QMCPy [1]
A Quasi-Monte Carlo (QMC) Software in Python 3

Aleksei Sorokin,
Sou-Cheng T. Choi, Fred J. Hickernell,
Mike McCourt, Jagadeeswaran Rathinavel

Illinois Institute of Technology (IIT),
Department of Applied Mathematics.
SigOpt Company.

August 27, 2020

A Little About Me ...

School

- 4 year undergrad at IIT
- Applied math and data science program

Research Interests

- Algorithm design and implementation
- MC / QMC
- ML models, neural networks applications

Free Time

- Programming
- Sports
- Exploring Chicago

ILLINOIS INSTITUTE OF TECHNOLOGY
1890

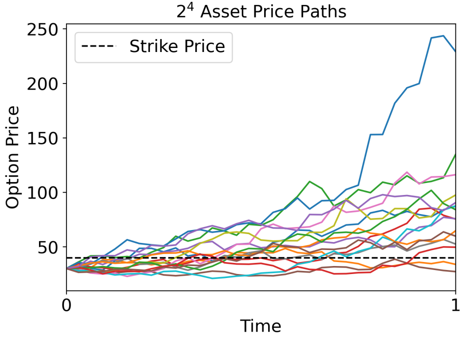
QMC Math Applications

Functions/simulations with possible outcomes $Y_1, Y_2, \ldots, Y_n$

- Financial market outcomes
- Power grid demand

Model that generates $Y = f(\boldsymbol{X})$ for $\boldsymbol{X} \sim \mathcal{U}(\mathbf{0}, \mathbf{1})$, so $Y_i = f(\boldsymbol{X}_i)$

- Numerical integration, $\mathbb{E}(Y) = \int_{[\mathbf{0},\mathbf{1}]} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{1}{n}(Y_1 + \ldots + Y_n)$
- Drawing samples from a probability distribution, $Y_1, Y_2, \ldots$



Extreme Weather Simulation

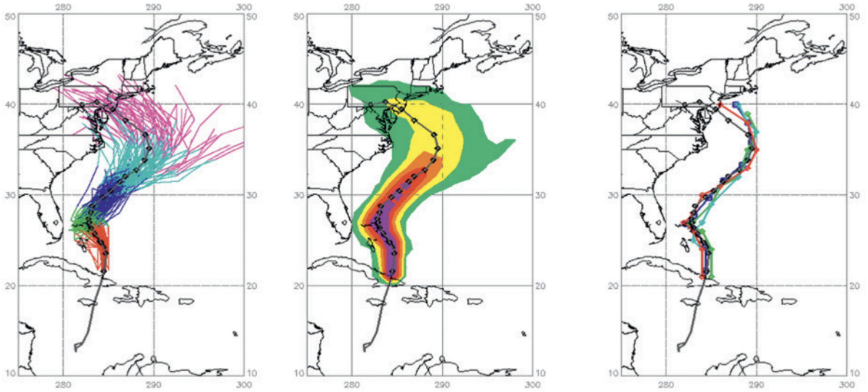


Image from [2]. See also this article from *The Wall Street Journal Article*.

Search and Rescue Optimal Planning System (SAROPS)

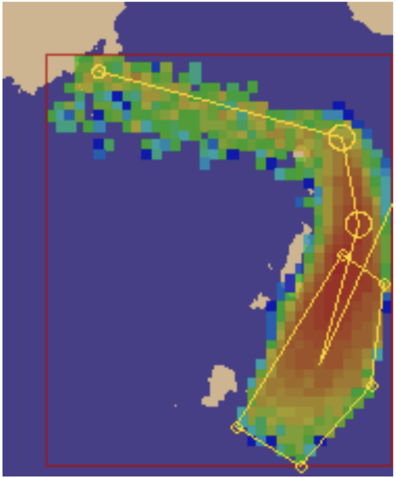


Image from [3]. See also the SAROPS Wiki page.

Computer Graphics

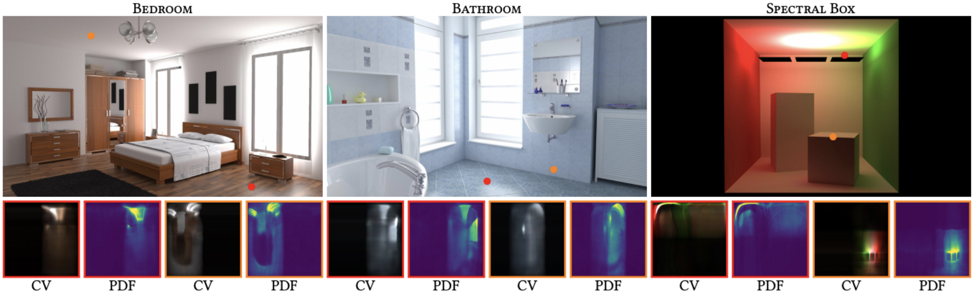


Image from [4]: uses neural network control variates for MC ray-tracing.

QMCPy

Why?

- Lack of MC / QMC software in Python
- Incompatible MC / QMC software
- Allow researchers / practitioners to utilize the work of others
- Guaranteed Automatic Integration Library (GAIL) [5]: MATLAB → Python

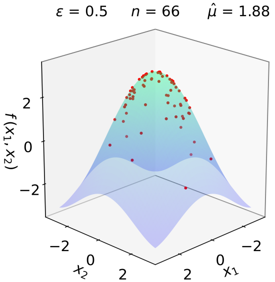
Goals

- Extensible
- Well tested
- Well documented
- Community developed
- Minimal dependencies

The (Quasi-)Monte Carlo Problem

$$\mu = \int_{\mathcal{T}} g(\boldsymbol{t})\,\rho(\boldsymbol{t})\mathrm{d}\boldsymbol{t} = \int_{\mathcal{X}} f(\boldsymbol{x})\,\psi(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \quad \approx \quad \int_{\mathcal{X}} f(\boldsymbol{x})\,\hat{\nu}(d\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}_i) = \hat{\mu}_n$$

$g : \mathcal{T} \to \mathbb{R} =$ original integrand
$\lambda =$ true measure with weight function ρ
$T : \mathcal{X} \to \mathcal{T} =$ change of variables
$f : \mathcal{X} \to \mathbb{R} =$ integrand after change of variables
$\nu =$ easy-to-sample measure with PDF ψ
$\approx \hat{\nu}_n = 1/n \sum_{i=1}^{n} \delta_{\boldsymbol{x}_i}(\cdot) =$ discrete distribution

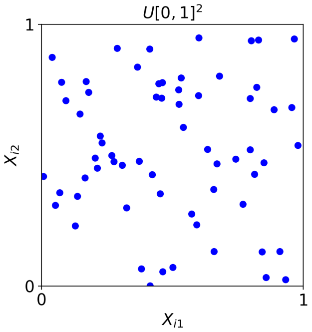


How to choose nodes $\{\boldsymbol{x}_i\}_{i=1}^{n}$ so that $|\mu - \hat{\mu}_n| < \epsilon =$ error tolerance?
Independent and identically distributed (**IID**) or low-discrepancy (**LD**)?

MC vs QMC

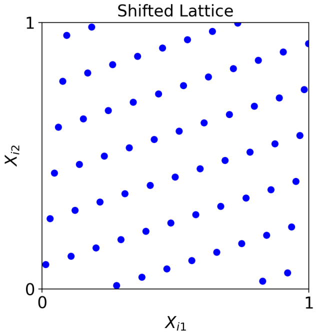
MC

- Nodes: independent, IID
- Cost: $\mathcal{O}(\varepsilon^{-2})$



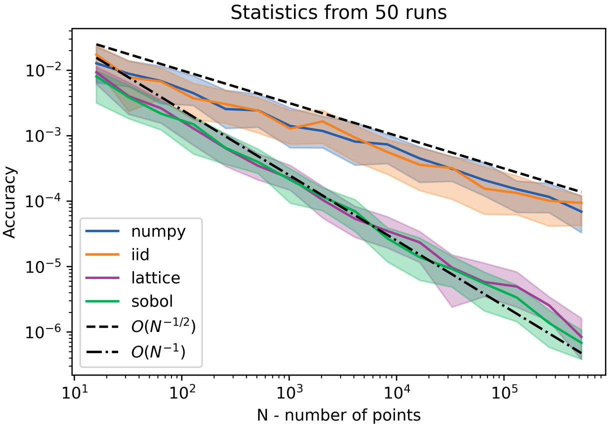
QMC

- Nodes: dependent, LD
- Cost: $\mathcal{O}(\varepsilon^{-1-\delta})$



MC vs QMC

QMC is (often) MUCH faster than MC!

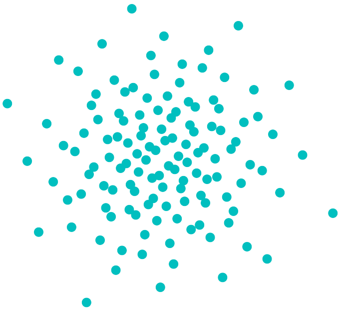


See the qEI with QMCPy blog for more info

Setting up QMCPy [1]

```
pip install qmcpy
```

- Install from PyPI:
  `pypi.org/project/qmcpy/`
- Clone from GitHub:
  `github.com/QMCSoftware/QMCSoftware`
- Documentation: `qmcpy.readthedocs.io`
- Blogs: `qmcpy.wordpress.com`
- Google Colab Quickstart:
  `tinyurl.com/QMCPyQuickstart`
- Google Colab MCQMC 2020 Tutorial:
  `tinyurl.com/QMCPyTutorial`



```
>>> from qmcpy import *
```

Abstract Classes and Some Implementations

Discrete Distribution

- (IID): IID Standard uniform/Gaussian
- (LD): Lattice, Sobol', Halton

True Measure

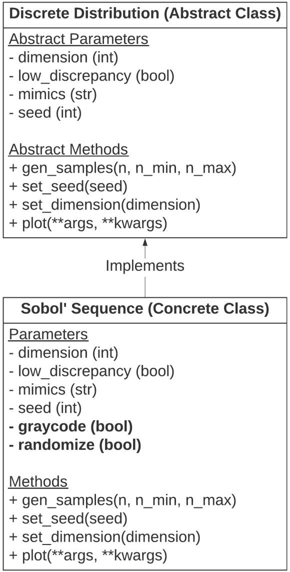
- Uniform
- Gaussian

Integrand

- Keister [6]
- European/Asian options

Stopping Criterion

- MC by Berry-Esseen inequalities (guaranteed)
- QMC for lattice/Sobol' sequences
  by Fourier/Walsh transform (guaranteed)
- MC/QMC multilevel algorithms



Discrete Distribution (Abstract Class)

Abstract Parameters
- dimension (int)
- low_discrepancy (bool)
- mimics (str)
- seed (int)

Abstract Methods
+ gen_samples(n, n_min, n_max)
+ set_seed(seed)
+ set_dimension(dimension)
+ plot(**args, **kwargs)

Implements

Sobol' Sequence (Concrete Class)

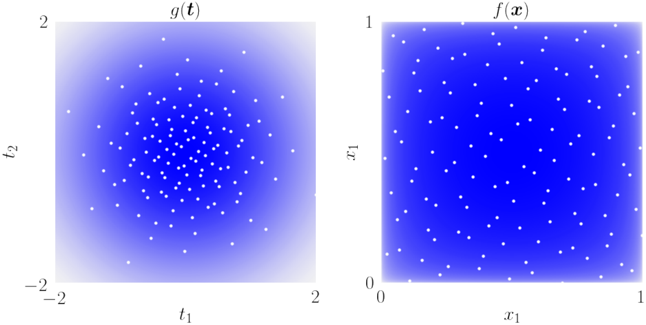
Parameters
- dimension (int)
- low_discrepancy (bool)
- mimics (str)
- seed (int)
- graycode (bool)
- randomize (bool)

Methods
+ gen_samples(n, n_min, n_max)
+ set_seed(seed)
+ set_dimension(dimension)
+ plot(**args, **kwargs)

Keister [6] Example

$$\mu = \int_{\mathbb{R}^d} \cos(\lVert \boldsymbol{t} \rVert) \exp(-\lVert \boldsymbol{t} \rVert^2) \, \mathrm{d}\boldsymbol{t} = \int_{\mathbb{R}^d} \underbrace{\pi^{d/2} \cos(\lVert \boldsymbol{t} \rVert)}_{g(\boldsymbol{t})} \underbrace{\pi^{-d/2} \exp(-\lVert \boldsymbol{t} \rVert^2) \, \mathrm{d}\boldsymbol{t}}_{\mathcal{N}(\mathbf{0}, \mathsf{I}/2)}$$

$$= \int_{[0,1]^d} \underbrace{\pi^{d/2} \cos(\lVert \Phi^{-1}(x_j)/\sqrt{2} \rVert)}_{f(\boldsymbol{x})} \underbrace{1}_{\mathcal{U}(\mathbf{0},\mathbf{1})} \, \mathrm{d}\boldsymbol{x}$$



$$g(\boldsymbol{t}) = \pi^{d/2}\cos(\|\boldsymbol{t}\|) \qquad \lambda \sim \mathcal{N}(\boldsymbol{0}, \mathsf{I}/2)$$

```
>>> from numpy import *
>>> d = 2 # dimension
>>> # discrete distribution
>>> dd = Lattice(d)
>>> # true measure
>>> tm = Gaussian(dd, covariance=1/2)
>>> # integrand
>>> i = CustomFun(tm,
>>>   lambda x: pi**(d/2)*cos(linalg.norm(x,axis=1)))
>>> # stopping criterion
>>> sc = CubQMCLatticeG(i, abs_tol=1e-3)
>>> # QMC integration
>>> solution,data = sc.integrate()
>>> # see results from integration process
>>> data # print(data)
```

```
Solution: 1.8081
Lattice (DiscreteDistribution Object)
        dimension       2
        scramble        1
        seed            None
        backend         gail
        mimics          StdUniform
Gaussian (TrueMeasure Object)
        mean            0
        covariance      0.5000
CubQMCLatticeG (StoppingCriterion Object)
        abs_tol         0.0010
        rel_tol         0
        n_init          1024
        n_max           34359738368
LDTransformData (AccumulateData Object)
        n_total         8192
        time_integrate  0.0204
```

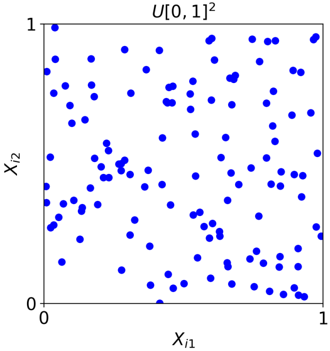
Discrete Distribution

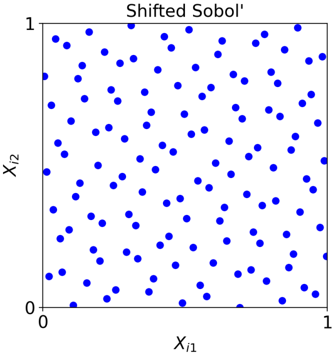
Independent Identically Distributed (IID) → MC

- IID standard uniform: `NumPy` [7] backend

Low Discrepancy (LD) → QMC

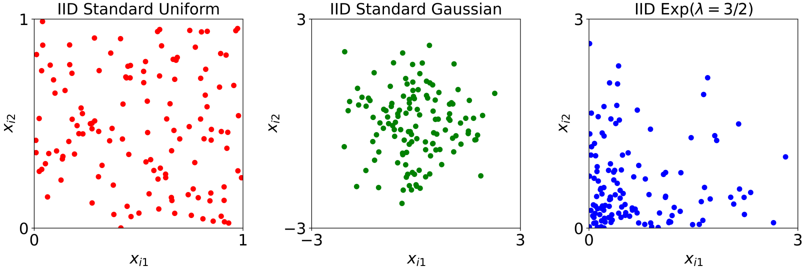
- Sobol': `QRNG` [8] or `PyTorch` [9] backends





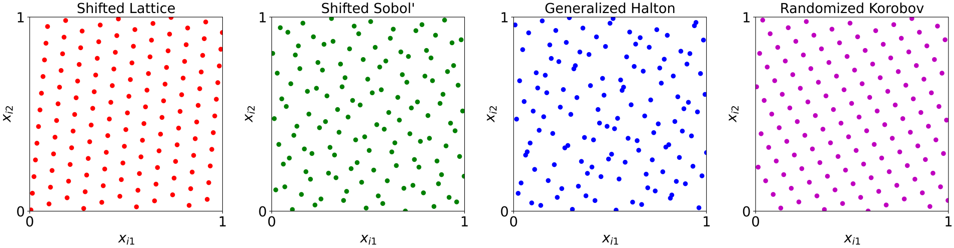
IID Discrete Distributions

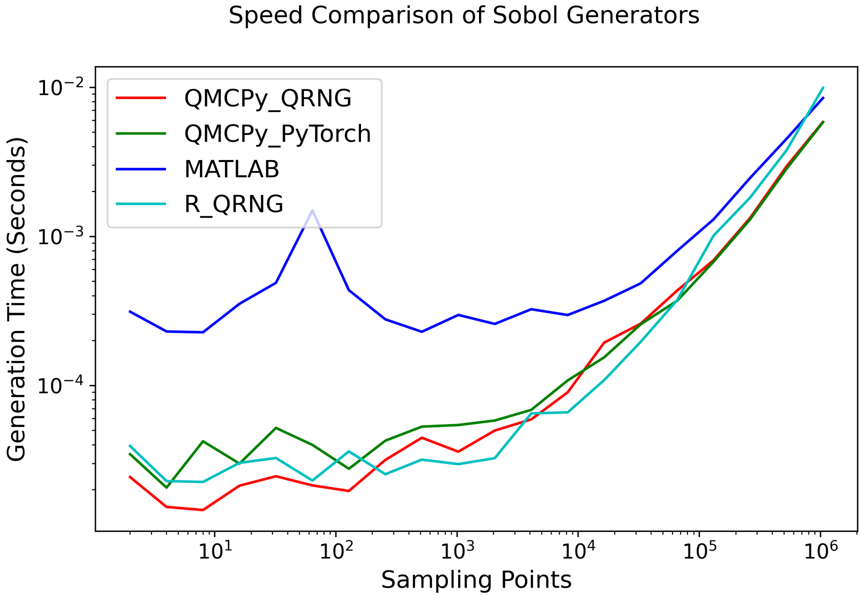
```
>>> from numpy.random import exponential
>>> d = 2 # dimension
>>> n = 2**7 # 2^7
>>> IIDStdUniform(d).gen_samples(n)
>>> IIDStdGaussian(d).gen_samples(n)
>>> ed = CustomIIDDistribution(
...    lambda n: exponential(scale=2/3 , size=(n,d)))
>>> ed.gen_samples(n)
```



LD Discrete Distributions

```
>>> d = 2 # dimension
>>> n = 2**7 # 2^7
>>> Lattice(d).gen_samples(n)
>>> Sobol(d).gen_samples(n)
>>> Halton(d).gen_samples(n)
>>> Korobov(d).gen_samples(n)
```

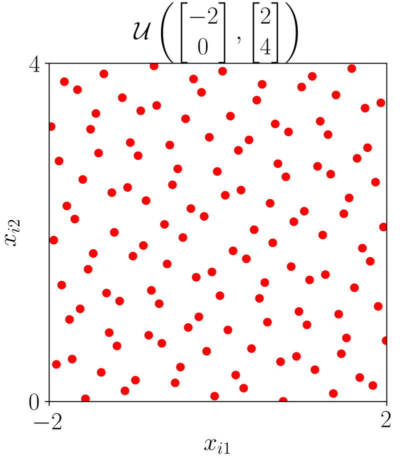



Speed Comparison of Sobol Generators
QMCPy_QRNG
QMCPy_PyTorch
MATLAB
R_QRNG
Generation Time (Seconds)
10^{-2}
10^{-3}
10^{-4}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
Sampling Points

Uniform True Measure

$$\lambda \sim \mathcal{U}(\boldsymbol{a}, \boldsymbol{b}) \qquad \nu \sim \mathcal{U}(\boldsymbol{0}, \boldsymbol{1}) \qquad T(\boldsymbol{x}) = (\boldsymbol{b} - \boldsymbol{a}) * \boldsymbol{x} + \boldsymbol{a}$$

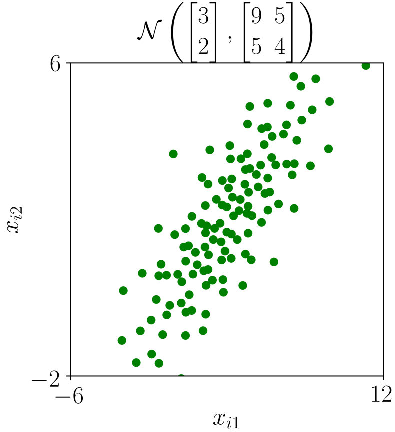
```
>>> u = Uniform(
...     Sobol(2),
...     lower_bound = [-2,0],
...     upper_bound = [2,4])
>>> u.gen_samples(2**7)
```



Gaussian True Measure

$$\lambda \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T) \qquad \nu \sim \mathcal{U}(\mathbf{0}, \mathbf{1}) \qquad T(\boldsymbol{x}) = \boldsymbol{A}^T \Phi^{-1}(\boldsymbol{x}) + \boldsymbol{\mu}$$

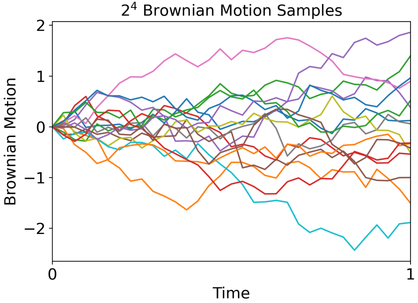
```
>>> g = Gaussian(
...     Sobol(2),
...     mean = [3.,2.],
...     covariance =
...       [[9.,5.],
...        [5.,4.]])
>>> g.gen_samples(2**7)
```



Geometric Brownian Motion True Measure, $\mathcal{B}$

$$\boldsymbol{\tau} = \left(\frac{j}{d}\right)_{j=1}^{d} \qquad \boldsymbol{\Sigma} = \left(\min(\tau_j, \tau_k)\right)_{j,k=1}^{d} \qquad \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$$

```
>>> bm = BrownianMotion(
...            Sobol(32))
>>> bm.gen_samples(2**4)
```

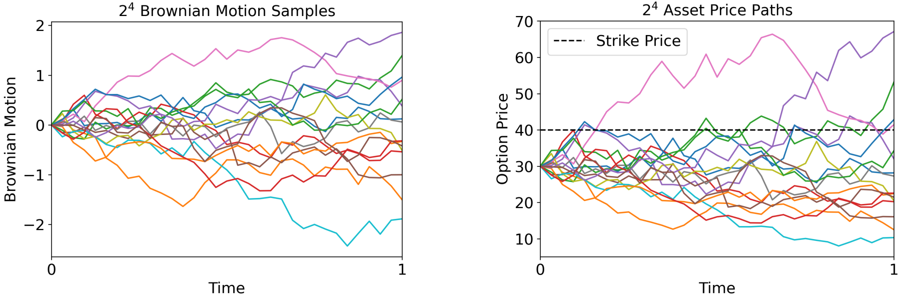


European Option Integrand

$$\text{Option price: } S(\tau_j, \boldsymbol{x}) = S_0 \exp\left((r - \sigma^2/2)\tau_j + \sigma \mathcal{B}(\tau_j, \boldsymbol{x})\right)$$

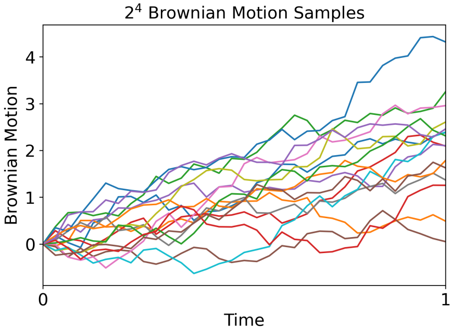
$$\text{Payoff: } P(\boldsymbol{x}) = \max\left(S(\tau_d, \boldsymbol{x}) - K\right), 0) \exp(-rT)$$

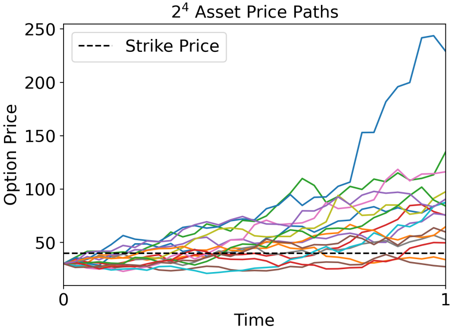
```
>>> opt = EuropeanOption(BrownianMotion(Sobol(32)),
...    start_price=30, strike_price=40)
>>> opt.f(opt.distribution.gen_samples(2**4))
```



Option Pricing with Importance Sampling

```
>>> opt = EuropeanOption(
...     BrownianMotion(Sobol(32), drift=2),
...     start_price=30, strike_price=40)
>>> opt.f(opt.distribution.gen_samples(2**4))
```


2^4 Brownian Motion Samples

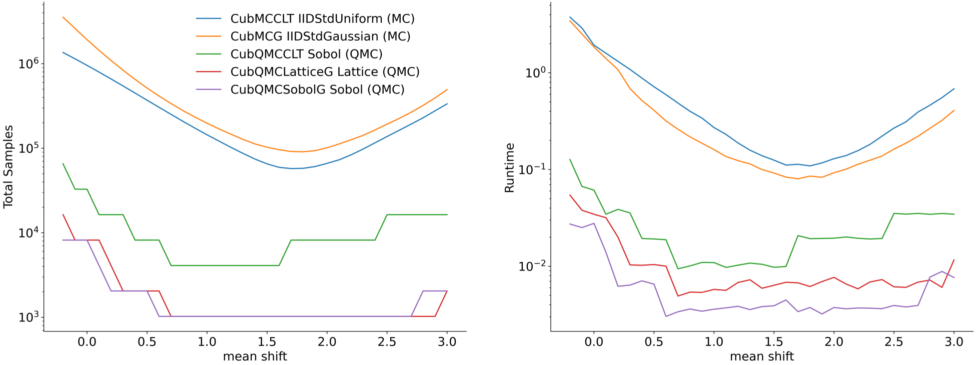

2^4 Asset Price Paths

QMCPy

QMC Applications
QMCPy Intro
Discrete Distribution
True Measure
Integrand
Stopping Criterion
Contributing
QMCPy Links
References

Stopping Criterion Comparison for Importance Sampling

European option with dimension $d = 32$ and absolute tolerance $\epsilon_{\text{abs}} = .025$



Single Level MC CLT

Initial Sample

$$\boldsymbol{x}_i \in \hat{\nu}, \qquad y_i = f(\boldsymbol{x}_i), \qquad i = 1, \dots, n_0$$

$$\hat{\mu}_{n_0} = \frac{1}{n_0} \sum_{i=1}^{n_0} y_i, \qquad \hat{\sigma}_{n_0} = \sqrt{\frac{1}{n_0} \sum_{i=1}^{n_0} (y_i - \hat{\mu}_0)^2}$$

CLT

$$\frac{\hat{\mu}_n - \mu}{\sigma / \sqrt{n}} \xrightarrow[n \to \infty]{} \mathcal{N}(0,1)$$

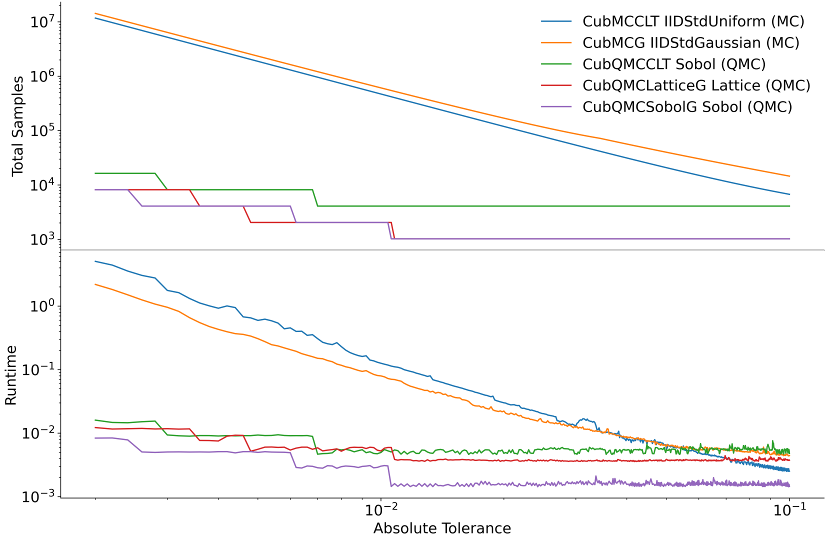
Final Sample

$$P\left(\hat{\mu}_{n_1} - \frac{\sigma z^*}{\sqrt{n_1}} \le \mu \le \hat{\mu}_{n_1} + \frac{\sigma z^*}{\sqrt{n_1}} \right) \approx 1 - \alpha$$

$$\sigma \le \mathcal{C} \hat{\sigma}_{n_0} \implies n_1 = \left\lceil \left(\frac{\mathcal{C} \hat{\sigma}_{n_0} z^*}{\epsilon} \right)^2 \right\rceil$$

MC vs QMC

Keister integrand with dimension $d = 3$



Contributions

- Guaranteed Automatic Integration Library (GAIL) [5]
- Marius Hofert and Christiane Lemieux's QRNG [8]
- Mike Giles MLMC [10] and MLQMC [11] software
- Art Owen's Halton sequences [12]
- Pierre L'Ecuyer's Lattice Builder [13]
- Dirk Nuyens's Magic Point Shop (MPS) [14]

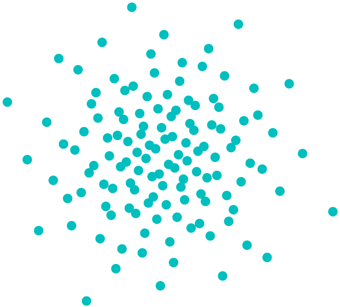
You can help by

- Using QMCPy
- Suggesting features and finding bugs github.com/QMCSoftware/QMCSoftware/issues
- Add features or use-cases github.com/QMCSoftware/QMCSoftware/pulls

QMCPy Links

pip install qmcpy

- Install from PyPI:
  `pypi.org/project/qmcpy/`
- Clone from GitHub:
  `github.com/QMCSoftware/QMCSoftware`
- Documentation: `qmcpy.readthedocs.io`
- Blogs: `qmcpy.wordpress.com`
- Google Colab Quickstart:
  `tinyurl.com/QMCPyQuickstart`
- Google Colab MCQMC 2020 Tutorial:
  `tinyurl.com/QMCPyTutorial`



```
>>> from qmcpy import *
```

References I

1. Choi, S.-C. T., Hickernell, F. J., McCourt, M. & Sorokin, A. *QMCPy: A quasi-Monte Carlo Python Library.* 2020+. https://github.com/QMCSoftware/QMCSoftware.
2. Swinbank, R. *et al.* The TIGGE project and its achievements. *Bulletin of the American Meteorological Society* **97,** 49–67 (Feb. 2016).
3. Kratzke, T., Stone, L. & Frost, J. *Search and Rescue Optimal Planning System.* in (Aug. 2010), 1–8.
4. Müller, T., Rousselle, F., Novák, J. & Keller, A. *Neural Control Variates.* 2020. arXiv: 2006.01524 [cs.LG].
5. Choi, S.-C. T. *et al. GAIL: Guaranteed Automatic Integration Library (Versions 1.0–2.2).* MATLAB software. 2013–2017. http://gailgithub.github.io/GAIL_Dev/.
6. Keister, B. D. Multidimensional Quadrature Algorithms. *Computers in Physics* **10,** 119–122 (1996).

References II

7. Oliphant, T. *Guide to NumPy.* `https://ecs.wgtn.ac.nz/foswiki/pub/Support/ManualPagesAndDocumentation/numpybook.pdf` (Trelgol Publishing USA, 2006).
8. Hofert, M. & Lemieux, C. *qrng: (Randomized) Quasi-Random Number Generators.* R package version 0.0-7 (2019). `https://CRAN.R-project.org/package=qrng`.
9. Paszke, A. *et al.* in *Advances in Neural Information Processing Systems 32* (eds Wallach, H. *et al.*) 8024–8035 (Curran Associates, Inc., 2019). `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.
10. Giles, M. Multilevel Monte Carlo Path Simulation. *Operations Research* **56,** 607–617 (June 2008).

References III

11. Giles, M. B. & Waterhouse, B. J. Multilevel quasi-Monte Carlo path simulation. *Advanced Financial Modelling, Radon Series on Computational and Applied Mathematics* **8,** 165–181 (2009).
12. Owen, A. B. *A randomized Halton algorithm in R.* 2017. arXiv: `1706.02808 [stat.CO]`.
13. L'ecuyer, P. & Munger, D. Algorithm 958: Lattice Builder: A General Software Tool for Constructing Rank-1 Lattice Rules. *ACM Trans. Math. Softw.* **42.** ISSN: 0098-3500. https://doi.org/10.1145/2754929 (May 2016).
14. Kuo, F. Y. & Nuyens, D. *Application of quasi-Monte Carlo methods to elliptic PDEs with random diffusion coefficients - a survey of analysis and implementation.* 2016. arXiv: `1606.06613 [math.NA]`.