Challenges in Developing Great Quasi-Monte Carlo Software

Sou-Cheng Terrya Choi, Yuhan Ding, Fred J. Hickernell, 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 to realize that implements QMC algorithms. We review what QMC software has been developed to date, propose some criteria for developing great QMC software, and suggest some steps toward achieving great software.

Sou-Cheng Terrya Choi
Department of Applied Mathematics, Illinois Institute of Technology,
RE 220, 10 W. 32nd St., Chicago, IL 60616 e-mail: `schoi32@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`

Yuhan Ding

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 a low discrepancy (LD) [12, 13, 34, 44] 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

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

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

In the next section we describe why QMC software is important. We then discuss 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),
- Sustainable by a community that owns it (Sect. 7), and
- Scalable to advanced hardware architectures (Sect. 8).

Developing software to implement such enhanced MC strategies poses a number of challenges. First, a development team must determine the value of the software to the users. We discuss why great (Q)MC software in necessary in Sect. 2. Second, a team must determine high level objectives for the software; the tenets of great software are described in Sect. 9. Third, an architecture for the (Q)MC software should be determined. We propose a modularization of the (Q)MC pipeline in Sect. 10. Next, a team must choose which programming languages, hardware environments, and distribution platforms they wish to support. We discuss the nuance and challenges in these decisions in Sect. 11. Finally, a project which hopes to stand the test of time must draw attention and support from the broader (Q)MC community. To this end, we encourage collaborative community development and describe its' best practices in Sect. 12.

2 Why Develop QMC Software

Recent interest in quality scientific software is exemplified by the 2021 US Department of Energy *Workshop on the Science of Scientific-Software Development and Use* [3,48].

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 that arose in this workshop were

- 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, numerical stable, and provides reasonable default choices of tuning parameters. Good QMC software relieves users of these concerns.

2.2 QMC Software to Theory

Good 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 leads to open theoretical questions. For example, the early application of QMC to high dimensional integrals arising in financial risk [40] led to a wave of theoretical results on the tractability of integration in weighted spaces [13, 35, 50].

We next discuss 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, $\hat{\mu}$:

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

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

Progressing from the original problem to a satisfactory solution requires several software components. Great QMC software environment allows different implementations of these components to be 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.

LD generators are the most prevalent generally available components of QMC software. These generators appear in Association of Computing Machinery publications [4,5,22], FinDer [38,40], libseq [14,15], BRODA [28], NAG [47], MATLAB [46], SamplePack [27], Grünschloß's website [19], the Magic Point Shop [36], R [21], Julia [1], SciPy [49], PyTorch [41], and TensorFlow [45], and MatBuilder [42].

Lattice and digital sequences are not unique, and improved or alternative versions continue to be constructed. Joe and Kuo [23–25] have proposed Sobol' generator direction numbers, which are now widely used, but were not part of the early LD sequence software. L'Ecuyer and colleagues have created LatNetBuilder [9] to construct good lattice and polynomial lattice sequences based on user defined criteria. As was raised during a lunch at MCQMC 2022, we should agree on a consistent format for storing the parameters that define good LD sequences so that they can by shared across software libraries.

3.2 Integrands and Variable Transformations

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

- g may represent the discounted payoff of a financial derivative and λ be the probability density function (PDF) for a discretized Brownian motion [18],

- g may represent the velocity of flow at a point in rock and λ is the PDF of the porosity field , or
- $g\lambda$ is the unnormalized Bayesian posterior density multiplied by parameter of interest.

, If λ is a probability density function for a random variable $\boldsymbol{T}$, then $\mu = \mathbb{E}[g(\boldsymbol{T})]$.

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

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

Such a transformation is typically non-unique, and the choice of a good one is equivalent to importance sampling.

Joe and Kuo

Finally, a *stopping criterion* is determines the sample size, n, required to satisfy the error criterion (3) given in terms of the absolute error tolerance, ε. One can also imagine error criteria involving a relative error tolerance or a combination of absolute and relative error tolerances.

No one software library contains all of these features, which is why great QMC software must be well integrated into other software libraries. This may mean large libraries including QMC routines and/or building connections between QMC libraries and other libraries.

This requires four components, which we implement as QMCPy classes.

Software should be *integrated* into the environment's scientific software ecosystem. More simply, libraries should play nicely together and utilize the mature developments of other teams. A simple example in CITE shows a straightforward integration between the QMC package QMCPy [7] and the PDE package FEniCS/Dolfin [33], both in Python.

* Standard definitions, e.g., for LD generator * UM-Bridge, Dolfin * Move to other libraries * Demos, bridges

4 Correct

* Matlab randomization * Don't omit first point *

5 Efficient

* New architectures * Memory and FLOPS

6 Accessible

* Documentation * Demos * Tutorials * Blogs * Supported, issues, features * Up to date *

7 Sustainable

Community Owned * Developers, contributors * Use to illustrate research code * Use cases, new applications * Referenced * Build STEM pipeline

8 Scalable

9 Tenets of Great Software

In this section we lay out common aspirations of scientific software. While some objectives are expected by practitioners, e.g. correctly written software, others may better be characterized as features that are pursued as a project matures, e.g. supporting scalable computation. The following paragraphs highlight aspirations with more important, expected behaviours coming earlier in the discussion.

Software is expected to be *correct*. A practitioner generating a digital sequence expects the routine to produce theoretically accurate points. For example, the QMC community has helped developers ensure implementations of low discrepancy sequence generators include the zeroth point [37,43] and have correct randomization routines.

Algorithms should be implemented in an *efficient* manner. Both the compute time and memory requirements should match theoretical developments which in turn should be as close to the state of the art as possible. As a classical example, it is expected that an implementation of the discrete Fourier transform for n points has complexity $O(n \log n)$.

Software should be *accessible* in terms of installation, presentation, navigation, exemplification, and communication. A library should be straightforward to install from a common distribution platform, consistent in user interface, intuitive to navigate to modules and documentation, comprehensible in its' examples, and supportive in user engagement. Software accessibility considerations are further discussed in Sect. 11.

Libraries should be *sustainable* for future development. The goal should be to have a sufficient user base and developer community for updates and maintenance. Similar to accessibility, sustainability is often overlooked for aspirations discussed later in this section. However, we contend that a sub-optimal but user-friendly library with community engagement may be more impactful than an optimal library which lacks support. We promote collaborative, community driven development further in Sect. 12.

Software should be *integrated* into the environment's scientific software ecosystem. More simply, libraries should play nicely together and utilize the mature developments of other teams. A simple example in CITE shows a straightforward integration between the QMC package QMCPy [7] and the PDE package FEniCS/Dolfin [33], both in Python.

Routines should be *scalable* to advanced computer architectures. The widespread availability of on demand computing with multi-core and multi-GPU machines encourages software that can take advantage of parallel computation. Scalable algorithms lend themselves to a larger class or problems and provide practitioners the opportunity to more quickly test ideas.

10 (Quasi-)Monte Carlo Software Architecture

An important step in the development of any software is to determine its' architecture. How should components interact with each other? Which aspects should be modular? Object oriented? Procedural? Answers to these questions form the blueprint of software design and guide implementation.

This section proposes an object oriented design for (Q)MC software. The authors have implemented this architecture into the QMCPy [7] Python package which is further detailed in the MCQMC 2020 proceedings article CITE. While this is not the only possible architecture for (Q)MC problems, we hope the high level object definitions facilitate easy collaboration and integration for research across the (Q)MC community.

The standard (Q)MC problem is to approximate the true mean $\mu := \mathbb{E}[g(\boldsymbol{T})]$ where $\boldsymbol{T}$ is some d dimensional random variable. For nice $\boldsymbol{T}$, one may perform a change of variables to write $\mu = \mathbb{E}[f(\boldsymbol{X})]$ where $\boldsymbol{X} \sim \mathcal{U}[0,1]^d$. For example, if $\boldsymbol{T} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, then one may set $f(\boldsymbol{x}) := g(\boldsymbol{\Phi}^{-1}(\boldsymbol{x}))$ for $\boldsymbol{x} \in [0,1]^d$ where $\boldsymbol{\Phi}^{-1}$ is the inverse CDF of a standard normal distribution taken element wise. One may then approximate μ by the n sample average $\hat{\mu} = 1/n \sum_{i=1}^n f(\boldsymbol{x}_i)$ where $\{\boldsymbol{x}_i\}_{i=1}^n$ are chosen so their empirical distribution is "close" to the uniform distribution. The approximation error $\varepsilon = |\mu - \hat{\mu}|$ is $O(n^{-1/2})$ when $\{\boldsymbol{x}_i\}_{i=1}^n$ are chosen to be IID standard uniform as in the case of simple Monte Carlo (MC) and almost $O(n^{-1})$ when $\{\boldsymbol{x}_i\}_{i=1}^n$ are chosen from a low discrepancy sequence as done for Quasi-Monte Carlo (QMC). Instead of selecting the number of samples n and then determining the error ε, it is often desirable to select ε and then adaptively determine n. From this view, n is $O(\varepsilon^{-2})$ for MC and almost $O(\varepsilon^{-1})$ for QMC. A more detailed account of MC and QMC can be found in CITE.

The (Q)MC pipeline above can be decomposed into 4 main components:

Generator: producing $\{\boldsymbol{x}_i\}_{i=1}^n$ whose empirical distribution is "close" to the standard uniform distribution in d dimensions for any $n \geq 1$. May produce IID or low discrepancy sequences depending on the problem.

Measure: that defines the distribution of $\boldsymbol{T}$ e.g., uniform, Gaussian, Lebesgue. This definition facilitates choosing a change of variables to determine f.
Integrand: g, which defines the original integral. When provided knowledge of the true measure, the integrand can also evaluate f via the change of variables.
Stopping Criterion: based on a data-driven error bound, which determines how large n should be to ensure that ε is less than some user-provided error threshold.

A variety of QMC materials and the their categorizations into the above framework are described in the following list.

- ACM low discrepancy point generators [4, 5, 22]
- FinDer [38, 40] and BRODA [28] targeting quantitative finance
- Korobov cubature and scrambled Sobol' generators in NAG [47] for decades
- Scrambled Sobol' and Halton generators in MATLAB [46] since 2008, and fixed a few years later
- LatMRG [31], RNGStreams [32], SSJ [29, 30], and LatNetBuilder [9]
- Low discrepancy generators [14, 15], SamplePack [27], [19], and MatBuilder [42]
- Sobol' direction numbers [23–25]
- Fast CBC, Magic Point Shop, QMC4PDE and other code since 2004 at [36]
- Multi-level software [16, 17]
- Data-driven error bounds and stopping criteria in GAIL [6] and QMCPy [7, 8]
- Uncertainty quantification libraries Dakota [2], UQTk [10, 11], and MUQ [39] have some basic level low discrepancy sampling
- QMC framework in Julia since 2019 [1] by Robbe and others
- Scrambled Sobol' in SciPy [49] and PyTorch [41] since several years ago
- TensorFlow QMC framework [45] since a year or so ago
- CUBA [20]

11 Choosing a Environment

This section discusses the challenges in choosing environments for development and distribution. The development environment includes the choice of which languages, architectures, testing methods, and documentation methods to support. The deployment environment will often be influenced by the choice of programming languages.

The choice of programming language should be aligned with the communities the team wishes to reach with their software. The choice should also account for developer familiarity and ability to quickly prototype ideas. A team should align their choice of language to balance the desire to have user-friendly prototyping code versus the desire to have high performance computing (HPC) "ninja" code written for production setting. To the second point, it is also important to consider which language will best facilitate the parallel and GPU computations the team wishes to support.

Many programming languages come bundled with their own preferred distribution systems. Python has PyPI, R has CRAN, Julia has Pkg. However, many projects support multiple languages and have complex dependency requirements. In such cases, it can be helpful to containerize your application using a service like Docker.

12 Encouraging Collaborative Development

We now discuss ways a team may encourage collaborative development while touching on a few best practices. As discussed earlier, collaborative development promotes greater community engagement, ecosystem integration, and software longevity.

The authors have found the Git version control system to be an invaluable for software development. Git hosts such as GitHub or Bitbucket may offer teams additional features such as tracking issues, managing discussion threads, and setting up automated workflows for jobs such as testing and documentation compilation.

Accessibility is key to community engagment. It is paramount that your package be easy to install, well documented, and tested regularly. Aside from static communications, a team should also be engaged in addressing users bugs and be attentive to collaborating with proposed code changes.

It is also important for a development team to make a clear path for collaborators to contributors to the software. Templates for issues and pull requests are a good place to start. Contributors should also be encouraged to write robust tests for their updates and highlight their new features in a demo for greater accessibility.

13 Example Collaboration with UMBridge and QMCPy

This section discusses a collaboration between the UM-Bridge (the UQ and Model Bridge) CITE and QMCPy (the Quasi-Monte Carlo Python library) [7] softwares. The UM-Bridge documentation describes its' mission is to, "provide a unified interface for numerical models that is accessible from virtually any programming language or framework. It is primarily intended for coupling advanced models (e.g. simulations of complex physical processes) to advanced statistics or optimization methods." QMCPy is a high level framework for combining QMC components in a compatible and consistent manner. The collaboration between these two packages focused on making the UQ methods of QMCPy compatible with the abstract model representations from UM-Bridge. Specifically, UM-Bridge models are made compatible with the QMCPy framework through an integrand wrapper for seamless compatibility with a variety of existing point generators, transformations, and stopping criterion already available in QMCPy.

To exemplify the integration, we focus on quantifying uncertainty propagation of material properties of a cantilevered beam. It is assumed the beam is split into 3 regions of random stiffness dependent on uncertain material properties. Given a sample of 3 stiffness values, the UM-Bridge model provides the displacement at 31 equidistant nodes along the beam. Our task is to quantify the uncertainty in displacement at each of these 31 locations along the beam.

A fully coded example of this problem is available at `https://github.com/QMCSoftware/QMCSoftware/blob/master/demos/umbridge.ipynb`. At a high level, one begins by running the UM-Bridge model in a Docker container. A model is then initialized in the UM-Bridge Python client which is then provided to QMCPy alongside a low discrepancy point generator and distribution governing the 3 beam stiffness values. QMCPy then performs adaptive QMC cubature to simultaneously approximate the expected displacement at the 31 locations of interest. When the cubature method queries the model, the QMCPy wrapper around UM-Bridge model sends requests to the Docker UM-Bridge model in parallel. The load balancing provided by UM-Bridge may be utilized to evaluate the model in parallel, a potentially advantageous feature for models which are costly to evaluate. Figure 1 depicts the 31 approximate expected displacement values subject to uniform stiffness.

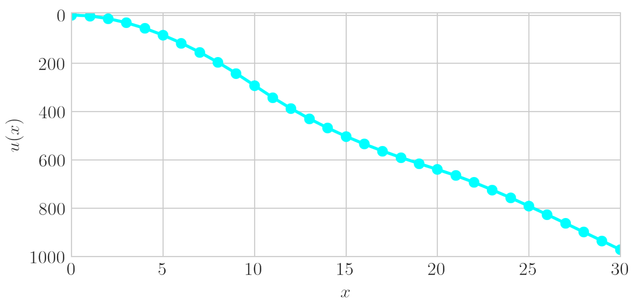

Fig. 1 QMCPy approximate expected displacement at 31 locations subject to random uniform stiffness.

References

1. SciML QuasiMonteCarlo.jl. 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., Jiang, L., Jiménez Rugama, Ll.A., Li, D., Jagadeeswaran, R., Tong, X., Zhang, K., Zhang, Y., Zhou, X.: GAIL: Guaranteed Automatic Integration Library (versions 1.0–2.3.2). MATLAB software, `http://gailgithub.github.io/GAIL_Dev/` (2021). DOI 10.5281/zenodo.4018189
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: Keller [26], pp. 23–50. `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. Debusschere, B.: UQ toolkit (2022). URL `https://www.sandia.gov/uqtoolkit/`
11. Debusschere, B., Najm, H.N., Pébay, P.P., Knio, O.M., Ghanem, R.G., Maître, O.P.L.: Numerical challenges in the use of polynomial chaos representations for stochastic processes. SIAM J. Sci. Comput. pp. 698–719 (2004). URL `10.1137/S1064827503427741`
12. 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
13. 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
14. Friedel, I., Keller, A.: libseq – low discrepancy sequence library (2001). URL `http://www.multires.caltech.edu/software/libseq/`
15. 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)
16. Giles, M.: Multi-level Monte Carlo software (2022). URL `https://people.maths.ox.ac.uk/gilesm/mlmc/`
17. Giles, M.: Multi-level quasi-Monte Carlo software (2022). URL `http://people.maths.ox.ac.uk/~gilesm/mlqmc/`
18. Glasserman, P.: Monte Carlo Methods in Financial Engineering, *Applications of Mathematics*, vol. 53. Springer-Verlag, New York (2004)
19. Grünschloß, L.: Leonhard grünschloß's webpage. URL `https://gruenschloss.org`
20. Hahn, T.: CUBA—a library for multidimensional numerical integration. Comput. Phys. Commun. **168**, 78–95 (2005). DOI 10.1016/j.cpc.2005.01.010. URL `https://feynarts.de/cuba/`
21. Hofert, M., Lemieux, C.: qrng R package (2017). URL `https://cran.r-project.org/web/packages/qrng/qrng.pdf`
22. 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
23. Joe, S., Kuo, F.Y.: Sobol sequence generator. URL `https://web.maths.unsw.edu.au/~fkuo/sobol/`
24. Joe, S., Kuo, F.Y.: Remark on algorithm 659: Implementing sobol's quasirandom sequence generator. ACM Trans. Math. Software **29**, 49–57 (2003)
25. Joe, S., Kuo, F.Y.: Constructing sobol sequences with better two-dimensional projections. SIAM J. Sci. Comput. **30**, 2635–2654 (2008)
26. Keller, A. (ed.): Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Oxford, England, August 2020, Springer Proceedings in Mathematics and Statistics. Springer, Cham (2022)
27. Kollig, T., Keller, A.: Samplepack (2002). URL `https://www.uni-kl.de/AG-Heinrich/SamplePack.html`
28. Kucherenko, S.: BRODA (2022). URL `https://www.broda.co.uk/index.html`
29. L'Ecuyer, P.: SSJ: A framework for stochastic simulation in Java. In: 2002 Winter Simulation Conference, pp. 234–242. IEEE Press (2002)
30. L'Ecuyer, P.: SSJ: Stochastic Simulation in Java (2022). URL `https://github.com/umontreal-simul/ssj`

31. L'Ecuyer, P., Couture, R.: An implementation of the lattice and spectral tests for multiple recursive linear random number generators. INFORMS J. Comput. **9**, 206–217 (1997)
32. L'Ecuyer, P., Simard, R., Chen, E.J., Kelton, W.D.: An objected-oriented random-number package with many long streams and substreams. Oper. Res. **50**, 1073–1075 (2002). DOI 10.1287/opre.50.6.1073.358
33. Logg, A., Wells, G.N., Hake, J.: DOLFIN: a C++/Python finite element library. In: K.M. A. Logg, G.N. Wells (eds.) Automated Solution of Differential Equations by the Finite Element Method, *Lecture Notes in Computational Science and Engineering*, vol. 84, chap. 10. Springer (2012)
34. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, Philadelphia (1992)
35. 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)
36. Nuyens, D.: Dirk's homepage. URL `https://people.cs.kuleuven.be/~dirk.nuyens/`
37. Owen, A.B.: On dropping the first Sobol' point. In: Keller [26], pp. 71–86. URL `https://arxiv.org/pdf/2008.08051.pdf`
38. Papageorgiou, A.: Finder. URL `http://www.cs.columbia.edu/~ap/html/finder.html`
39. Parno, M., Davis, A., Seelinger, L., Marzouk, Y.: MIT uncertainty quantification (muq) library (2014). URL `https://mituq.bitbucket.io/source/_site/index.html`
40. Paskov, S., Traub, J.: Faster valuation of financial derivatives. J. Portfolio Management **22**, 113–120 (1995)
41. 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)
42. 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
43. SciPy Developers: scipy discussion of Sobol' sequence implementation (2020). URL `https://github.com/scipy/scipy/pull/10844`
44. Sloan, I.H., Joe, S.: Lattice Methods for Multiple Integration. Oxford University Press, Oxford (1994)
45. TF Quant Finance Contributors: Quasi Monte-Carlo methods (2021). URL `https://github.com/google/tf-quant-finance/math/qmc`
46. The MathWorks, Inc.: MATLAB R2022b. Natick, MA (2022)
47. The Numerical Algorithms Group: The NAG Library. Oxford, Mark 27 edn. (2021)
48. 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`
49. 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)

50. 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)

Contents

TODO

- Flesh out Section 12
- QMC Software Architecture section: better way then description of components and list of softwares?

- For Section 9, the slides also include
  - Complete—contain the components or easily access components in other libraries to solve real, complex problems
  - Current—include the latest and best algorithms

  which I think fall under the umbrella of software being *integrated* with the community.
- update QMCPy citation [7]
- figures?
- I have left out or been breif about material from the following slides: QMCPy + FEniCS/Dolfin, Building a Developer Team for Your Library