads
216     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
217 ↪ 0., 0.061, 0.087, .117, 0.144])
218     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
219 ↪ .1, .075, .05, .025])
220     monthly_load_heating = monthly_load_heating_percentage * 100 * 10 ** 3 # kWh
221     monthly_load_cooling_init = monthly_load_cooling_percentage * 100 * 10 ** 3 # kWh
222     peak_cooling_init = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0.,
223 ↪ 0.]) # Peak cooling in kW
224     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
225 ↪ ])
226
227     # set heating loads
228     borefield.set_peak_heating(peak_heating)
229     borefield.set_baseload_heating(monthly_load_heating)
230
231     results = []
232     for i, config in enumerate(configs):
233         depth_array = []
234         for imbalance in imbalance_array:
235             # initiate ground data
236             ground_data = GroundData(2.4, 10, 0.12)
237
238             # set ground data
239             borefield.set_ground_parameters(ground_data)
240
241             # set borefield
242             borefield.create_rectangular_borefield(config[0], config[1], 7, 7, 100, 4)
243
244             # calculate loads
245             extra_load = imbalance / 12 * 10 ** 3 # kWh
246             monthly_load_cooling = monthly_load_cooling_init + extra_load
247             peak_cooling = peak_cooling_init + extra_load / 730
248
249             # set cooling loads
250             borefield.set_peak_cooling(peak_cooling)
251             borefield.set_baseload_cooling(monthly_load_cooling)
252             try:
253                 depth = borefield.size()
254                 depth_array.append(depth)
255             except:

```

(continues on next page)

(continued from previous page)

```

252         pass
253
254         results.append(depth_array)
255
256         # create figure
257         plt.figure()
258         for i, config in enumerate(configs):
259             plt.plot(imbalance_array[:len(results[i])], results[i], label=str(config[0]) + "x
↳ " + str(config[1]))
260
261         # plt.title("Depth for different imbalances")
262         plt.xlabel("Imbalance (MWh/y)")
263         plt.ylabel("Depth (m)")
264
265         plt.legend()
266         plt.show()
267
268
269 def figure_5():
270     """
271     This function creates the fifth figure of the article.
272     """
273
274     # initiate borefield
275     borefield = Borefield()
276
277     # set the correct sizing method
278     borefield.sizing_setup(L2_sizing=True)
279
280     # initiate array with imbalances
281     imbalance_array = np.linspace(200, 1600, 20)
282
283     # initiate list of borefield configurations
284     configs = [(10, 10), (11, 11), (12, 12), (14, 14),
285               (15, 15), (18, 18), (20, 20)]
286
287     # initiate loads
288     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
↳ 0., 0.061, 0.087, .117, 0.144])
289     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↳ .1, .075, .05, .025])
290     monthly_load_heating = monthly_load_heating_percentage * 100 * 10 ** 3 # kWh
291     monthly_load_cooling_init = monthly_load_cooling_percentage * 100 * 10 ** 3 # kWh
292     peak_cooling_init = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0.,
↳ 0.]) # Peak cooling in kW
293     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↳ ])
294
295     # set heating loads
296     borefield.set_peak_heating(peak_heating)
297     borefield.set_baseload_heating(monthly_load_heating)
298

```

(continues on next page)

(continued from previous page)

```

299 results = []
300 for i, config in enumerate(configs):
301     Ra_array = []
302     for imbalance in imbalance_array:
303         # initiate ground data
304         ground_data = GroundData(2.4, 10, 0.12)
305
306         # set ground data
307         borefield.set_ground_parameters(ground_data)
308
309         # set borefield
310         borefield.create_rectangular_borefield(config[0], config[1], 7, 7, 100, 4)
311
312         # calculate loads
313         extra_load = imbalance / 12 * 10 ** 3 # kWh
314         monthly_load_cooling = monthly_load_cooling_init + extra_load
315         peak_cooling = peak_cooling_init + extra_load / 730
316
317         # set cooling loads
318         borefield.set_peak_cooling(peak_cooling)
319         borefield.set_base_load_cooling(monthly_load_cooling)
320         try:
321             depth = borefield.size()
322
323             # calculate gfunction
324             gfunction = borefield.gfunction(borefield.time, depth)
325
326             # calculate Ra
327             Ra = (gfunction[2] - gfunction[1]) / (2 * math.pi * borefield.k_s)
328
329             Ra_array.append(Ra)
330         except:
331             pass
332
333     results.append(Ra_array)
334
335     # create figure
336     plt.figure()
337     for i, config in enumerate(configs):
338         plt.plot(imbalance_array[:len(results[i])], results[i], label=str(config[0]) + "x
339         ↪ " + str(config[1]))
340
341     # plt.title("Ra for different borefield configurations")
342     plt.xlabel("Imbalance (MWh/y)")
343     plt.ylabel("Ra (mK/W)")
344
345     plt.legend()
346     plt.show()
347
348 def figure_7():
349     """

```

(continues on next page)

(continued from previous page)

```

350     This function creates the seventh figure in the article.
351     """
352
353     # initiate borefield
354     borefield = Borefield()
355
356     # set the correct sizing method
357     borefield.sizing_setup(L2_sizing=True)
358
359     # initiate array with imbalances
360     imbalance_array = np.linspace(100, 500, 10)
361
362     # initiate list of borefield configurations
363     configs = [(7, 15), (15, 15)]
364
365     # initiate loads
366     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
↪ 0., 0.061, 0.087, .117, 0.144])
367     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪ .1, .075, .05, .025])
368     monthly_load_heating = monthly_load_heating_percentage * 100 * 10 ** 3 # kWh
369     monthly_load_cooling_init = monthly_load_cooling_percentage * 100 * 10 ** 3 # kWh
370     peak_cooling_init = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0.,
↪ 0.]) # Peak cooling in kW
371     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↪ ])
372
373     # set heating loads
374     borefield.set_peak_heating(peak_heating)
375     borefield.set_baseload_heating(monthly_load_heating)
376
377     results = []
378     for i, config in enumerate(configs):
379         Ra_array = []
380         for imbalance in imbalance_array:
381             # initiate ground data
382             ground_data = GroundData(2.4, 10, 0.12)
383
384             # set ground data
385             borefield.set_ground_parameters(ground_data)
386
387             # set borefield
388             borefield.create_rectangular_borefield(config[0], config[1], 7, 7, 100, 4)
389
390             # calculate loads
391             extra_load = imbalance / 12 * 10 ** 3 # kWh
392             monthly_load_cooling = monthly_load_cooling_init + extra_load
393             peak_cooling = peak_cooling_init + extra_load / 730
394
395             # set cooling loads
396             borefield.set_peak_cooling(peak_cooling)
397             borefield.set_baseload_cooling(monthly_load_cooling)

```

(continues on next page)

(continued from previous page)

```

398     try:
399         depth = borefield.size()
400
401         # calculate gfunction
402         gfunction = borefield.gfunction(borefield.time, depth)
403
404         # calculate Ra
405         Ra = (gfunction[2] - gfunction[1]) / (2 * math.pi * borefield.k_s)
406
407         Ra_array.append(Ra)
408     except:
409         pass
410
411     results.append(Ra_array)
412
413     # create figure
414     plt.figure()
415     for i, config in enumerate(configs):
416         plt.plot(imbalance_array[:len(results[i])], results[i], label=str(config[0]) + "x
↪ " + str(config[1]))
417
418     # plt.title("Ra for different borefield configurations")
419     plt.xlabel("Imbalance (MWh/y)")
420     plt.ylabel("Ra (mK/W)")
421
422     plt.annotate(' ', xy=(300, 2.167), xytext=(350, 2.205),
423                 arrowprops=dict(arrowstyle='<-', color='black'))
424     plt.annotate(' ', xy=(280, 2.15), xytext=(240, 2.02),
425                 arrowprops=dict(arrowstyle='<-', color='black'))
426
427     plt.legend()
428     plt.show()
429
430
431 def figure_8():
432     """
433     This function creates the eight figure of the article.
434     """
435
436     # initiate borefield
437     borefield1 = Borefield()
438     borefield2 = Borefield()
439
440     # set the correct sizing method
441     borefield1.sizing_setup(L2_sizing=True)
442     borefield2.sizing_setup(L2_sizing=True)
443
444     # initiate array with imbalances percentages
445     imbalance_array = np.linspace(30, 70, 20)
446
447     # initiate list of borefield configurations
448     configs = [(20, 6), (20, 6)],

```

(continues on next page)

(continued from previous page)

```

449         ((18, 8), (16, 6)),
450         ((14, 12), (18, 4)),
451         ((16, 12), (16, 5))]
452
453     # initiate imbalance
454     imbalance = 800
455
456     # initiate loads
457     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
↪ 0., 0.061, 0.087, 0.117, 0.144])
458     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪ .1, .075, .05, .025])
459     monthly_load_heating = monthly_load_heating_percentage * (100 + imbalance) * 10 ** 3
460     monthly_load_cooling = monthly_load_cooling_percentage * 100 * 10 ** 3 # kWh
461     peak_cooling = np.array([0., 0., 22., 44., 83., 117., 134., 150., 100., 23., 0., 0.])
462     peak_heating = np.array([300., 268., 191., 103., 75., 0., 0., 38., 76., 160., 224.,
↪ 255.])
463
464     results = []
465     for i, config_pair in enumerate(configs):
466         config1, config2 = config_pair
467         ratio_of_nb_of_boreholes = config1[0] * config1[1] / (config2[0] * config2[0] +
↪ config1[0] * config1[1])
468
469         # initiate ground data
470         ground_data1 = GroundData(2.4, 10, 0.14)
471         ground_data2 = GroundData(2.4, 10, 0.14)
472
473         # set ground data
474         borefield1.set_ground_parameters(ground_data1)
475         borefield2.set_ground_parameters(ground_data2)
476
477         # set borefields
478         borefield1.create_rectangular_borefield(config1[0], config1[1], 7, 7, 100, 4)
479         borefield2.create_rectangular_borefield(config2[0], config2[1], 7, 7, 100, 4)
480
481         # set cooling peak according to the ratio of nb_of_boreholes
482         borefield1.set_peak_cooling(peak_cooling * ratio_of_nb_of_boreholes)
483         borefield2.set_peak_cooling(peak_cooling * (1-ratio_of_nb_of_boreholes))
484
485         results_temp = []
486
487         for imbalance_percentage in imbalance_array:
488
489             # set the imbalance loads
490             borefield1.set_baseload_heating(monthly_load_heating * imbalance_percentage/
↪ 100)
491             borefield2.set_baseload_heating(monthly_load_heating * (100 - imbalance_
↪ percentage)/100)
492
493             # set peak load heating
494             borefield1.set_peak_heating((peak_heating + imbalance_percentage/100 *

```

(continues on next page)

(continued from previous page)

```

↪ imbalance / 12 / 730 * 10 ** 3))
495     borefield2.set_peak_heating((peak_heating + (100 - imbalance_percentage) /
↪ 100 * imbalance / 12 / 730 * 10 ** 3))
496
497     # set baseload cooling equally over the fields
498     borefield1.set_baseload_cooling(monthly_load_cooling * imbalance_percentage/
↪ 100)
499     borefield2.set_baseload_cooling(monthly_load_cooling * (100 - imbalance_
↪ percentage)/100)
500
501     try:
502         depth1 = borefield1.size()
503         depth2 = borefield2.size()
504
505         results_temp.append(depth1 * config1[0] * config1[1] + depth2 *
↪ config2[0] * config2[1])
506     except:
507         results_temp.append(0)
508
509     results.append(results_temp)
510
511     # create figure
512     plt.figure()
513     for i, config in enumerate(configs):
514         plt.plot(imbalance_array[:len(results[i])], results[i], label=str(config[0]) + "x
↪ " + str(config[1]))
515
516     # plt.title("Effect of imbalance distribution on total borefield length")
517     plt.xlabel("Percentage of imbalance on field with largest number of boreholes")
518     plt.ylabel("Total borefield length (m)")
519
520     plt.legend()
521     plt.show()
522
523
524 if __name__ == "__main__":
525     figure_1()
526     figure_2()
527     figure_3()
528     figure_4()
529     figure_5()
530     figure_7()
531     figure_8()

```

References

- Meertens, L. (2024). Invloed van dynamische korte-termijneffecten op de dimensionering van geothermische boorvelden. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.
- Coninx, M., De Nies, J., Hermans, L., Peere, W., Boydens, W., Helsen, L. (2024). Cost-efficient cooling of buildings by means of geothermal borefields with active and passive cooling. *Applied Energy*, 355, Art. No. 122261, <https://doi.org/10.1016/j.apenergy.2023.122261>.
- Peere, W., Hermans, L., Boydens, W., and Helsen, L. (2023). Evaluation of the oversizing and computational speed of different open-source borefield sizing methods. In *Proceedings of International Building Simulation Conference 2023*. Shanghai (Belgium), 4-6 September 2023.
- Weynjes, J. (2023). Methode voor het dimensioneren van een geothermisch systeem met regeneratie binnen verschillende ESCO-structuren. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.
- Hermans, L., Haesen, R., Uytterhoeven, A., Peere, W., Boydens, W., Helsen, L. (2023). Pre-design of collective residential solar districts with seasonal thermal energy storage: Importance of level of detail. *Applied thermal engineering* 226, Art.No. 120203, 10.1016/j.applthermaleng.2023.120203
- Cimmino, M., Cook., J. C. (2022). pygfunction 2.2 : New Features and Improvements in Accuracy and Computational Efficiency. In *Proceedings of IGSHPA Research Track 2022*. Las Vegas (USA), 6-8 December 2022. <https://doi.org/10.22488/okstate.22.000015>
- Verleyen, L., Peere, W., Michiels, E., Boydens, W., Helsen, L. (2022). The beauty of reason and insight: a story about 30 years old borefield equations. *IEA HPT Magazine* 40(3), 36-39, <https://doi.org/10.23697/6q4n-3223>
- Peere, W., Boydens, W., Helsen, L. (2022). GHEtool: een open-sourcetool voor boorvelddimensionering. Presented at the 15e warmtepompsymposium: van uitdaging naar aanpak, Quadrivium, Heverlee, België.
- Peere, W., Coninx, M., De Nies, J., Hermans, L., Boydens, W., Helsen, L. (2022). Cost-efficient Cooling of Buildings by means of Borefields with Active and Passive Cooling. Presented at the 15e warmtepompsymposium: van uitdaging naar aanpak, Quadrivium, Heverlee, België.
- Peere, W. (2022). Technologieën voor de energietransitie. Presented at the Energietransitie in meergezinswoningen en kantoorgebouwen: uitdagingen!, VUB Brussel Bruxelles - U Residence.
- Peere, W., Blanke, T.(2022). GHEtool: An open-source tool for borefield sizing in Python. *Journal of Open Source Software*, 7(76), 4406, <https://doi.org/10.21105/joss.04406>
- Sharifi., M. (2022). Early-Stage Integrated Design Methods for Hybrid GEOTABS Buildings. PhD thesis, Department of Architecture and Urban Planning, Faculty of Engineering and Architecture, Ghent University.
- Coninx M., De Nies J. (2022). Cost-efficient Cooling of Buildings by means of Borefields with Active and Passive Cooling. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.
- Michiels, E. (2022). Dimensionering van meerdere gekoppelde boorvelden op basis van het type vraagprofiel en de verbinding met de gebruikers. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.
- Vanpoucke, B. (2022). Optimale dimensionering van boorvelden door een variabel massadebiet. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.
- Haesen, R., Hermans, L. (2021). Design and Assessment of Low-carbon Residential District Concepts with (Collective) Seasonal Thermal Energy Storage. Master thesis, Departement of Mechanical Engineering, KU Leuven, Belgium.

Peere, W., Picard, D., Cupeiro Figueroa, I., Boydens, W., and Helsen, L. (2021). Validated combined first and last year borefield sizing methodology. In *Proceedings of International Building Simulation Conference 2021*. Brugge (Belgium), 1-3 September 2021. <https://doi.org/10.26868/25222708.2021.30180>

Peere, W. (2020). Methode voor economische optimalisatie van geothermische verwarmings- en koelsystemen. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.

1.2.3 GHEtool's Changelog and future developments

All notable changes to this project will be documented in this file. For future developments, please visit our [project board](#) on GitHub.

The format is based on [Keep a Changelog](#).

[2.2.2] - unpublished

Added

- Added multiple ground layers (issue #97).
- Function to create box, U and L-shaped borefields (issue #224).
- Multiple year validation for L3 and L4 sizing (issue #227).
- Added MonthlyGeothermalLoadMultiYear (issue #227).
- Added optimise_load_profile_energy (issue #229).
- Added k_p_out to Coaxial Pipe class (issue #239).

changed

- Removed set_peak_length from Borefield class (issue #227).
- Definition of the optimise_load_profile_class (issue #229).
- Changed number_of_boreholes to an attribute (issue #233).
- Definition of H when loading a borefield is now the average borehole depth (issue #233).
- Changed store_previous_values in GFunction class to be a property (issue #233).
- Changed temperature database to a json-file (issue #235).
- Changed optimise_load_profile_power to be able to work with hourly data directly (issue #237).

fixed

- Small typo's in functions (issue #224).
- Bug when using borefield with different borehole lengths (issue #233).

2.2.1 - 2024-01-27

Added

- GHEtool is available on conda-forge (issue #107).
- Possibility to start in another month (issue #140).
- Equal functions for HourlyGeothermalLoad and MonthlyGeothermalLoadAbsolute (issue #189).
- Cylindrical borehole correction (issue #187).
- **add** functionality for the load classes (issue #202).

changed

- Negative reference temperatures for the fluid are now possible (issue #192).
- Move code related to the GUI to a separate repo (issue #210).
- Autorelease to PyPi and testPyPi (issue #212).

fixed

- Problem with multiyear hourly data and L3 sizing (issue #153).
- Problem with negative g-function values (issue #187).
- Bug in load-duration curve when not working with optimize load profile (issue #189).
- Bug in hourly data (issue #196).
- Bug in saving after a file has been moved (issue #198).
- Bug in DHW and peak heating power(issue #202).

2.2.0 - 2023-10-17

Added

- Extra warning message if one wants to load a GHEtool file that was created with a newer version.
- Borehole thermal resistance is now visible at the borehole thermal resistance page (issue #51).
- New class of GroundData: GroundTemperatureGradient added (issue #145).
- Load classes (issue #45).
- Pipe classes (single, double, coaxial, Multiple U Tube) (issue #40 and #45).
- Added another methodology for sizing with a variable ground temperature (issue #144).
- Custom error when the field cannot be sized due to a ground temperature gradient (issue #156).
- Interpolation option in calculate function in Gfunction class (issue #159).
- Absolute and relative tolerances for the sizing methods even as a maximum number of iterations is added, so there is more transparency and flexibility in the trade-off between accuracy and speed (issue #159).
- Added advanced options to GHEtool GUI (issue #165).

- Added a result class so all calculated temperatures are now in a separate Result class object within the borefield object (issue #167).
- Added domestic hot water (DHW) to GHEtool (issue #172).
- Glycol-water mixtures can now be selected from within the GUI (issue #174).
- Pygfunction media object can be imported into the FluidData object in GHEtool (issue #174).
- Temperature and flux database (Europe) implemented (issue #178).
- Yearly heating/cooling load in LoadClass (issue #180).

Changed

- GUI was moved to a separate project: [ScenarioGUI](#).
- H_init was removed from the sizing functions since it was not used.
- Rb is now solely handled by the borehole object.
- load_hourly_profile is moved to the separate load classes (issue #45).
- Removed 'set_hourly_cooling_load', 'set_hourly_heating_load' from main_class and move it to separate load class (issue #45).
- Moved draw_borehole_internals to PipeClass (issue #45).
- Borehole equivalent resistances is now calculated in one step, centralised in the pipe class (issue #45).
- Go to 100% code coverage with 350 tests.
- Threshold interpolation for g-functions set to a relative threshold of 25% relative to the demanded depth (issue #144).
- Implemented a custom error for crossing the maximum number of iterations: 'MaximumNumberOfIterations' (issue #144).
- _size_based_on_temperature_profile now returns two arguments: the required depth and a boolean flag to check if the field is sized (issue #144).
- Speed up of L3/L4 sizing by halving calculation time due to intermediate checks if the field is sized (issue #144).
- Changed ValueError when the field cannot be sized due to a temperature gradient to the custom UnsolvableViewToTemperatureGradient Exception (issue #156).
- Rename SizingSetup class to CalculationSetup class (issue #159).
- Move H_init to CalculationSetup class (issue #159).
- Move use_precalculated_data to CalculationSetup class and rename to: 'use_precalculate_dataset' (issue #159).
- Changed 'set_max_ground_temperature' and 'set_min_ground_temperature' to correct names: 'set_max_avg_fluid_temperature' and 'set_min_avg_fluid_temperature'
- Changed 'minimal average fluid temperature' to 'minimum average fluid temperature' in GUI (issue #172).
- Max value of SEER is now 1000 (issue #178).

Fixed

- Fixed problem with L2 sizing, when the peak load was the same in all months (issue #146).
- Small bug in faster g-function calculation solved. When changing the borefield, the previously calculated g-functions were not removed.
- When using interpolation for the g-functions, the results could vary a little bit based on the previous sizings. By reinstating the H_init parameter, this is solved.
- Borehole internals can no longer overlap in the GUI.
- Optimise load profile crashes with small borefields (issue #180).

2.1.2 - 2023-04-28

Added

- Logger for GHEtool (issue #96).
- Examples are now also in RTD.
- Reynolds number is shown on the result page (issue #112).
- Example for the combination of active and passive cooling (issue #114).
- It is now possible to use building loads (with a SCOP/SEER) instead of ground loads (issue #115).

Changed

- In figure plotting, the interval[x[i], x[i+1]] now has the value y[i] (instead of y[i-1]).
- Scroll behaviour on the result page (issue #99).
- Changed icon of GHEtool.
- Imbalance changed to property so it can handle hourly loads as well (issue #106).
- Remove recalculation option (issue #109).
- When data is loaded in a two-column format, the button for 'two columns' is set (issue #133).
- GUI doesn't crash anymore when wrong separator and decimal points are selected when loading a .csv.
- One can now use monthly calculations which do not assume equal month length.

Fixed

- Sizing doesn't crash when either no heating or cooling load is present (issue #91).
- Wrong heating load in april in GUI (issue #94).
- Results are now cleared when new loads are loaded (issue #106).
- Options for g-function calculations are not working (issue #119).
- Wrong naming aim optimise load profile.
- GHEtool now can start after a crash without removing the backup file.
- Some translations were not correct.

- Solves issue with loading .csv file and optimise load profile (issue #130).
- Figure in optimize load profile keeps getting bigger and bigger (issue #131).
- Problem with sizing with temperature gradients (issue #136).
- Problem solved with calculate_multiple_scenarios.

2.1.1 - 2023-01-30

Added

- Added NavigationToolbar to figure (issue #55).
- Added different peak lengths for heating and cooling separately (issue #72).
- Readable saving format for gui (JSON).
- A super class that contains functions relevant for all GHEtool classes.
- Exe can be installed either locally for one user without admin permission or for all users using admin permission.
- Saved files (*.GHEtool) can be loaded from GHEtool by double click.

Changed

- Created a class for the custom g-functions (issue #57).
- Created a class for g-function calculation that stores the previously calculated g-values to speed up the iterative algorithms (issue #57).
- Created a class for sizing_setup to clean up the code. The speed improvement is over a factor 10 for heavy iterative procedures (like optimise load profile). A full speed improvement report can be found under: code version > speed improvements > v2.1.1.
- The sizing methods themselves are now faster due to the fact that only the first and last year are calculated (issue #44). For more info, one can check: code version > speed improvements > v2.1.1.
- Faster loading time of the GUI.
- Further documentation for optimise_load_profile functionality.
- Smaller exe-file size by setting up a virtual environment and using a pyinstall folder instead of a single file.

Fixed

- The hourly_heating_load_on_the_borefield and hourly_cooling_load_on_the_borefield are now correctly calculated.
- When an hourly temperature profile is plotted after an optimise_load_profile optimisation, the hourly load on the borefield (and not the entire hourly load) is shown.
- Correct conversion from hourly to monthly load (issue #62).
- Problem with np.float16 when using simulation periods >80 years due to overflow errors.
- Implemented FIFO-class to prevent cycling in iterative sizing.
- A scenario name cannot occur twice in the scenario list.
- Sometimes some gui options were not shown.

- The drag-and-drop behaviour of the scenario list is fixed (issue #80).
- The renaming of a scenario was not possible (issue #86).
- Problems with borehole internals and pipe overlaps.

2.1.0 - 2022-11-30

Added

- Documentation with ReadTheDocs
- GUI Documentation
- Changelog
- New features in the GUI

Changed

- GUI workflow to be simpler
- precalculated data is removed
- general speed improvements

Removed

- size by length and width for it is not compatible with the just-in-time calculation of the g-functions.

2.0.6 - 2022-10-07

Added

- new functionalities for g-function calculation (inherited from pygfunction) are implemented

Changed

- just-in-time calculation of g-functions is included (and will be expanded later)
- custom borefields can be way faster calculated

Fixed

- Hyperlinks in PyPi should work now
- Sizing by length and width had problems with temperatures below the minimum temperature

2.0.5 - 2022-08-31

Added

- Hourly sizing method (L4) is implemented
- Hourly plotting method
- Volumetric heat capacity is included in the ground data

Changed

- Implemented numpy arrays everywhere
- Implemented convolution instead of matrix multiplication
- New implementation for L3 sizing

Fixed

- No more problems with iteration (L2) and sub 1m depth fields
- Fixed bug in main_functionalities example

Varia

- New validation document for the effective thermal borehole resistance, comparison with EED

2.0.4 - 2022-08-17

Fixed

- Final JOSS paper update

2.0.3 - 2022-08-12

Added

- Variable ground temperature
- Sizing with dynamic R_b^*

Fixed

- General bug fixes

Changed

- Sizing setup with more streamlined sizing options

2.0.2 - 2022-06-12

Added

- Included a function (and example) on sizing a borefield by length and width

2.0.1 - 2022-06-12

Added

- Included a pytest document to check if package is correctly installed

2.0.0 - 2022-04-01

Added

- GUI
- Borehole thermal resistance (based on the pygfunction package)

Changed

- More documentation and examples

1.0.1 - 2021-12-11

Changed

- longer simulation period up to 100 years

Fixed

- fixed bug in interpolation

1.2.4 Legal Notice

1.2.5 Installation

Requirements

This code is tested with Python 3.8, 3.9, 3.10, 3.11 and 3.12 and requires the following libraries (the versions mentioned are the ones with which the code is tested)

- matplotlib \geq 3.5.2
- numpy \geq 1.23.1
- pandas \geq 1.4.3
- pygfunction \geq 2.2.1
- scipy \geq 1.8.1

For the tests

- Pytest \geq 7.1.2

For the active/passive example

- scikit-optimize \geq 0.9.0

Installation

One can install GHEtool by running Pip and running the command

```
pip install GHEtool
```

or one can install a newer development version using

```
pip install --extra-index-url https://test.pypi.org/simple/ GHEtool
```

GHEtool is also available as a conda package. Therefore, you can install GHEtool with the command:

```
conda install GHEtool
```

Developers can clone this repository.

It is a good practise to use virtual environments (venv) when working on a (new) Python project so different Python and package versions don't conflict with eachother. For GHEtool, Python 3.8 or higher is recommended. General information about Python virtual environments can be found [here](#) and in [this article](#).

Check installation

To check whether everything is installed correctly, run the following command

```
pytest --pyargs GHEtool
```

This runs some predefined cases to see whether all the internal dependencies work correctly. 9 test should pass successfully.

1.2.6 Get started with GHEtool

Building blocks of GHEtool

GHEtool is a flexible package that can be extended with methods from [pygfunction](#) (and [ScenarioGUI](#) for the GUI part). To work efficiently with GHEtool, it is important to understand the main structure of the package.

Borefield

The Borefield object is the central object within GHEtool. It is within this object that all the calculations and optimizations take place. All attributes (ground properties, load data ...) are set inside the borefield object.

Ground properties

Within GHEtool, there are multiple ways of setting the ground data. Currently, your options are:

- *GroundConstantTemperature*: if you want to model your borefield with a constant, known ground temperature.
- *GroundFluxTemperature*: if you want to model your ground with a varying ground temperature due to a constant geothermal heat flux.
- *GroundTemperatureGradient*: if you want to model your ground with a varying ground temperature due to a geothermal gradient.
- You can also use multiple ground layers to define your ground model. Please take a look at [our example](#).

Please note that it is possible to add your own ground types by inheriting the attributes from the abstract `_GroundData` class.

Pipe data

Within GHEtool, you can use different structures for the borehole internals: U-tubes or coaxial pipes. Concretely, the classes you can use are:

- *Multiple U-tubes*
- *Single U-tubes (special case of multiple U-tubes)*
- *Double U-tubes (special case of multiple U-tubes)*
- *Coaxial pipe*

Please note that it is possible to add your own pipe types by inheriting the attributes from the abstract `_PipeData` class.

Fluid data

You can set the fluid data by using the FluidData class. In the future, more fluid data classes will be made available.

Load data

One last element which you will need in your calculations, is the load data. Currently, you can only set the primary (i.e. geothermal) load of the borefield. In a future version of GHEtool, also secondary building loads will be included. For now, you can use the following inputs:

- *MonthlyGeothermalLoadAbsolute*: You can set one the monthly baseload and peak load for heating and cooling for one standard year which will be used for all years within the simulation period.
- *HourlyGeothermalLoad*: You can set (or load) the hourly heating and cooling load of a standard year which will be used for all years within the simulation period.
- *HourlyGeothermalLoadMultiYear*: You can set (or load) the hourly heating and cooling load for multiple years (i.e. for the whole simulation period). This way, you can use secondary loads already with GHEtool as shown in [this example](#).
- *MonthlyGeothermalLoadMultiYear*: You can set the monthly heating and cooling load for multiple years (i.e. for the whole simulation period).

All load classes also have the option to add a yearly domestic hot water usage.

Please note that it is possible to add your own load types by inheriting the attributes from the abstract `_LoadData` class.

Simple example

To show how all the pieces of GHEtool work together, below you can find a step-by-step example of how, traditionally, one would work with GHEtool. Start by importing all the relevant classes. In this case we are going to work with a ground model which assumes a constant ground temperature (e.g. from a TRT-test), and we will provide the load with a monthly resolution.

```
from GHEtool import Borefield, GroundDataConstantTemperature,
↳MonthlyGeothermalLoadAbsolute
```

After importing the necessary classes, the relevant ground data parameters are set.

```
data =
GroundDataConstantTemperature(3, # ground thermal conductivity (W/mK)
                              10, # initial/undisturbed
↳ground temperature (deg C)
                              2.4*10**6) # volumetric heat capacity of the ground (J/
↳m3K)
```

Furthermore, for our loads, we need to set the peak loads as well as the monthly base loads for heating and cooling.

```
peak_cooling = [0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.] # Peak
↳cooling in kW
peak_heating = [160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.] # Peak
↳heating in kW

monthly_load_heating = [46500.0, 44400.0, 37500.0, 29700.0, 19200.0, 0.0, 0.0, 0.0,
↳18300.0, 26100.0, 35100.0, 43200.0] # in kWh
```

(continues on next page)

(continued from previous page)

```
monthly_load_cooling = [4000.0, 8000.0, 8000.0, 8000.0, 12000.0, 16000.0, 32000.0, 32000.
↪0, 16000.0, 12000.0, 8000.0, 4000.0] # in kWh

# set load object
load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling, peak_
↪heating, peak_cooling)
```

Next, we create the borefield object in GHEtool and set the temperature constraints and the ground data. Here, since we do not use a pipe and fluid model (see [Examples](#) if you need examples where no borehole thermal resistance is given), we set the borehole equivalent thermal resistance.

```
# create the borefield object
borefield = Borefield(load=load
peak_heating = peak_heating,
peak_cooling = peak_cooling,
baseload_heating = monthly_load_heating,
baseload_cooling = monthly_load_cooling)

# set ground parameters
borefield.set_ground_parameters(data)

# set the borehole equivalent resistance
borefield.Rb = 0.12

# set temperature boundaries
borefield.set_max_avg_fluid_temperature(16) # maximum temperature
borefield.set_min_avg_fluid_temperature(0) # minimum temperature
```

Next we create a rectangular borefield.

```
# set a rectangular borefield
borefield.create_rectangular_borefield(10, 12, 6, 6, 110, 4, 0.075)
```

Note that the borefield can also be set using the `pygfunction` package, if you want more complex designs.

```
import pygfunction as gt

# set a rectangular borefield
borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 110, 1, 0.075)
borefield.set_borefield(borefield_gt)
```

Once a Borefield object is created, one can make use of all the functionalities of GHEtool. One can for example size the borefield using:

```
depth = borefield.size()
print("The borehole depth is: ", depth, "m")
```

Or one can plot the temperature profile by using

```
borefield.print_temperature_profile(legend=True)
```


1.2.7 Functionalities of GHEtool

GHEtool has a couple of different functionalities, all centered around borefield sizing. In the examples listed below, one can find example code on how to use the different functionalities.

Sizing the borefield (i.e. calculating the required depth) for a given injection and extraction load for the borefield (three sizing methods are available).

Main functionalities of GHEtool

```

1  """
2  This file contains all the main functionalities of GHEtool being:
3      * sizing of the borefield
4      * sizing of the borefield for a specific quadrant
5      * plotting the temperature evolution
6      * plotting the temperature evolution for a specific depth
7      * printing the array of the temperature
8  """
9
10 import numpy as np
11
12 # import all the relevant functions
13 from GHEtool import Borefield, FluidData, DoubleUTube, GroundConstantTemperature, ↵
14 ↵MonthlyGeothermalLoadAbsolute
15
16 def main_functionalities():
17     # relevant borefield data for the calculations
18     data = GroundConstantTemperature(3,          # conductivity of the soil (W/mK)
19                                     10,          # Ground temperature at infinity ↵
20 ↵(degrees C)
21                                     2.4 * 10**6) # ground volumetric heat capacity (J/
22 ↵m3K)
23
24     # monthly loading values
25     peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.
26 ↵]) # Peak cooling in kW
27     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
28 ↵]) # Peak heating in kW
29
30     # annual heating and cooling load
31     annual_heating_load = 300 * 10 ** 3 # kWh
32     annual_cooling_load = 160 * 10 ** 3 # kWh
33
34     # percentage of annual load per month (15.5% for January ...)
35     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0., ↵
36 ↵0., 0.061, 0.087, 0.117, 0.144])
37     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
38 ↵.1, .075, .05, .025])
39
40     # resulting load per month
41     monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
42     monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh

```

(continues on next page)

(continued from previous page)

```

37
38     # set the load
39     load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
↪peak_heating, peak_cooling)
40
41     # create the borefield object
42     borefield = Borefield(load=load)
43
44     # one can activate or deactivate the logger, by default it is deactivated
45     # borefield.activate_logger()
46     # borefield.deactivate_logger()
47
48     borefield.set_ground_parameters(data)
49     borefield.create_rectangular_borefield(10, 12, 6, 6, 100, 4, 0.075)
50
51     borefield.Rb = 0.12 # equivalent borehole resistance (K/W)
52
53     # set temperature boundaries
54     borefield.set_max_avg_fluid_temperature(16) # maximum temperature
55     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
56
57     # size borefield
58     depth = borefield.size()
59     print("The borehole depth is: ", depth, "m")
60
61     # print imbalance
62     print("The borefield imbalance is: ", borefield._borefield_load.imbalance, "kWh/y.
↪(A negative imbalance means the the field is heat extraction dominated so it cools
↪down year after year.)") # print imbalance
63
64     # plot temperature profile for the calculated depth
65     borefield.print_temperature_profile(legend=True)
66
67     # plot temperature profile for a fixed depth
68     borefield.print_temperature_profile_fixed_depth(depth=75, legend=False)
69
70     # print gives the array of monthly temperatures for peak cooling without showing the
↪plot
71     borefield.calculate_temperatures(depth=90)
72     print("Result array for cooling peaks")
73     print(borefield.results.peak_cooling)
74     print("-----")
75
76     # size the borefield for quadrant 3
77     # for more information about borefield quadrants, see (Peere et al., 2021)
78     depth = borefield.size(quadrant_sizing=3)
79     print("The borehole depth is: ", str(round(depth, 2)), "m for a sizing in quadrant 3
↪")
80     # plot temperature profile for the calculated depth
81     borefield.print_temperature_profile(legend=True)
82
83     # size with a dynamic Rb* value

```

(continues on next page)

(continued from previous page)

```

84     # note that the original Rb* value will be overwritten!
85
86     # this requires pipe and fluid data
87     fluid_data = FluidData(0.2, 0.568, 998, 4180, 1e-3)
88     pipe_data = DoubleUTube(1, 0.015, 0.02, 0.4, 0.05)
89     borefield.set_fluid_parameters(fluid_data)
90     borefield.set_pipe_parameters(pipe_data)
91
92     # disable the use of constant_Rb with the setup, in order to plot the profile_
↪ correctly
93     # when it is given as an argument to the size function, it will size correctly, but_
↪ the plot will be with
94     # constant Rb* since it has not been changed in the setup function
95     borefield.calculation_setup(use_constant_Rb=False)
96     depth = borefield.size()
97     print("The borehole depth is: ", str(round(depth, 2)), "m for a sizing with dynamic_
↪ Rb*.")
98     borefield.print_temperature_profile(legend=True)
99
100
101 if __name__ == "__main__": # pragma: no cover
102     main_functionalities()

```

Using dynamically calculated borehole thermal resistance (this is directly based on the code of pygfunction).

Sizing with equivalent borehole resistance calculation

```

1  """
2  This document compares the sizing with a constant Rb*-value with sizing where the Rb*-
↪ value is being recalculated.
3  For the test, the L2 sizing method is used.
4  The comparison is based on speed and relative accuracy in the result.
5  It is shown that the speed difference is significant, but so is the difference in the_
↪ result. With a constant Rb* value, it is important that the initial depth is rather_
↪ accurate.
6  """
7
8  import time
9
10 import numpy as np
11 import pygfunction as gt
12
13 from GHEtool import Borefield, FluidData, GroundConstantTemperature, DoubleUTube,
↪ MonthlyGeothermalLoadAbsolute
14
15
16 def sizing_with_Rb():
17     number_of_iterations = 50
18     max_value_cooling = 700
19     max_value_heating = 800
20

```

(continues on next page)

(continued from previous page)

```

21  # initiate the arrays
22  results_Rb_static = np.empty(number_of_iterations)
23  results_Rb_dynamic = np.empty(number_of_iterations)
24  difference_results = np.empty(number_of_iterations)
25
26  monthly_load_cooling_array = np.empty((number_of_iterations, 12))
27  monthly_load_heating_array = np.empty((number_of_iterations, 12))
28  peak_load_cooling_array = np.empty((number_of_iterations, 12))
29  peak_load_heating_array = np.empty((number_of_iterations, 12))
30
31  # populate arrays with random values
32  for i in range(number_of_iterations):
33      for j in range(12):
34          monthly_load_cooling_array[i, j] = np.random.randint(0, max_value_cooling)
35          monthly_load_heating_array[i, j] = np.random.randint(0, max_value_heating)
36          peak_load_cooling_array[i, j] = np.random.randint(monthly_load_cooling_
↪ array[i, j], max_value_cooling)
37          peak_load_heating_array[i, j] = np.random.randint(monthly_load_heating_
↪ array[i, j], max_value_heating)
38
39  # initiate borefield model
40  data = GroundConstantTemperature(3, 10) # ground data with an inaccurate guess of
↪ 100m for the depth of the borefield
41  fluid_data = FluidData(0.2, 0.568, 998, 4180, 1e-3)
42  pipe_data = DoubleUTube(1, 0.015, 0.02, 0.4, 0.05)
43
44  borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 100, 1, 0.075)
45
46  # Monthly loading values
47  peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.
↪ ]) # Peak cooling in kW
48  peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↪ ]) # Peak heating in kW
49
50  # annual heating and cooling load
51  annual_heating_load = 300 * 10 ** 3 # kWh
52  annual_cooling_load = 160 * 10 ** 3 # kWh
53
54  # percentage of annual load per month (15.5% for January ...)
55  monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
↪ 0., 0.061, 0.087, 0.117, 0.144])
56  monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪ .1, .075, .05, .025])
57
58  # resulting load per month
59  monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
60  monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh
61
62  # set the load
63  load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
↪ peak_heating, peak_cooling)
64

```

(continues on next page)

(continued from previous page)

```

65  # create the borefield object
66  borefield = Borefield(load=load)
67
68  borefield.set_ground_parameters(data)
69  borefield.set_fluid_parameters(fluid_data)
70  borefield.set_pipe_parameters(pipe_data)
71  borefield.Rb = 0.2
72  borefield.set_borefield(borefield_gt)
73
74  # create custom gfunction to speed up the calculation
75  borefield.create_custom_dataset()
76
77  # set temperature boundaries
78  borefield.set_max_avg_fluid_temperature(16)  # maximum temperature
79  borefield.set_min_avg_fluid_temperature(0)   # minimum temperature
80
81  # size with constant Rb* value
82  borefield.calculation_setup(use_constant_Rb=True)
83
84  # calculate the Rb* value
85  borefield.Rb = borefield.borehole.calculate_Rb(100, 1, 0.075, data.k_s)
86
87  start_Rb_constant = time.time()
88  for i in range(number_of_iterations):
89      # set the load
90      load = MonthlyGeothermalLoadAbsolute(monthly_load_heating_array[i], monthly_load_
↪cooling_array[i],
91                                          peak_load_heating_array[i], peak_load_
↪cooling_array[i])
92      borefield.load = load
93      results_Rb_static[i] = borefield.size()
94      end_Rb_constant = time.time()
95
96  # size with a dynamic Rb* value
97  borefield.calculation_setup(use_constant_Rb=False)
98
99  start_Rb_dynamic = time.time()
100  for i in range(number_of_iterations):
101      # set the load
102      load = MonthlyGeothermalLoadAbsolute(monthly_load_heating_array[i], monthly_load_
↪cooling_array[i],
103                                          peak_load_heating_array[i], peak_load_
↪cooling_array[i])
104      borefield.load = load
105      results_Rb_dynamic[i] = borefield.size()
106      end_Rb_dynamic = time.time()
107      print(results_Rb_dynamic[1])
108
109  print("These are the results when an inaccurate constant Rb* value is used.")
110  print("Time for sizing with a constant Rb* value:", end_Rb_constant - start_Rb_
↪constant, "s")
111  print("Time for sizing with a dynamic Rb* value:", end_Rb_dynamic - start_Rb_dynamic,

```

(continues on next page)

(continued from previous page)

```

112 ↪ "s")
113     # calculate differences
114     for i in range(number_of_iterations):
115         difference_results[i] = results_Rb_dynamic[i] - results_Rb_static[i]
116
117     print("The maximal difference between the sizing with a constant and a dynamic Rb*
118 ↪value:", np.round(np.max(difference_results), 3), "m or", np.round(np.max(difference_
119 ↪results) / results_Rb_static[np.argmax(difference_results)] * 100, 3), "% w.r.t. the
120 ↪constant Rb* approach.")
121     print("The mean difference between the sizing with a constant and a dynamic Rb*
122 ↪value:", np.round(np.mean(difference_results), 3), "m or", np.round(np.mean(difference_
123 ↪results) / np.mean(results_Rb_static) * 100, 3), "% w.r.t. the constant Rb* approach.")
124     print("-----")
125 ↪")
126
127     # Do the same thing but with another constant Rb* value based on a borehole depth of
128 ↪185m.
129
130     borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 185, 1, 0.075) # borefield
131 ↪with an accurate guess of 185m for the depth of the borefield
132     borefield.set_borefield(borefield_gt)
133
134     # size with a constant Rb* value
135     borefield.calculation_setup(use_constant_Rb=True)
136
137     # calculate the Rb* value
138     borefield.Rb = borefield.borehole.calculate_Rb(100, 1, 0.075, data.k_s)
139
140     start_Rb_constant = time.time()
141     for i in range(number_of_iterations):
142         # set the load
143         load = MonthlyGeothermalLoadAbsolute(monthly_load_heating_array[i], monthly_load_
144 ↪cooling_array[i],
145 ↪peak_load_heating_array[i], peak_load_
146 ↪cooling_array[i])
147         borefield.load = load
148         results_Rb_static[i] = borefield.size()
149         end_Rb_constant = time.time()
150
151     # size with a dynamic Rb* value
152     borefield.calculation_setup(use_constant_Rb=False)
153
154     start_Rb_dynamic = time.time()
155     for i in range(number_of_iterations):
156         # set the load
157         load = MonthlyGeothermalLoadAbsolute(monthly_load_heating_array[i], monthly_load_
158 ↪cooling_array[i],
159 ↪peak_load_heating_array[i], peak_load_
160 ↪cooling_array[i])
161         borefield.load = load
162         results_Rb_dynamic[i] = borefield.size()

```

(continues on next page)

(continued from previous page)

```

151     end_Rb_dynamic = time.time()
152
153     print("These are the results when an accurate constant Rb* value is used.")
154     print("Time for sizing with a constant Rb* value:", end_Rb_constant - start_Rb_
↪ constant, "s")
155     print("Time for sizing with a dynamic Rb* value:", end_Rb_dynamic - start_Rb_dynamic,
↪ "s")
156
157     # calculate differences
158     for i in range(number_of_iterations):
159         difference_results[i] = results_Rb_dynamic[i] - results_Rb_static[i]
160
161     print("The maximal difference between the sizing with a constant and a dynamic Rb*
↪ value:",
162           np.round(np.max(difference_results), 3), "m or",
163           np.round(np.max(difference_results) / results_Rb_static[np.argmax(difference_
↪ results)] * 100, 3),
164           "% w.r.t. the constant Rb* approach.")
165     print("The mean difference between the sizing with a constant and a dynamic Rb*
↪ value:",
166           np.round(np.mean(difference_results), 3), "m or",
167           np.round(np.mean(difference_results) / np.mean(results_Rb_static) * 100, 3),
168           "% w.r.t. the constant Rb* approach.")
169
170
171 if __name__ == "__main__": # pragma: no cover
172     sizing_with_Rb()

```

Optimising the load profile for a given heating and cooling load.

Optimise load profile

```

1  """
2  This document is an example of load optimisation.
3  First an hourly profile is imported and a fixed borefield size is set.
4  Then, based on a load-duration curve, the heating and cooling load is altered in order
↪ to fit as much load as possible on the field.
5  The results are returned.
6
7  """
8  import numpy as np
9
10 # import all the relevant functions
11 from GHEtool import *
12
13
14 def optimise():
15
16     # initiate ground data
17     data = GroundConstantTemperature(3, 10)
18

```

(continues on next page)

(continued from previous page)

```

19  # initiate borefield
20  borefield = Borefield()
21
22  # set ground data in borefield
23  borefield.set_ground_parameters(data)
24
25  # set Rb
26  borefield.Rb = 0.12
27
28  # set borefield
29  borefield.create_rectangular_borefield(10, 10, 6, 6, 150, 1, 0.075)
30
31  # load the hourly profile
32  load = HourlyGeothermalLoad()
33  load.load_hourly_profile("hourly_profile.csv", header=True, separator=";")
34
35  # optimise the load for a 10x10 field (see data above) and a fixed depth of 150m.
36  # first for an optimisation based on the power
37  borefield.optimise_load_profile_power(building_load=load, depth=150)
38
39  print(f'Max heating power (primary): {borefield.load.max_peak_heating:,.0f}kW')
40  print(f'Max cooling power (primary): {borefield.load.max_peak_cooling:,.0f}kW')
41
42  print(f'Total energy extracted from the borefield over simulation period: {np.
↳ sum(borefield.load.baseload_heating_simulation_period):,.0f}MWh')
43  print(f'Total energy injected in the borefield over simulation period: {np.
↳ sum(borefield.load.baseload_cooling_simulation_period):,.0f}MWh')
44  print('-----')
45  borefield.calculate_temperatures(hourly=True)
46  borefield.print_temperature_profile(plot_hourly=True)
47
48  # first for an optimisation based on the energy
49  borefield.optimise_load_profile_energy(building_load=load, depth=150)
50
51  print(f'Max heating power (primary): {borefield.load.max_peak_heating:,.0f}kW')
52  print(f'Max cooling power (primary): {borefield.load.max_peak_cooling:,.0f}kW')
53
54  print(
55      f'Total energy extracted from the borefield over simulation period: {np.
↳ sum(borefield.load.baseload_heating_simulation_period):,.0f}MWh')
56  print(
57      f'Total energy injected in the borefield over simulation period: {np.
↳ sum(borefield.load.baseload_cooling_simulation_period):,.0f}MWh')
58
59  borefield.calculate_temperatures(hourly=True)
60  borefield.print_temperature_profile(plot_hourly=True)
61
62
63  if __name__ == "__main__": # pragma: no cover
64      optimise()

```

Importing heating and cooling loads from .csv files.

Import data

```

1  """
2  This document is an example on how to import hourly load profiles into GHEtool.
3  It uses the hourly_profile.csv data.
4  """
5  import pygfunction as gt
6
7  # import all the relevant functions
8  from GHEtool import *
9
10 # initiate ground data
11 data = GroundConstantTemperature(3, 10, 2.4*10**6)
12 borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 110, 1, 0.075)
13
14 # initiate borefield
15 borefield = Borefield()
16
17 # set ground data in borefield
18 borefield.set_ground_parameters(data)
19
20 # set Rb
21 borefield.Rb = 0.12
22
23 # set borefield
24 borefield.set_borefield(borefield_gt)
25
26 # load the hourly profile
27 load = HourlyGeothermalLoad()
28 load.load_hourly_profile("hourly_profile.csv", header=True, separator=";")
29 borefield.load = load
30
31 # size the borefield and plot the resulting temperature evolution
32 depth = borefield.size(100, L2_sizing=True)
33 print(depth)
34 borefield.print_temperature_profile()

```

Using your custom borefield configurations.

Custom borefield configuration

```

1  """
2  This file gives an example on how to work with a custom borefield within GHEtool using
3  ↪ pygfunction.
4
5  When working on a custom borefield configuration, one needs to import this configuration
6  ↪ into the GHEtool.
7  Based on the pygfunction, one creates his custom borefield and gives it as an argument
8  ↪ to the class initiator Borefield of GHEtool.
9
10 You also need a custom g-function file for interpolation. This can also be given as an
11 ↪ argument to the class initiator as _custom_gfunction.

```

(continues on next page)

(continued from previous page)

```

8  This custom variable, must contain gfunctions for all time steps in Borefield.DEFAULT_
   ↪ TIME_ARRAY, and should be structured as follows:
9  {"Time":Borefield.DEFAULT_TIME_ARRAY,"Data":[[Depth1,[Gfunc1,Gfunc2 ...]], [Depth2,
   ↪ [Gfunc1, Gfunc2 ...]]]}.
10
11 However, one can use the function 'create_custom_dataset' when a custom borefield is_
   ↪ given. This will make the required dataset for the optimisation.
12  """
13
14 import numpy as np
15 import pygfunction as gt
16
17 # import all the relevant functions
18 from GHEtool import *
19
20
21 def custom_borefield_configuration():
22     # set the relevant ground data for the calculations
23     data = GroundConstantTemperature(3, 10)
24
25     # Monthly loading values
26     peak_cooling = np.array([0., 0, 3.4, 6.9, 13., 18., 21., 50., 16., 3.7, 0., 0.]) #_
   ↪ Peak cooling in kW
27     peak_heating = np.array([60., 42., 10., 5., 0., 0., 0., 0., 4.4, 8.5, 19., 36.]) #_
   ↪ Peak heating in kW
28
29     # annual heating and cooling load
30     annual_heating_load = 30 * 10 ** 3 # kWh
31     annual_cooling_load = 16 * 10 ** 3 # kWh
32
33     # percentage of annual load per month (15.5% for January ...)
34     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,_
   ↪ 0., 0.061, 0.087, 0.117, 0.144])
35     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
   ↪ .1, .075, .05, .025])
36
37     # resulting load per month
38     monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
39     monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh
40
41     # set the load
42     load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,_
   ↪ peak_heating, peak_cooling)
43
44     # create the borefield object
45     borefield = Borefield(load=load)
46
47     borefield.set_ground_parameters(data)
48     borefield.Rb = 0.2
49
50     # set temperature boundaries
51     borefield.set_max_avg_fluid_temperature(16) # maximum temperature

```

(continues on next page)

(continued from previous page)

```

52     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
53
54     # create custom borefield based on pygfunction
55     custom_field = gt.boreholes.L_shaped_field(N_1=4, N_2=5, B_1=5., B_2=5., H=100., D=4,
↪ r_b=0.05)
56
57     # set the custom borefield (so the number of boreholes is correct)
58     borefield.set_borefield(custom_field)
59     borefield.create_custom_dataset()
60
61     # size borefield
62     depth = borefield.size()
63     print("The borehole depth is: ", depth, "m")
64
65     # print imbalance
66     print("The borefield imbalance is: ", borefield.load.imbalance, "kWh/y. (A negative_
↪ imbalance means the the field is heat extraction dominated so it cools down year after_
↪ year.)") # print imbalance
67
68     # plot temperature profile for the calculated depth
69     borefield.print_temperature_profile(legend=True)
70
71     # plot temperature profile for a fixed depth
72     borefield.print_temperature_profile_fixed_depth(depth=75, legend=False)
73
74     # print gives the array of monthly temperatures for peak cooling without showing the_
↪ plot
75     borefield.calculate_temperatures(depth=90)
76     print("Result array for cooling peaks")
77     print(borefield.results.peak_cooling)
78
79
80 if __name__ == "__main__": # pragma: no cover
81     custom_borefield_configuration()

```

1.2.8 Modules

Main class

This file contains all the code for the borefield calculations.

```

class GHEtool.main_class.Borefield(peak_heating: Optional[Union[_SupportsArray[dtype[Any]],
    _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float,
    complex, str, bytes, _NestedSequence[Union[bool, int, float, complex,
    str, bytes]]]] = None, peak_cooling:
    Optional[Union[_SupportsArray[dtype[Any]],
    _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float,
    complex, str, bytes, _NestedSequence[Union[bool, int, float, complex,
    str, bytes]]]] = None, baseload_heating:
    Optional[Union[_SupportsArray[dtype[Any]],
    _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float,
    complex, str, bytes, _NestedSequence[Union[bool, int, float, complex,
    str, bytes]]]] = None, baseload_cooling:
    Optional[Union[_SupportsArray[dtype[Any]],
    _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float,
    complex, str, bytes, _NestedSequence[Union[bool, int, float, complex,
    str, bytes]]]] = None, borefield:
    Optional[list[pygfunction.boreholes.Borehole]] = None,
    custom_gfunction: Optional[CustomGFunction] = None, load:
    Optional[_LoadData] = None)

```

Bases: [BaseClass](#)

Main borefield class

Parameters

peak_heating

[list, numpy array] Monthly peak heating values [kW]

peak_cooling

[list, numpy array] Monthly peak cooling values [kW]

baseload_heating

[list, numpy array] Monthly baseload heating values [kWh]

baseload_cooling

[list, numpy array] Monthly baseload heating values [kWh]

borefield

[pygfunction borehole/borefield object] Set the borefield for which the calculations will be carried out

custom_gfunction

[CustomGFunction] Custom gfunction dataset

Examples

monthly peak values [kW]

```

>>> peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0.,
    ↳ 0.])
>>> peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119.,
    ↳ 136.])

```

annual heating and cooling load [kWh]

```

>>> annual_heating_load = 300 * 10 ** 3
>>> annual_cooling_load = 160 * 10 ** 3

```

percentage of annual load per month (15.5% for January ...)

```
>>> monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0.,
↪ 0., 0., 0.061, 0.087, 0.117, 0.144])
>>> monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .
↪ 2, .2, .1, .075, .05, .025])
```

resulting load per month [kWh]

```
>>> monthly_load_heating = annual_heating_load * monthly_load_heating_percentage
>>> monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage
```

create the borefield object

```
>>> borefield = Borefield()
```

set the load

```
>>> load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
↪ peak_heating, peak_cooling)
>>> borefield.load = load
```

property H: float

This function returns the borehole depth.

Returns

float

Borehole depth [meters]

property Rb: float

This function returns the equivalent borehole thermal resistance.

Returns

Rb

[float] Equivalent borehole thermal resistance [mK/W]

property Re: float

Reynolds number.

Returns

float

Reynolds number

static activate_logger() → None

This function activates the logging.

Returns

None

property borefield

Returns the hidden _borefield variable.

Returns

Hidden _borefield object

calculate_next_depth_deep_sizing(*current_depth: float*) → float

This method is a slower but more robust way of calculating the next depth in the sizing iteration when the borefield is sized for the maximum fluid temperature when there is a non-constant ground temperature. The method is based (as can be seen in its corresponding validation document) on the assumption that the difference between the maximum temperature in peak cooling and the average undisturbed ground temperature is irreversibly proportional to the depth. In this way, given this difference in temperature and the current depth, a new depth can be calculated.

Parameters

current_depth

[float] The current depth of the borefield [m]

Returns

float

New depth of the borefield [m]

calculate_quadrant() → int

This function returns the borefield quadrant (as defined by Peere et al., 2021¹) based on the calculated temperature profile. If there is no limiting quadrant, None is returned.

Quadrant 1 is limited in the first year by the maximum temperature

Quadrant 2 is limited in the last year by the maximum temperature

Quadrant 3 is limited in the first year by the minimum temperature

Quadrant 4 is limited in the last year by the maximum temperature

Returns

quadrant

[int] The quadrant which limits the borefield

References

calculate_temperatures(*depth: Optional[float] = None, hourly: bool = False*) → None

Calculate all the temperatures without plotting the figure. When depth is given, it calculates it for a given depth.

Parameters

depth

[float] Depth for which the temperature profile should be calculated for [m]

hourly

[bool] True when the temperatures should be calculated based on hourly data

Returns

None

calculation_setup(*calculation_setup: Optional[CalculationSetup] = None, use_constant_Rb: Optional[bool] = None, **kwargs*) → None

This function sets the options for the sizing function.

¹ Peere, W., Picard, D., Cupeiro Figueroa, I., Boydens, W., and Helsen, L. (2021) Validated combined first and last year borefield sizing methodology. In Proceedings of International Building Simulation Conference 2021. Brugge (Belgium), 1-3 September 2021. <https://doi.org/10.26868/25222708.2021.30180>

- The L2 sizing is the one explained in (Peere et al., 2021)² and is the quickest method (it uses 3 pulses)
- The L3 sizing is a more general approach which is slower but more accurate (it uses 24 pulses/year)
- The L4 sizing is the most exact one, since it uses hourly data (8760 pulses/year)

Parameters

calculation_setup

[CalculationSetup] An instance of the CalculationSetup class. When this argument differs from None, all the other parameters are set based on this calculation_setup

use_constant_Rb

[bool] True if a constant borehole equivalent resistance (Rb*) value should be used

kwargs

Dictionary with all the other options that can be set within GHEtool. For a complete list, see the documentation in the CalculationSetup class.

Returns

None

References

create_L_shaped_borefield(*N_1: int, N_2: int, B_1: float, B_2: float, H: float, D: float = 1, r_b: float = 0.075*)

This function creates a L shaped borefield. It calls the pygfunction module in the background. The documentation of this function is based on pygfunction.

Parameters

N_1

[int] Number of boreholes in the x direction

N_2

[int] Number of boreholes in the y direction

B_1

[int] Distance between adjacent boreholes in the x direction [m]

B_2

[int] Distance between adjacent boreholes in the y direction [m]

H

[float] Borehole depth [m]

D

[float] Borehole buried depth [m]

r_b

[float] Borehole radius [m]

Returns

pygfunction borefield object

² Peere, W., Picard, D., Cupeiro Figueroa, I., Boydens, W., and Helsen, L. (2021) Validated combined first and last year borefield sizing methodology. In Proceedings of International Building Simulation Conference 2021. Brugge (Belgium), 1-3 September 2021. <https://doi.org/10.26868/25222708.2021.30180>

create_U_shaped_borefield(*N_1: int, N_2: int, B_1: float, B_2: float, H: float, D: float = 1, r_b: float = 0.075*)

This function creates a U shaped borefield. It calls the pygfunction module in the background. The documentation of this function is based on pygfunction.

Parameters

N_1

[int] Number of boreholes in the x direction

N_2

[int] Number of boreholes in the y direction

B_1

[int] Distance between adjacent boreholes in the x direction [m]

B_2

[int] Distance between adjacent boreholes in the y direction [m]

H

[float] Borehole depth [m]

D

[float] Borehole buried depth [m]

r_b

[float] Borehole radius [m]

Returns

pygfunction borefield object

create_box_shaped_borefield(*N_1: int, N_2: int, B_1: float, B_2: float, H: float, D: float = 1, r_b: float = 0.075*)

This function creates a box shaped borefield. It calls the pygfunction module in the background. The documentation of this function is based on pygfunction.

Parameters

N_1

[int] Number of boreholes in the x direction

N_2

[int] Number of boreholes in the y direction

B_1

[int] Distance between adjacent boreholes in the x direction [m]

B_2

[int] Distance between adjacent boreholes in the y direction [m]

H

[float] Borehole depth [m]

D

[float] Borehole buried depth [m]

r_b

[float] Borehole radius [m]

Returns

pygfunction borefield object

create_circular_borefield(*N: int, R: float, H: float, D: float = 1, r_b: float = 0.075*)

This function creates a circular borefield. It calls the pygfunction module in the background. The documentation of this function is based on pygfunction.

Parameters

- N**
[int] Number of boreholes in the borefield
- R**
[float] Distance of boreholes from the center of the field
- H**
[float] Borehole depth [m]
- D**
[float] Borehole buried depth [m]
- r_b**
[float] Borehole radius [m]

Returns

pygfunction borefield object

create_custom_dataset(*time_array: Optional[Union[_SupportsArray[dtype[Any]],
_NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str,
bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]] = None,
depth_array: Optional[Union[_SupportsArray[dtype[Any]],
_NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str,
bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]] = None,
options: dict = {}*) → None

This function makes a datafile for a given custom borefield and sets it for the borefield object. It automatically sets this datafile in the current borefield object, so it can be used as a source for the interpolation of g-values.

Parameters

- time_array**
[list, np.array] Time values (in seconds) used for the calculation of the datafile
- depth_array**
[list, np.array] List or arrays of depths for which the datafile should be created
- options**
[dict] Options for the g-function calculation (check pygfunction.gfunction.gFunction() for more information)

Returns

None

Raises

- ValueError**
When no borefield or ground data is set

create_rectangular_borefield(*N_1: int, N_2: int, B_1: float, B_2: float, H: float, D: float = 1, r_b: float = 0.075*)

This function creates a rectangular borefield. It calls the pygfunction module in the background. The documentation of this function is based on pygfunction.

Parameters

N_1
[int] Number of boreholes in the x direction

N_2
[int] Number of boreholes in the y direction

B_1
[int] Distance between adjacent boreholes in the x direction [m]

B_2
[int] Distance between adjacent boreholes in the y direction [m]

H
[float] Borehole depth [m]

D
[float] Borehole buried depth [m]

r_b
[float] Borehole radius [m]

Returns

pygfunction borefield object

static deactivate_logger() → None

This function deactivates the logging.

Returns

None

gfunction(*time_value: Union[_SupportsArray[dtype[Any]],
_NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes,
_NestedSequence[Union[bool, int, float, complex, str, bytes]]], H: Optional[float] = None*) →
ndarray

This function returns the gfunction value. It can do so by either calculating the gfunctions just-in-time or by interpolating from a loaded custom data file.

Parameters

time_value
[list, float, np.ndarray] Time value(s) in seconds at which the gfunctions should be calculated

H
[float] Depth [m] at which the gfunctions should be calculated. If no depth is given, the current depth is taken.

Returns

gvalue
[np.ndarray] 1D array with the g-values for all the requested time_value(s)

property ground_data: [*_GroundData*](#)

This function returns the ground data.

Returns

ground data
[GroundData]

property investment_cost: float

This function calculates the investment cost based on a cost profile linear to the total borehole length.

Returns

float

Investment cost

property load: `GHEtool.VariableClasses.LoadData._LoadData._LoadData` | `GHEtool.VariableClasses.LoadData.GeothermalLoad.HourlyGeothermalLoad.HourlyGeothermalLoad` | `GHEtool.VariableClasses.LoadData.GeothermalLoad.MonthlyGeothermalLoadAbsolute.MonthlyGeothermalLoadAbsolute`

This returns the LoadData object.

Returns

Load data: LoadData

load_custom_gfunction(*location: str*) → None

This function loads the custom gfunction.

Parameters

location

[str] Path to the location of the custom gfunction file

Returns

None

property number_of_boreholes: int

This returns the number of boreholes in the borefield attribute.

Returns

int

Number of boreholes

optimise_load_profile_energy(*building_load: HourlyGeothermalLoad, depth: Optional[float] = None, SCOP: float = 1000000, SEER: float = 1000000, temperature_threshold: float = 0.05*) → None

This function optimises the load based on the given borefield and the given hourly load. (When the load is not geothermal, the SCOP and SEER are used to convert it to a geothermal load.) It does so based on a load-duration curve. The temperatures of the borefield are calculated on a monthly basis, even though we have hourly data, for an hourly calculation of the temperatures would take a very long time.

Parameters

building_load

[_LoadData] Load data used for the optimisation

depth

[float] Depth of the boreholes in the borefield [m]

SCOP

[float] SCOP of the geothermal system (needed to convert hourly building load to geothermal load)

SEER

[float] SEER of the geothermal system (needed to convert hourly building load to geothermal load)

temperature_threshold

[float] The maximum allowed temperature difference between the maximum and minimum fluid temperatures and their respective limits. The lower this threshold, the longer the convergence will take.

Returns

None

Raises**ValueError**

ValueError if no hourly load is given or the threshold is negative

optimise_load_profile_power(*building_load: HourlyGeothermalLoad, depth: Optional[float] = None, SCOP: float = 1000000, SEER: float = 1000000, temperature_threshold: float = 0.05, use_hourly_resolution: bool = True*) → None

This function optimises the load based on the given borefield and the given hourly load. (When the load is not geothermal, the SCOP and SEER are used to convert it to a geothermal load.) It does so based on a load-duration curve. The temperatures of the borefield are calculated on a monthly basis, even though we have hourly data, for an hourly calculation of the temperatures would take a very long time.

Parameters**building_load**

[_LoadData] Load data used for the optimisation

depth

[float] Depth of the boreholes in the borefield [m]

SCOP

[float] SCOP of the geothermal system (needed to convert hourly building load to geothermal load)

SEER

[float] SEER of the geothermal system (needed to convert hourly building load to geothermal load)

temperature_threshold

[float] The maximum allowed temperature difference between the maximum and minimum fluid temperatures and their respective limits. The lower this threshold, the longer the convergence will take.

use_hourly_resolution

[bool] If use_hourly_resolution is used, the hourly data will be used for this optimisation. This can take some more time than using the monthly resolution, but it will give more accurate results.

Returns

None

Raises**ValueError**

ValueError if no hourly load is given or the threshold is negative

plot_load_duration(*legend: bool = False*) → Tuple[Figure, Axes]

This function makes a load-duration curve from the hourly values.

Parameters

legend

[bool] True if the figure should have a legend

Returns**Tuple**

plt.Figure, plt.Axes

print_temperature_profile(*legend: bool = True, plot_hourly: bool = False*) → None

This function plots the temperature profile for the calculated depth. It uses the available temperature profile data.

Parameters**legend**

[bool] True if the legend should be printed

plot_hourly

[bool] True if the temperature profile printed should be based on the hourly load profile.

Returns**fig, ax**

Figure object

print_temperature_profile_fixed_depth(*depth: float, legend: bool = True, plot_hourly: bool = False*)

This function plots the temperature profile for a fixed depth. It uses the already calculated temperature profile data, if available.

Parameters**depth**

[float] Depth at which the temperature profile should be shown

legend

[bool] True if the legend should be printed

plot_hourly

[bool] True if the temperature profile printed should be based on the hourly load profile.

Returns**fig, ax**

Figure object

set_Rb(*Rb: float*) → None

This function sets the constant equivalent borehole thermal resistance.

Parameters**Rb**

[float] Equivalent borehole thermal resistance (mK/W)

Returns

None

set_borefield(*borefield: Optional[list[pygfunction.boreholes.Borehole]] = None*) → None

This function set the borefield object. When None is given, the borefield will be deleted.

Parameters**borefield**

[List[pygfunction.boreholes.Borehole]] Borefield created with the pygfunction package

Returns**None****set_fluid_parameters**(*data*: FluidData) → None

This function sets the fluid parameters.

Parameters**data**

[FluidData] All the relevant fluid data

Returns**None****set_ground_parameters**(*data*: _GroundData) → None

This function sets the relevant ground parameters.

Parameters**data**

[GroundData] All the relevant ground data

Returns**None****set_investment_cost**(*investment_cost*: Optional[list] = None) → None

This function sets the investment cost. This is linear with respect to the total field length. If None, the default is set.

Parameters**investment_cost**

[list] 1D array of polynomial coefficients (including coefficients equal to zero) from highest degree to the constant term

Returns**None****set_load**(*load*: _LoadData) → None

This function sets the _load attribute.

Parameters**load**

[_LoadData] Load data object

Returns**None****set_max_avg_fluid_temperature**(*temp*: float) → None

This functions sets the maximal average fluid temperature to temp.

Parameters**temp**

[float] Maximal average fluid temperature [deg C]

Returns**None**

Raises**ValueError**

When the maximal average fluid temperature is lower than the minimal average fluid temperature

set_min_avg_fluid_temperature(*temp: float*) → None

This function sets the minimal average fluid temperature to temp.

Parameters**temp**

[float] Minimal average fluid temperature [deg C]

Returns

None

Raises**ValueError**

When the maximal average temperature is lower than the minimal average temperature

set_options_gfunction_calculation(*options: dict*) → None

This function sets the options for the gfunction calculation of pygfunction. This dictionary is directly passed through to the gFunction class of pygfunction. For more information, please visit the documentation of pygfunction.

Parameters**options**

[dict] Dictionary with options for the gFunction class of pygfunction

Returns

None

set_pipe_parameters(*data: _PipeData*) → None

This function sets the pipe parameters.

Parameters**data**

[PipeData] All the relevant pipe parameters

Returns

None

property simulation_period: int

This returns the simulation period from the LoadData object.

Returns**Simulation period [years]**

[int]

size(*H_init: Optional[float] = None, use_constant_Rb: Optional[bool] = None, L2_sizing: Optional[bool] = None, L3_sizing: Optional[bool] = None, L4_sizing: Optional[bool] = None, quadrant_sizing: Optional[int] = None, **kwargs*) → float

This function sets the options for the sizing function.

- The L2 sizing is the one explained in (Peere et al., 2021)^{Page 43, 2} and is the quickest method (it uses 3 pulses)

- The L3 sizing is a more general approach which is slower but more accurate (it uses 24 pulses/year)
- The L4 sizing is the most exact one, since it uses hourly data (8760 pulses/year)

Please note that the changes sizing setup changes here are not saved! Use `self.setupSizing` for this.

Parameters

H_init

[float] Initial depth for the iteration. If None, the default H_init is chosen.

use_constant_Rb

[bool] True if a constant borehole equivalent resistance (Rb*) value should be used

L2_sizing

[bool] True if a sizing with the L2 method is needed

L3_sizing

[bool] True if a sizing with the L3 method is needed

L4_sizing

[bool] True if a sizing with the L4 method is needed

quadrant_sizing

[int] Differs from 0 when a sizing in a certain quadrant is desired. Quadrants are developed by (Peere et al., 2021)^{Page 43, 2, 3},

kwargs

[dict] Dictionary with all the other options that can be set within GHEtool. For a complete list, see the documentation in the CalculationSetup class.

Returns

borehole depth

[float]

Raises

ValueError

ValueError when no ground data is provided

size_L2(H_init: Optional[float] = None, quadrant_sizing: int = 0) → float

This function sizes the of the given configuration according to the methodology explained in (Peere et al., 2021)^{Page 43, 2}, which is a L2 method. When quadrant sizing is other than 0, it sizes the field based on the asked quadrant. It returns the borefield depth.

Parameters

H_init

[float] Initial depth from where to start the iteration [m]

quadrant_sizing

[int] If a quadrant is given the sizing is performed for this quadrant else for the relevant

Returns

H

[float] Required depth of the borefield [m]

Raises

³ Peere, W. (2020) Methode voor economische optimalisatie van geothermische verwarmings- en koelsystemen. Master thesis, Department of Mechanical Engineering, KU Leuven, Belgium.

ValueError

ValueError when no ground data is provided or quadrant is not in range.

size_L3(*H_init: Optional[float] = None, quadrant_sizing: int = 0*) → float

This function sizes the borefield based on a monthly (L3) method.

Parameters**H_init**

[float] Initial depth from where to start the iteration [m]

quadrant_sizing

[int] If a quadrant is given the sizing is performed for this quadrant else for the relevant

Returns**H**

[float] Required depth of the borefield [m]

Raises**ValueError**

ValueError when no ground data is provided or quadrant is not in range.

UnsolvableDueToTemperatureGradient

Error when the field cannot be sized.

size_L4(*H_init: Optional[float] = None, quadrant_sizing: int = 0*) → float

This function sizes the borefield based on an hourly (L4) sizing methodology.

Parameters**H_init**

[float] Initial depth from where to start the iteration [m]

quadrant_sizing

[int] If a quadrant is given the sizing is performed for this quadrant else for the relevant

Returns**H**

[float] Required depth of the borefield [m]

Raises**ValueError**

ValueError when no ground data is provided or quadrant is not in range.

UnsolvableDueToTemperatureGradient

When the field cannot be sized due to the temperature gradient.

BaseClass

This document contains the information for the BaseClass. This class is used as a super class for different variable classes.

class GHEtool.VariableClasses.BaseClass.**BaseClass**

Bases: object

This class contains basic functionality of different classes within GHEtool. It contains the code to generate a dictionary from the class (in order to be able to export to JSON), to load a class based on a dictionary and to check whether or not all attributes differ from None.

This class should only be altered whenever a highly general method should be implemented.

check_values() → bool

This functions checks if the class attributes differ from None.

Returns

bool

True if all values are correct. False otherwise

from_dict(dictionary: dict) → None

This function converts the dictionary values to the class attributes. Currently, it can handle np.ndarray, list, set, str, int, float, tuple, pygfunction.Borehole and classes within GHEtool.

Parameters

dictionary

Dictionary with all the attributes of the class

Returns

None

to_dict() → dict

This function converts the class variables to a dictionary so it can be saved in a JSON format. Currently, it can handle np.ndarray, list, set, str, int, float, tuple, pygfunction.Borehole and classes within GHEtool.

Returns

dict

Dictionary with all the attributes of the class

exception GHEtool.VariableClasses.BaseClass.**MaximumNumberOfIterations**(iter: int)

Bases: RuntimeError

This Error occurs when the maximum number of iteration is reacted.

exception GHEtool.VariableClasses.BaseClass.**UnsolvableDueToTemperatureGradient**

Bases: Exception

This Exception occurs when there is an unsizeble borefield due to incompatibility between 1) peak cooling, which requires a deeper borefield and 2) a temperature gradient, which causes a higher ground temperature when the field is drilled deeper. This leads to unsizeble solutions.

Borehole

This document contains all the information of the borehole class.

class GHEtool.VariableClasses.Borehole.**Borehole**(fluid_data: Optional[FluidData] = None, pipe_data: Optional[_PipeData] = None)

Bases: *BaseClass*

The borehole class contains all the functionalities related to the calculation of the equivalent borehole thermal resistance and contains a fluid and pipe class object.

Parameters

fluid_data

[FluidData] Fluid data

pipe_data

[PipeData] Pipe data

property Rb: float

This returns the constant, equivalent borehole thermal resistance [mK/W].

Returns**Rb***

[float] Equivalent borehole thermal resistance [mK/W]

property Re: float

Reynolds number.

Returns**Reynolds number**

[float]

calculate_Rb(*H: float, D: float, r_b: float, k_s: Union[float, callable]*) → float

This function calculates the equivalent borehole thermal resistance.

Parameters**H**

[float] Borehole depth [m]

D

[float] Borehole burial depth [m]

r_b

[float] Borehole radius [m]

k_s

[float or callable] (Function to calculate the) ground thermal conductivity [mk/W]

Returns**Rb**

[float] Equivalent borehole thermal resistance

Raises**ValueError**

ValueError when the pipe and/or fluid data is not set correctly.

property fluid_data: FluidData

This function returns the fluid data object.

Returns**FluidData****get_Rb**(*H: float, D: float, r_b: float, k_s: Union[callable, float]*) → float

This function returns the equivalent borehole thermal resistance. If use_constant_Rb is True, self._Rb is returned, otherwise the resistance is calculated.

Parameters**H**

[float] Borehole depth [m]

D

[float] Borehole burial depth [m]

r_b

[float] Borehole radius [m]

k_s
[float or callable] (Function to calculate) ground thermal conductivity in function of depth
[mk/W]

Returns

Rb*
[float] Equivalent borehole thermal resistance [mK/W]

property pipe_data: *[_PipeData](#)*

This function returns the pipe data object.

Returns

PipeData

Variable Classes

GHEtool uses a couple of variable classes for handling the load data, ground and pipe properties. Please find below the different classes and their modules.

Ground data

The ground data classes you can use have either:

1. A constant ground temperature
2. A constant ground geothermal heat flux
3. A constant ground temperature gradient

All of these three classes are children of the abstract class `_GroundData`.

class `GHEtool.VariableClasses.GroundData._GroundData.GroundLayer`(*k_s: Optional[float] = None, volumetric_heat_capacity: float = 2400000.0, thickness: Optional[float] = None*)

Bases: *[BaseClass](#)*

Contains the information about a certain ground layer.

Parameters

k_s
[float] Layer thermal conductivity [W/mK]

volumetric_heat_capacity
[float] Layer volumetric heat capacity [J/m³K]

thickness
[float] Layer thickness [m]. None is assumed infinite depth

non_negative(*value*) → float

This function returns the value if the value > 0. Otherwise, an error is raised.

Parameters

value
[float] Value to be checked

Returns

float
Value

Raises

ValueError
When the value equals 0 or is smaller

```
class GHEtool.VariableClasses.GroundData._GroundData._GroundData(k_s: Optional[float] = None,  
                                                                volumetric_heat_capacity: float  
                                                                = 2400000.0)
```

Bases: [BaseClass](#), [ABC](#)

Contains information regarding the ground data of the borefield.

Parameters

k_s
[float] Ground thermal conductivity [W/mK]

volumetric_heat_capacity
[float] The volumetric heat capacity of the ground [J/m³K]

```
add_layer_on_bottom(layer: Union[GroundLayer, List[GroundLayer]]) → None
```

This function adds a ground layer on the bottom of the array. This hence becomes the deepest ground layer.

Parameters

layer
[GroundLayer or list of ground layers] GroundLayer object with thermal properties of this layer

Returns

None

Raises

ValueError
When you add a ground layer on the bottom of a layer which has no predefined depth

```
add_layer_on_top(layer: Union[GroundLayer, List[GroundLayer]]) → None
```

This function adds a ground layer on the top of the array. This hence becomes the highest ground layer.

Parameters

layer
[GroundLayer or list of ground layers] GroundLayer object with thermal properties of this layer

Returns

None

Raises

ValueError
When you add a ground layer with no specified depth and there are already ground layers in the array

```
alpha(H: float = 100) → float
```

Returns the ground thermal diffusivity in m²/s for a given depth. If no volumetric heat capacity or conductivity is given, None is returned.

Parameters**H**

[float] Depth in meters.

Returns**float**

Ground thermal diffusivity in m^2/s for a given depth.

abstract calculate_Tg(*H: float*) \rightarrow float

This function gives back the ground temperature

Parameters**H**

[float] Depth of the borefield [m]

Returns**Tg**

[float] Ground temperature [deg C]

abstract calculate_delta_H(*temperature_diff: float*) \rightarrow float

This function calculates the difference in depth for a given difference in temperature.

Parameters**temperature_diff**

[float] Difference in temperature [deg C]

Returns**Difference in depth [m]**

[float]

calculate_value(*thickness_list: list, cumulative_thickness_list: list, y_range: list, H: float*) \rightarrow float

This function calculates the average value of a certain *y_range* of values for a certain depth, given the thickness of the ground layers.

Parameters**thickness_list**

[list] List of all the layer thicknesses

cumulative_thickness_list

[list] Cumulative sum of all the layer thicknesses

y_range

[list] Range with the values for each layer

H

[float] Depth [m]

Returns**float**

Calculated value for either *k_s* or volumetric heat capacity

check_depth(*H: float*) \rightarrow bool

Checks if the depth is correct. A depth is False when it is lower than 0 or it exceeds the deepest ground layer and *last_layer_infinite* is set to False.

Parameters

H

[float] Depth [m]

Returns**bool**

True if the depth is valid

Raises**ValueError**

When a depth is requested that is either smaller than zero or larger than the maximum depth.

k_s(*H: float = 100*) → float

Returns the ground thermal conductivity in W/mK for a given depth.

Parameters**H**

[float] Depth in meters.

Returns**float**

Ground thermal conductivity in W/mK for a given depth.

max_depth(*max_temp: float*) → float

This function returns the maximum depth, based on the maximum temperature. The maximum is the depth where the ground temperature equals the maximum temperature limit.

Parameters**max_temp**

[float] Maximum temperature [deg C]

Returns**Depth**

[float] Maximum depth [m]

volumetric_heat_capacity(*H: float = 100*) → floatReturns the ground volumetric heat capacity in J/m³K for a given depth.**Parameters****H**

[float] Depth in meters.

Returns**float**Ground volumetric heat capacity in J/m³K for a given depth.

```
class GHEtool.VariableClasses.GroundData.GroundConstantTemperature.GroundConstantTemperature(k_s:
    Optional[float]
    =
    None,
    T_g:
    Optional[float]
    =
    None,
    volumetric_heat_capacity:
    float
    =
    2400000.0)
```

Bases: [_GroundData](#)

Parameters

k_s
[float] Ground thermal conductivity [W/mK]

T_g
[float] Ground temperature at infinity [deg C]

volumetric_heat_capacity
[float] The volumetric heat capacity of the ground [J/m³K]

calculate_Tg(*H*: *Optional[float] = None*) → float

This function gives back the ground temperature.

Parameters

H
[float] Depth of the borefield [m] (not used)

Returns

Tg
[float] Ground temperature [deg C]

calculate_delta_H(*temperature_diff*: float) → float

This function calculates the difference in depth for a given difference in temperature.

Parameters

temperature_diff
[float] Difference in temperature [deg C]

Returns

float
Difference in depth [m]


```

class GHEtool.VariableClasses.GroundData.GroundFluxTemperature.GroundFluxTemperature(k_s:
    Optional[float]
    =
    None,
    T_g:
    Optional[float]
    =
    None,
    volumetric_heat_capacity:
    float
    =
    2400000.0,
    flux:
    float
    =
    0.06)

```

Bases: [_GroundData](#)

Parameters

k_s
[float] Ground thermal conductivity [W/mK]

T_g
[float] Surface ground temperature [deg C]

volumetric_heat_capacity
[float] The volumetric heat capacity of the ground [J/m³K]

flux
[float] The geothermal heat flux [W/m²]

calculate_Tg(H: float) → float

This function gives back the ground temperature at a depth H.

Parameters

H
[float] Depth at which the temperature should be calculated [m]

Returns

Tg
[float] Ground temperature [deg C]

calculate_delta_H(temperature_diff: float, H: float = 100) → float

This function calculates the difference in depth for a given difference in temperature.

Parameters

temperature_diff
[float] Difference in temperature [deg C]

H
[float] Depth at which the average ground thermal conductivity should be taken [m]

Returns

float

Difference in depth [m]

```
class GHEtool.VariableClasses.GroundData.GroundTemperatureGradient.GroundTemperatureGradient(k_s:
    Optional[float]
    =
    None,
    T_g:
    Optional[float]
    =
    None,
    volumetric_heat_capacity:
    float
    =
    2400000.0,
    gradient:
    float
    =
    3.0)
```

Bases: [_GroundData](#)

Parameters

k_s

[float] Ground thermal conductivity [W/mK]

T_g

[float] Surface ground temperature [deg C]

volumetric_heat_capacity

[float] The volumetric heat capacity of the ground [J/m³K]

gradient

[float] The geothermal temperature gradient [K/100m]

calculate_Tg(H: float) → float

This function gives back the ground temperature at a depth H.

Parameters

H

[float] Depth at which the temperature should be calculated [m]

Returns

Tg

[float] Ground temperature [deg C]

calculate_delta_H(temperature_diff: float) → float

This function calculates the difference in depth for a given difference in temperature.

Parameters

temperature_diff
[float] Difference in temperature [deg C]

Returns

float
Difference in depth [m]

Fluid data

This document contains the variable classes for the fluid data.

```
class GHEtool.VariableClasses.FluidData.FluidData(mfr: Optional[float] = None, k_f: Optional[float] =
None, rho: Optional[float] = None, Cp:
Optional[float] = None, mu: Optional[float] =
None)
```

Bases: *BaseClass*

Contains information regarding the fluid data of the borefield.

Parameters

mfr
[float] Mass flow rate per borehole [kg/s]

k_f
[float] Thermal Conductivity of the fluid [W/mK]

rho
[float] Density of the fluid [kg/m3]

Cp
[float] Thermal capacity of the fluid [J/kgK]

mu
[float] Dynamic viscosity of the fluid [Pa/s]

```
import_fluid_from_pygfunction(fluid_object: Fluid) → None
```

This function loads a fluid object from pygfunction and imports it into GHEtool. Note that this object does not contain the mass flow rate!

Parameters

fluid_object
[Fluid object from pygfunction]

Returns

None

```
set_mass_flow_rate(mfr: float) → None
```

This function sets the mass flow rate per borehole.

Parameters

mfr
[fluid] Mass flow rate per borehole [kg/s]

Returns

None

Load data

GHETool supports different types of load data (and more are coming, check our project for more information: <https://github.com/users/wouterpeere/projects/2>) Currently you can use:

1. Geothermal loads with a monthly resolution for one year (so it repeats itself every year)
2. Geothermal loads with an hourly resolution for one year (so it repeats itself every year)
3. Geothermal loads with an hourly resolution but multiple years (it does not repeat itself)

All of the load classes are based children of the abstract `_LoadData` class.

class `GHETool.VariableClasses.LoadData._LoadData._LoadData`(*hourly_resolution: bool*,
simulation_period: int = 20)

Bases: `BaseClass`, `ABC`

This class contains information w.r.t. load data for the borefield sizing.

Parameters

hourly_resolution

[bool] True if the load class uses an hourly resolution

simulation_period

[int] Length of the simulation period in years

property UPM: ndarray

Depending on whether all months are assumed to have equal length, the UPM are either constant or vary during the year.

Returns

Hours per month

[np.ndarray]

_calculate_first_year_params(*HC: bool*) → tuple

This function calculates the parameters for the sizing based on the first year of operation. This is needed for the L2 sizing.

Parameters

HC

[bool] True if the borefield is limited by extraction load

Returns

th, tpm, tcm, qh, qpm, qcm

[float] Peak duration [s], cumulative time passed at the start of the month [s], cumulative time passed at the end of the month [s], peak load [W], average cumulative load of the past months [W avg], average load of the current month [W avg]

_calculate_last_year_params(*HC: bool*) → tuple

This function calculates the parameters for the sizing based on the last year of operation. This is needed for the L2 sizing.

Parameters

HC

[bool] True if the borefield is limited by extraction load

Returns

th, qh, qm, qa

[float] Peak length [s], peak load [W], corresponding monthly load [W], yearly imbalance [W]

abstract _check_input(*input: Union[ndarray, list, tuple]*) → bool

This function checks whether the input is valid or not.

Parameters

input

[np.ndarray, list, tuple] Thermal load input

Returns

bool

True if the input is correct for the load class

add_dhw(*dhw: float*) → None

This function adds the domestic hot water (dhw). An error is raised if the dhw is not positive.

Parameters

dhw

[float] Yearly consumption of domestic hot water [kWh/year]

Returns

None

property all_months_equal: bool

Returns the attribute all months are equal

Returns

bool

True if the months are assumed to be of equal length (i.e. 730 hours/month). False if the correct number of hours is used.

abstract baseload_cooling() → ndarray

This function returns the baseload cooling in kWh/month.

Returns

baseload cooling

[np.ndarray]

property baseload_cooling_power: ndarray

This function returns the baseload cooling in kW avg/month.

Returns

baseload cooling

[np.ndarray]

property baseload_cooling_power_simulation_period: ndarray

This function returns the average cooling power in kW avg/month for a whole simulation period.

Returns

average cooling power

[np.ndarray] average cooling for the whole simulation period

property baseload_cooling_simulation_period: ndarray

This function returns the baseload cooling in kWh/month for a whole simulation period.

Returns

baseload cooling

[np.ndarray] baseload cooling for the whole simulation period

abstract baseload_heating() → ndarray

This function returns the baseload heating in kWh/month.

Returns

baseload heating

[np.ndarray]

property baseload_heating_power: ndarray

This function returns the baseload heating in kW avg/month.

Returns

baseload heating

[np.ndarray]

property baseload_heating_power_simulation_period: ndarray

This function returns the average heating power in kW avg/month for a whole simulation period.

Returns

average heating power

[np.ndarray] average heating power for the whole simulation period

property baseload_heating_simulation_period: ndarray

This function returns the baseload heating in kWh/month for a whole simulation period.

Returns

baseload heating

[np.ndarray] baseload heating for the whole simulation period

abstract correct_for_start_month(array: ndarray) → ndarray

This function corrects the load for the correct start month. If the simulation starts in september, the start month is 9 and hence the array should start at index 9.

Parameters

array

[np.ndarray] Load array

Returns

load

[np.ndarray]

property dhw: float

This function returns the yearly domestic hot water consumption.

Returns

dhw

[float] Yearly domestic hot water consumption [kWh/year]

property dhw_power: float

This function returns the power related to the dhw production.

Returns

dhw power
[float]

static get_month_index(peak_load, avg_load) → int

This function calculates and returns the month index (i.e. the index of the month in which the field should be sized). It does so by taking 1) the month with the highest peak load. 2) if all the peak loads are the same, it takes the month with the highest average load 3) if all average loads are the same, it takes the last month

Parameters

peak_load
[np.ndarray] array with the peak loads [kW]
avg_load
[np.ndarray] array with the monthly average loads [kW]

Returns

month_index
[int] 0 = jan, 1 = feb ...

property imbalance: float

This function calculates the ground imbalance. A positive imbalance means that the field is injection dominated, i.e. it heats up every year.

Returns

imbalance
[float]

property max_peak_cooling: float

This returns the max peak cooling in kW.

Returns

max peak cooling
[float]

property max_peak_heating: float

This returns the max peak heating in kW.

Returns

max peak heating
[float]

property monthly_average_load: ndarray

This function calculates the average monthly load in kW.

Returns

monthly average load
[np.ndarray]

property monthly_average_load_simulation_period: ndarray

This function calculates the average monthly load in kW for the whole simulation period.

Returns

monthly average load

[np.ndarray]

abstract peak_cooling() → ndarray

This function returns the peak cooling load in kW/month.

Returns

peak cooling

[np.ndarray]

property peak_cooling_duration: float

Duration of the peak in cooling.

Returns

Duration of the peak in cooling [s]

property peak_cooling_simulation_period: ndarray

This function returns the peak cooling in kW/month for a whole simulation period.

Returns

peak cooling

[np.ndarray] peak cooling for the whole simulation period

property peak_duration: None

Dummy object to set the length peak for both heating and cooling.

Returns

None

abstract peak_heating() → ndarray

This function returns the peak heating load in kW/month.

Returns

peak heating

[np.ndarray]

property peak_heating_duration: float

Length of the peak in heating.

Returns

Length peak in heating [s]

property peak_heating_simulation_period: ndarray

This function returns the peak heating in kW/month for a whole simulation period.

Returns

peak heating

[np.ndarray] peak heating for the whole simulation period

property start_month: int

This function returns the start month.

Returns

float

Start month

property time_L3: ndarray

Time for L3 sizing, i.e. an array with monthly the cumulative seconds that have passed. [744, 1416 ...] * 3600

Returns**Times for the L3 sizing**

[np.ndarray]

property time_L4: ndarray

Times for the L4 sizing, i.e. an array with hourly the cumulative seconds that have passed. [1, 2, 3, 4 ...] * 3600

Returns**Times for the L4 sizing**

[np.ndarray]

property ty: float

Simulation period in seconds.

Returns**Simulation period in seconds****property yearly_cooling_load: float**

This function returns the yearly cooling load in kWh/year.

Returns**float**

Yearly cooling load kWh/year

property yearly_heating_load: float

This function returns the yearly heating load in kWh/year.

Returns**float**

Yearly heating load kWh/year

This class contains all the information for geothermal load data with a monthly resolution and absolute input. This means that the inputs are both in kWh/month and kW/month. This class contains all the information for geothermal load data with a monthly resolution and absolute input. This means that the inputs are both in kWh/month and kW/month.

This class contains all the information for geothermal load data with a monthly resolution and absolute input. This means that the inputs are both in kWh/month and kW/month.

Pipe data

GHEtool supports both U-type and coaxial type heat exchangers. You can use:

1. Multiple U-tubes
2. Single U-tubes (special case of multiple U-tubes)
3. Double U-tubes (special case of multiple U-tubes)
4. Coaxial pipe

All of the classes above are children from the abstract `_PipeData` class.

```
class GHEtool.VariableClasses.PipeData._PipeData._PipeData(k_g: Optional[float] = None, k_p:  
Optional[float] = None, epsilon: float  
= 1e-06)
```

Bases: [BaseClass](#), ABC

Contains information regarding the pipe data of the borefield.

Parameters

k_g
[float] Grout thermal conductivity [W/mK]

k_p
[float] Pipe thermal conductivity [W/mK]

epsilon
[float] Pipe roughness [m]

abstract **Re**(*fluid_data: FluidData*) → float

Reynolds number.

Parameters

fluid_data: FluidData
Fluid data

Returns

Reynolds number
[float]

abstract **calculate_resistances**(*fluid_data: FluidData*) → None

This function calculates the conductive and convective resistances, which are constant.

Parameters

fluid_data
[FluidData] Fluid data

Returns

None

abstract **draw_borehole_internal**(*r_b: float*) → None

This function draws the internal structure of a borehole. This means, it draws the pipes inside the borehole.

Parameters

r_b
[float] Borehole radius [m]

Returns

None

abstract **pipe_model**(*fluid_data: FluidData, k_s: float, borehole: Borehole*) → _BasePipe

This function returns the BasePipe model.

Parameters

fluid_data
[FluidData] Fluid data

k_s
[float] Ground thermal conductivity

borehole
[Borehole] Borehole object

Returns

BasePipe

```
class GHEtool.VariableClasses.PipeData.MultipleUTube.MultipleUTube(k_g: Optional[float] = None,  
r_in: Optional[float] =  
None, r_out: Optional[float]  
= None, k_p: Optional[float]  
= None, D_s:  
Optional[float] = None,  
number_of_pipes: int = 1,  
epsilon: float = 1e-06)
```

Bases: [_PipeData](#)

Contains information regarding the Multiple U-Tube class.

Parameters

k_g
[float] Grout thermal conductivity [W/mK]

r_in
[float] Inner pipe radius [m]

r_out
[float] Outer pipe radius [m]

k_p
[float] Pipe thermal conductivity [W/mK]

D_s
[float] Distance of the pipe until center [m]

number_of_pipes
[int] Number of pipes [#] (single U-tube: 1, double U-tube:2)

epsilon
[float] Pipe roughness [m]

Re(*fluid_data: FluidData*) → float

Reynolds number.

Parameters

fluid_data: FluidData
fluid data

Returns

Reynolds number
[float]

calculate_resistances(*fluid_data: FluidData*) → None

This function calculates the conductive and convective resistances, which are constant.

Parameters

fluid_data
[FluidData] Fluid data

Returns

None

draw_borehole_internal(*r_b*: float) → None

This function draws the internal structure of a borehole. This means, it draws the pipes inside the borehole.

Parameters

r_b
[float] Borehole radius [m]

Returns

None

pipe_model(*fluid_data*: FluidData, *k_s*: float, *borehole*: Borehole) → _BasePipe

This function returns the BasePipe model.

Parameters

fluid_data
[FluidData] Fluid data

k_s
[float] Ground thermal conductivity

borehole
[Borehole] Borehole object

Returns

BasePipe

```
class GHEtool.VariableClasses.PipeData.SingleUTube.SingleUTube(k_g: Optional[float] = None,  
                                                             r_in: Optional[float] = None,  
                                                             r_out: Optional[float] = None,  
                                                             k_p: Optional[float] = None, D_s:  
                                                             Optional[float] = None, epsilon:  
                                                             float = 1e-06)
```

Bases: [MultipleUTube](#)

Class for the single U-Tube borehole.

Parameters

k_g
[float] Grout thermal conductivity [W/mK]

r_in
[float] Inner pipe radius [m]

r_out
[float] Outer pipe radius [m]

k_p
[float] Pipe thermal conductivity [W/mK]

D_s
[float] Distance of the pipe until center [m]

epsilon
[float] Pipe roughness [m]

```
class GHEtool.VariableClasses.PipeData.DoubleUTube.DoubleUTube(k_g: Optional[float] = None,
                                                             r_in: Optional[float] = None,
                                                             r_out: Optional[float] = None,
                                                             k_p: Optional[float] = None, D_s:
                                                             Optional[float] = None, epsilon:
                                                             float = 1e-06)
```

Bases: [MultipleUTube](#)

Class for the double U-Tube borehole.

Parameters

k_g
[float] Grout thermal conductivity [W/mK]

r_in
[float] Inner pipe radius [m]

r_out
[float] Outer pipe radius [m]

k_p
[float] Pipe thermal conductivity [W/mK]

D_s
[float] Distance of the pipe until center [m]

epsilon
[float] Pipe roughness [m]

```
class GHEtool.VariableClasses.PipeData.CoaxialPipe.CoaxialPipe(r_in_in: Optional[float] = None,
                                                             r_in_out: Optional[float] = None,
                                                             r_out_in: Optional[float] = None,
                                                             r_out_out: Optional[float] =
                                                             None, k_p: Optional[float] =
                                                             None, k_g: Optional[float] =
                                                             None, epsilon: float = 1e-06,
                                                             is_inner_inlet: bool = True,
                                                             k_p_out: Optional[float] = None)
```

Bases: [_PipeData](#)

Contains information regarding the Coaxial pipe class.

Parameters

r_in_in
[float] Inner radius of the inner annulus [m]

r_in_out
[float] Outer radius of the inner annulus [m]

r_out_in
[float] Inner radius of the outer annulus [m]

r_out_out
[float] Outer radius of the outer annulus [m]

k_p
[float] Pipe thermal conductivity of the inner and outer pipe [W/mK]. If *k_p_out* is set, *k_p* is only used for the conductivity of the inner pipe.

k_g
[float] Thermal conductivity of the grout [W/mK]

epsilon
[float] Pipe roughness of the tube [m]

is_inner_inlet
[bool] True if the inlet of the fluid is through the inner annulus

k_p_out
[float] Pipe conductivity of the outer pipe [W/mK]. If None, it is assumed that the outer pipe has the same conductivity as the inner pipe (k_p).

Re(*fluid_data*: FluidData) → float

Reynolds number. Note: This code is based on pygfunction, 'convective_heat_transfer_coefficient_concentric_annulus' in the Pipes class.

Parameters

fluid_data: FluidData
fluid data

Returns

Reynolds number
[float]

calculate_resistances(*fluid_data*: FluidData) → None

This function calculates the conductive and convective resistances, which are constant.

Parameters

fluid_data
[FluidData] Fluid data

Returns

None

draw_borehole_internal(*r_b*: float) → None

This function draws the internal structure of a borehole. This means, it draws the pipes inside the borehole.

Parameters

r_b
[float] Borehole radius [m]

Returns

None

pipe_model(*fluid_data*: FluidData, *k_s*: float, *borehole*: Borehole) → _BasePipe

This function returns the BasePipe model.

Parameters

fluid_data
[FluidData] Fluid data

k_s
[float] Ground thermal conductivity

borehole
[Borehole] Borehole object

Returns**BasePipe****CustomGFunction**

This file contains both the CustomGFunction class and all the relevant information w.r.t. custom gfunctions.

```
class GHEtool.VariableClasses.CustomGFunction.CustomGFunction(time_array: Optional[ndarray] =
                                                                None, depth_array:
                                                                Optional[ndarray] = None,
                                                                options: Optional[dict] = None)
```

Bases: object

This class contains all the functionalities related to custom gfunctions.

Parameters**time_array**

[np.ndarray] Time value(s) in seconds at which the gfunctions should be calculated

depth_array

[np.ndarray] Depths [m] for which the gfunctions should be calculated

options

[dict] Dictionary with options for the gFunction class of pygfunction

```
calculate_gfunction(time_value: Union[list, float, ndarray], H: float, check: bool = False) → ndarray
```

This function returns the gfunction value, based on interpolation between precalculated values.

Parameters**time_value**

[list, float, np.ndarray] Time value(s) in seconds at which the gfunctions should be calculated

H

[float] Depth [m] at which the gfunctions should be calculated. If no depth is given, the current depth is taken.

check

[bool] True if it should be check whether or not the requested gvalues can be interpolated based on the precalculated values

Returns**gvalues**

[np.ndarray] 1D array with all the requested gvalues. False is returned if the check is True and the requested values are out of range for interpolation

```
create_custom_dataset(borefield: List[Borehole], alpha: Union[float, callable]) → None
```

This function creates the custom dataset.

Parameters**borefield**

[list[pygfunction.boreholes.Borehole]] Borefield object for which the custom dataset should be created

alpha

[float or callable] Ground thermal diffusivity [m2/s] or function to calculate it at a certain depth

Returns**None****delete_custom_gfunction()** → None

This function deletes the custom gfunction.

Returns**None****dump_custom_dataset**(*path: str, name: str*) → None

This function dumps the current custom dataset.

Parameters**path**

[str] Location where the dataset should be saved

name

[str] Name under which the dataset should be saved

Returns**None****set_options_gfunction_calculation**(*options: dict*) → None

This function sets the options for the gfunction calculation of pygfunction. This dictionary is directly passed through to the gFunction class of pygfunction. For more information, please visit the documentation of pygfunction.

Parameters**options**

[dict] Dictionary with options for the gFunction class of pygfunction

Returns**None****within_range**(*time_value: Union[list, float, ndarray], H: float*) → bool

This function checks whether or not the requested data can be calculated using the custom dataset.

Parameters**time_value**

[list, float, np.ndarray] Time value(s) in seconds at which the gfunctions should be calculated

H

[float] Depth [m] at which the gfunctions should be calculated. If no depth is given, the current depth is taken.

Returns**bool**

True if the requested values are within the range of the precalculated data, False otherwise

GHEtool.VariableClasses.CustomGFunction.**load_custom_gfunction**(*path: str*) → *CustomGFunction*

This function loads a custom gfunction dataset.

Parameters**path**

[str] Location of the dataset

Returns**CustomGFunction**

Dataset with the custom gfunction data

GFunction

class GHEtool.VariableClasses.GFunction.FIFO(*length: int = 2*)

Bases: object

This class is a container with n elements. If the n+1th element is added, the first is removed

Parameters**length**

[int] Length of the fifo-array

add(*value: float*) → None

This function adds the value to the fifo array. If the array is full, the first element is removed.

Parameters**value**

[float] Value to be added to the array

Returns

None

clear() → None

This function clears the fifo_array.

Returns

None

in_fifo_list(*value: float*) → bool

This function checks whether the value is in the fifo list, but not the last element!

Parameters**value**

[float] Value potentially in the fifo list

Returns**bool**

True if the value is in the fifo list, false otherwise

class GHEtool.VariableClasses.GFunction.GFunction

Bases: object

Class that contains the functionality to calculate gfunctions and to store previously calculated values that can potentially be used for interpolation to save time. This is done by storing previously calculated gvalues.

calculate(*time_value: Union[list, float, ndarray]*, *borefield: List[Borehole]*, *alpha: float*, *interpolate: Optional[bool] = None*)

This function returns the gvalues either by interpolation or by calculating them. It does so by calling the function gvalues which does this calculation. This calculation function also stores the previous calculated data and makes interpolations whenever the requested list of time_value are longer then DEFAULT_NUMBER_OF_TIMESTEPS.

Parameters**time_value**

[list, float, np.ndarray] Array with all the time values [s] for which gvalues should be calculated

borefield

[list[pygfunction.boreholes.Borehole]] Borefield model for which the gvalues should be calculated

alpha

[float] Thermal diffusivity of the ground [m²/s]

interpolate

[bool] True if results should be interpolated when possible, False otherwise. If None, the default is chosen.

Returns**gvalues**

[np.ndarray] 1D array with all the requested gvalues

interpolate_gfunctions(*time_value*: Union[list, float, ndarray], *depth*: float, *alpha*: float, *borefield*: List[Borehole]) → ndarray

This function returns the gvalues by interpolation them. If interpolation is not possible, an empty array is returned.

Parameters**time_value**

[list, float, np.ndarray] Time value(s) [s] for which gvalues should be calculated

depth

[float] Depth of the borefield [m]

alpha

[float] Thermal diffusivity of the ground [m²/s]

borefield

[list[pygfunction.boreholes.Borehole]] Borefield model for which the gvalues should be calculated

Returns**gvalues**

[np.ndarray] 1D array with all the requested gvalues

remove_previous_data() → None

This function removes the previous calculated data by setting the depth_array, time_array and previous_gfunctions back to empty arrays.

Returns**None**

set_new_calculated_data(*time_values*: ndarray, *depth*: float, *gvalues*: ndarray, *borefield*, *alpha*) → bool

This function stores the newly calculated gvalues if this is needed.

Parameters**time_values**

[np.ndarray] Array with all the time values [s] for which gvalues should be calculated

depth

[float] Depth of the borefield [m]

gvalues

[np.ndarray] Array with all the calculated gvalues for the corresponding borefield, alpha and time_values

borefield

[list[pygfunction.borehole]] Borefield model for which the gvalues should be calculated

alpha

[float] Thermal diffusivity of the ground [m2/s]

Returns**bool**

True if the data is saved, False otherwise

set_options_gfunction_calculation(*options: dict, add: bool = True*) → None

This function sets the options for the gfunction calculation of pygfunction. This dictionary is directly passed through to the gFunction class of pygfunction. For more information, please visit the documentation of pygfunction.

Parameters**options**

[dict] Dictionary with options for the gFunction class of pygfunction

add

[bool] True if the options should be added, False if the options should be replaced.

Returns**None****property store_previous_values: bool**

This returns the truth value of the store_previous_values attribute.

Returns**bool**

True if the previously calculated gfunction values should be saved.

SizingSetup**ghe_logger**

Script to create the GHEtool logger for the console and the text file

```
class GHEtool.logger.ghe_logger.CustomFormatter(fmt: str)
```

Bases: Formatter

Class to create a special console coloring of the messages

Parameters**fmt**

[str] Format of the log message

format(*record: LogRecord*) → str

Formats the record.

Parameters

record: logging.LogRecord
record to be formatted

Returns

str
Formatted log message

GHEtool.logger.ghe_logger.**addLoggingLevel**(*levelName, levelNum, methodName=None*)

Comprehensively adds a new logging level to the *logging* module and the currently configured logging class.

levelName becomes an attribute of the *logging* module with the value *levelNum*. *methodName* becomes a convenience method for both *logging* itself and the class returned by *logging.getLoggerClass()* (usually just *logging.Logger*). If *methodName* is not specified, *levelName.lower()* is used.

To avoid accidental clobberings of existing attributes, this method will raise an *AttributeError* if the level name is already an attribute of the *logging* module or if the method name is already present

1.2.9 Examples

Combination of active and passive cooling

```

1  """
2  This file contains an example on how GHEtool can be used to size a borefield
3  using a combination of active and passive cooling.
4  This example is based on the work of Coninx and De Nies, 2021.
5  Coninx, M., De Nies, J. (2022). Cost-efficient Cooling of Buildings by means of ↵
   ↵Borefields
6  with Active and Passive Cooling. Master thesis, Department of Mechanical Engineering, KU ↵
   ↵Leuven, Belgium.
7  It is also published as: Coninx, M., De Nies, J., Hermans, L., Peere, W., Boydens, W., ↵
   ↵Helsen, L. (2024).
8  Cost-efficient cooling of buildings by means of geothermal borefields with active and ↵
   ↵passive cooling.
9  Applied Energy, 355, Art. No. 122261, https://doi.org/10.1016/j.apenergy.2023.122261.
10 """
11
12 from GHEtool import Borefield, GroundConstantTemperature, HourlyGeothermalLoadMultiYear
13
14 import pandas as pd
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from skopt import gp_minimize
18
19
20 def active_passive_cooling(location='Active_passive_example.csv'):
21
22     # load data
23     columnNames = ['HeatingSpace', 'HeatingAHU', 'CoolingSpace', 'CoolingAHU']
24     df = pd.read_csv(location, names=columnNames, header=0)

```

(continues on next page)

(continued from previous page)

```

25 heating_data = df.HeatingSpace + df.HeatingAHU
26 cooling_data = df.CoolingSpace + df.CoolingAHU
27
28 # variable COP and EER data
29 COP = [0.122, 4.365] # ax+b
30 EER = [-3.916, 17.901] # ax+b
31 threshold_active_cooling = 16
32
33 # set simulation period
34 SIMULATION_PERIOD: int = 50
35 heating_building: np.ndarray = np.tile(np.array(heating_data), SIMULATION_PERIOD)
36 cooling_building: np.ndarray = np.tile(np.array(cooling_data), SIMULATION_PERIOD)
37
38 def update_load_COP(temp_profile: np.ndarray,
39                    COP: np.ndarray,
40                    load_profile: np.ndarray) -> np.ndarray:
41     """
42     This function updates the geothermal load for heating based on a variable COP
43
44     Parameters
45     -----
46     temp_profile : np.ndarray
47         Temperature profile of the fluid
48     COP : np.ndarray
49         Variable COP i.f.o. temperature
50     load_profile : np.ndarray
51         Heating load of the building
52
53     Returns
54     -----
55     Geothermal heating load : np.ndarray
56     """
57     COP_array = temp_profile * COP[0] + COP[1]
58     return load_profile * (1 - 1/COP_array)
59
60
61 def update_load_EER(temp_profile: np.ndarray,
62                    EER: np.ndarray,
63                    threshold_active_cooling: float,
64                    load_profile: np.ndarray) -> np.ndarray:
65     """
66     This function updates the geothermal load for cooling based on a threshold for
67     ↪ active/passive cooling,
68     and a variable EER.
69
70     Parameters
71     -----
72     temp_profile : np.ndarray
73         Temperature profile of the fluid
74     EER : np.ndarray
75         Variable EER i.f.o. temperature
76     threshold_active_cooling : float

```

(continues on next page)

(continued from previous page)

```

76         Threshold of the temperature above which active cooling is needed
77         load_profile : np.ndarray
78         Cooling load of the building
79
80     Returns
81     -----
82     Geothermal cooling load : np.ndarray
83     """
84     EER_array = temp_profile * EER[0] + EER[1]
85     passive: np.ndarray = temp_profile < threshold_active_cooling
86     active = np.invert(passive)
87     return active * load_profile * (1 + 1/EER_array) + passive * load_profile
88
89
90     costs = {"C_elec": 0.2159,      # electricity cost (EUR/kWh)
91             "C_borefield": 35,     # inv cost per m borefield (EUR/m)
92             "DR": 0.0011,         # discount rate(-)
93             "sim_period": SIMULATION_PERIOD}
94
95
96     def calculate_costs(borefield: Borefield, heating_building: np.ndarray, heating_
97     ↪geothermal: np.ndarray,
98                          cooling_building: np.ndarray, cooling_geothermal: np.ndarray,
99     ↪costs: dict) -> tuple:
100         """
101         This function calculates the relevant costs for the borefield.
102
103         Parameters
104         -----
105         borefield : Borefield
106             Borefield object
107         heating_building : np.ndarray
108             Heating demand for the building
109         heating_geothermal : np.ndarray
110             Heating demand coming from the ground
111         cooling_building : np.ndarray
112             Cooling demand for the building
113         cooling_geothermal : np.ndarray
114             Cooling demand coming from the ground
115         costs : dict
116             Dictionary with investment cost for borefield/m, electricity cost, annual_
117     ↪discount rate
118
119         Returns
120         -----
121         investment cost borefield, operational cost heating, operational cost cooling,
122     ↪total operational cost:
123         float, float, float, float
124         """
125         # calculate investment cost
126         investment_borefield = costs["C_borefield"] * borefield.H * borefield.number_of_
127     ↪boreholes

```

(continues on next page)

(continued from previous page)

```

123
124     # calculate working costs
125     elec_heating = heating_building - heating_geothermal
126     elec_cooling = cooling_geothermal - cooling_building
127
128     discounted_cooling_cost = []
129     discounted_heating_cost = []
130     for i in range(SIMULATION_PERIOD):
131         tempc = costs["C_elec"] * (elec_cooling[730 * 12 * i:730 * 12 * (i + 1)])
132         tempc = tempc * (1 / (1 + costs["DR"])) ** (i + 1)
133
134         temp_h = costs["C_elec"] * (elec_heating[730 * 12 * i:730 * 12 * (i + 1)])
135         temp_h = temp_h * (1 / (1 + costs["DR"])) ** (i + 1)
136         discounted_cooling_cost.append(tempc)
137         discounted_heating_cost.append(temp_h)
138     cost_cooling = np.sum(discounted_cooling_cost)
139     cost_heating = np.sum(discounted_heating_cost)
140
141     return investment_borefield, cost_heating, cost_cooling, cost_heating+cost_
↪cooling
142
143     borefield = Borefield()
144     borefield.simulation_period = SIMULATION_PERIOD
145     borefield.set_max_avg_fluid_temperature(17)
146
147     borefield.create_rectangular_borefield(12, 12, 6, 6, 100)
148     borefield.set_ground_parameters(GroundConstantTemperature(2.1, 11))
149     borefield.Rb = 0.12
150
151     ### PASSIVE COOLING
152     depths = [0.9, 0]
153
154     # set initial loads
155     cooling_ground = cooling_building.copy()
156     heating_ground = heating_building.copy()
157
158     while abs(depths[0] - depths[1]) > 0.1:
159         # set loads
160         load = HourlyGeothermalLoadMultiYear()
161         load.hourly_heating_load = heating_ground
162         load.hourly_cooling_load = cooling_ground
163         borefield.load = load
164
165         # size borefield
166         depth_passive = borefield.size_L4()
167         depths.insert(0, depth_passive)
168
169         # get temperature profile
170         temp_profile = borefield.results.peak_heating
171
172         # recalculate heating load
173         heating_ground = update_load_COP(temp_profile, COP, heating_building)

```

(continues on next page)

(continued from previous page)

```

174
175
176 ### ACTIVE COOLING
177 depths = [0.9, 0]
178
179 # set initial loads
180 cooling_ground = cooling_building.copy()
181 heating_ground = heating_building.copy()
182
183 borefield.set_max_avg_fluid_temperature(25)
184 while abs(depths[0] - depths[1]) > 0.1:
185     # set loads
186     load = HourlyGeothermalLoadMultiYear()
187     load.hourly_heating_load = heating_ground
188     load.hourly_cooling_load = cooling_ground
189     borefield.load = load
190
191     # size borefield
192     depth_active = borefield.size_L4()
193     depths.insert(0, depth_active)
194
195     # get temperature profile
196     temp_profile = borefield.results.peak_heating
197
198     # recalculate heating load
199     heating_ground = update_load_COP(temp_profile, COP, heating_building)
200     cooling_ground = update_load_EER(temp_profile, EER, 16, cooling_building)
201
202
203 ### RUN OPTIMISATION
204
205 # initialise parameters
206 operational_costs = []
207 operational_costs_cooling = []
208 operational_costs_heating = []
209 investment_costs = []
210 total_costs = []
211 depths = []
212
213
214 def f(depth: list) -> float:
215     """
216     Optimisation function.
217
218     Parameters
219     -----
220     depth : list
221         List with one element: the depth of the borefield in mm
222
223     Returns
224     -----
225     total_cost : float

```

(continues on next page)

(continued from previous page)

```

226     """
227     # convert to meters
228     depth = depth[0] / 1000
229     borefield._update_borefield_depth(depth)
230     borefield.H = depth
231     depths.append(depth)
232
233     # initialise
234     heating_ground = heating_building.copy()
235     cooling_ground = cooling_building.copy()
236
237     heating_ground_prev = np.zeros(len(heating_ground))
238     cooling_ground_prev = np.zeros(len(cooling_ground))
239
240     # iterate until convergence in the load
241     while np.sum(cooling_ground + heating_ground - heating_ground_prev - cooling_
↪ground_prev) > 100:
242         # set loads
243         load = HourlyGeothermalLoadMultiYear()
244         load.hourly_heating_load = heating_ground
245         load.hourly_cooling_load = cooling_ground
246         borefield.load = load
247
248         # get temperature profile
249         borefield.calculate_temperatures(depth, hourly=True)
250         temp_profile = borefield.results.peak_heating
251
252         # set previous loads
253         heating_ground_prev = heating_ground.copy()
254         cooling_ground_prev = cooling_ground.copy()
255
256         # recalculate heating load
257         heating_ground = update_load_COP(temp_profile, COP, heating_building)
258         cooling_ground = update_load_EER(temp_profile, EER, 16, cooling_building)
259
260         # calculate costs
261         investment, cost_heating, cost_cooling, operational_cost = calculate_
↪costs(borefield,
262                                             heating_
↪building, heating_ground,
263                                             cooling_
↪building, cooling_ground,
264                                             costs)
265         total_costs.append(investment + operational_cost)
266         operational_costs.append(operational_cost)
267         operational_costs_cooling.append(cost_cooling)
268         operational_costs_heating.append(cost_heating)
269         investment_costs.append(investment)
270         return investment + operational_cost
271
272     # add boundaries to figure
273     # multiply with 1000 for numerical stability

```

(continues on next page)

(continued from previous page)

```

274     f([depth_active * 10 ** 3])
275     f([depth_passive * 10 ** 3])
276
277     res = gp_minimize(f, # the function to minimize
278                      [(depth_active * 10 ** 3, depth_passive * 10 ** 3)], # the bounds_
↳ on each dimension of x
279                      acq_func="EI", # the acquisition function
280                      n_calls=30, # the number of evaluations of f
281                      initial_point_generator="lhs",
282                      n_random_starts=15, # the number of random initialization points
283                      # noise=0, # the noise level (optional)
284                      random_state=1234) # the random seed
285
286     # plot figures
287     fig = plt.figure()
288     ax1 = fig.add_subplot(111)
289     ax1.plot(depths, [x/1000 for x in total_costs], marker = 'o', label = "TC")
290     ax1.plot(depths, [x/1000 for x in investment_costs], marker = 'o', label="IC")
291     ax1.plot(depths, [x/1000 for x in operational_costs], marker = 'o', label="OC")
292     ax1.plot(depths, [x/1000 for x in operational_costs_cooling], marker='o', label="OCc
↳ ")
293     ax1.plot(depths, [x/1000 for x in operational_costs_heating], marker='o', label="OCh
↳ ")
294     ax1.set_xlabel(r'Depth (m)', fontsize=14)
295     ax1.set_ylabel(r'Costs ($k€$)', fontsize=14)
296     ax1.legend(loc='lower left', ncol=3)
297     ax1.tick_params(labelsize=14)
298     plt.show()
299
300
301 if __name__ == "__main__": # pragma: no cover
302     active_passive_cooling()

```

Different borehole configurations

```

1  """
2  This document is an example of how the borefield configuration can influence the total_
↳ borehole length and hence the cost of the borefield.
3  """
4
5  # import all the relevant functions
6  from GHEtool import GroundConstantTemperature, Borefield, MonthlyGeothermalLoadAbsolute
7  import numpy as np
8  import pygfunction as gt
9
10
11 def effect_borefield_configuration():
12     # GroundData for an initial field of 11 x 11
13     data = GroundConstantTemperature(3, 10)
14     borefield_gt = gt.boreholes.rectangle_field(11, 11, 6, 6, 110, 1, 0.075)

```

(continues on next page)

(continued from previous page)

```

15
16     # Monthly loading values
17     peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.
↪]) # Peak cooling in kW
18     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↪]) # Peak heating in kW
19
20     # annual heating and cooling load
21     annual_heating_load = 150 * 10 ** 3 # kWh
22     annual_cooling_load = 400 * 10 ** 3 # kWh
23
24     # percentage of annual load per month (15.5% for January ...)
25     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
↪0., 0.061, 0.087, 0.117, 0.144])
26     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪.1, .075, .05, .025])
27
28     # resulting load per month
29     monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
30     monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh
31
32     # set the load
33     load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
↪peak_heating, peak_cooling)
34
35     # create the borefield object
36     borefield = Borefield(load=load)
37
38     borefield.set_ground_parameters(data)
39     borefield.set_borefield(borefield_gt)
40     borefield.Rb = 0.2
41
42     # set temperature boundaries
43     borefield.set_max_avg_fluid_temperature(16) # maximum temperature
44     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
45
46     # size borefield
47     depth = borefield.size()
48     print("The borehole depth is:", depth, "m for a 11x11 field")
49     print("The total length is:", int(depth * 11 * 11), "m")
50     print("-----")
51
52
53     # borefield of 6x20
54     data = GroundConstantTemperature(3, 10)
55     borefield_gt = gt.boreholes.rectangle_field(6, 20, 6, 6, 110, 1, 0.075)
56
57     # set ground parameters to borefield
58     borefield.set_borefield(borefield_gt)
59     borefield.set_ground_parameters(data)
60
61     # set Rb

```

(continues on next page)

(continued from previous page)

```

62     borefield.Rb = 0.2
63
64     # size borefield
65     depth6_20 = borefield.size()
66     print("The borehole depth is:", depth6_20, "m for a 6x20 field")
67     print("The total length is:", int(depth6_20 * 6 * 20), "m")
68     print("The second field is hence", -int(depth6_20 * 6 * 20) + int(depth * 11 * 11),
69     ↪ "m shorter")
69
70     borefield.print_temperature_profile()
71
72
73 if __name__ == "__main__": # pragma: no cover
74     effect_borefield_configuration()

```

Start simulation in a different month

```

1  """
2  This example illustrates the importance of when a borefield is 'started' (i.e. when the
3  ↪ first month of operation is).
4  """
5
6  from GHEtool import *
7  from GHEtool.Validation.cases import load_case
8
9  import matplotlib.pyplot as plt
10
11 def start_in_different_month():
12     # set data
13     ground_data = GroundTemperatureGradient(2.5, 10)
14     load = MonthlyGeothermalLoadAbsolute(*load_case(1))
15
16     # create borefield object
17     borefield = Borefield(load=load)
18     borefield.ground_data = ground_data
19     borefield.create_rectangular_borefield(10, 8, 6, 6, 100)
20
21     borefield.set_max_avg_fluid_temperature(17)
22     borefield.set_min_avg_fluid_temperature(3)
23     borefield.calculation_setup(max_nb_of_iterations=100)
24
25     depth_list = []
26
27     # iterate over all the start months
28     for month in range(1, 13, 1):
29         borefield.load.start_month = month
30         depth_list.append(borefield.size_L3())
31
32     plt.figure()

```

(continues on next page)

(continued from previous page)

```

33 plt.bar(range(1, 13, 1), depth_list)
34 plt.ylabel('Required depth [m]')
35 plt.xlabel('First month of operation')
36 plt.xlim(0)
37 plt.ylim(0)
38 plt.title('Required depth as a function of the first month of operation')
39 plt.show()
40
41
42 if __name__ == "__main__": # pragma: no cover
43     start_in_different_month()

```

Work with multiple ground layers

```

1  """
2  This example sizes a borefield using the advanced option of using multiple ground layers.
3  """
4
5  from GHEtool import *
6  from GHEtool.Validation.cases import load_case
7
8  def multiple_ground_layers():
9      # initiate borefield model
10     borefield = Borefield()
11     borefield.create_rectangular_borefield(10, 10, 6, 6, 110, 1, 0.075)
12     borefield.set_Rb(0.12)
13
14     # set temperature boundaries
15     borefield.set_max_avg_fluid_temperature(16) # maximum temperature
16     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
17     borefield.load = MonthlyGeothermalLoadAbsolute(*load_case(4))
18
19     # create two ground classes
20     constant_ks = GroundFluxTemperature(1.7, 10)
21
22     layer_1 = GroundLayer(k_s=1.7, thickness=4.9)
23     layer_2 = GroundLayer(k_s=2.3, thickness=1.9)
24     layer_3 = GroundLayer(k_s=2.1, thickness=3)
25     layer_4 = GroundLayer(k_s=1.5, thickness=69.7)
26     layer_5 = GroundLayer(k_s=2.1, thickness=16.1)
27     layer_6 = GroundLayer(k_s=1.7, thickness=None)
28
29     layered_ground = GroundFluxTemperature(T_g=10)
30     layered_ground.add_layer_on_bottom([layer_1, layer_2, layer_3, layer_4, layer_5,
31     ↪ layer_6])
32
33     # size borefield according to the two different ground data variables
34     borefield.ground_data = constant_ks
35     print(f'The required borehole depth is {borefield.size():.3f}m if you use a constant_
36     ↪ approximation for the ground conductivity.')

```

(continues on next page)

(continued from previous page)

```

35     borefield.ground_data = layered_ground
36     print(f'The required borehole depth is {borefield.size():.3f}m if you use a detailed_
37     ↳ground model with multiple layers.')
38
39
40 if __name__ == "__main__": # pragma: no cover
41     multiple_ground_layers()

```

1.2.10 Validation

GHEtool is validated in a couple of ways. The goal is to increase the number of validation documents in the future.

First of all, it is internally checked for coherence, meaning that different methodology give (more or less) the same result. Examples of this can be found in the validation files below.

Sizing method comparison (L2/L3)

```

1  """
2  This document compares both the L2 sizing method of (Peere et al., 2021) with a more_
3  ↳general L3 sizing.
4  The comparison is based on speed and relative accuracy in the result.
5  """
6
7  import time
8
9  import numpy as np
10 import pygfunction as gt
11
12 from GHEtool import Borefield, GroundConstantTemperature, MonthlyGeothermalLoadAbsolute
13
14 def sizing_method_comparison():
15     number_of_iterations = 50
16     max_value_cooling = 700
17     max_value_heating = 800
18
19     # initiate the arrays
20     results_L2 = np.zeros(number_of_iterations)
21     results_L3 = np.zeros(number_of_iterations)
22     difference_results = np.zeros(number_of_iterations)
23
24     monthly_load_cooling_array = np.empty((number_of_iterations, 12))
25     monthly_load_heating_array = np.empty((number_of_iterations, 12))
26     peak_load_cooling_array = np.empty((number_of_iterations, 12))
27     peak_load_heating_array = np.empty((number_of_iterations, 12))
28
29     # populate arrays with random values
30     for i in range(number_of_iterations):
31         for j in range(12):

```

(continues on next page)

(continued from previous page)

```

32         monthly_load_cooling_array[i, j] = np.random.randint(0, max_value_cooling)
33         monthly_load_heating_array[i, j] = np.random.randint(0, max_value_heating)
34         peak_load_cooling_array[i, j] = np.random.randint(monthly_load_cooling_
↪ array[i, j], max_value_cooling)
35         peak_load_heating_array[i, j] = np.random.randint(monthly_load_heating_
↪ array[i, j], max_value_heating)
36
37     # initiate borefield model
38     data = GroundConstantTemperature(3, 10)
39     borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 110, 1, 0.075)
40
41     # Monthly loading values
42     peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.
↪ ]) # Peak cooling in kW
43     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↪ ]) # Peak heating in kW
44
45     # annual heating and cooling load
46     annual_heating_load = 300 * 10 ** 3 # kWh
47     annual_cooling_load = 160 * 10 ** 3 # kWh
48
49     # percentage of annual load per month (15.5% for January ...)
50     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0., 0.,
↪ 0., 0.061, 0.087, 0.117, 0.144])
51     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪ .1, .075, .05, .025])
52
53     # resulting load per month
54     monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
55     monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh
56
57     # set the load
58     load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
↪ peak_heating, peak_cooling)
59
60     # create the borefield object
61     borefield = Borefield(load=load)
62     borefield.set_ground_parameters(data)
63     borefield.set_borefield(borefield_gt)
64     borefield.Rb = 0.2
65
66     # set temperature boundaries
67     borefield.set_max_avg_fluid_temperature(16) # maximum temperature
68     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
69
70     # size according to L2 method
71     start_L2 = time.time()
72     for i in range(number_of_iterations):
73         # set the load
74         load = MonthlyGeothermalLoadAbsolute(monthly_load_heating_array[i], monthly_load_
↪ cooling_array[i],
75                                             peak_load_heating_array[i], peak_load_

```

(continues on next page)

(continued from previous page)

```

↪cooling_array[i])
    borefield.load = load
    results_L2[i] = borefield.size(L2_sizing=True)
    end_L2 = time.time()

    # size according to L3 method
    start_L3 = time.time()
    for i in range(number_of_iterations):
        # set the load
        load = MonthlyGeothermalLoadAbsolute(monthly_load_heating_array[i], monthly_load_
↪cooling_array[i],
        peak_load_heating_array[i], peak_load_
↪cooling_array[i])
        borefield.load = load
        results_L3[i] = borefield.size(L3_sizing=True)
        end_L3 = time.time()

    print("Time for sizing according to L2:", end_L2 - start_L2, "s (or ", round((end_L2_
↪start_L2) / number_of_iterations * 1000, 3), "ms/sizing)")
    print("Time for sizing according to L3:", end_L3 - start_L3, "s (or ", round((end_L3_
↪start_L3) / number_of_iterations * 1000, 3), "ms/sizing)")

    # calculate differences
    for i in range(number_of_iterations):
        difference_results[i] = results_L3[i] - results_L2[i]

    print("The maximal difference between the sizing of L2 and L3 was:", np.round(np.
↪max(difference_results), 3), "m or", np.round(np.max(difference_results) / results_
↪L2[np.argmax(difference_results)] * 100, 3), "% w.r.t. the L2 sizing.")
    print("The mean difference between the sizing of L2 and L3 was:", np.round(np.
↪mean(difference_results), 3), "m or", np.round(np.mean(difference_results) / np.
↪mean(results_L2) * 100, 3), "% w.r.t. the L2 sizing.")

if __name__ == "__main__": # pragma: no cover
    sizing_method_comparison()

```

Sizing method comparison (L2/L3/L4)

```

1  """
2  This document is an example of the different sizing methods in GHEtool.
3  The example load profile is for a profile limited in the first year of operation.
4  """
5  import time
6
7  import numpy as np
8  import pygfunction as gt
9
10 # import all the relevant functions
11 from GHEtool import *

```

(continues on next page)

(continued from previous page)

```

12
13
14 def compare():
15     # initiate ground data
16     data = GroundConstantTemperature(3, 10)
17
18     # initiate borefield
19     borefield = Borefield()
20
21     # set ground data in borefield
22     borefield.set_ground_parameters(data)
23
24     # set Rb
25     borefield.Rb = 0.12
26
27     # set the borefield
28     borefield.create_rectangular_borefield(10, 10, 6, 6, 110, 1, 0.075)
29
30     # load the hourly profile
31     load = HourlyGeothermalLoad()
32     load.load_hourly_profile(FOLDER.joinpath("Examples/hourly_profile.csv"), header=True,
↪ separator=";")
33     borefield.load = load
34     borefield.simulation_period = 100
35
36     ### size the borefield
37     # according to L2
38     L2_start = time.time()
39     depth_L2 = borefield.size(100, L2_sizing=True)
40
41     L2_stop = time.time()
42
43     # according to L3
44     L3_start = time.time()
45     depth_L3 = borefield.size(100, L3_sizing=True)
46     L3_stop = time.time()
47
48     # according to L4
49     L4_start = time.time()
50     depth_L4 = borefield.size(100, L4_sizing=True)
51     L4_stop = time.time()
52
53     ### print results
54     print("The sizing according to L2 took", round((L2_stop-L2_start) * 1000, 4), "ms_
↪ and was", depth_L2, "m.")
55     print("The sizing according to L3 took", round((L3_stop-L3_start) * 1000, 4), "ms_
↪ and was", depth_L3, "m.")
56     print("The sizing according to L4 took", round((L4_stop-L4_start) * 1000, 4), "ms_
↪ and was", depth_L4, "m.")
57
58     borefield.plot_load_duration()
59     borefield.print_temperature_profile(plot_hourly=True)

```

(continues on next page)

(continued from previous page)

```

60
61
62 if __name__ == '__main__':    # pragma: no cover
63     compare()

```

Speed comparison

```

1  """
2  This document compares the speed of the L2 sizing method of (Peere et al., 2021) with
3  and without the precalculated gfunction data.
4  This is done for two fields with different sizes. It shows that, specifically for the
5  larger fields, the precalculated data is way faster.
6  """
7
8  import time
9
10 import pygfunction as gt
11
12 from GHEtool import Borefield, GroundConstantTemperature, MonthlyGeothermalLoadAbsolute
13
14 def test_64_boreholes():
15     data = GroundConstantTemperature(3, 10)
16     borefield_64 = gt.boreholes.rectangle_field(8, 8, 6, 6, 110, 1, 0.075)
17
18     # monthly loading values
19     peak_cooling = [0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.] # Peak
20     cooling in kW
21     peak_heating = [160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.] # Peak
22     heating in kW
23
24     # annual heating and cooling load
25     annual_heating_load = 300 * 10 ** 3 # kWh
26     annual_cooling_load = 160 * 10 ** 3 # kWh
27
28     # percentage of annual load per month (15.5% for January ...)
29     monthly_load_heating_percentage = [0.155, 0.148, 0.125, .099, .064, 0., 0., 0., 0.
30     061, 0.087, 0.117, 0.144]
31     monthly_load_cooling_percentage = [0.025, 0.05, 0.05, .05, .075, .1, .2, .2, .1, .
32     075, .05, .025]
33
34     # resulting load per month
35     monthly_load_heating = list(map(lambda x: x * annual_heating_load, monthly_load_
36     heating_percentage)) # kWh
37     monthly_load_cooling = list(map(lambda x: x * annual_cooling_load, monthly_load_
38     cooling_percentage)) # kWh
39
40     # set the load
41     load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
42     peak_heating, peak_cooling)

```

(continues on next page)

(continued from previous page)

```

35
36 # create the borefield object
37 borefield = Borefield(load=load)
38
39 borefield.set_ground_parameters(data)
40 borefield.set_borefield(borefield_64)
41 borefield.Rb = 0.2
42
43 # set temperature boundaries
44 borefield.set_max_avg_fluid_temperature(16) # maximum temperature
45 borefield.set_min_avg_fluid_temperature(0) # minimum temperature
46
47 # precalculate
48 borefield.create_custom_dataset()
49
50 # size borefield
51 t1 = time.time()
52 depth_precalculated = borefield.size()
53 t1_end = time.time()
54
55 # delete precalculated data
56 borefield.custom_gfunction.delete_custom_gfunction()
57
58 ### size without the precalculation
59 t2 = time.time()
60 depth_calculated = borefield.size()
61 t2_end = time.time()
62
63 print("With precalculated data, the sizing took", round(t1_end - t1, 3), "s for 64_
↳boreholes.")
64 print("Without the precalculated data, the sizing took", round(t2_end - t2, 3), "s_
↳for 64 boreholes.")
65 print("The difference in accuracy between the two results is",
66       round((depth_calculated - depth_precalculated) / depth_calculated * 100, 3), "
↳%.")
67
68
69 def test_10_boreholes():
70     data = GroundConstantTemperature(3, 10)
71     borefield_10 = gt.boreholes.rectangle_field(2, 5, 6, 6, 110, 1, 0.075)
72
73     # monthly loading values
74     peak_cooling = [0., 0, 3., 9., 13., 20., 43., 30., 16., 7., 0., 0.] # Peak cooling_
↳in kW
75     peak_heating = [16., 14, 10., 5., 0., 0., 0., 0., 4, 8., 19., 13.] # Peak heating_
↳in kW
76
77     # annual heating and cooling load
78     annual_heating_load = 16 * 10 ** 3 # kWh
79     annual_cooling_load = 24 * 10 ** 3 # kWh
80
81     # percentage of annual load per month (15.5% for January ...)

```

(continues on next page)

(continued from previous page)

```

82     monthly_heating_load_percentage = [0.155, 0.148, 0.125, .099, .064, 0., 0., 0., 0.
↪ 0.61, 0.087, 0.117, 0.144]
83     monthly_load_cooling_percentage = [0.025, 0.05, 0.05, .05, .075, .1, .2, .2, .1, .
↪ 0.75, .05, .025]
84
85     # resulting load per month
86     monthly_load_heating = list(map(lambda x: x * annual_heating_load, monthly_heating_
↪ load_percentage)) # kWh
87     monthly_load_cooling = list(map(lambda x: x * annual_cooling_load, monthly_load_
↪ cooling_percentage)) # kWh
88
89     # set the load
90     load = MonthlyGeothermalLoadAbsolute(monthly_load_heating, monthly_load_cooling,
↪ peak_heating, peak_cooling)
91
92     # create the borefield object
93     borefield = Borefield(load=load)
94
95     borefield.set_ground_parameters(data)
96     borefield.set_borefield(borefield_10)
97     borefield.Rb = 0.2
98
99     # set temperature boundaries
100    borefield.set_max_avg_fluid_temperature(16) # maximum temperature
101    borefield.set_min_avg_fluid_temperature(0) # minimum temperature
102
103    # precalculate
104    borefield.create_custom_dataset()
105
106    # size borefield
107    t1 = time.time()
108    depth_precalculated = borefield.size()
109    t1_end = time.time()
110
111    # delete precalculated data
112    borefield.custom_gfunction.delete_custom_gfunction()
113
114    ### size without the precalculation
115    t2 = time.time()
116    depth_calculated = borefield.size()
117    t2_end = time.time()
118
119    print("With precalculated data, the sizing took", round(t1_end - t1, 3), "s for 10_
↪ boreholes.")
120    print("Without the precalculated data, the sizing took", round(t2_end - t2, 3), "s_
↪ for 10 boreholes.")
121    print("The difference in accuracy between the two results is",
122          round((depth_calculated - depth_precalculated) / depth_calculated * 100, 3), "%.\n"
↪ )
123
124
125 if __name__ == "__main__": # pragma: no cover

```

(continues on next page)

(continued from previous page)

```

126 test_10_boreholes()
127 test_64_boreholes()
128

```

Peere et al. (2021) validates the hybrid sizing method of GHEtool. The validation code can be found below.

Four different sizing cases

```

1  """
2  This document contains checks to see whether or not adaptations to the code still comply
3  ↳with some specific cases.
4  It also shows the difference between the original L2 sizing methode (Peere et al., 2021)
5  ↳and a more general L3 one.
6
7  This document contains 4 different cases referring to the paper: Peere, W., Picard, D.,
8  ↳Cupeiro Figueroa, I., Boydens, W., and Helsen, L. Validated combined first and last
9  ↳year borefield sizing methodology. In Proceedings of International Building Simulation
10 ↳Conference 2021 (2021). Brugge (Belgium), 1-3 September 2021.
11
12  """
13
14  import numpy as np
15  import pygfunction as gt
16
17  from GHEtool import Borefield, GroundConstantTemperature, MonthlyGeothermalLoadAbsolute
18
19  # relevant borefield data for the calculations
20  data = GroundConstantTemperature(3.5, # conductivity of the soil (W/mK)
21                                  10)  # Ground temperature at infinity (degrees C)
22
23  borefield_gt = gt.boreholes.rectangle_field(10, 12, 6.5, 6.5, 100, 4, 0.075)
24
25  def load_case(number):
26      """This function returns the values for one of the four cases."""
27
28      if number == 1:
29          # case 1
30          # limited in the first year by cooling
31          monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0.,
32 ↳0., 0., 0.061, 0.087, 0.117, 0.144])
33          monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2,
34 ↳.2, .1, .075, .05, .025])
35          monthly_load_heating = monthly_load_heating_percentage * 300 * 10 ** 3 # kWh
36          monthly_load_cooling = monthly_load_cooling_percentage * 150 * 10 ** 3 # kWh
37          peak_cooling = np.array([0., 0., 22., 44., 83., 117., 134., 150., 100., 23., 0.,
38 ↳0.])
39          peak_heating = np.zeros(12)
40
41      elif number == 2:
42          # case 2

```

(continues on next page)

(continued from previous page)

```

36     # limited in the last year by cooling
37     monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0.,
↪ 0., 0., 0.061, 0.087, .117, 0.144])
38     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2,
↪ .2, .1, .075, .05, .025])
39     monthly_load_heating = monthly_load_heating_percentage * 160 * 10 ** 3 # kWh
40     monthly_load_cooling = monthly_load_cooling_percentage * 240 * 10 ** 3 # kWh
41     peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0.,
↪ 0.]) # Peak cooling in kW
42     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119.,
↪ 136.])
43
44     elif number == 3:
45         # case 3
46         # limited in the first year by heating
47         monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0.,
↪ 0., 0., 0.061, 0.087, .117, 0.144])
48         monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2,
↪ .2, .1, .075, .05, .025])
49         monthly_load_heating = monthly_load_heating_percentage * 160 * 10 ** 3 # kWh
50         monthly_load_cooling = monthly_load_cooling_percentage * 240 * 10 ** 3 # kWh
51         peak_cooling = np.zeros(12)
52         peak_heating = np.array([300.0, 266.25, 191.25, 103.125, 0.0, 0.0, 0.0, 0.0, 75.
↪ 75, 159.375, 223.125, 255.0])
53
54     else:
55         # case 4
56         # limited in the last year by heating
57         monthly_load_heating_percentage = np.array([0.155, 0.148, 0.125, .099, .064, 0.,
↪ 0., 0., 0.061, 0.087, 0.117, 0.144])
58         monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2,
↪ .2, .1, .075, .05, .025])
59         monthly_load_heating = monthly_load_heating_percentage * 300 * 10 ** 3 # kWh
60         monthly_load_cooling = monthly_load_cooling_percentage * 150 * 10 ** 3 # kWh
61         peak_cooling = np.array([0., 0., 22., 44., 83., 117., 134., 150., 100., 23., 0.,
↪ 0.])
62         peak_heating = np.array([300., 268., 191., 103., 75., 0., 0., 38., 76., 160.,
↪ 224., 255.])
63
64     return monthly_load_heating, monthly_load_cooling, peak_heating, peak_cooling
65
66
67 def check_cases():
68     """
69
70     This function checks whether the borefield sizing gives the correct (i.e. validated)
↪ results for the 4 cases.
71     If not, an assertion error is raised.
72     NOTE: these values differ slightly from the values in the mentioned paper. This is
↪ due to the fact that GHEtool uses slightly different precalculated data.
73     """
74

```

(continues on next page)

(continued from previous page)

```

75 correct_answers_L2 = (56.75, 117.23, 66.94, 91.32)
76 correct_answers_L3 = (56.77, 118.74, 66.47, 91.24)
77
78 for i in (1, 2, 3, 4):
79     borefield = Borefield(load=MonthlyGeothermalLoadAbsolute(*load_case(i)))
80
81     borefield.set_ground_parameters(data)
82     borefield.set_borefield(borefield_gt)
83     borefield.Rb = 0.2
84
85     # set temperature boundaries
86     borefield.set_max_avg_fluid_temperature(16) # maximum temperature
87     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
88
89     borefield.size(100, L2_sizing=True)
90     print(f'correct answer L2: {correct_answers_L2[i-1]}; calculated answer L2:
↪ {round(borefield.H,2)}; error: '
91         f'{round(abs(1 - borefield.H / correct_answers_L2[i - 1]) * 100, 4)} %')
92     assert np.isclose(borefield.H, correct_answers_L2[i-1], rtol=0.001)
93
94     borefield.size(100, L3_sizing=True)
95     print(f'correct answer L3: {correct_answers_L3[i - 1]}; calculated answer L3:
↪ {round(borefield.H, 2)}; error: '
96         f'{round(abs(1 - borefield.H / correct_answers_L3[i - 1]) * 100, 4)} %')
97     assert np.isclose(borefield.H, correct_answers_L3[i-1], rtol=0.001)
98
99
100 def check_custom_datafile():
101     """
102     This function checks whether the borefield sizing gives the correct (i.e. validated)
↪ results for the 4 cases given the custom datafile.
103     If not, an assertion error is raised.
104     """
105
106     # create custom datafile
107
108     correct_answers = (56.75, 117.23, 66.94, 91.32)
109
110     custom_field = gt.boreholes.rectangle_field(N_1=12, N_2=10, B_1=6.5, B_2=6.5, H=110.,
↪ D=4, r_b=0.075)
111
112     for i in (1, 2, 3, 4):
113         borefield = Borefield(load=MonthlyGeothermalLoadAbsolute(*load_case(i)))
114
115         borefield.set_ground_parameters(data)
116         borefield.set_borefield(custom_field)
117         borefield.Rb = 0.2
118
119         # set temperature boundaries
120         borefield.set_max_avg_fluid_temperature(16) # maximum temperature
121         borefield.set_min_avg_fluid_temperature(0) # minimum temperature
122

```

(continues on next page)

(continued from previous page)

```

123     borefield.size(100, L3_sizing=True)
124     print(f'correct answer: {correct_answers[i-1]}; calculated '
125           f'answer: {round(borefield.H,2)}; error: '
126           f'{round(abs(1-borefield.H/correct_answers[i - 1])*100,4)} %')
127
128
129 if __name__ == "__main__": # pragma: no cover
130     check_cases() # check different cases
131     check_custom_datafile() # check if the custom datafile is correct

```

The equivalent borehole thermal resistance is validated with the commercial software of Earth Energy Designer (EED) and can be found here.

Validation equivalent borehole thermal resistance

```

1  """
2  This document contains the code to compare the equivalent borehole thermal resistance,
3  ↪calculated with GHEtool
4  (based on pygfunction) with the results from Earth Energy Designer. The differences can,
5  ↪be explained by using other
6  correlations and another assumption for the Nusselt number in the laminar regime.
7  """
8
9  import math
10
11 import matplotlib.pyplot as plt
12 import numpy as np
13 import pandas as pd
14 import pygfunction as gt
15
16 from GHEtool import Borefield, FOLDER
17 from GHEtool.VariableClasses import FluidData, GroundConstantTemperature, DoubleUTube
18
19 def validate():
20     # initiate parameters
21     ground_data = GroundConstantTemperature(3, 10) # ground data with an inaccurate,
22     ↪guess of 100m for the depth of the borefield
23     borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 100, 1, 0.075)
24     pipe_data = DoubleUTube(1, 0.015, 0.02, 0.4, 0.05, epsilon=1e-6)
25
26     # initiate borefield model
27     borefield = Borefield()
28     borefield.set_ground_parameters(ground_data)
29     borefield.set_pipe_parameters(pipe_data)
30     borefield.set_borefield(borefield_gt)
31     borefield.Rb = 0.12
32
33     # initialise variables
34     R_fp = []
35     R_p = []

```

(continues on next page)

(continued from previous page)

```

34 Rb = []
35
36 # load data EED
37 data_EED = pd.read_csv(FOLDER.joinpath("Validation/resistances_EED.csv"), sep=";")
38
39 mfr_range = np.arange(0.05, 0.55, 0.05)
40
41 # calculate effective borehole thermal resistance (Rb*)
42 for mfr in mfr_range:
43     fluid_data = FluidData(mfr, 0.568, 998, 4180, 1e-3)
44     borefield.set_fluid_parameters(fluid_data)
45     Rb.append(borefield.Rb)
46     R_p.append(borefield.borehole.pipe_data.R_p)
47     R_fp.append(borefield.borehole.pipe_data.R_f)
48
49
50 # make figure
51 plt.figure()
52 plt.plot(R_fp, 'r+', label="GHEtool")
53 plt.plot(data_EED["R_fp"], 'bo', label="EED")
54 plt.xlabel("Mass flow rate per borehole l/s")
55 plt.ylabel("Fluid-pipe resistance resistance mK/W")
56 plt.title("Comparison R_fp from GHEtool with EED")
57 plt.legend()
58
59 plt.figure()
60 plt.plot(mfr_range, (R_fp - data_EED["R_fp"])/data_EED["R_fp"]*100, 'bo')
61 plt.xlabel("Mass flow rate per borehole l/s")
62 plt.ylabel("Difference in fluid-pipe resistance %")
63 plt.title("Comparison R_fp from GHEtool with EED (relative)")
64
65 plt.figure()
66 plt.plot(Rb, 'r+', label="GHEtool")
67 plt.plot(data_EED["Rb*"], 'bo', label="EED")
68 plt.xlabel("Mass flow rate per borehole l/s")
69 plt.ylabel("Effective borehole thermal resistance mK/W")
70 plt.title("Comparison Rb* from GHEtool with EED")
71 plt.legend()
72
73 plt.figure()
74 plt.plot(mfr_range, (Rb - data_EED["Rb*"])/data_EED["Rb*"]*100, 'bo')
75 plt.xlabel("Mass flow rate per borehole l/s")
76 plt.ylabel("Difference in effective borehole thermal resistance %")
77 plt.title("Comparison Rb* from GHEtool with EED (relative)")
78
79 plt.show()
80
81
82 if __name__ == '__main__': # pragma: no cover
83     validate()

```

The deep sizing methodology, using a 1/depth assumption for the convergence of the borefield sizing when there is a temperature gradient can be found here.

Deep sizing

```

1  """
2  This file contains the reasoning behind the sizing method when the field is limited by
3  ↪ injection (i.e. cooling)
4  and there is a non-constant ground temperature. This is based on the assumption that the
5  ↪ difference between the
6  maximum peak temperature in injection and the average, undisturbed ground temperature
7  ↪ scales like 1/depth.
8  """
9
10 from GHEtool import *
11 import matplotlib.pyplot as plt
12 import numpy as np
13 from scipy.optimize import curve_fit
14
15
16 def validate():
17     ground_data = GroundFluxTemperature(3, 10)
18     fluid_data = FluidData(0.2, 0.568, 998, 4180, 1e-3)
19     pipe_data = DoubleUTube(1, 0.015, 0.02, 0.4, 0.05)
20     borefield = Borefield()
21     borefield.create_rectangular_borefield(5, 4, 6, 6, 110, 4, 0.075)
22     borefield.set_ground_parameters(ground_data)
23     borefield.set_fluid_parameters(fluid_data)
24     borefield.set_pipe_parameters(pipe_data)
25     borefield.calculation_setup(use_constant_Rb=False)
26     borefield.set_max_avg_fluid_temperature(17)
27     borefield.set_min_avg_fluid_temperature(3)
28     hourly_load = HourlyGeothermalLoad()
29     hourly_load.load_hourly_profile(FOLDER.joinpath("test\\methods\\hourly_data\\
30     ↪ auditorium.csv"), header=True, separator=";",
31                                     col_cooling=0, col_heating=1)
32     borefield.load = hourly_load
33
34     # initiate lists
35     Tg_list = []
36     max_Tf_list = []
37     depth_list = range(20, 450, 20)
38
39     for depth in depth_list:
40         print(f'The current depth is {depth} m.')
41         borefield.calculate_temperatures(depth)
42         Tg_list.append(borefield.ground_data.calculate_Tg(depth))
43         max_Tf_list.append(np.max(borefield.results.peak_cooling))
44
45     def f(x, a, b):
46         return a/x + b

```

(continues on next page)

(continued from previous page)

```

47     # determine temperature difference between peak cooling temperature and ground
↪temperature
48     diff = np.array(max_Tf_list) - np.array(Tg_list)
49
50     # fit to curve
51     popt, pcov = curve_fit(f, depth_list, diff)
52     print(popt, pcov)
53
54     plt.figure()
55     plt.plot(depth_list, Tg_list, label='Ground')
56     plt.plot(depth_list, max_Tf_list, label='Fluid')
57     plt.hlines(borefield.Tf_max, 0, depth_list[-1], label='Maximum temperature limit')
58     plt.xlabel('Depth [m]')
59     plt.ylabel('Temperature [deg C]')
60     plt.legend()
61     # plt.show()
62
63     plt.figure()
64     plt.plot(depth_list, diff, label='Actual calculated difference')
65     plt.plot(depth_list, f(np.array(depth_list), *popt), label='Fitted difference')
66     plt.xlabel('Depth [m]')
67     plt.ylabel('Temperature difference [deg C]')
68     plt.title('Temperature difference between maximum peak cooling fluid\ntemperature
↪and undistrubed ground temperature')
69     plt.legend()
70     plt.show()
71
72
73 if __name__ == "__main__":    # pragma: no cover
74     validate()

```

1.2.11 Speed improvements

This folder contains information w.r.t. speed improvements in new versions of GHEtool.

v2.1.1

In GHEtool v2.1.1 there are two major code changes that reduce the computational time significantly. One has to do with the way the sizing methodology (L3/L4) is implemented and another with the new Gfunction class. Both improvements are explained below.

Improvement in sizing

Previously, in v2.1.0, for the sizing methodology L3 and L4 (i.e. monthly and hourly), the temperatures were calculated every time step. This however (especially for long simulation periods) requires a lot of time due to the convolution step, especially for an hourly sizing. In v2.1.1 this is changed so that only the first and last year are calculated, since only these years are relevant for the sizing¹. This means that however long the simulation period may be, only two years are calculated. In the table below, the time required for one sizing iteration (i.e. one g-value convolution) for an hourly sizing is shown in μs . The code to come up with these numbers is added below.

Table 1: Speed improvement for g-value convolution in hourly sizing

Simulation period [years]	Required time old method [μs]	Required time new method [μs]
5 years	15625 μs	0 μs
15 years	15625 μs	0 μs
25 years	31250 μs	0 μs
35 years	46875 μs	0 μs
45 years	62500 μs	0 μs
55 years	78125 μs	0 μs
65 years	78125 μs	0 μs
75 years	78125 μs	0 μs
85 years	109375 μs	0 μs
95 years	125000 μs	0 μs

Gfunction class

Due to the implementation of the GFunction class in GHEtool, a substantial speed improvement is made w.r.t. GHEtool v2.1.0 for computationally expensive tasks. In the tables below, one can find this speed improvement for the different sizing methods and for several of the examples documents. The results can be recreated by running the code below. The computational times as shown in the table below, are an average of 5 runs.

Table 2: Speed benchmark sizing methods

Sizing method	Time v2.1.0 [ms]	Time v2.1.1 [ms]	Improvement [%]
L2 (three pulse) sizing	1.84 ms	1.43 ms	28%
L3 (monthly) sizing	12.34 ms	5.8 ms	113%
L4 (hourly) sizing	4.21 ms	3.63 ms	16%

Table 3: Speed benchmark examples

Example	Time v2.1.0 [ms]	Time v2.1.1 [ms]	Improvement [%]
Main functionalities	3.38 ms	2.57 ms	32%
Optimise load profile	15.07 ms	0.63 ms	2305%
Sizing with Rb calculation	9.97 ms	9.99 ms	0%
Effect borefield configuration	1.57 ms	1.5 ms	4%

```

1 import numpy as np
2 import pygfunction as gt
3 from GHEtool import GroundConstantTemperature, Borefield, HourlyGeothermalLoad, FOLDER

```

(continues on next page)

¹ Peere, W., Picard, D., Cupeiro Figueroa, I., Boydens, W., and Helsen, L. (2021) Validated combined first and last year borefield sizing methodology. In Proceedings of International Building Simulation Conference 2021. Brugge (Belgium), 1-3 September 2021. <https://doi.org/10.26868/25222708.2021.30180>

(continued from previous page)

```

4 from scipy.signal import convolve
5 from math import pi
6 from time import process_time_ns
7
8
9 def test_new_calc_method(simulation_period: int):
10     """
11     Test the new calculation method which is just considering the first and last year.
12
13     Parameters
14     -----
15     simulation_period : int
16         simulation period [years]
17
18     Returns
19     -----
20         None
21
22     Raises
23     -----
24         AssertionError
25     """
26
27     h = 110
28
29     # initiate ground data
30     data = GroundConstantTemperature(3, 10)
31
32     # initiate pygfunction borefield model
33     borefield_gt = gt.boreholes.rectangle_field(10, 10, 6, 6, 110, 1, 0.075)
34
35     # initiate borefield
36     borefield = Borefield(100)
37
38     # set borehole thermal equivalent resistance
39     borefield.Rb = 0.12
40
41     # set ground data in borefield
42     borefield.set_ground_parameters(data)
43
44     # set pygfunction borefield model
45     borefield.set_borefield(borefield_gt)
46
47
48     # load the hourly profile
49     load = HourlyGeothermalLoad(simulation_period=simulation_period)
50     load.load_hourly_profile(f'hourly_profile.csv', header=True, separator=";")
51
52     borefield.load = load
53
54     # borefield.g-function is a function that uses the precalculated data to interpolate
55     ↪ the correct values of the

```

(continues on next page)

(continued from previous page)

```

55  # g-function. This dataset is checked over and over again and is correct
56  g_values = borefield.gfunction(borefield.time_L4, borefield.H)
57
58  # get process time at start of new method
59  dt1 = process_time_ns()
60  # determine load
61  loads_short = borefield.hourly_cooling_load - borefield.hourly_heating_load
62  # reverse the load
63  loads_short_rev = loads_short[::-1]
64  # init results vector
65  results = np.zeros.loads_short.size * 2)
66  # calculation of needed differences of the g-function values. These are the weight_
↳ factors in the calculation
67  # of Tb.
68  g_value_differences = np.diff(g_values, prepend=0)
69
70  # convolution to get the results for the first year
71  results[:8760] = convolve.loads_short * 1000, g_value_differences[:8760])[:8760]
72  # sum up g_values until the pre last year
73  g_sum_n1 = g_value_differences[:8760 * (borefield.simulation_period - 1)].
↳ reshape(borefield.simulation_period - 1, 8760).sum(axis=0)
74  # add up last year
75  g_sum = g_sum_n1 + g_value_differences[8760 * (borefield.simulation_period - 1):]
76  # add zero at start and reverse the order
77  g_sum_n2 = np.concatenate((np.array([0]), g_sum_n1[::-1]))[: -1]
78  # determine results for the last year by the influence of the year (first term) and_
↳ the previous years (last term)
79  results[8760:] = convolve.loads_short * 1000, g_sum)[:8760] + convolve.loads_short_
↳ rev * 1000, g_sum_n2)[:8760]][: -1]
80  # calculation the borehole wall temperature for every month i
81  t_b = results / (2 * pi * borefield.ground_data.k_s) / (borefield.H * borefield.
↳ number_of_boreholes) + borefield._Tg(borefield.H)
82
83  # get process time
84  dt2 = process_time_ns()
85  # determine hourly load
86  hourly_load = np.tile(borefield.hourly_cooling_load - borefield.hourly_heating_load,
↳ borefield.simulation_period)
87  # calculation of needed differences of the g-function values. These are the weight_
↳ factors in the calculation
88  # of Tb.
89  g_value_differences = np.diff(g_values, prepend=0)
90
91  # convolution to get the monthly results
92  results = convolve(hourly_load * 1000, g_value_differences)[:hourly_load.size]
93
94  # calculation the borehole wall temperature for every month i
95  t_b_new = results / (2 * pi * borefield.ground_data.k_s) / (h * borefield.number_of_
↳ boreholes) + borefield._Tg(h)
96
97  # print time for the different methods
98  print(f'simulation period: {simulation_period}; old: {(process_time_ns() - dt2)/

```

(continues on next page)

(continued from previous page)

```

↪1000:.0f} μs;'
    f'new: {(dt2 - dt1)/1000:.0f} μs')
# compare results to ensure they are the same
assert np.allclose(t_b[:8760], t_b_new[:8760])
assert np.allclose(t_b[8760:], t_b_new[8760*(borefield.simulation_period - 1):])

if __name__ == "__main__":
    for sim_year in np.arange(5, 101, 10):
        test_new_calc_method(sim_year)

```

```

1 from GHEtool import *
2 import numpy as np
3 import pygfunction as gt
4 import time
5 import os, contextlib
6
7 from GHEtool.Examples.main_functionalities import main_functionalities
8 from GHEtool.Examples.sizing_with_Rb_calculation import sizing_with_Rb
9 from GHEtool.Examples.effect_of_borehole_configuration import effect_borefield_
↪configuration
10 from GHEtool import HourlyGeothermalLoad
11
12 # disable the plot function by monkey patching over it
13 Borefield._plot_temperature_profile = lambda *args, **kwargs: None
14
15
16 # disable the print outputs
17 def supress_stdout(func):
18     def wrapper(*a, **ka):
19         with open(os.devnull, 'w') as devnull:
20             with contextlib.redirect_stdout(devnull):
21                 return func(*a, **ka)
22     return wrapper
23
24
25 @supress_stdout
26 def run_without_messages(callable) -> None:
27     """
28     This function runs the callable without messages.
29
30     Parameters
31     -----
32     callable : Callable
33         Function to be called
34
35     Returns
36     -----
37     None
38     """
39     callable()
40

```

(continues on next page)

(continued from previous page)

```

41 def optimise_load_profile() -> None:
42     """
43     This is a benchmark for the optimise load profile method.
44
45     Returns
46     -----
47     None
48     """
49     # initiate ground data
50     data = GroundData(3, 10, 0.2)
51
52     # initiate pygfunction borefield model
53     borefield_gt = gt.boreholes.rectangle_field(10, 10, 6, 6, 110, 1, 0.075)
54
55     # initiate borefield
56     borefield = Borefield()
57
58     # set ground data in borefield
59     borefield.set_ground_parameters(data)
60
61     # set pygfunction borefield
62     borefield.set_borefield(borefield_gt)
63
64     # load the hourly profile
65     load = HourlyGeothermalLoad()
66     load.load_hourly_profile("hourly_profile.csv", header=True, separator=";")
67     borefield.load = load
68
69     # optimise the load for a 10x10 field (see data above) and a fixed depth of 150m.
70     borefield.optimise_load_profile(depth=150, print_results=False)
71
72
73 def size_L2() -> None:
74     """
75     This is a benchmark for the L2 sizing method.
76
77     Returns
78     -----
79     None
80     """
81
82     number_of_iterations = 5
83     max_value_cooling = 700
84     max_value_heating = 800
85
86     monthly_load_cooling_array = np.empty((number_of_iterations, 12))
87     monthly_load_heating_array = np.empty((number_of_iterations, 12))
88     peak_load_cooling_array = np.empty((number_of_iterations, 12))
89     peak_load_heating_array = np.empty((number_of_iterations, 12))
90
91     # populate arrays with random values
92     for i in range(number_of_iterations):

```

(continues on next page)

(continued from previous page)

```

93     for j in range(12):
94         monthly_load_cooling_array[i, j] = np.random.randint(0, max_value_cooling)
95         monthly_load_heating_array[i, j] = np.random.randint(0, max_value_heating)
96         peak_load_cooling_array[i, j] = np.random.randint(monthly_load_cooling_
↪ array[i, j], max_value_cooling)
97         peak_load_heating_array[i, j] = np.random.randint(monthly_load_heating_
↪ array[i, j], max_value_heating)
98
99     # initiate borefield model
100     data = GroundData(3, 10, 0.2)
101     borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 110, 1, 0.075)
102
103     # Monthly loading values
104     peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.
↪ ]) # Peak cooling in kW
105     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↪ ]) # Peak heating in kW
106
107     # annual heating and cooling load
108     annual_heating_load = 300 * 10 ** 3 # kWh
109     annual_cooling_load = 160 * 10 ** 3 # kWh
110
111     # percentage of annual load per month (15.5% for January ...)
112     monthly_load_heating_percentage = np.array(
113         [0.155, 0.148, 0.125, .099, .064, 0., 0., 0., 0.061, 0.087, 0.117, 0.144])
114     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪ .1, .075, .05, .025])
115
116     # resulting load per month
117     monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
118     monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh
119
120     # create the borefield object
121     borefield = Borefield(simulation_period=20,
122                           peak_heating=peak_heating,
123                           peak_cooling=peak_cooling,
124                           baseload_heating=monthly_load_heating,
125                           baseload_cooling=monthly_load_cooling)
126     borefield.set_ground_parameters(data)
127     borefield.set_borefield(borefield_gt)
128
129     # set temperature boundaries
130     borefield.set_max_avg_fluid_temperature(16) # maximum temperature
131     borefield.set_min_avg_fluid_temperature(0) # minimum temperature
132
133     # size according to L2 method
134     for i in range(number_of_iterations):
135         borefield.set_baseload_cooling(monthly_load_cooling_array[i])
136         borefield.set_baseload_heating(monthly_load_heating_array[i])
137         borefield.set_peak_cooling(peak_load_cooling_array[i])
138         borefield.set_peak_heating(peak_load_heating_array[i])
139         borefield.size(100, L2_sizing=True)

```

(continues on next page)

(continued from previous page)

```

140
141
142 def size_L3() -> None:
143     """
144     This is a benchmark for the L3 sizing method.
145
146     Returns
147     -----
148     None
149     """
150     number_of_iterations = 5
151     max_value_cooling = 700
152     max_value_heating = 800
153
154     monthly_load_cooling_array = np.empty((number_of_iterations, 12))
155     monthly_load_heating_array = np.empty((number_of_iterations, 12))
156     peak_load_cooling_array = np.empty((number_of_iterations, 12))
157     peak_load_heating_array = np.empty((number_of_iterations, 12))
158
159     # populate arrays with random values
160     for i in range(number_of_iterations):
161         for j in range(12):
162             monthly_load_cooling_array[i, j] = np.random.randint(0, max_value_cooling)
163             monthly_load_heating_array[i, j] = np.random.randint(0, max_value_heating)
164             peak_load_cooling_array[i, j] = np.random.randint(monthly_load_cooling_
↪ array[i, j], max_value_cooling)
165             peak_load_heating_array[i, j] = np.random.randint(monthly_load_heating_
↪ array[i, j], max_value_heating)
166
167     # initiate borefield model
168     data = GroundData(3, 10, 0.2)
169     borefield_gt = gt.boreholes.rectangle_field(10, 12, 6, 6, 110, 1, 0.075)
170
171     # Monthly loading values
172     peak_cooling = np.array([0., 0, 34., 69., 133., 187., 213., 240., 160., 37., 0., 0.
↪ ]) # Peak cooling in kW
173     peak_heating = np.array([160., 142, 102., 55., 0., 0., 0., 0., 40.4, 85., 119., 136.
↪ ]) # Peak heating in kW
174
175     # annual heating and cooling load
176     annual_heating_load = 300 * 10 ** 3 # kWh
177     annual_cooling_load = 160 * 10 ** 3 # kWh
178
179     # percentage of annual load per month (15.5% for January ...)
180     monthly_load_heating_percentage = np.array(
181         [0.155, 0.148, 0.125, .099, .064, 0., 0., 0., 0.061, 0.087, 0.117, 0.144])
182     monthly_load_cooling_percentage = np.array([0.025, 0.05, 0.05, .05, .075, .1, .2, .2,
↪ .1, .075, .05, .025])
183
184     # resulting load per month
185     monthly_load_heating = annual_heating_load * monthly_load_heating_percentage # kWh
186     monthly_load_cooling = annual_cooling_load * monthly_load_cooling_percentage # kWh

```

(continues on next page)

(continued from previous page)

```

187
188 # create the borefield object
189 borefield = Borefield(simulation_period=20,
190                       peak_heating=peak_heating,
191                       peak_cooling=peak_cooling,
192                       baseload_heating=monthly_load_heating,
193                       baseload_cooling=monthly_load_cooling)
194 borefield.set_ground_parameters(data)
195 borefield.set_borefield(borefield_gt)
196
197 # set temperature boundaries
198 borefield.set_max_avg_fluid_temperature(16) # maximum temperature
199 borefield.set_min_avg_fluid_temperature(0) # minimum temperature
200
201 # size according to L3 method
202 for i in range(number_of_iterations):
203     borefield.set_baseload_cooling(monthly_load_cooling_array[i])
204     borefield.set_baseload_heating(monthly_load_heating_array[i])
205     borefield.set_peak_cooling(peak_load_cooling_array[i])
206     borefield.set_peak_heating(peak_load_heating_array[i])
207     borefield.size(100, L3_sizing=True)
208
209
210 def size_L4() -> None:
211     """
212     This is a benchmark for the L4 sizing method.
213
214     Returns
215     -----
216     None
217     """
218     # initiate ground data
219     data = GroundData(3, 10, 0.2)
220
221     # initiate pygfunction borefield model
222     borefield_gt = gt.boreholes.rectangle_field(10, 10, 6, 6, 110, 1, 0.075)
223
224     # initiate borefield
225     borefield = Borefield()
226
227     # set ground data in borefield
228     borefield.set_ground_parameters(data)
229
230     # set pygfunction borefield
231     borefield.set_borefield(borefield_gt)
232
233     # load the hourly profile
234     load = HourlyGeothermalLoad()
235     load.load_hourly_profile("hourly_profile.csv", header=True, separator=";")
236     borefield.load = load
237
238     borefield.size(L4_sizing=True)

```

(continues on next page)

(continued from previous page)

```

239
240
241 def benchmark(callable, name: str) -> None:
242     """
243     This function calls the callable five times and outputs the time needed to run the_
↳ callable.
244
245     Parameters
246     -----
247     callable : Callable
248         Function to be called
249     name : str
250         Name of the function
251
252     Returns
253     -----
254     None
255     """
256     diff, diff_without = 0., 0.
257
258     for i in range(5):
259         GFunction.DEFAULT_STORE_PREVIOUS_VALUES = True
260
261         start_time = time.time()
262         run_without_messages(callable)
263         end_time = time.time()
264         diff = diff * i/(i+1) + (end_time - start_time)/(i+1)
265
266         GFunction.DEFAULT_STORE_PREVIOUS_VALUES = False
267
268         start_time_without = time.time()
269         run_without_messages(callable)
270         end_time_without = time.time()
271         diff_without = diff_without * i/(i+1) + (end_time_without - start_time_without)/
↳ (i+1)
272
273     print(f'{name} took {round(diff_without, 2)} ms in v2.1.0 and '
274           f'{round(diff, 2)} ms in v2.1.1. This is an improvement of {round((diff_
↳ without-diff)/diff*100)}%.')
275
276
277 # run examples
278 benchmark(main_functionalities, "Main functionalities")
279 benchmark(optimise_load_profile, "Optimise load profile")
280 benchmark(sizing_with_Rb, "Sizing with Rb calculation")
281 benchmark(effect_borefield_configuration, "Effect borehole configuration")
282
283 # run benchmark sizing methods
284 benchmark(size_L2, "Sizing with L2 method")
285 benchmark(size_L3, "Sizing with L3 method")
286 benchmark(size_L4, "Sizing with L4 (hourly) method")

```

References

PYTHON MODULE INDEX

g

`GHEtool.logger.ghe_logger`, 79
`GHEtool.main_class`, 39
`GHEtool.VariableClasses.BaseClass`, 53
`GHEtool.VariableClasses.Borehole`, 54
`GHEtool.VariableClasses.CustomGFunction`, 75
`GHEtool.VariableClasses.FluidData`, 63
`GHEtool.VariableClasses.GeothermalLoad.HourlyGeothermalLoad`, 69
`GHEtool.VariableClasses.GeothermalLoad.HourlyGeothermalLoadMultiYear`, 69
`GHEtool.VariableClasses.GeothermalLoad.MonthlyGeothermalLoadAbsolute`, 69
`GHEtool.VariableClasses.GFunction`, 77
`GHEtool.VariableClasses.GroundData._GroundData`, 56
`GHEtool.VariableClasses.GroundData.GroundConstantTemperature`, 59
`GHEtool.VariableClasses.GroundData.GroundFluxTemperature`, 60
`GHEtool.VariableClasses.GroundData.GroundTemperatureGradient`, 62
`GHEtool.VariableClasses.LoadData._LoadData`, 64
`GHEtool.VariableClasses.PipeData._PipeData`, 69
`GHEtool.VariableClasses.PipeData.CoaxialPipe`, 73
`GHEtool.VariableClasses.PipeData.DoubleUTube`, 72
`GHEtool.VariableClasses.PipeData.MultipleUTube`, 71
`GHEtool.VariableClasses.PipeData.SingleUTube`, 72

Symbols

_GroundData (class in *GHEtool.VariableClasses.GroundData._GroundData*), 57
 _LoadData (class in *GHEtool.VariableClasses.LoadData._LoadData*), 64
 _PipeData (class in *GHEtool.VariableClasses.PipeData._PipeData*), 69
 _calculate_first_year_params() (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* method), 64
 _calculate_last_year_params() (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* method), 64
 _check_input() (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* method), 65
A
 activate_logger() (*GHEtool.main_class.Borefield* static method), 41
 add() (*GHEtool.VariableClasses.GFunction.FIFO* method), 77
 add_dhw() (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* method), 65
 add_layer_on_bottom() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* method), 57
 add_layer_on_top() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* method), 57
 addLoggingLevel() (in module *GHEtool.logger.ghe_logger*), 80
 all_months_equal (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 65
 alpha() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* method), 57
B
 BaseClass (class in *GHEtool.VariableClasses.BaseClass*), 53
 baseload_cooling() (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* method), 65
 baseload_cooling_power (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 65
 baseload_cooling_power_simulation_period (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 65
 baseload_cooling_simulation_period (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 65
 baseload_heating() (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* method), 66
 baseload_heating_power (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 66
 baseload_heating_power_simulation_period (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 66
 baseload_heating_simulation_period (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* property), 66
 Borefield (class in *GHEtool.main_class*), 39
 borefield (*GHEtool.main_class.Borefield* property), 41
 Borehole (class in *GHEtool.VariableClasses.Borehole*), 54
C
 calculate() (*GHEtool.VariableClasses.GFunction.GFunction* method), 77
 calculate_delta_H() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* method), 58
 calculate_delta_H() (*GHEtool.VariableClasses.GroundData.GroundConstantTemperature* method), 60
 calculate_delta_H() (*GHEtool.VariableClasses.GroundData.GroundFluxTemperature* method), 61
 calculate_delta_H() (*GHEtool.VariableClasses.GroundData.GroundTemperatureGradient* method), 62
 calculate_gfunction() (*GHEtool.VariableClasses.CustomGFunction.CustomGFunction* method), 62

`method)`, 75
`calculate_next_depth_deep_sizing()`
`(GHEtool.main_class.Borefield method)`, 41
`calculate_quadrant()`
`(GHEtool.main_class.Borefield method)`, 42
`calculate_Rb()` (*GHEtool.VariableClasses.Borehole.Borehole* `method`), 55
`calculate_resistances()`
`(GHEtool.VariableClasses.PipeData._PipeData._PipeData` `method`), 70
`calculate_resistances()`
`(GHEtool.VariableClasses.PipeData.CoaxialPipe.CoaxialPipe` `method`), 74
`calculate_resistances()`
`(GHEtool.VariableClasses.PipeData.MultipleUTube.MultipleUTube` `method`), 71
`calculate_temperatures()`
`(GHEtool.main_class.Borefield method)`, 42
`calculate_Tg()` (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* `method`), 58
`calculate_Tg()` (*GHEtool.VariableClasses.GroundData.GroundConstantTemperature.GroundConstantTemperature* `method`), 60
`calculate_Tg()` (*GHEtool.VariableClasses.GroundData.GroundFluxTemperature.GroundFluxTemperature* `method`), 61
`calculate_Tg()` (*GHEtool.VariableClasses.GroundData.GroundTemperatureGradient.GroundTemperatureGradient* `method`), 62
`calculate_value()` (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* `method`), 58
`calculation_setup()` (*GHEtool.main_class.Borefield* `method`), 42
`check_depth()` (*GHEtool.VariableClasses.GroundData._GroundData._GroundData* `method`), 58
`check_values()` (*GHEtool.VariableClasses.BaseClass.BaseClass* `method`), 54
`clear()` (*GHEtool.VariableClasses.GFunction.FIFO* `method`), 77
CoaxialPipe (class in *GHEtool.VariableClasses.PipeData.CoaxialPipe*), 73
`correct_for_start_month()`
`(GHEtool.VariableClasses.LoadData._LoadData._LoadData` `method`), 66
`create_box_shaped_borefield()`
`(GHEtool.main_class.Borefield method)`, 44
`create_circular_borefield()`
`(GHEtool.main_class.Borefield method)`, 44
`create_custom_dataset()`
`(GHEtool.main_class.Borefield method)`, 45
`create_custom_dataset()`
`(GHEtool.VariableClasses.CustomGFunction.CustomGFunction` `method`), 75
`create_L_shaped_borefield()`
`(GHEtool.main_class.Borefield method)`, 43
`create_rectangular_borefield()`
`(GHEtool.main_class.Borefield method)`, 45
`create_U_shaped_borefield()`
`(GHEtool.main_class.Borefield method)`, 43
CustomFormatter (class in *GHEtool.logger.ghe_logger*), 79
CustomGFunction (class in *GHEtool.VariableClasses.CustomGFunction*), 75

D

`deactivate_logger()` (*GHEtool.main_class.Borefield* `static method`), 46
`delete_custom_dataset()`
`(GHEtool.VariableClasses.CustomGFunction.CustomGFunction` `method`), 75
`dhw` (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* `property`), 66
`dhw_power` (*GHEtool.VariableClasses.LoadData._LoadData._LoadData* `property`), 66
DoubleUTube (class in *GHEtool.VariableClasses.PipeData.DoubleUTube*), 72
`draw_borehole_internal()`
`(GHEtool.VariableClasses.PipeData._PipeData._PipeData` `method`), 74
`draw_borehole_internal()`
`(GHEtool.VariableClasses.PipeData.CoaxialPipe.CoaxialPipe` `method`), 74
`draw_borehole_internal()`
`(GHEtool.VariableClasses.PipeData.MultipleUTube.MultipleUTube` `method`), 72
`dump_custom_dataset()`
`(GHEtool.VariableClasses.CustomGFunction.CustomGFunction` `method`), 76

F

FIFO (class in *GHEtool.VariableClasses.GFunction*), 77
`fluid_data` (*GHEtool.VariableClasses.Borehole.Borehole* `property`), 55
FluidData (class in *GHEtool.VariableClasses.FluidData*), 63
`format()` (*GHEtool.logger.ghe_logger.CustomFormatter* `method`), 79
`from_dict()` (*GHEtool.VariableClasses.BaseClass.BaseClass* `method`), 54

G

get_month_index() (*GHEtool.VariableClasses.LoadData._LoadData* *static method*), 67

get_Rb() (*GHEtool.VariableClasses.Borehole.Borehole method*), 55

GFunction (*class in GHEtool.VariableClasses.GFunction*), 77

gfunction() (*GHEtool.main_class.Borefield method*), 46

GHEtool.logger.ghe_logger module, 79

GHEtool.main_class module, 39

GHEtool.VariableClasses.BaseClass module, 53

GHEtool.VariableClasses.Borehole module, 54

GHEtool.VariableClasses.CustomGFunction module, 75

GHEtool.VariableClasses.FluidData module, 63

GHEtool.VariableClasses.GeothermalLoad.HourlyGeothermalLoad module, 69

GHEtool.VariableClasses.GeothermalLoad.HourlyGeothermalLoadMultiYear module, 69

GHEtool.VariableClasses.GeothermalLoad.MonthlyGeothermalLoad module, 69

GHEtool.VariableClasses.GFunction module, 77

GHEtool.VariableClasses.GroundData._GroundData module, 56

GHEtool.VariableClasses.GroundData.GroundConstantTemperature module, 59

GHEtool.VariableClasses.GroundData.GroundFluxTemperature module, 60

GHEtool.VariableClasses.GroundData.GroundTemperatureGradient module, 62

GHEtool.VariableClasses.LoadData._LoadData module, 64

GHEtool.VariableClasses.PipeData._PipeData module, 69

GHEtool.VariableClasses.PipeData.CoaxialPipe module, 73

GHEtool.VariableClasses.PipeData.DoubleUTube module, 72

GHEtool.VariableClasses.PipeData.MultipleUTube module, 71

GHEtool.VariableClasses.PipeData.SingleUTube module, 72

ground_data (*GHEtool.main_class.Borefield property*), 46

GroundConstantTemperature (*class in GHEtool.VariableClasses.GroundData.GroundConstantTemperature*), 59

GroundFluxTemperature (*class in GHEtool.VariableClasses.GroundData.GroundFluxTemperature*), 60

GroundLayer (*class in GHEtool.VariableClasses.GroundData._GroundData*), 56

GroundTemperatureGradient (*class in GHEtool.VariableClasses.GroundData.GroundTemperatureGradient*), 62

H

H (*GHEtool.main_class.Borefield property*), 41

I

imbalance (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 67

import_fluid_from_pygfunction() (*GHEtool.VariableClasses.FluidData.FluidData method*), 63

in_fifo_list() (*GHEtool.VariableClasses.GFunction.FIFO method*), 77

in_geothermal_load_functions() (*GHEtool.VariableClasses.GFunction.GFunction method*), 77

investment_cost (*GHEtool.main_class.Borefield property*), 41

K

k_s() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData method*), 59

L

load (*GHEtool.main_class.Borefield property*), 47

load_custom_gfunction() (*GHEtool.main_class.Borefield method*), 47

load_custom_gfunction() (*in module GHEtool.VariableClasses.CustomGFunction*), 76

M

max_depth() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData method*), 59

max_peak_cooling (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 67

max_peak_heating (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 67

MaximumNumberOfIterations, 54

module

GHEtool.logger.ghe_logger, 79

GHEtool.main_class, 39

GHEtool.VariableClasses.BaseClass, 53

GHEtool.VariableClasses.Borehole, 54

GHEtool.VariableClasses.CustomGFunction, 75
 GHEtool.VariableClasses.FluidData, 63
 GHEtool.VariableClasses.GeothermalLoad.HourlyGeothermalLoad, 69
 GHEtool.VariableClasses.GeothermalLoad.HourlyGeothermalLoadMultiYear, 69
 GHEtool.VariableClasses.GeothermalLoad.MonthlyGeothermalLoadAbsolute, 69
 GHEtool.VariableClasses.GFunction, 77
 GHEtool.VariableClasses.GroundData._GroundData, 56
 GHEtool.VariableClasses.GroundData.GroundConstantTemperature, 59
 GHEtool.VariableClasses.GroundData.GroundFluxTemperature, 60
 GHEtool.VariableClasses.GroundData.GroundTemperatureGradient, 62
 GHEtool.VariableClasses.LoadData._LoadData, 64
 GHEtool.VariableClasses.PipeData._PipeData, 69
 GHEtool.VariableClasses.PipeData.CoaxialPipe, 73
 GHEtool.VariableClasses.PipeData.DoubleUTube, 72
 GHEtool.VariableClasses.PipeData.MultipleUTube, 71
 GHEtool.VariableClasses.PipeData.SingleUTube, 72
 monthly_average_load (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 67
 monthly_average_load_simulation_period (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 67
 MultipleUTube (class in GHEtool.VariableClasses.PipeData.MultipleUTube), 71
 N
 non_negative() (GHEtool.VariableClasses.GroundData._GroundData.GroundLayer method), 56
 number_of_boreholes (GHEtool.main_class.Borefield property), 47
 O
 optimise_load_profile_energy() (GHEtool.main_class.Borefield method), 47
 optimise_load_profile_power() (GHEtool.main_class.Borefield method), 48
 P
 peak_cooling() (GHEtool.VariableClasses.LoadData._LoadData._LoadData method), 68
 peak_cooling_duration (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 68
 peak_cooling_simulation_period (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 68
 peak_duration (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 68
 peak_heating() (GHEtool.VariableClasses.LoadData._LoadData._LoadData method), 68
 peak_heating_duration (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 68
 peak_heating_simulation_period (GHEtool.VariableClasses.LoadData._LoadData._LoadData property), 68
 pipe_data (GHEtool.VariableClasses.Borehole.Borehole property), 56
 pipe_model() (GHEtool.VariableClasses.PipeData._PipeData._PipeData method), 70
 pipe_model() (GHEtool.VariableClasses.PipeData.CoaxialPipe.CoaxialPipe method), 74
 pipe_model() (GHEtool.VariableClasses.PipeData.MultipleUTube.MultipleUTube method), 72
 plot_load_duration() (GHEtool.main_class.Borefield method), 48
 print_temperature_profile() (GHEtool.main_class.Borefield method), 49
 print_temperature_profile_fixed_depth() (GHEtool.main_class.Borefield method), 49
 R
 Rb (GHEtool.main_class.Borefield property), 41
 Rb (GHEtool.VariableClasses.Borehole.Borehole property), 55
 Re (GHEtool.main_class.Borefield property), 41
 Re (GHEtool.VariableClasses.Borehole.Borehole property), 55
 Re() (GHEtool.VariableClasses.PipeData._PipeData._PipeData method), 70
 Re() (GHEtool.VariableClasses.PipeData.CoaxialPipe.CoaxialPipe method), 74
 Re() (GHEtool.VariableClasses.PipeData.MultipleUTube.MultipleUTube method), 71
 remove_previous_data() (GHEtool.VariableClasses.GFunction.GFunction method), 78

S

set_borefield() (*GHEtool.main_class.Borefield method*), 49
 set_fluid_parameters() (*GHEtool.main_class.Borefield method*), 50
 set_ground_parameters() (*GHEtool.main_class.Borefield method*), 50
 set_investment_cost() (*GHEtool.main_class.Borefield method*), 50
 set_load() (*GHEtool.main_class.Borefield method*), 50
 set_mass_flow_rate() (*GHEtool.VariableClasses.FluidData.FluidData method*), 63
 set_max_avg_fluid_temperature() (*GHEtool.main_class.Borefield method*), 50
 set_min_avg_fluid_temperature() (*GHEtool.main_class.Borefield method*), 51
 set_new_calculated_data() (*GHEtool.VariableClasses.GFunction.GFunction method*), 78
 set_options_gfunction_calculation() (*GHEtool.main_class.Borefield method*), 51
 set_options_gfunction_calculation() (*GHEtool.VariableClasses.CustomGFunction.CustomGFunction method*), 76
 set_options_gfunction_calculation() (*GHEtool.VariableClasses.GFunction.GFunction method*), 79
 set_pipe_parameters() (*GHEtool.main_class.Borefield method*), 51
 set_Rb() (*GHEtool.main_class.Borefield method*), 49
 simulation_period (*GHEtool.main_class.Borefield property*), 51
 SingleUTube (class in *GHEtool.VariableClasses.PipeData.SingleUTube*), 72
 size() (*GHEtool.main_class.Borefield method*), 51
 size_L2() (*GHEtool.main_class.Borefield method*), 52
 size_L3() (*GHEtool.main_class.Borefield method*), 53
 size_L4() (*GHEtool.main_class.Borefield method*), 53
 start_month (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 68
 store_previous_values (*GHEtool.VariableClasses.GFunction.GFunction property*), 79

T

time_L3 (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 68
 time_L4 (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 69
 to_dict() (*GHEtool.VariableClasses.BaseClass.BaseClass method*), 54
 ty (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 69

U

UnsolvableDueToTemperatureGradient, 54
 UPM (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 64

V

volumetric_heat_capacity() (*GHEtool.VariableClasses.GroundData._GroundData._GroundData method*), 59

W

within_range() (*GHEtool.VariableClasses.CustomGFunction.CustomGFunction method*), 76

Y

yearly_cooling_load (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 69
 yearly_heating_load (*GHEtool.VariableClasses.LoadData._LoadData._LoadData property*), 69