FASTER COMPUTATION OF G-FUNCTIONS USED FOR MODELING OF GROUND HEAT EXCHANGERS WITH REDUCED MEMORY CONSUMPTION

Pre-print

Cook, J.C. and J.D. Spitler. 2021. Faster computation of g-functions used for modeling of ground heat exchangers with reduced memory consumption. Accepted for publication in Proceedings of Building Simulation 2021, Bruges, Belgium. September 1-3, 2021.

Jack Cook and Jeffrey Spitler

Jack.cook@okstate.edu, spitler@okstate.edu

Faster computation of g-functions used for modeling of ground heat exchangers with reduced memory consumption

Jack C. Cook[1], Jeffrey D. Spitler[1],
[1]Oklahoma State University, Stillwater, Oklahoma, USA

Abstract

Temperature response functions, known as g-functions, are a computationally efficient method for simulating ground heat exchangers (GHEs), used with ground-source heat pump (GSHP) systems or direct ground cooling systems as part of a whole-building energy simulation. In fact, at present, there are no other methods that have sufficient accuracy and are fast enough to simulate a ground-source heat pump system in a whole-building energy simulation.

The concept, mathematical derivation and an implementation of a g-function calculation program were originally developed by Claesson and Eskilson (1985). More recently (Cimmino 2018a, Cimmino 2018b, Cimmino 2019) developed an open-source g-function calculation tool known as pygfunction. This tool offers great flexibility for the user to compute g-functions for specific configurations of boreholes. However, for large borehole configurations (with ~1000 boreholes), the required time to compute a single g-function can take several hours, and the required RAM can be on the order of 100 GB, greatly exceeding most desktop PCs. In order to develop libraries of g-functions and training sets for machine learning approaches, we are computing hundreds of thousands of g-functions. This paper describes further development of Cimmino's methodology to speed the computation and reduce the memory requirements.

Key Innovations

- Significant reduction in memory requirements for computing g-functions.
- Significant increase in g-function computation speed for ground heat exchangers with irregular configurations.

Practical Implications

Developments in this paper make it more practical to compute g-functions for larger ground heat exchangers on a desktop computer. These same developments significantly increase throughput of calculations on a high-performance computing cluster, bringing us closer to being able to utilize machine learning to very quickly calculate g-functions. In the future, this will support automated optimization of ground heat exchanger designs.

Introduction

Ground source heat pump systems commonly use vertical borehole ground heat exchangers to serve as a heat sink and source. Since the ground heat exchanger (GHE) represents a significant portion of the system cost, it is important to accurately "right-size" the GHE. This is particularly the case as large systems become more common. Current interest in GSHP technology began in the 1970s with residential applications, but there are many large commercial systems with hundreds of boreholes, and some "district-scale" systems with more than a thousand boreholes. Two large district-scale examples are the Ball State University system with more than 3600 boreholes and the Epic Systems system with more than 6000 boreholes. (Florea et al. 2017)

For both large systems serving commercial and institutional buildings, g-functions are needed for design and simulation purposes. Calculation of these g-functions, whether done for a specific design, or as part of development of a machine learning training set, is of interest. Due to available computing resources, the work in this paper is focused on borehole configurations with a maximum number of boreholes around 1000.

Dusseault and Pasquier (2019) demonstrated use of artificial neural networks to compute g-functions. 500,000 g-functions computed for fields of up to 10 boreholes were used to train the neural network. Pasquier (2019) has estimated that a million g-functions might be needed to train a neural network. As described by Spitler et al. (2020), the computing time and memory scale approximately with the square of the number of segments used. Each borehole is divided into multiple segments treated as finite line sources and the accuracy depends on the number of segments used. To compute g-functions for ~1000 borehole configurations with deep (~400m) boreholes even the resources of our university's high-performance computing cluster (OSUHPCC 2020)are insufficient, as we have very few nodes with available RAM exceeding 96 GB.

Therefore, any effort to develop neural network training sets, or even libraries of g-functions that include very large borehole configurations will require development of an improved tool that is faster and uses less memory than pygfunction. This is also necessary for individual users wishing to compute custom g-functions on their desktop PC. The work described in this paper represents a

significant improvement in both speed and memory requirements.

Background

Cimmino and Bernier (2014) developed a semi-analytical methodology for calculating g-functions of a ground heat exchanger consisting of one or more borehole heat exchangers. The semi-analytical method relies on discretizing each borehole into discrete finite line sources. The single integral analytical finite line source model presented by Claesson and Javed (2011) was adapted by Cimmino and Bernier (2014) to calculate the response between two buried segments. The g-function is calculated for a uniform borehole wall temperature throughout the field. This requires that the heat injection rate to each finite line source be adjusted over time to maintain a constant heat injection rate for the entire ground heat exchanger, while also meeting a prescribed boundary condition at any time for all segments.

Different boundary conditions may be applied to the ground heat exchanger in order to calculate the g-function. It is important to understand the differences between the boundary conditions:

- Uniform heat flux (UHF). The heat is uniformly distributed, and all boreholes have the same heat flux, i.e. the total heat input used to calculate the g-function is divided by the total borehole length. This is relatively easy to calculate, but as shown by Malayappan and Spitler (2013), the resulting g-function may significantly overpredict the temperature response as the number of boreholes increases.
- Uniform borehole wall temperature (UBHWT). The borehole wall temperatures have time-varying but uniform temperature (i.e., the same borehole wall temperature for all boreholes at any given time). Calculating the heat input rates for each segment at each time requires solution of a large equation set.
- Uniform inlet fluid temperature (UIFT). All of the boreholes receive fluid at the same temperature. The actual distribution of heat is then calculated as part of the calculation of the g-function. Again, calculating the individual heat inputs requires solution of a large equation set. G-functions calculated with the UIFT boundary condition show some sensitivity to the borehole thermal resistance and fluid flow rates.

The UIFT boundary conditions are arguably the closest to the physical situation – in most system designs, the fluid returning from the heat pump(s) to the GHE is well-mixed and other than minor heat losses or gains in the horizontal connecting piping, the inlet fluid temperatures should be the same for all boreholes. For design and simulation purposes g-functions calculated with the UBHWT boundary conditions have been used for many years and the results for design purposes have been validated by Cullin et al. (2015).

Further development (Cimmino 2018a, Cimmino 2019) led to an open source g-function toolbox, written in Python, known as pygfunction (Cimmino 2018b). pygfunction can calculate g-functions for any user-defined ground heat exchanger, for all three boundary conditions.

Pygfunction is a very powerful tool for calculating g-functions, but for larger borehole fields with hundreds of boreholes and thousands of finite line sources, the required memory and computational time can be quite high, exceeding several hours and requiring hundreds of Gigabytes of RAM. Figure 1 shows the computational time and memory requirements for calculation of g-functions for rectangular borehole fields up to 1024 boreholes, with up to 24 finite line sources per borehole. The computational time and memory are plotted against the total number of finite line source segments, n_q, for each ground heat exchanger. (I.e. n_q is the number of boreholes multiplied by the number of finite line source segments per borehole.) Cimmino and Bernier (2014) recommended 12 finite line sources per borehole. As shown by Spitler et al. (2020), the g-functions have some sensitivity to the discretization and for larger borefields and deeper boreholes, more than 12 finite line sources may be needed. Further research will explore this sensitivity in more detail and show how this sensitivity can be exploited to reduce computation time and memory requirements.

The calculations used for Figure 1 were run with Pygfunction 1.1.1 on Oklahoma State University's High Performance Computing Cluster (OSUHPCC). All compute nodes on the cluster use dual Intel Skylake 6130 CPUs. (OSUHPCC 2020) There are 164 nodes with 96 GB RAM, 12 nodes with 768 GB RAM and one node with 1.5 TB RAM. For larger fields with several hundred boreholes or more, the memory required can easily exceed what's available on most desktop computers, and even exceed the 96 GB available on the most commonly available OSUHPCC nodes.

Therefore, the amount of memory required can be very important, and, in our view, the most important feature of the new software is a nearly 8-fold reduction in memory requirements.

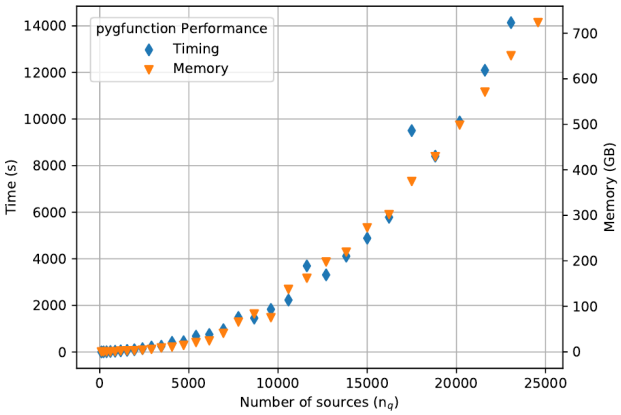

Figure 1 CPU time and memory requirements for g-function calculation with pygfunction

Methodology

This section gives an overview of the cpgfunction methodology, much of which is derived directly from

pygfunction. In order to benefit from the increased speed of a compiled language, cpgfunction was written in C++. Both programs require calculation of segment-to-segment response functions, using Cimmino and Bernier's adaptation (2014) of Claesson and Javed's (2011) single integral analytical finite line source formulation. Letting $h_{ij}(t_k)$ represent the dimensionless mean temperature rise of the *j*th segment due to a unit heat flux input at the *i*th segment, after time t_k, a matrix $H[i,j,t_k]$ is formed in pygfunction and later used to calculate the g-function. This matrix is responsible for much of pygfunction's memory consumption.

The *H* matrix has a form of symmetry across the diagonal; if L_i, L_j are the lengths of the *i*th and *j*th segments, respectively:

$$L_i \cdot H[i,j,t_k] = L_j \cdot H[j,i,t_k] \quad (1)$$

Given the need to reduce the memory consumption, only the lower triangular half of the matrix is stored; when needed, values from the upper triangular half are recalculated using a reordered form of Eqn. 1. The lower half of the matrix is stored in a vector using an index transformation:

$$I_{dx}(i,j) = \frac{i*(2*n_q-i-1)}{2} + j \quad (2)$$

Where n_q is the total number of finite line sources, and I_{dx} is the index for the vector.

In order to calculate the UBHWT and UIFT boundary conditions, pygfunction allocates memory and computes variants of the segment response matrix H, which are used in the spatial and temporal superposition processes. It is not currently clear if this is done for readability and organizational purposes, or if it is necessary to maintain speed performance. Cpgfunction does not allocate additional space for these matrices, but accesses the *H* matrix itself. The resulting memory savings of using a single triangular half matrix and eliminating the two additional matrices gives approximately a six-fold reduction in memory required. Beyond this reduction, the containers (specifically the vector container available in the standard C++ library) used in cpgfunction are minimized. C++ containers handle the memory allocation and deallocation that would have to be explicitly described in C. Each container is only allocated once, and the locations in those arrays overwritten with new values rather than replaced. The net impacts, as shown later, are, in practice, reductions between five-fold and nine-fold.

These memory reductions come at a computational time cost once the problem becomes large enough, but in our view, this is needed for large ground heat exchangers where the memory requirements can easily exceed available machine memory. The extra computational time of cpgfunction stems from the departure from complete vectorization (as utilized in pygfunction) after the segment response matrix is filled. This had to be done due to the reduction in size of the *H* matrix, and the need to access it by Eqn. 2. Cpgfunction makes extensive use of multithreading to perform calculations in parallel. The possibility of introducing critical race conditions make development of mathematical computation programs containing multithreading tedious in nature. Additionally, the cpgfunction code is not as straightforwardly readable as that of pygfunction.

Having spoken of the general symmetry of the *H* matrix, it is important to note that both programs take advantages of other symmetries for reducing the number of calculations of segment-to-segment response functions. In large rectangular-grid configurations, there are many, many identical pairs of segments. (By "identical", we mean they have the same horizontal distance and same vertical distance, within some tolerance.) As discussed by Cimmino (2018a), use of these similarities greatly speeds the g-function computation for cases where there are large numbers of identical pairs.

Regarding the boundary conditions, it should be noted that cpgfunction currently only implements calculation of the UBHWT boundary conditions; calculation of UIFT boundary conditions would be a desirable capability.

Results

The results include a validation of accuracy; comparisons of RAM; comparisons of timing for high-similarities and low-similarities cases.

Verification and Timing for Example Cases

Cpgfunction has been verified against pygfunction for a variety of cases. The UBHWT boundary condition in cpgfunction matches the results of pygfunction quite accurately. Consider the four cases shown in Figure 2 – an L-shape, U-shape, open rectangle, and a configuration generated with Poisson disk sampling.

Hill's (2017) implementation of Bridson's (2007) Poisson disk sampling algorithm is used to generate this field and other fields (below) with reduced similarities. Compared to a true random placement of boreholes, using the Poisson disk sampling algorithm eliminates the possibilities of having boreholes closer together than a minimum distance, or farther apart than twice that distance. Here we used 5 m as the minimum distance.

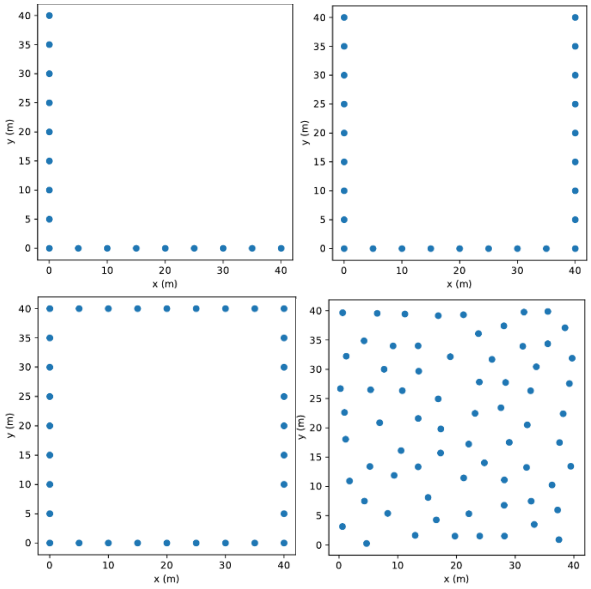

Figure 2 Borehole configurations - illustrative test cases

Figure 3 shows the g-functions computed with each program; the RMSE differences between the 27 points of each g-function, is less than 0.1%.

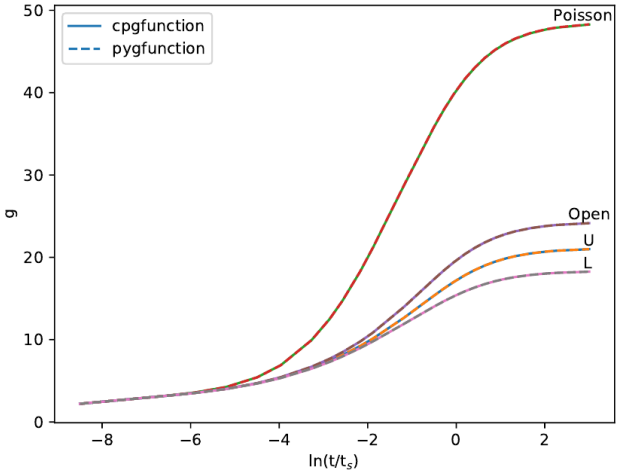

Figure 3 Comparison of g-functions calculated with cpgfunction and pygfunction

Table 1 summarizes the time required by the two programs; the last column gives the ratio of the time required by pygfunction to that required by cpgfunction. It could be considered the "speed ratio". The g-function is represented as a series of *g*-values, computed for specific values of $\ln(t/t_s)$. The amount of time to compute a g-function varies linearly with the number of g-values computed. All g-functions in this paper are represented by 27 *g*-values. This ratio is related to the relative prevalence of similar segment pairs. Cases with higher numbers of similar pairs have lower ratios. Ordering the cases as Poisson Disk, L, U, Open rectangle, orders the ratio from highest to lowest and the prevalence of similar pairs from lowest to highest.

This trend can be explained as follows. The g-function calculation occurs in two parts: (1) computing segment-to-segment responses and filling the segment matrix, and (2) building and solving systems of equations. In process (1), cpgfunction calculates segment-to-segment response factors notably faster than pygfunction, due to the advantage of using a compiled language (C++ instead of Python.) Both programs incorporate similarity identification presented by Cimmino (2018a) to reduce the number of necessary segment-to-segment responses (FLS integrations) to be computed. Segment-to-segment responses that are similar contain pairs of segments with identical or near-identical horizontal and vertical offsets, such that the segment-to-segment responses are identical or near-identical. However, a high prevalence of similar segments reduces the advantage of cpgfunction. This occurs because less time is spent in process (1), and the remainder of cpgfunction contains less vectorization than pygfunction. Therefore, the building and solving systems of equations that adjust the distribution of heat are comparatively slower in cpgfunction than in pygfunction. The segment response matrix must be indexed by Equation 2, thus cpgfunction's speed relies heavily on multithreading (by use of thread pools). The superposition process in pygfunction occurs significantly faster than the multithreaded version of cpgfunction; there are simply too many operations occurring in series in this function. Additionally, multidimensional arrays are being used to sum into a vector of segment wall temperatures. This compression into one matrix could introduce a critical race condition if not threaded properly. However, in future development, the multithreaded loop could be completely unwrapped by copying the necessary values out of the segment response matrix into matrices that could use vectorization. Unravelling the loop and copying rather than operating would give a significant increase in speed. Once the full matrix for the current time step is built, vectorization could be used (like pygfunction). It is anticipated that this would significantly increase the speed of cpgfunction.

Table 1: Summary of example cases

Config.	Number sources	Pygf. time (s)	Cpgf. time (s)	Ratio
L	204	19.7	1.6	12.7
U	300	22.9	2.3	9.9
Open	384	23.3	3.7	6.4
Poisson Disk	840	168.0	11.0	15.3

Results for fields with high similarities

As discussed in the last section, large rectangular borehole configurations have large numbers of identical segment pairs, which reduces the speed advantage of cpgfunction compared to pygfunction.

To compare the performance of cpgfunction and pygfunction for cases with high similarities, g-functions were calculated for uniformly spaced square configurations from 2x2 to 32x32 boreholes. A horizontal spacing of 5m and a depth of 96m were utilized, with 24 segments per borehole. For each case, the computation time was determined within the code by polling the system clock at the beginning and end of the calculation. The RAM consumption is provided by the Simple Linux Utility for Resource Management (SLURM) (Jette et al. 2003), which OSUHPCC makes use of for job scheduling.

Figure 4 shows a comparison of the computational time for the 30 of the 31 cases, plotted against the total number of sources (n_q) in the field. (Even using a compute node with 768 GB of RAM, pygfunction ran out of RAM for the 32x32 borehole case, so pygfunction results are not shown for this case.) The ratio of the time required for pygfunction to that required by cpgfunction is plotted in Figure 5. Up to about 10,000 sources, cpgfunction is faster than pygfunction; above 10,000 sources the two programs have similar computational speed, with cpgfunction being slightly slower than pygfunction for some cases.

As can be seen in Figures 4 and 5, there is some "wobble" in the results – slight deviations from a smooth curve with pygfunction. The reason is not known, but it seems likely that other processes on the compute node may have small effects on the computational time.

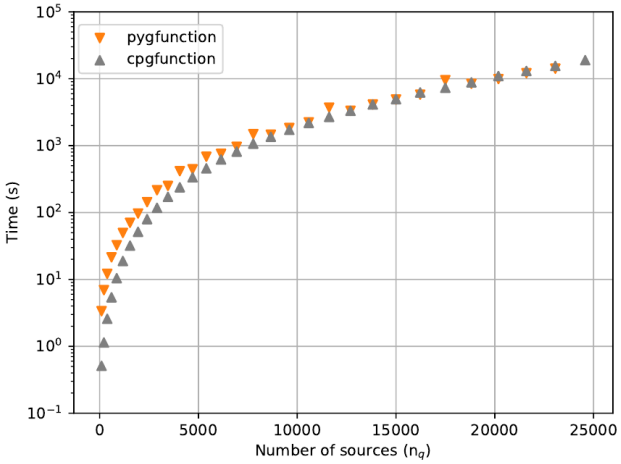

Figure 4 Timing comparisons for square configurations

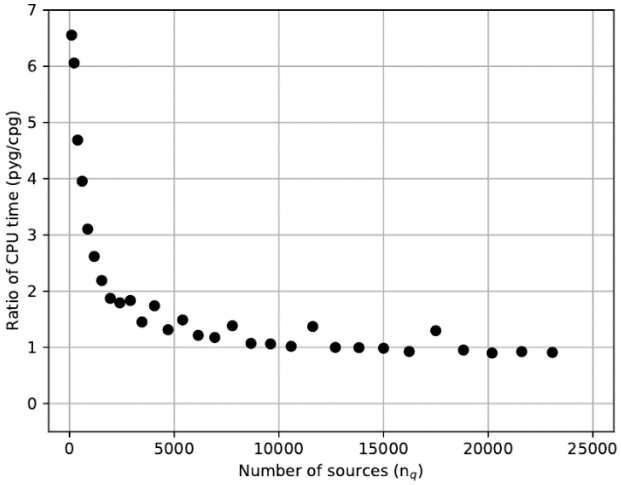

Figure 5 Ratio (pygf./cpgf.) of computing time

Pygfunction uses the NumPy package (Harris et al. 2020) which provides compiled-language-like speed when performing matrix operations. Here, the computational speed of pygfunction exceeds that of cpgfunction with configurations containing high number of sources and similarities in large part due to the similarity identification methods of Cimmino (2018a), which reduce the amount of time spent in pure Python code while calculating segment-to-segment response factors. (Again, the more similar segment pairs, the greater the possibility for pygfunction to have a computational edge in speed.) After the segment responses are computed, the remainder of pygfunction's UBHWT g-function calculation depends heavily on NumPy matrix operations which provide access to low-level highly optimized FORTRAN or C functions.

Figure 6 compares the memory requirements for the square borehole configurations. Figure 7 shows the ratio of peak RAM consumed by pygfunction to that consumed by cpgfunction. For larger fields where the RAM becomes a limiting factor, pygfunction uses about seven to nine times as much RAM. Picking just one example – the case with 5400 sources corresponds to a borehole field with 225 boreholes in a 15x15 configuration. For this case, pygfunction requires 20.5GB of RAM – a value that will exceed the available RAM on all but the highest spec desktop computers. Cpgfunction will only require 3.6 GB – a value that is commonly available on modern desktop computers.

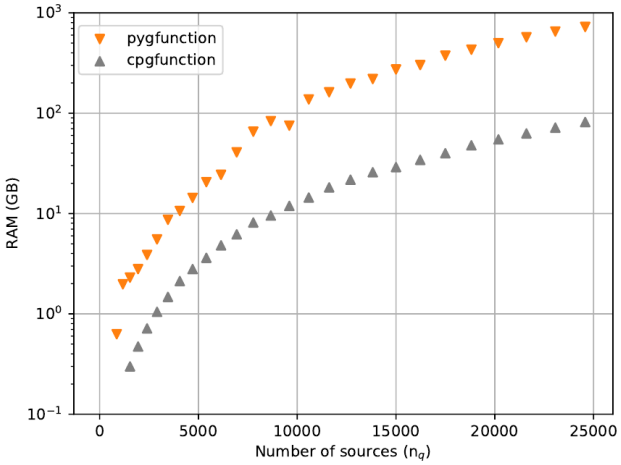

Figure 6 Comparison of peak RAM usage

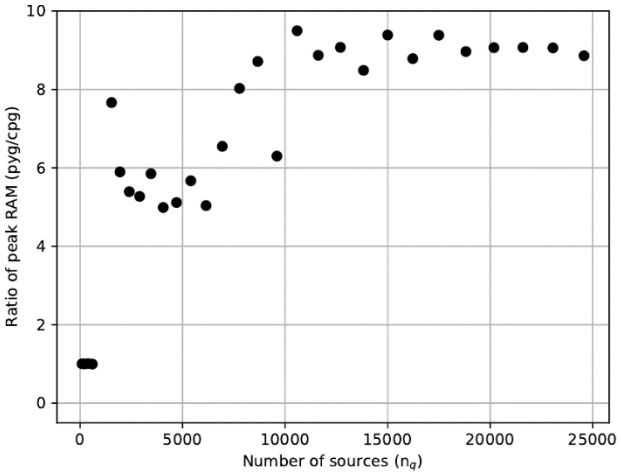

Figure 7 Ratio (pygf./cpgf.) of peak RAM usage

Results for fields with low similarities

To investigate the speed of the two programs for more irregular cases, a series of Poisson disk configurations were calculated. These fields were chosen based on having the same surface area as the rectangular cases above, but the aspect ratio (width to length of the field) was chosen as 4. Then, for each rectangle, a Poisson disk distribution was created. A sample field of 900 m^2 area, with 28 boreholes is shown in Figure 8.

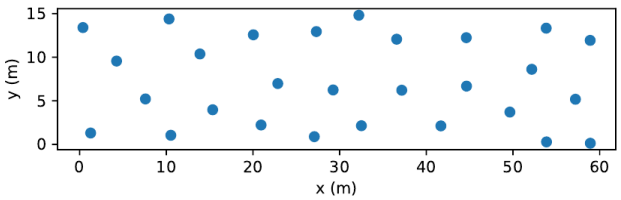

Figure 8 Sample Poisson disk field

Figure 9 shows a comparison of the computation time required for the two programs with the Poisson disk fields, where there are comparatively low numbers of similar pairs. As shown in Figure 10, cpgfunction is significantly faster than pygfunction for all cases; as the number of

sources gets high, cpgfunction is about 4 times faster than pygfunction. This suggests that giving pygfunction access to the segment-to-segment response factor calculation in the cpgfunction library could significantly speed pygfunction.

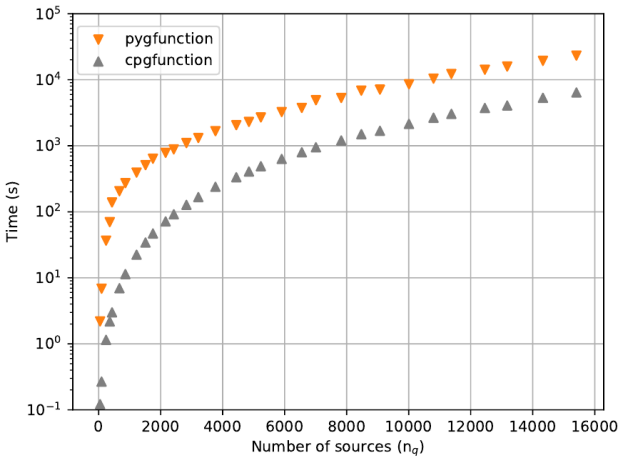

Figure 9 Timing results for the Poisson disk fields

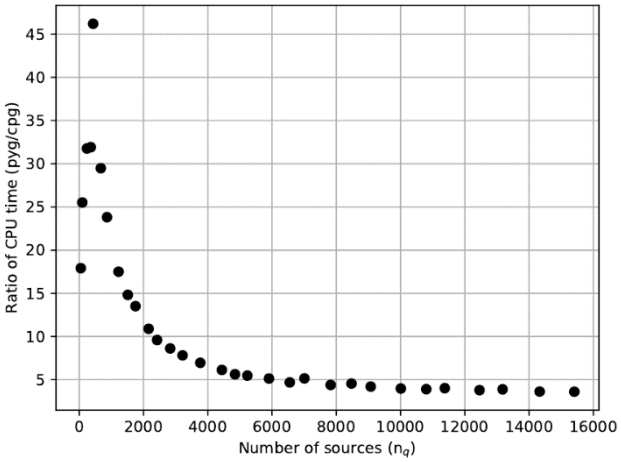

Figure 10 Ratio (pygf./cpgf.) of computing time

Example

For purposes of example, a g-function for a system installed in the 1990s was calculated with both pygfunction and cpgfunction. The 320 borehole system (Dinse 1998)was installed at Daniel Boone High School in Washington County, Tennessee. The boreholes are 46m deep and are laid out in 16 groups of 20 boreholes each. Each group is a 4x5 rectangle with 4.6m spacing, The field consists of two rows of 8 groups, with spacing of 6.1m between each group, as illustrated in Figure 11. Thus, there are considerably more similarities than there would be in a 320-borehole Poisson disk field, but less than there would be in a uniformly-spaced 32x10 rectangular field. For both programs, 12 segments per borehole were used for a total of 3840 segments.

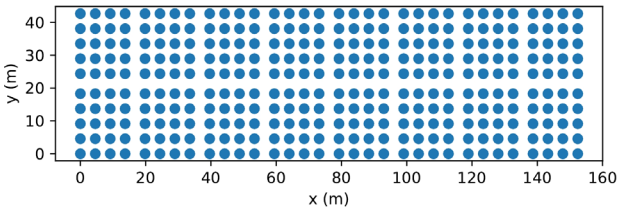

Figure 11 Example borehole field

The two g-functions are shown in Figure 12. The RMSE difference between the two g-functions, each with 27 points, is less than 1%. For this case, pygfunction required 10.8 GB of RAM and cpgfunction required 2.0 GB. The computation time for each step is summarized in Table 2. As can be seen, cpgfunction has a significant advantage in calculating the segment-to-segment response factors. This might be incorporated in pygfunction by making use of a compiled version of this portion of cpgfunction as part of a library used by pygfunction. Also noticeable is the fact that pygfunction, by being able to take fuller advantage of very fast linear algebra libraries is faster at building and solving the system of equations. This comes at the cost of using about 5.3 times the RAM of cpgfunction.

Table 2: Summary of 320-borehole example

Step	Pygf. time (s)	Cpgf. time (s)	Ratio
Identifying similarities	258.5	29.9	8.6
Calculating segment-to-segment response factors	293.1	2.1	139.6
Building and solving system of equations	108.3	197.7	0.55
Total	662	229.8	2.8

One may also infer that, for cpgfunction, it may not be worth the computational effort expended to identify similarities. This needs further investigation.

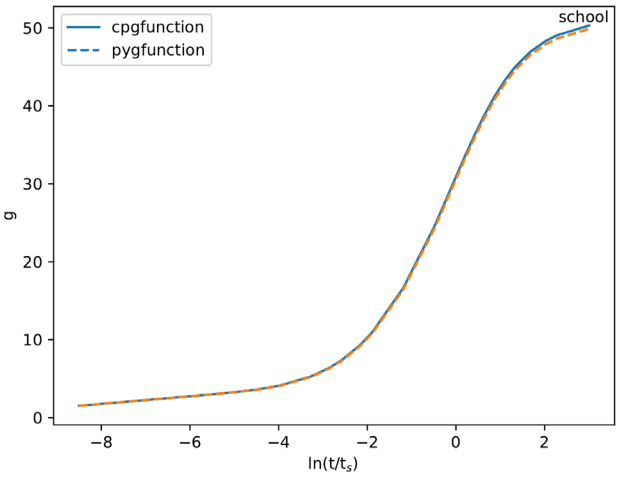

Figure 12 Example borehole field g-functions

Conclusion

The authors anticipate three uses for cpgfunction:

1. Calculation of custom g-functions for use in design and simulation programs.
2. Calculation of large libraries of g-functions for use in design and simulation programs.
3. Calculation of training sets for use in developing ANN that can very quickly calculate g-functions.

For all three applications, the RAM requirements can represent a significant hurdle to calculation of g-functions. For the first application, the improvements shown in cpgfunction can allow calculation of g-functions for larger borehole fields on desktop computers, for which pygfunction would exceed the available RAM. For the second two applications, the reduced RAM requirements allow much higher throughput on a high-performance computing cluster where the availability of compute nodes with RAM exceeding 100 GB is limited.

With regards to computational time, the relative performance of pygfunction and cpgfunction is highly dependent on the arrangement of the field. With a high degree of similarities (large numbers of segment pairs with the same horizontal and vertical distances), pygfunction is extremely efficient for large fields, achieving computational speeds up to 10% faster than cpgfunction. However, for less regular fields where the degree of similarities is lower, cpgfunction can be significantly faster. In the case of a highly irregular field with few similarities, cpgfunction is about four times faster than pygfunction.

Cpgfunction currently only calculates g-functions using the UBHWT boundary conditions, so further work to incorporate the UIFT boundary conditions is recommended. For pygfunction, it seems likely that some of the modifications that reduced memory consumption could be applied to reduce pygfunction's memory consumption. It also appears that pygfunction's speed could be increased by using the segment-to-segment response factor calculation in the cpgfunction library.

Acknowledgements

The work in this paper builds upon theory developed by Prof. Johan Claesson and his graduate students, Per Eskilson and Göran Hellström, of Lund University in Sweden. It builds and even more heavily on the further developments of Dr. Massimo Cimmino, Assistant Professor at Polytechnique Montreal in Canada. Their contributions to the field are gratefully acknowledged.

References

Bridson, R. (2007). Fast Poisson disk sampling in arbitrary dimensions. ACM SIGGRAPH 2007 Sketches, SIGGRAPH'07, August 5, 2007 - August 9, 2007, San Diego (United States), Association for Computing Machinery.

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

Cimmino, M. (2018b). pygfunction: an open-source toolbox for the evaluation of thermal. eSim 2018, Montreál, IBPSA Canada.

Cimmino, M. (2019). "Semi-Analytical Method for g-Function Calculation of bore fields with series- and parallel-connected boreholes." Science and Technology for the Built Environment **25**(8): 1007-1022.

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

Claesson, J. and P. Eskilson (1985). Thermal analysis of heat extraction boreholes. Enerstock 85 (the 3rd International Conference on Energy Storage for Building Heating and Cooling), Toronto, Canada Public Works and Government Services Canada.

Claesson, J. and S. Javed (2011). "An Analytical Method to Calculate Borehole Fluid Temperatures for Time-scales from Minutes to Decades." ASHRAE Transactions **117**(2): 279-288.

Cullin, J. R., J. D. Spitler, C. Montagud, F. Ruiz-Calvo, S. J. Rees, S. S. Naicker, P. Konečný and L. E. Southard (2015). "Validation of vertical ground heat exchanger design methodologies." Science and Technology for the Built Environment **21**(2): 137-149.

Dinse, D. R. (1998). "Geothermal System for School." ASHRAE Journal **40**(5): 52-54.

Dusseault, B. and P. Pasquier (2019). "Efficient g-function approximation with artificial neural networks for a varying number of boreholes on a regular or irregular layout." Science and Technology for the Built Environment **25**(8): 1023-1035.

Florea, L. J., D. Hart, J. Tinjum and C. Choi (2017). "Potential Impacts to Groundwater from Ground-Coupled Geothermal Heat Pumps in District Scale." Groundwater **55**(1): 8-9.

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant (2020). "Array programming with NumPy." Nature **585**(7825): 357-362.

Hill, C. (2017). "Poisson disc sampling in Python." Retrieved 29 January, 2021, from https://scipython.com/blog/poisson-disc-sampling-in-python/.

Jette, M. A., A. B. Yoo and M. Grondona (2003). SLURM: Simple Linux Utility for Resource Management. Job Scheduling Strategies for Parallel Processing, Seattle, Washington (USA).

Malayappan, V. and J. D. Spitler (2013). Limitations of Using Uniform Heat Flux Assumptions in Sizing Vertical Borehole Heat Exchanger Fields. Clima 2013. Prague (Czech Republic).

OSUHPCC. (2020). "OSU's newest supercomputer "Pete" is available for all OSU researchers." Retrieved 29 January, 2021, from https://hpcc.okstate.edu/pete-supercomputer.html.

Pasquier, P. (2019). E-mail. J. D. Spitler.

Spitler, J. D., J. C. Cook and X. Liu (2020). "Recent Experiences Calculating g-functions for Use in Simulation of Ground Heat Exchangers." Geothermal Resources Council Transactions **44**: 296-315.