Challenges in Developing Great Quasi-Monte Carlo Software

Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Jagadeeswaran Rathinavel, and Aleksei G. Sorokin

Abstract Quasi-Monte Carlo (QMC) methods have developed over several decades. With the explosion in computational science, there is a need for great software that implements QMC algorithms. We summarize the QMC software that has been developed to date, propose some criteria for developing great QMC software, and suggest some steps toward achieving great software. We illustrate these criteria and steps in practice with the Quasi-Monte Carlo Python library (QMCPy), an open-source community software framework, extensible by design with common programming interfaces to an increasing number of existing or emerging QMC libraries developed by the greater community of QMC researchers.

Sou-Cheng T. Choi
Department of Applied Mathematics, Illinois Institute of Technology; SAS Institute Inc.
RE 220, 10 W. 32nd St., Chicago, IL 60616 e-mail: schoi32@iit.edu

Yuhan Ding
Department of Applied Mathematics, Illinois Institute of Technology,
RE 220, 10 W. 32nd St., Chicago, IL 60616 e-mail: yding2@hawk.iit.edu

Fred J. Hickernell
Center for Interdisciplinary Scientific Computation and
Department of Applied Mathematics, Illinois Institute of Technology
RE 220, 10 W. 32nd St., Chicago, IL 60616 e-mail: hickernell@iit.edu

Jagadeeswaran Rathinavel
Department of Applied Mathematics, Illinois Institute of Technology,
RE 220, 10 W. 32nd St., Chicago, IL 60616 e-mail: jrathin1@hawk.iit.edu

Aleksei G. Sorokin
Department of Applied Mathematics, Illinois Institute of Technology,
RE 220, 10 W. 32nd St., Chicago, IL 60616 e-mail: asorokin@hawk.iit.edu

1 Introduction

A QMC approximation of $\mu := \mathbb{E}[f(\boldsymbol{X})]$, $\boldsymbol{X} \sim \mathcal{U}[0,1]^d$ can be implemented in a few steps:

1. Draw a sequence of n low-discrepancy (LD) [11, 12, 35, 47] nodes, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ that mimic $\mathcal{U}[0,1]^d$.
2. Evaluate the integrand f at these nodes to obtain $f(\boldsymbol{x}_i)$, $i = 1, \ldots, n$.
3. Estimate the true mean, μ, by the sample mean,

$$\hat{\mu}_n := \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}_i). \tag{1}$$

However, the practice of QMC is often more complicated. The original problem may need to be rewritten to fit the above form and/or to facilitate a good approximation with a small number of sampling points, n. Practitioners may wish to determine n adaptively to satisfy a prescribed error tolerance.

In the next section, we describe why QMC software is important. We then discuss the characteristics of great QMC software:

- Integrated with related libraries (Sect. 3),
- Correct (Sect. 4),
- Efficient in computational time and memory (Sect. 5),
- Accessible to practitioners and theorists alike (Sect. 6), and
- Sustainable by a community that owns it (Sect. 7).

In each section, we describe the challenges faced and how they might be overcome.

We draw on our collective experiences as members of the academic QMC community, developers of open-source and commercial scientific software, and users of a wide range of scientific software as research scientists or data scientists. Many of the insights that we have gained have come through our development of the Guaranteed Automatic Integration Library (GAIL) [8] in MATLAB and the Quasi-Monte Carlo Python library (QMCPy) [7].

2 Why Develop QMC Software

Recent interest in quality scientific software is exemplified by the 2021 US Department of Energy report, *Workshop on the Science of Scientific-Software Development and Use* [3, 53]. This workshop not only discussed diagnostics and treatments for the challenges of developing great scientific software, but also emphasized the importance of implementing theoretical advancements into well-written, accessible software libraries. Three cross-cutting themes arose in this workshop:

- We need to consider both human and technical elements to better understand how to improve the development and use of scientific software.

- We need to address urgent challenges in workforce recruitment and retention in the computing sciences with growth through expanded diversity, stable career paths, and the creation of a community and culture that attract and retain new generations of scientists.
- Scientific software has become essential to all areas of science and technology, creating opportunities for expanded partnerships, collaboration, and impact.

These themes apply to QMC software in particular, as well as scientific software in general.

2.1 QMC Theory to Software

QMC software makes theoretical advances in QMC available to practitioners. However, translating pseudo-code into good executable code is nuanced. Software developers must write code that is computationally efficient, numerically stable, and provides reasonable default choices of tuning parameters. Great QMC software relieves users of these concerns.

Great QMC software eliminates the need for researchers to resort to "do-it-yourself" for established algorithms. The less code we have to write, the fewer errors.

2.2 QMC Software to Theory

Great QMC software opens up new application areas for QMC methods by allowing practitioners to compare new methods to existing ones. QMC software has been successful in quantitative finance, uncertainty quantification, and image rendering. Unexpectedly good or bad computational results lead to open theoretical questions. For example, the early application of QMC to high dimensional integrals arising in financial risk [41] led to a wave of theoretical results on the tractability of integration in weighted spaces [12, 36, 55].

We next discuss the characteristics of great QMC software outlined in the introduction. For each characteristic, we identify what is lacking and what might be done to remedy the lack.

3 Integrated

Expanding on the problem formulation in the introduction, we want to approximate well the integral or expectation, μ, by the sample mean, $\hat{\mu}_n$:

$$\mu := \int_{\mathcal{T}} g(\boldsymbol{t})\, \lambda(\boldsymbol{t})\, \mathrm{d}\boldsymbol{t} = \mathbb{E}[f(\boldsymbol{X})] = \int_{[0,1]^d} f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \approx \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}_i) =: \hat{\mu}_n, \quad (2)$$

$$|\mu - \hat{\mu}_n| \le \varepsilon. \quad (3)$$

Progressing from the original problem to a satisfactory solution requires several software components, which we describe in the subsections below. Great QMC software environments allow different implementations of these components to be freely interchanged.

3.1 LD Sequence Generators

A lattice, digital sequence, or Halton sequence generator typically supplies the sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$. These sequences may be deterministic or random and are intended to have an empirical distribution that approximates well the uniform distribution. Figure 1 illustrates an example of a (randomly shifted) lattice sequence.

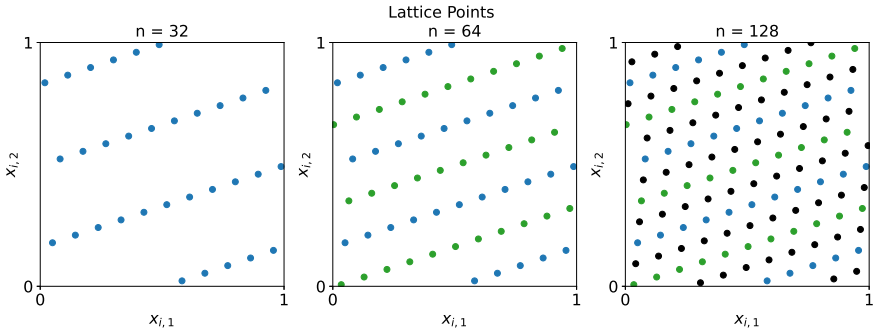

Fig. 1 A two-dimensional projection of lattice points for increasing sample size. The later points fill in gaps because they are highly correlated with the earlier points. These plots may be reproduced using the Jupyter notebook [6].

LD generators are the most prevalent, generally available components of QMC software. These generators appear in Association of Computing Machinery publications [4,5,23], FinDer [39,41], libseq [13,14], BRODA [33], NAG [52], MATLAB [51], SamplePack [32], Grünschloß's website [17], the Magic Point Shop [37], R [22], Julia [1], PyTorch [42], SAS [45], SciPy [54], TensorFlow [50], MatBuilder [43], and QMCPy [7].

Lattice and digital sequences are not unique, and improved or alternate versions continue to be constructed. Stephen Joe and Frances Kuo [29–31] have proposed Sobol' generator direction numbers, which are now widely used, but were not part of early LD sequence software. Pierre L'Ecuyer and collaborators have created LatNetBuilder [9] to construct good lattice and polynomial lattice sequences based

on user-defined criteria. During lunch at MCQMC 2022, it was proposed that we agree on a consistent format for storing the parameters that define good LD sequences so that they can be shared across software libraries.

Users computing solutions in comprehensive software environments, such as NAG, MATLAB, R, SciPy, and TensorFlow, have convenient access to LD sequence generators. However, most of these implementations in large libraries would benefit a wider menu of LD offerings, including custom generators from LatNetBuilder or other sources. Some large environments such as Dakota [2] have quite limited LD generators and should expand their offerings. SAS facilitates flexible integration of third-party open-source software; see `https://www.sas.com/en_us/software/viya/open.html`. This presents opportunities for developers of smaller libraries with more QMC features, like BRODA and QMCPy, to connect their libraries with larger software environments. An example of QMCPy connecting with other software is given in the next subsection.

In fact, QMCPy is designed not simply to contain our own algorithms. It is designed to serve as a middleware framework, allowing users to tap into a collection of QMC libraries originally written in various languages by different researchers at different times. Using the object-oriented design, QMCPy specifies a few extensible high-level mathematical objects as described in this section and in [8]. Developers of other (Q)MC software can extend these objects to incorporate their existing algorithms by a combination of mechanisms: object inheritance, Python package inclusion, direct code translation, and/or interfaces to compiled C or Java libraries.

3.2 Integrands and Variable Transformations

The original integral or expectation arising from an application is defined in terms of the integrand, g, and the non-negative weight, λ. For example,

- g is the discounted payoff of a financial derivative and λ is the probability density function (PDF) for a discretized Brownian motion [16],
- g is the velocity of a fluid at a point in rock with a random porosity field whose discretized PDF is λ [34], or
- $g\lambda$ is the unnormalized Bayesian posterior density multiplied by the parameter of interest [15].

To put μ into the form that is directly accessible to QMC methods requires a variable transformation, $\boldsymbol{\Psi}$, such that

$$\boldsymbol{\Psi}((0,1)^d) = \mathcal{T}, \qquad f(\boldsymbol{x}) = g(\boldsymbol{\Psi}(\boldsymbol{x}))\lambda(\boldsymbol{\Psi}(\boldsymbol{x}))\left|\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{x}}\right|.$$

Such a transformation is typically non-unique, and the choice of a good one is equivalent to importance sampling. If λ is a PDF for a random variable $\boldsymbol{T}$, then $\mu = \mathbb{E}[g(\boldsymbol{T})]$. In this case, one might choose $\boldsymbol{\Psi}$ such that $\boldsymbol{T} \sim \boldsymbol{\Psi}(\boldsymbol{X})$, in which case $\lambda(\boldsymbol{\Psi}(\boldsymbol{x}))\,|\partial\boldsymbol{\Psi}/\partial\boldsymbol{x}| = 1$ and $f(\boldsymbol{x}) = g(\boldsymbol{\Psi}(\boldsymbol{x}))$. Software such as SciPy and QMCPy

facilitate such variable transformations by providing default $\boldsymbol{\Psi}$ for common $\boldsymbol{T}$. This alleviates the burden on the user to determine a transformation and implement the Jacobian.

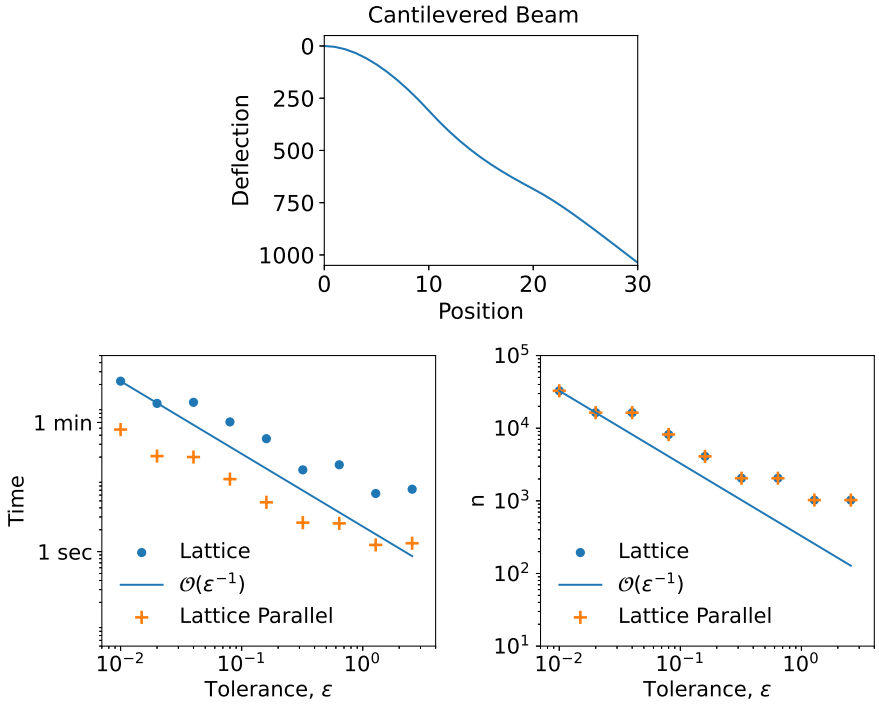

Fig. 2 The expected deflection of a cantilevered beam with random Young's modulus. QMCPy computes the answer in serial and in parallel using UM-Bridge and docker containers. For small error tolerances, the time and sample size is $O(\varepsilon^{-1})$, and parallel processing gives a speed-up factor of four to five. These plots may be reproduced using the Jupyter notebook [6].

Complex integrands might be best evaluated by libraries that have no direct connection to QMC software libraries. In the uncertainty quantification (UQ) space, the recently developed UQ and Model Bridge (UM-Bridge) [10], connects libraries that generate sequences of model parameters, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ with libraries that evaluate models, $f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots$. We illustrate in Figure 2 how UM-Bridge connects QMCPy with a differential equation boundary value problem solver for a cantilevered beam with random Young's modulus [40], where the randomness is given by $\boldsymbol{X} \sim \mathcal{U}[1, 1.2]^3$. The beam position is discretized at 31 points. QMCPy computes the expected deflection of the beam at these points for a sequence of error tolerances using lattice rules. By taking advantage of virtual machines and docker containers, UM-Bridge provides automatic load balancing to facilitate seamless parallel model evaluation.

We need more connections of QMC software to other libraries, like what has been done with QMCPy and UM-Bridge. This will allow newer and better QMC algorithms to tackle new applications.

3.3 Stopping Criteria

The cantilevered beam example illustrated in Figure 2 utilizes QMCPy's stopping criterion to choose the sample size, n, based on the input error tolerance, ε, and the observed discrete Fourier series coefficients of the integrand, f. See [20] for details of this stopping criterion for lattice rules and [18] for a review on automatic QMC stopping criteria. Theoretically justified stopping criteria for choosing sample sizes are important and a relatively recent development [19, 20, 24, 25, 28]. Some of these stopping criteria for absolute error criteria, (3), have been extended to relative error criteria or a combination of absolute and relative error criteria, as well as stopping criteria for approximating functions of several integrals [21, 26, 27, 49].

We emphasize again that no one software library contains all of the best LD generators, integrands, variable transformations, and stopping criteria, which is why great QMC software must be well integrated into other software libraries. This may take the form of large libraries incorporating more QMC routines and/or building connections among QMC libraries and with other non-QMC libraries.

4 Correct

Although correctness is an obvious feature of great QMC software, it should not always be assumed in popular packages. Here are two examples.

MATLAB's original randomization of Sobol' points was incorrectly implemented. This was discovered by Lluís Antoni Jiménez Rugama, who informed the developer. The error was corrected in MATLAB R2018a.

SciPy and PyTorch's original implementations of Sobol' sequences omitted the first point. The rationale was that the unscrambled first point of the Sobol' sequence is $\boldsymbol{x}_1 = \boldsymbol{0}$, which becomes the undesirable $-\infty$ when transformed to mimic a Gaussian distribution. A vigorous discussion ensued [44, 46], which prompted an article by Art Owen [38] on why one should not drop the first point and destroy the digital net structure. Instead one should randomly scramble the Sobol' sequence (and other LD sequences) so that $\boldsymbol{x}_1 \neq \boldsymbol{0}$. Although SciPy and PyTorch were corrected, other software libraries with LD sequence generators still omit the first point.

QMC software is sometimes written by those not intimately familiar with QMC. Those of us most familiar with good QMC practices need to pay attention to QMC algorithms in popular software libraries and inform the developers when their software contains bugs or deviates from best practices.

The above examples and discussion also underscore the importance of continuous testing in QMC software development. This involves creating a comprehensive test suite that covers a wide range of scenarios and input parameters. Additionally, QMC software should be benchmarked against known results to ensure that it is both accurate and efficient.

5 Efficient

Another obvious feature of great QMC software is computational efficiency. This requires developers to identify bottlenecks in performance. Intensive computations may need to be rewritten in lower-level languages, like C. Code may need to be refactored to avoid excessive calls to memory. As new hardware architectures arise, QMC software should adapt to take advantage of them.

Efficiency gains in QMC software can also result from clever algorithm choices. Adaptive QMC algorithms as in [20, 21, 24, 25, 28] can be used to dynamically increase the number of sampling points based on estimated error bounds. They can lead to significant efficiency gains by reducing the number of unnecessary function evaluations. The Bayesian stopping rules for lattice and Sobol sequences in [24, 25] choose covariance kernels that match the corresponding LD sequence. This reduces the computations from what would be $O(n^3)$ for arbitrary covariance kernels to only $O(n \log n)$ for these well-chosen kernels. Consequently, computing the stopping criterion is roughly the same order of magnitude as computing the solution.

In spite of the desire for efficiency, great QMC software will be written in a variety of computer languages since different communities tend to favor different languages. Trade-offs will be made between efficiency and convenience.

6 Accessible

One attribute of great QMC software that often receives insufficient attention is its accessibility. Developers may write code for their own purposes and move on.

Great software should be stored in a repository, where the latest version can be downloaded. There should be documentation that guides the user through installation and explains the library's features. Demos are crucial for showing how the software works and for providing a template for new users to write their own code. Tutorials help newcomers to QMC understand its advantages. Blogs educate a wider audience and enlarge our QMC community. There should be a place for users to report bugs and make feature requests.

Accessible QMC software has greater impact. Accessibility also depends on the next characteristic of great QMC software.

7 Sustainable

Great QMC software libraries should be alive. They should be under active development or merged into other libraries that are actively maintained. Commercial software has an infrastructure to maintain their algorithms, but these algorithms need periodic refreshing. Open source software requires a community of active users and developers to keep the library up-to-date and bug-free.

The QMC community can help sustain great QMC software in several ways.

- Let's build our research code and showcase our new algorithms using existing QMC software libraries or build and maintain new libraries, if needed. This allows the next generation to more easily reproduce our results and build upon them.
- Many of us QMC researchers rely on transient team members (students and postdocs) to write the code that illustrates our new ideas. Building our own QMC libraries or contributing to existing ones helps ensure that the code our team members write lasts beyond their involvement in our teams.
- It is relatively easy to post demos of our new ideas on our own web pages, and if possible in the software libraries that we are using. QMCPy welcomes blogs, short articles introducing a broader audience to important ideas, illustrated by software.
- When we see bugs or missing features in QMC software, let's persistently request fixes and improvements. The more robust our QMC software, the easier it is to open new application areas to QMC and grow our QMC community.
- Let's publish our QMC software with journals such as ACM TOMS, JORS, etc. to obtain code review and feedback.
- When we find QMC scientific software useful in our research or applications, let's encourage and recognize such software by properly citing it [48].
- Encourage QMC community members to learn about software development through interdisciplinary education, internships, or hackathons.

Encouraging students and postdocs to invest time in developing QMC software will require us as their mentors valuing software as a research output. Great software should be published with a digital object identifier (DOI).

The term research software engineer (RSE) is becoming a more valued vocation. According to the website for Research Software Engineers International, `https://researchsoftware.org`, *Research Software Engineers are people who combine professional software expertise with an understanding of research.*

8 Summary

QMC has experienced decades of exciting growth in theory since the late 1950s. The earliest publicly available QMC software dates to the early 1990s and its growth has not matched that of theory. As a QMC community, we have a responsibility to implement our great theory into great QMC software.

References

1. SciML QuasiMonteCarlo.jl (2022). URL https://github.com/SciML/QuasiMonteCarlo.jl
2. Adams, B.M., Bohnhoff, W.J., Dalbey, K.R., Ebeida, M.S., Eddy, J.P., Eldred, M.S., Hooper, R.W., Hough, P.D., Hu, K.T., Jakeman, J.D., Khalil, M., Maupin, K.A., Monschke, J.A., Ridgway, E.M., Rushdi, A.A., Seidl, D.T., Stephens, J.A., Swiler, L.P., Tran, A., Winokur, J.G.: Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.16 User's Manual. Sandia National Laboratories (2022)
3. Bernholdt, D.E., Cary, J., Heroux, M., McInnes, L.C.: The science of scientific-software development and use (2022). DOI 10.2172/1846008. URL https://www.osti.gov/biblio/1846008
4. Bratley, P., Fox, B.L.: Algorithm 659: Implementing Sobol's quasirandom sequence generator. ACM Trans. Math. Software **14**, 88–100 (1988)
5. Bratley, P., Fox, B.L., Niederreiter, H.: Implementation and tests of low-discrepancy sequences. ACM Trans. Model. Comput. Simul. **2**, 195–213 (1992)
6. Choi, S.C.T., Ding, Y., Hickernell, F.J., Jagadeeswaran, R., Sorokin, A.G.: MCQMC 2022 figures. URL https://github.com/QMCSoftware/QMCSoftware/tree/develop/demos/talk_paper_demos/MCQMC2022_Article_Figures
7. Choi, S.C.T., Hickernell, F.J., Jagadeeswaran, R., McCourt, M., Sorokin, A.: QMCPy: A quasi-Monte Carlo Python library (2022). DOI 10.5281/zenodo.3964489. URL https://qmcsoftware.github.io/QMCSoftware/
8. Choi, S.C.T., Hickernell, F.J., Jagadeeswaran, R., McCourt, M.J., Sorokin, A.G.: Quasi-Monte Carlo software. In: A. Keller (ed.) Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Oxford, England, August 2020, Springer Proceedings in Mathematics and Statistics, pp. 23–50. Springer, Cham (2022). https://arxiv.org/abs/2102.07833
9. Darmon, Y., Godin, M., L'Ecuyer, P., Jemel, A., Marion, P., Munger, D.: LatNet builder (2018). URL https://github.com/umontreal-simul/latnetbuilder
10. Davis, A., Parno, M., Reinarz, A., Seelinger, L.: UQ and model bridge (UM-bridge) (2022). URL https://um-bridge-benchmarks.readthedocs.io/en/docs/
11. Dick, J., Kritzer, P., Pillichshammer, F.: Lattice Rules: Numerical Integration, Approximation, and Discrepancy. Springer Series in Computational Mathematics. Springer Cham (2022). DOI https://doi.org/10.1007/978-3-031-09951-9
12. Dick, J., Kuo, F., Sloan, I.H.: High dimensional integration — the Quasi-Monte Carlo way. Acta Numer. **22**, 133–288 (2013). DOI 10.1017/S0962492913000044
13. Friedel, I., Keller, A.: libseq – low discrepancy sequence library (2001). URL http://www.multires.caltech.edu/software/libseq/
14. Friedel, I., Keller, A.: Fast generation of randomized low-discrepancy point sets. In: K.T. Fang, F.J. Hickernell, H. Niederreiter (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2000, pp. 257–273. Springer-Verlag, Berlin (2002)
15. Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.: Bayesian Data Analysis. CRC Texts in Statistical Science. Chapman & Hall (2013)
16. Glasserman, P.: Monte Carlo Methods in Financial Engineering, *Applications of Mathematics*, vol. 53. Springer-Verlag, New York (2004)
17. Grünschloß, L.: Leonhard grünschloß's webpage. URL https://gruenschloss.org
18. Hickernell, F.J., Choi, S.C.T., Jiang, L., Jiménez Rugama, Ll.A.: Monte Carlo simulation, automatic stopping criteria for. In: M. Davidian, B. Everitt, R.S. Kenett, G. Molenberghs, W. Piegorsch, F. Ruggeri (eds.) Wiley StatsRef-Statistics Reference Online. John Wiley & Sons Ltd. (2018). DOI 10.1002/9781118445112.stat08035
19. Hickernell, F.J., Jiang, L., Liu, Y., Owen, A.B.: Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling. In: J. Dick, F.Y. Kuo, G.W. Peters, I.H. Sloan (eds.) Monte Carlo and Quasi-Monte Carlo Methods 2012, *Springer Proceedings in Mathematics and Statistics*, vol. 65, pp. 105–128. Springer-Verlag, Berlin (2013). DOI 10.1007/978-3-642-41095-6

20. Hickernell, F.J., Jiménez Rugama, Ll.A.: Reliable adaptive cubature using digital sequences. In: R. Cools, D. Nuyens (eds.) Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014, *Springer Proceedings in Mathematics and Statistics*, vol. 163, pp. 367–383. Springer-Verlag, Berlin (2016). ArXiv:1410.8615 [math.NA]
21. Hickernell, F.J., Jiménez Rugama, Ll.A., Li, D.: Adaptive quasi-Monte Carlo methods for cubature. In: J. Dick, F.Y. Kuo, H. Woźniakowski (eds.) Contemporary Computational Mathematics — a celebration of the 80th birthday of Ian Sloan, pp. 597–619. Springer-Verlag (2018). DOI 10.1007/978-3-319-72456-0
22. Hofert, M., Lemieux, C.: qrng R package (2017). URL `https://cran.r-project.org/web/packages/qrng/qrng.pdf`
23. Hong, H.S., Hickernell, F.J.: Algorithm 823: Implementing scrambled digital nets. ACM Trans. Math. Software **29**, 95–109 (2003). DOI 10.1145/779359.779360
24. Jagadeeswaran, R., Hickernell, F.J.: Fast automatic Bayesian cubature using lattice sampling. Stat. Comput. **29**, 1215–1229 (2019). DOI 10.1007/s11222-019-09895-9
25. Jagadeeswaran, R., Hickernell, F.J.: Fast automatic Bayesian cubature using Sobol' sampling. In: Z. Botev, A. Keller, C. Lemieux, B. Tuffin (eds.) Advances in Modeling and Simulation: Festschrift in Honour of Pierre L'Ecuyer, pp. 301–318. Springer, Cham (2022). DOI 10.1007/978-3-031-10193-9_15
26. Jiang, L.: Guaranteed adaptive Monte Carlo methods for estimating means of random variables. Ph.D. thesis, Illinois Institute of Technology (2016)
27. Jiménez Rugama, Ll.A., Gilquin, L.: Reliable error estimation for Sobol' indices. Statistics and Computing **28**, 725–738 (2018). DOI 10.1007/s11222-017-9759-1
28. Jiménez Rugama, Ll.A., Hickernell, F.J.: Adaptive multidimensional integration based on rank-1 lattices. In: R. Cools, D. Nuyens (eds.) Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014, *Springer Proceedings in Mathematics and Statistics*, vol. 163, pp. 407–422. Springer-Verlag, Berlin (2016). ArXiv:1411.1966
29. Joe, S., Kuo, F.Y.: Sobol sequence generator. URL `https://web.maths.unsw.edu.au/~fkuo/sobol/`
30. Joe, S., Kuo, F.Y.: Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator. ACM Trans. Math. Software **29**, 49–57 (2003)
31. Joe, S., Kuo, F.Y.: Constructing Sobol sequences with better two-dimensional projections. SIAM J. Sci. Comput. **30**, 2635–2654 (2008)
32. Kollig, T., Keller, A.: Samplepack (2002). URL `https://www.uni-kl.de/AG-Heinrich/SamplePack.html`
33. Kucherenko, S.: BRODA (2022). URL `https://www.broda.co.uk/index.html`
34. Kuo, F.Y., Nuyens, D.: Application of quasi-Monte Carlo methods to elliptic PDEs with random diffusion coefficients – a survey of analysis and implementation. Found. Comput. Math. **16**, 1631–1696 (2016). URL `https://people.cs.kuleuven.be/~dirk.nuyens/qmc4pde/`
35. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, Philadelphia (1992)
36. Novak, E., Woźniakowski, H.: Tractability of Multivariate Problems Volume II: Standard Information for Functionals. No. 12 in EMS Tracts in Mathematics. European Mathematical Society, Zürich (2010)
37. Nuyens, D.: Dirk's homepage. URL `https://people.cs.kuleuven.be/~dirk.nuyens/`
38. Owen, A.B.: On dropping the first Sobol' point. In: A. Keller (ed.) Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Oxford, England, August 2020, Springer Proceedings in Mathematics and Statistics, pp. 71–86. Springer, Cham (2022). URL `https://arxiv.org/pdf/2008.08051.pdf`
39. Papageorgiou, A.: Finder. URL `http://www.cs.columbia.edu/~ap/html/finder.html`
40. Parno, M., Seelinger, L.: Uncertainty propagation of material properties of a cantilevered beam (2022). URL `https://um-bridge-benchmarks.readthedocs.io/en/docs/forward-benchmarks/muq-beam-propagation.html`
41. Paskov, S., Traub, J.: Faster valuation of financial derivatives. J. Portfolio Management **22**, 113–120 (1995)

42. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32**, 8026–8037 (2019)
43. Paulin, L., Bonneel, N., Coeurjolly, D., Iehl, J.C., Keller, A., Ostromoukhov, V.: Matbuilder: Mastering sampling uniformity over projections. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **41**(4) (2022). DOI https://doi.org/10.1145/3528223.3530063
44. PyTorch Developers: Sobol point implementation (2020). URL `https://github.com/pytorch/pytorch/issues/32047`
45. SAS Institute: Sas documentation for the model procedure (2023). URL `https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.4/etsug/etsug_model_sect197.htm`
46. SciPy Developers: SciPy discussion of Sobol' sequence implementation (2020). URL `https://github.com/scipy/scipy/pull/10844`
47. Sloan, I.H., Joe, S.: Lattice Methods for Multiple Integration. Oxford University Press, Oxford (1994)
48. Smith, A.M., Katz, D.S., Niemeyer, K.E.: Software citation principles. PeerJ Computer Science **2**, e86 (2016)
49. Sorokin, A.G., Jagadeeswaran, R.: Monte Carlo for vector functions of integrals. In: A. Hinrichs, P. Kritzer, F. Pillichshammer (eds.) Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Linz, Austria, July 2022, Springer Proceedings in Mathematics and Statistics. Springer, Cham (2022+). In preparation for submission for publication
50. TF Quant Finance Contributors: Quasi Monte-Carlo methods (2021). URL `https://github.com/google/tf-quant-finance/math/qmc`
51. The MathWorks, Inc.: MATLAB R2022b. Natick, MA (2022)
52. The Numerical Algorithms Group: The NAG Library. Oxford, Mark 27 edn. (2021)
53. U.S. Department of Energy, Office of Advanced Scientific Computing Research: Workshop on the science of scientific-software development and use (2021). URL `https://web.cvent.com/event/1b7d7c3a-e9b4-409d-ae2b-284779cfe72f/summary`
54. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, I., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., Vijaykumar, A., Bardelli, A.P., Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C.N., Fulton, C., Masson, C., Häggström, C., Fitzgerald, C., Nicholson, D.A., Hagen, D.R., Pasechnik, D.V., Olivetti, E., Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G.A., Ingold, G.L., Allen, G.E., Lee, G.R., Audren, H., Probst, I., Dietrich, J.P., Silterra, J., Webber, J.T., Slavič, J., Nothman, J., Buchner, J., Kulick, J., Schönberger, J.L., de Miranda Cardoso, J., Reimer, J., Harrington, J., Rodríguez, J.L.C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kümmerer, M., Bolingbroke, M., Tartre, M., Pak, M., Smith, N.J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb, P.A., Lee, P., McGibbon, R.T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson, S., More, S., Pudlik, T., Oshima, T., Pingel, T.J., Robitaille, T.P., Spura, T., Jones, T.R., Cera, T., Leslie, T., Zito, T., Krauss, T., Upadhyay, U., Halchenko, Y.O., Vázquez-Baeza, Y., Contributors: Scipy 1.0: fundamental algorithms for scientific computing in python. Nature Methods **17**(3), 261–272 (2020)
55. Woźniakowski, H.: Efficiency of quasi-Monte Carlo algorithms for high dimensions. In: H. Niederreiter, J. Spanier (eds.) Monte Carlo and Quasi-Monte Carlo Methods 1998, pp. 114–136. Springer-Verlag, Berlin (2000)