Guaranteed Automatic Integration Library (GAIL): An Open-Source MATLAB Library for Function Approximation, Optimization, and Integration

Journal of
open research software

SOFTWARE METAPAPER

]u[ubiquity press

XIN TONG
SOU-CHENG T. CHOI
YUHAN DING
FRED J. HICKERNELL
LAN JIANG
LLUÍS ANTONI JIMÉNEZ RUGAMA
JAGADEESWARAN RATHINAVEL
KAN ZHANG
YIZHI ZHANG
XUAN ZHOU

*Author affiliations can be found in the back matter of this article

ABSTRACT

Function approximation, integration, and optimization are three fundamental mathematical problems. They are especially challenging when the functions involved fluctuate wildly in certain parts of the domain, or if the domain is high dimensional. Ideally, algorithms to solve these problems should possess a rigorous mathematical framework, data-based (probabilistic) error bounds, and advanced sampling strategies for efficiency.

The Guaranteed Automatic Integration Library (GAIL) is our multi-year research effort addressing these aforementioned challenges. GAIL is a free, open-source MATLAB software library with nine main algorithms undergirded by over a dozen peer-reviewed publications. GAIL solves problems in univariate and multivariate integration, and in univariate function approximation and optimization. GAIL algorithms adaptively sample data values of the input function and automatically stop when the error tolerance has been reached. In some cases, GAIL algorithms are proven to have asymptotically optimal computational cost. We consistently employ good software development practices for GAIL such as unit tests, searchable online documentation, and Git version control. GAIL is available at https://gailgithub.github.io/GAIL_Dev/.

CORRESPONDING AUTHOR:

Xin Tong

Department of Scientific Computing, Florida State University, Tallahassee, FL 32306, US

xtong5@hawk.iit.edu

KEYWORDS:

Univariate Function Approximation; Multivariate Integration; Univariate Optimization; Adaptive; (Quasi-)Monte Carlo; Bayesian Cubature

TO CITE THIS ARTICLE:

Tong X, Choi S-CT, Ding Y, Hickernell FJ, Jiang L, Jiménez Rugama Ll A, Rathinavel J, Zhang K, Zhang Y, Zhou X 2022 Guaranteed Automatic Integration Library (GAIL): An Open-Source MATLAB Library for Function Approximation, Optimization, and Integration. *Journal of Open Research Software*, 10: 7. DOI: https://doi.org/10.5334/jors.381

(1) OVERVIEW

INTRODUCTION

Function approximation, integration, and optimization are fundamental problems requiring numerical solutions that come from iterative algorithms. A crucial question is how and when to stop the computation.

Theoretical error bounds typically contain unknown quantities, such as the norm of the input function. This makes them impractical as stopping criteria.

Therefore, practical algorithms that adapt the computation to the error requirement are often based on heuristics. These include a popular adaptive quadrature algorithm of Shampine [25], which is a part of MATLAB [27], and the Chebfun library [7]. Heuristics based on function data tend to lack theoretical support; one does not know when they work and when they do not. A warning against commonly used adaptive quadrature stopping criteria is given by Lyness [24].

To address these shortcomings, we have developed adaptive stopping criteria for univariate function approximation, integration, and optimization; and for multivariate integration. We have implemented them in the Guaranteed Automatic Integration Library (GAIL) [2]. In contrast to other automatic or adaptive algorithms, our GAIL algorithms have theoretical foundations that are detailed in a series of articles and graduate theses [3, 5, 6, 8, 9, 10, 16, 17, 18, 19, 20, 23, 28, 29].

The underlying idea in our GAIL algorithms is that for reasonable functions *what you see is nearly what you get*. The initial sampling of the function to be approximated, integrated, or optimized tells us enough about its norm so that we can compute data-driven error bounds. For each algorithm, there is an associated set of reasonable functions corresponding to a cone, C. Mathematically, "cone" means that a constant multiple of every function in C is also in C. For adaptive simple (i.e., independent and identically distributed) Monte Carlo integration algorithms [10, 18], C corresponds to a function whose kurtosis is no larger than some bound. The bound reflects the user's definition of "reasonable". For adaptive univariate algorithms [3, 5, 6, 28, 29], C corresponds to functions for which a stronger norm is bounded in terms of a weaker one. For quasi-Monte Carlo integration algorithms [8, 9, 16, 17, 19, 20], C corresponds to functions whose Fourier complex exponential or Walsh coefficients decay steadily. For Bayesian quasi-Monte Carlo algorithms [16, 17], C corresponds to typical (non-outlier) Gaussian processes within the sample space.

IMPLEMENTATION AND ARCHITECTURE

GAIL includes the following algorithms. All core algorithm names end with "_g" to denote some form of accuracy *guarantee*. The last one, **meanMC_CLT**, is the only exception and is a stopping criterion based on the Central Limit Theorem for pedagogical purposes. Figure 1 shows the structure of GAIL.

1. One-dimensional algorithms:
   (a) **funappx_g** [3, 5, 6]: One-dimensional function approximation on a closed, bounded interval;
   (b) **funmin_g** [3, 28]: Global minimum value of univariate function on a closed, bounded interval;
   (c) **integral_g** [5, 29]: One-dimensional integration on a bounded interval.
2. Multi-dimensional algorithms:
   (a) **meanMC_g** [10, 18]: simple Monte Carlo method for estimating mean of a random variable;
   (b) **cubMC_g** [10, 18]: simple Monte Carlo method for numerical multiple integration;
   (c) **cubLattice_g** [20]: Quasi-Monte Carlo method using rank-1 lattice cubature for d-dimensional integration;
   (d) **cubSobol_g** [9, 20, 23]: Quasi-Monte Carlo method using Sobol' cubature for d-dimensional integration;
   (e) **cubBayesLattice_g** [17]: Bayesian cubature method for d-dimensional integration using lattice points;
   (f) **cubBayesNet_g** [16, 17]: Bayesian cubature method for d-dimensional integration using Sobol points;
   (g) **meanMC_CLT**: Monte Carlo method with Central Limit Theorem (CLT) confidence intervals for estimating mean of a random variable.

Figure 2 shows the architectural design of GAIL algorithms. A GAIL algorithm typically takes in

- a real-valued function, f,
- its domain, D (e.g., finite interval or hyperbox),
- user tolerance, $\epsilon > 0$,
- an initial number and a maximum number of sample points in D at which f are evaluated, n_0,
- a maximum number of sample points n_N, and
- a maximum number of iterations, I.

Among all the inputs, only f is compulsory. Other inputs are optional and implemented with default values as specified in the documentation. We note that in some multiple integration algorithms, the user tolerance may be a generalized error tolerance function, $\max(\epsilon_a, |y|\epsilon_r)$, where $\epsilon_a > 0$ and $\epsilon_r > 0$ are respectively absolute and relative tolerances, and y is the unknown true solution. Each algorithm may have its unique additional inputs. For instance, a (quasi-)Monte Carlo algorithm typically has an input dimension, d.

In the ith iteration, a GAIL algorithm evaluates an estimated solution sol_i and its error bound, e_i, using n_i function samples. When $e_i \le \epsilon$, the algorithm designates the iteration as the last iteration, l and returns the outputs $sol = sol_l$, $e = e_l$, and an exit flag that indicates algorithmic success, along with other outputs that are specific to the algorithm. Other less satisfactory stopping conditions are $i == I$ or $n_i == n_N$, and the returned exit flag would note

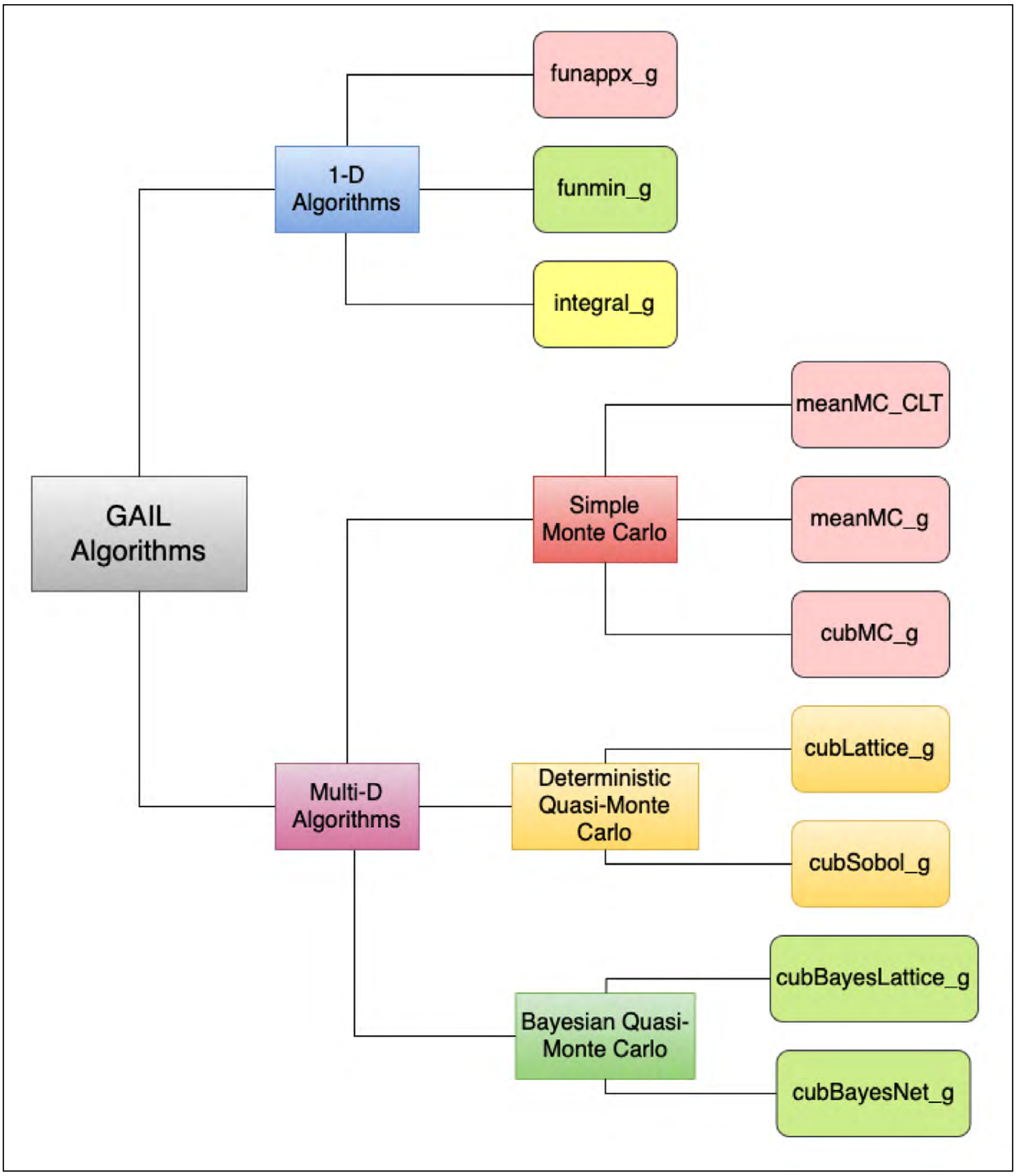

Figure 1 Structure of GAIL Algorithms.

such non-success. We refer readers to [11] for details of stopping criteria in GAIL's algorithms.

Each one of our key GAIL algorithms, except for **cubBayesLattice_g** and **cubBayesNet_g**, can parse inputs with the following three patterns of Application Programming Interfaces (APIs), where `f` is a real-valued MATLAB function or function handle; `in_param` and `out_param` are MATLAB structure arrays; and x is an estimated output:

1. Ordered input values: `[x, out_param] = algo(f, val1, val2, val3,...)`
2. Input structure array: `[x, out_param] = algo(f, in_param)`
3. Ordered input values, followed by optional name-value pairs: `[x, out_param] = algo(f, 'input1', val1, 'input2', val2,...)`

For **cubBayesLattice_g** and **cubBayesNet_g**, they are implemented in object-oriented design (whereas others are in modular functions). Hence, their interfaces are slightly different. We refer readers to Table 4 for an example and GAIL documentation for details. The three forms of aforementioned inputs are still applicable, but the outputs of the Bayes methods are `[obj, x]` instead, where `obj` is an instance of the object class. To obtain the same output form of other algorithms, users can simply do an additional, optional step: `[x, out_param] = compInteg(obj)`. Another small difference is that the second input parameter, dimension `d`, in the two Bayes algorithms is compulsory.

We note that almost all high-dimensional integration algorithms in GAIL have been re-implemented (with extensions) in the open-source Python software library, QMCPy [4].

In the following, we showcase GAIL's performance with two examples on univariate function optimization and cubature.

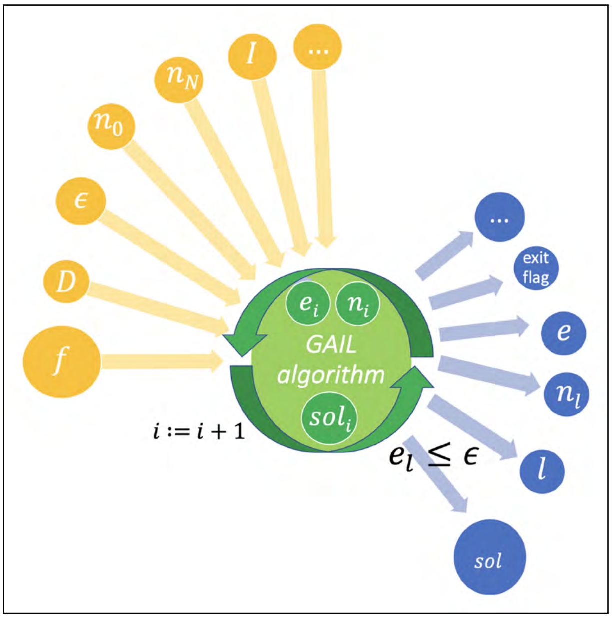

Figure 2 GAIL architectural design. The largest yellow circle contains a compulsory input function f. The other inputs in small yellow circles are typically optional and, when absent, set to default values in the GAIL algorithms. Each GAIL algorithm is iterative in nature. In the ith iteration, a solution estimate sol_i is computed along with its error estimate e_i obtained by n_i sampling points in the input domain D. When e_i is not greater than the tolerance, ϵ, GAIL iterations stop and return the final numerical solution sol (in the largest blue circle). Other outputs (in small blue circles) are bundled in a MATLAB structure array.

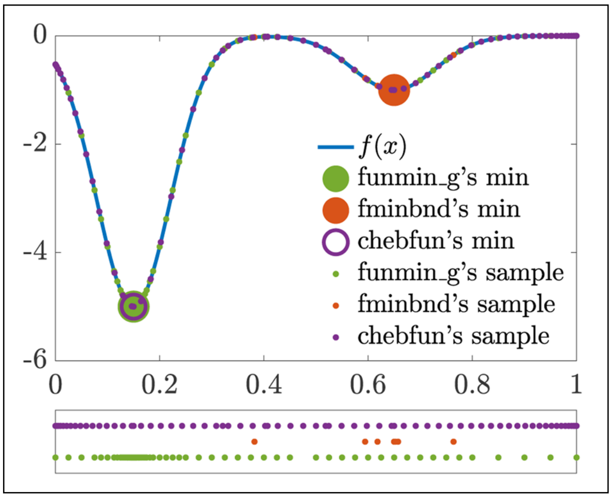

Figure 3 Function f defined in (1), sampling points and best estimates retured by solvers MATLAB's `fminbnd`, Chebfun's `min`, and GAIL's `funmin_g`. This figure is reproducible by the MATLAB script, `demo_funmin_g2_samplepoints.m` available in GAIL's 'develop' branch at https://github.com/GailGithub/GAIL_Dev/tree/develop/GAIL_Matlab/Papers/GAIL_JORS.

Example 1. We want to find the global minimum of the following function:

$$f(x) \quad = -5e^{-100(x-0.15)^2} - e^{-80(x-0.65)^2} \text{ for } x \in [0,1]. \tag{1}$$

We plot the function f in (1), along with the sampling points and best estimates, $(\hat{x}, f(\hat{x}))$ of true minimum $(x^*, f(x^*))$ from three solvers, MATLAB's `fminbnd`, Chebfun's `min`, and GAIL's `funmin_g` in Figure 3. For (1), `funmin_g` automatically samples the function more often in spiky areas and locates the global minimum accurately. In contrast, MATLAB's `fminbnd` [1, 12] returns a local minimum that is not a global minimum. That said, `fminbnd` is designed for seeking a local minimum. Chebfun [14] approximates f with Chebyshev polynomials and samples f at Chebyshev points. Its `min` function is capable of returning all local minima that it can find, out of which we extract the global minimum. Both `min` and `funmin_g` succeeded in locating the global minimum to the required accuracy, with the former being more efficient using fewer sampling points and the latter more accurate, but both met the tolerance of $\epsilon = 10^{-6}$. In addition, Table 1 summarizes the solvers' performance. In Table 2, we show the essential code for setting up `funmin_g` for this example.

Example 2. In this example, we compare GAIL's Monte Carlo and quasi-Monte Carlo methods in similar ways as in Section 4 in [15] with the Keister integrals [21]:

$$\int_{[0,1]^d} \pi^{d/2} \cos\left(\sqrt{\frac{1}{2}\sum_{j=1}^{d} \Phi^{-1}(x_j)}\right) d\boldsymbol{x}. \tag{2}$$

where $\boldsymbol{x}$ represents vectors in the d-dimensional unit hypercube. In Table 3, we summarize the performance of the methods MC, Lattice, Sobol, Bayes Lattice, and Bayes Net—they refer to the GAIL cubatures, `cubMC_g`, `cubLattice_g`, `cubSobol_g`, `cubBayesLattice_g`, and `cubBayesNet_g`, respectively. In the case of $d = 3$, all five methods succeeded completely meaning the absolute error is less than given tolerance, i.e., $|\mu - \hat{\mu}| \le \epsilon$, where $\hat{\mu}$ is a cubature's approximated value and μ is the true value of (2). In the case of $d = 8$, success rate is at least 98% for each GAIL cubature. The fastest method was `cubSobol_g` for the two cases, whereas `cubBayesNet_g` used the least number of sampling points. `cubBayesLattice_g` and `cubSobol_g` achieved the smallest average absolute error for $d = 3$ and $d = 8$, respectively. The code in Table 4 shows how the problem with $d = 3$ is solved with the GAIL solvers.

QUALITY CONTROL

The testing of GAIL library is automated as scheduled tasks. There are two kinds of tests run: fast tests and long tests.

As aptly named, the fast tests take a relatively short time to run so that a user can quickly test the sanity of the library and the installation. Essential capabilities of

METHOD	FUNMIN_G	FMINBND	MIN
$\lvert\hat{x} - x^*\rvert$	1.0×10^{-10}	0.5	1.0×10^{-8}
$\lvert f(\hat{x}) - f(x^*)\rvert$	0	4.0	1.3×10^{-7}
n	113	10	37
Time (seconds)	0.042	0.048	0.022

Table 1 Performance of `funmin_g, fminbnd`, and `min` with automatic stopping criteria for optimizing the function defined in Example 1. This table is reproducible by the MATLAB script, `demo_funmin_g2_samplepoints.m`.

```
f = @(x) -5*exp(-100*(x-0.15).^2) - exp(-80*(x-0.65).^2);
[fmin,out] = funmin_g(f);
```

Table 2 Essential code in the MATLAB script, `gail_jors_eg1.m`, for invoking `funmin_g` in Example 1.

	$d = 3$, $\epsilon = 0.005$				
METHOD	MC	LATTICE	SOBOL	BAYES LATTICE	BAYES NET
Absolute Error	1.1×10^{-3}	5.2×10^{-4}	5.2×10^{-4}	3.4×10^{-7}	5.8×10^{-4}
Tolerence Met	100%	100%	100%	100%	100%
n	2500000	4100	3900	4100	1800
Time (seconds)	0.1700	0.0097	0.0065	0.0100	0.1200
	$d = 8$, $\epsilon = 0.050$				
METHOD	MC	LATTICE	SOBOL	BAYES LATTICE	BAYES NET
Absolute Error	1.2×10^{-2}	1.4×10^{-2}	6.9×10^{-3}	2.1×10^{-1}	8.8×10^{-3}
Tolerence Met	100%	99%	100%	98%	100%
n	7400000	15000	16000	1000000	8200
Time (seconds)	1.1000	0.0380	0.0240	2.4000	0.3600

Table 3 Average performance of cubatures with automatic stopping criteria for estimating the integrals in (2) for 1000 independent runs. These results can be conditionally reproduced with the MATLAB command, `KeisterCubatureExampleJORS(1000)`, in GAIL.

```
a = 1/sqrt(2);
d = 3;
abstol = 0.005;
reltol = 0;
normsqd = @(t) sum(t.*t,2); % squared l_2 norm of t
replaceZeros = @(t) (t+(t==0)*eps); % to avoid getting Inf
yinv = @(t) erfcinv(replaceZeros(abs(t)));
f1 = @(t,d) cos(sqrt(normsqd(yinv(t)))) * (sqrt(pi))^d;
fKeister = @(x) f1(x,d);
inputArgs = {'absTol', abstol, 'relTol', reltol};
hyperbox = [zeros(1,d); ones(1,d)];
[u1,~] = cubMC_g(fKeister, hyperbox, inputArgs{:});
[u2,~] = cubSobol_g(fKeister, hyperbox, inputArgs{:});
[u3,~] = cubLattice_g(fKeister, hyperbox, inputArgs{:});
[~,u4] = cubBayesNet_g(fKeister, d, inputArgs{:});
[~,u5] = cubBayesLattice_g(fKeister, d, inputArgs{:});
```

Table 4 Essential code in the MATLAB script, `gail_jors_eg2.m,` for invoking GAIL's (Q)MC algorithms in Example 2.

the algorithms are quickly checked with carefully chosen tests to make sure the new code has not broken existing algorithms.

The long tests are more rigorous use cases that take much longer time, up to several hours to finish. These tests also typically include the examples from the papers and theses associated with the GAIL algorithms. The long tests are meant to test all the features and capabilities of the algorithms which cannot be covered in the fast tests.

Both types of tests are executed on the Karlin computing cluster hosted at the Illinois Institute of Technology (IIT). These machines run Centos Release 6.10. The Portable Batch System (PBS) is used to schedule the tasks. GAIL library is tested with seven recent MATLAB versions at least. The fast tests are automatically run once everyday.

The fast tests take less than two minutes to finish in our test setup. The long tests are run everyday for at least one version of MATLAB so that all the recent seven MATLAB release versions are covered in a circular rotation. As of this writing, both the fast tests and long tests are run with these MATLAB versions: R2017a, R2017b, R2018a, R2018b, R2019a, R2019b, and R2020a.

Before the tests begin, the 'develop' branch of the GAIL git repository is pulled in. Then the fast tests are run first, followed by the long tests. Automatically the test results are sent as emails to the maintainers.

All of the GAIL code-base is hosted and version-controlled in GitHub at https://github.com/GailGithub/GAIL_Dev. There are three major git branches used: 1) master, 2) develop and, 3) feature. The major releases come out of the 'master' branch after regression testing. A 'feature' branch is where one or more developers host their own rudimentary work and start developing an algorithm. Once the feature branch code reaches a satisfactory level of completion with all the tests passing, it gets merged into the 'develop' branch. The 'develop' branch is used to curate the candidate release algorithms. Periodically, all the developers get together and review the status of the 'develop' branch such as the documentation, code cleanliness, and tests completion before voting to merge with 'master'.

(2) AVAILABILITY

OPERATING SYSTEM

Our software is expected to run on multiple operating systems including but not limited to Windows, Mac, and Linux. Any operating system that is compatible with the MATLAB versions below should be able to run GAIL successfully; please see System Requirements and Supported Compilers at https://www.mathworks.com/support/requirements/previous-releases.html. Our automated test suites are executed daily on CentOS Linux release 6.10.

PROGRAMMING LANGUAGE

MATLAB, versions R2017a–R2021a.

ADDITIONAL SYSTEM REQUIREMENTS

We refer readers to the following page for MATLAB system requirements, which depend on MATLAB version and machine type: https://au.mathworks.com/support/requirements/previous-releases.html.

In addition, the installation of GAIL requires approximately 42 megabytes (MB) of disk space. The memory requirement of executing GAIL applications depends on various factors such as choice of algorithms, user tolerance, and the number of function sampling points. We recommend at least 2 gigabytes (GB) of memory allocated for MATLAB and GAIL.

DEPENDENCIES

GAIL is developed in MATLAB versions R2016a to R2021a. In particular, three of our core algorithms, `cubSobol_g`, `cubBayesNet_g`, and `cubBayesLattice_g` require the following MATLAB add-on toolboxes: Signal Processing Toolbox, Optimization Toolbox, Statistics and Machine Learning Toolbox. As each MATLAB release is associated with a specific version of a MATLAB toolbox, we do not detail the toolbox versions here — if necessary, the toolbox version numbers can be simply determined with the MATLAB command `ver`.

For development and testing purposes, we use the third-party toolboxes, Chebfun [14] and Doctest for MATLAB [26].

LIST OF CONTRIBUTORS

We thank the contributions of current, former, and visiting students in the Department of Applied Mathematics at Illinois Institute of Technology: Noah Grudowski, Cu Hauw Hung, Yueyi Li, Xincheng Sheng, Aleksei Sorokin, Xiaoyang Zhao, and Tianci Zhu.

SOFTWARE LOCATION

Name: MathWorks File Exchange
Persistent identifier: 10.5281/zenodo.4018189
Licence: IIT License; see LICENSE.m in the archive
Publisher: Kan Zhang
Version published: 2.3.2
Date published: 05/09/2021

Code repository

Name: GitHub
Persistent identifier: https://github.com/GailGithub/GAIL_Dev
Licence: IIT License; see LICENSE.m in the zip or tar.gz archives
Date published: 05/09/2021

LANGUAGE

English

(3) REUSE POTENTIAL

GAIL is publicly available as a Git repository hosted on GitHub at https://gailgithub.github.io/GAIL_Dev/. Since GAIL is written in MATLAB, it is accessible by all MATLAB users whose work requires numerical function approximation, integration, or optimization. Multivariate integration arises in fields such as quantitative finance [13] and uncertainty quantification [22].

Users with questions can submit an issue through GitHub Issues. Developers who wish to add algorithms to or enhance GAIL can submit a pull request, or email to the mailing list, gail-users@googlegroups.com.

ADDITIONAL FILE

The additional files for reproducing the results in Tables 1 – 4 and Figure 3 this article can be found as follows:

- **GAIL.** Develop branch. URL: https://github.com/GailGithub/GAIL_Dev/tree/develop/GAIL_Matlab/Papers/GAIL_JORS

ACKNOWLEDGEMENTS

Hickernell, Ding, and Choi wish to thank students from the following IIT courses for discussion: SCI 498 Adaptive Monte Carlo Algorithms with Applications to Financial Risk Management, Summer 2016; MATH 491 Reading & Research, Summer 2015; SCI 498/MATH 491 Computational Social Sciences, Summer 2016; MATH 491-195 Solving Problems in the Social Sciences Using Tools from Computational Mathematics and Statistics, Summer 2015; Math 573/SCI 498 Reliable Mathematical Software, Fall 2013 and Fall 2018.

FUNDING STATEMENT

Our work is supported in part by grants NSF-DMS-1115392 and NSF-DMS-1522687. The publication costs for this article were funded by a gift from a generous Illinois Institute of Technology alumnus.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Xin Tong orcid.org/0000-0003-4718-1198
Department of Scientific Computing, Florida State University, Tallahassee, FL 32306, US

Sou-Cheng T. Choi orcid.org/0000-0002-6190-2986
Department of Applied Mathematics, Room 220, 10 W. 32nd St., Illinois Institute of Technology, Chicago, IL 60616, US; SAS Institute Inc., 2222 Kalakaua Ave, Suite 1400, Honolulu, HI 96815, US

Yuhan Ding orcid.org/0000-0002-7807-1752
Department of Applied Mathematics, Room 220, 10 W. 32nd St., Illinois Institute of Technology, Chicago, IL 60616, US

Fred J. Hickernell orcid.org/0000-0001-6677-1324
Department of Applied Mathematics, Room 220, 10 W. 32nd St., Illinois Institute of Technology, Chicago, IL 60616, US

Lan Jiang
Compass Inc., 90 5th Avenue, 3rd Floor, New York, NY 10011, US

Lluís Antoni Jiménez Rugama
Virtu Financial, 2530 Walsh Tarlton Ln, Austin, TX 78746, US

Jagadeeswaran Rathinavel
Wi-Tronix, LLC, 631 E Boughton Rd #240, Bolingbrook, IL 60440, US

Kan Zhang
Department of Applied Mathematics, Room 220, 10 W. 32nd St., Illinois Institute of Technology, Chicago, IL 60616, US

Yizhi Zhang
Department of Applied Mathematics, Room 220, 10 W. 32nd St., Illinois Institute of Technology, Chicago, IL 60616, US

Xuan Zhou
Morgan Stanley, 1585 Broadway, New York, NY 10036, US

REFERENCES

1. **Brent RP.** *Algorithms for minimization without derivatives.* Courier Dover Publications; 2013.
2. **Choi S-CT,** et al. *GAIL: Guaranteed Automatic Integration Library (Versions 1.0-2.3.2)*. MATLAB software, http://gailgithub.github.io/GAIL_Dev/. 2021. DOI: https://doi.org/10.5281/zenodo.4018189
3. **Choi S-CT,** et al. Local Adaption for Approximation and Minimization of Univariate Functions. In: *J. Complexity,* 2017; 40: 17–33. DOI: https://doi.org/10.1016/j.jco.2016.11.005
4. **Choi S-CT,** et al. *QMCPy: A Quasi-Monte Carlo Python Library;* 2021. URL: https://github.com/QMCSoftware/QMCSoftware. DOI: https://doi.org/10.5281/zenodo.3964489
5. **Clancy N,** et al. The Cost of Deterministic, Adaptive, Automatic Algorithms: Cones, Not Balls. In: *J. Complexity,* 2014; 30: 21–45. DOI: https://doi.org/10.1016/j.jco.2013.09.002
6. **Ding Y.** Guaranteed Adaptive Univariate Function Approximation. PhD thesis. Illinois Institute of Technology; 2015.
7. **Driscoll TA, Hale N, Trefethen LN,** eds. *Chebfun Guide.* Oxford: Pafnuty Publications; 2014.
8. **Hickernell FJ, Jiménez Rugama Ll A.** Reliable Adaptive Cubature Using Digital Sequences. In: *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*. Ed. by R. Cools and D. Nuyens. Vol. 163. Springer Proceedings in Mathematics and Statistics. Springer-Verlag, Berlin, 2016; 367–383. DOI: https://doi.org/10.1007/978-3-319-33507-0_18
9. **Hickernell FJ, Jiménez Rugama Ll A, Li D.** Adaptive Quasi-Monte Carlo Methods for Cubature. In: *Contemporary Computational Mathematics – a celebration of the 80th birthday of Ian Sloan*, Dick J, Kuo FY, Woźniakowski H (eds.). 2018; 597–619. Springer-Verlag. DOI: https://doi.org/10.1007/978-3-319-72456-0

10. **Hickernell FJ,** et al. Guaranteed Conservative Fixed Width Confidence Intervals Via Monte Carlo Sampling. In: *Monte Carlo and Quasi-Monte Carlo Methods* 2012, Dick J, et al. (eds.). 2013; 65: 105–128. Springer Proceedings in Mathematics and Statistics. Springer-Verlag, Berlin. DOI: https://doi.org/10.1007/978-3-642-41095-6
11. **Hickernell FJ,** et al. Monte Carlo simulation, automatic stopping criteria for. In: *Wiley StatsRef-Statistics Reference Online*, Davidian M, et al. (eds.). 2018; John Wiley & Sons Ltd. DOI: https://doi.org/10.1002/9781118445112.stat08035
12. **Forsythe GE, Malcolm MA, Moler CB.** *Computer methods for mathematical computations.* Vol. 8. Prentice-Hall Englewood Cliffs, NJ; 1977.
13. **Glasserman P.** *Monte Carlo Methods in Financial Engineering.* Vol. 53. Applications of Mathematics. New York: Springer-Verlag; 2004. DOI: https://doi.org/10.1007/978-0-387-21617-1_5
14. **Hale N, Trefethen LN, Driscoll TA.** *Chebfun Version 5.7.;* 2017.
15. **Hickernell FJ,** et al. Monte Carlo simulation, automatic stopping criteria for. In: *Wiley StatsRef: Statistics Reference Online,* 2018; 1–7. DOI: https://doi.org/10.1002/9781118445112.stat08035
16. **Jagadeeswaran R.** Fast Automatic Bayesian Cubature Using Matching Kenrels and Designs. PhD thesis. Illinois Institute of Technology; 2019. DOI: https://doi.org/10.1007/s11222-019-09895-9
17. **Jagadeeswaran R, Hickernell FJ.** Fast Automatic Bayesian Cubature Using Lattice Sampling. In: *Stat. Comput,* 2019; 29: 1215–1229. DOI: https://doi.org/10.1007/s11222-019-09895-9
18. **Jiang L.** Guaranteed Adaptive Monte Carlo Methods for Estimating Means of Random Variables. PhD thesis. Illinois Institute of Technology; 2016.
19. **Jiménez Rugama Ll A.** Adaptive Quasi-Monte Carlo Cubature. PhD thesis. Illinois Institute of Technology; 2016.
20. **Jiménez Rugama Ll A, Hickernell FJ.** Adaptive Multidimensional Integration Based on Rank-1 Lattices. In: *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, Cools R, Nuyens D (eds.). 2016; 163: 407–422. Springer Proceedings in Mathematics and Statistics. Springer-Verlag, Berlin. DOI: https://doi.org/10.1007/978-3-319-33507-0_20
21. **Keister BD.** Multidimensional quadrature algorithms. In: *Computers in Physics,* 1996; 10(2): 119–128. DOI: https://doi.org/10.1063/1.168565
22. **Kuo FY, Schwab C, Sloan IH.** Quasi-Monte Carlo Finite Element Methods for a Class of Elliptic Partial Differential Equations with Random Coefficients. In: *SIAM J. Numer. Anal.* 2012; 50: 3351–3374. DOI: https://doi.org/10.1137/110845537
23. **Li D.** Reliable Quasi-Monte Carlo with Control Variates. MA thesis. Illinois Institute of Technology; 2016.
24. **Lyness JN.** When Not to Use an Automatic Quadrature Routine. In: *SIAM Rev.* 1983; 25: 63–87. DOI: https://doi.org/10.1137/1025003
25. **Shampine LF.** Vectorized Adaptive Quadrature in MATLAB. In: *J. Comput. Appl. Math.* 2008; 211: 131–140. DOI: https://doi.org/10.1016/j.cam.2006.11.021
26. **Smith T.** *Doctest – embed testable examples in your function's help, version 1.1.0.0.* MATLAB Central File Exchange; 2010. URL: https://www.mathworks.com/matlabcentral/fileexchange/28862-doctest-embed-testable-examples-in-your-function-s-help-comments.
27. **The MathWorks, Inc.** *MATLAB R2021a*. Natick, MA; 2021.
28. **Tong X.** A Guaranteed, Adaptive, Automatic Algorithm for Univariate Function Minimization. MA thesis. Illinois Institute of Technology; 2014.
29. **Zhang Y.** Guaranteed, Adaptive, Automatic Algorithms for Univariate Integration: Methods, Costs and Implementation. PhD thesis. Illinois Institute of Technology; 2018.

TO CITE THIS ARTICLE:

Tong X, Choi S-CT, Ding Y, Hickernell FJ, Jiang L, Jiménez Rugama Ll A, Rathinavel J, Zhang K, Zhang Y, Zhou X 2022 Guaranteed Automatic Integration Library (GAIL): An Open-Source MATLAB Library for Function Approximation, Optimization, and Integration. *Journal of Open Research Software,* 10: 7. DOI: https://doi.org/10.5334/jors.381

Submitted: 24 June 2021 **Accepted:** 13 April 2022 **Published:** 29 July 2022