# Building QMCPy's Quasi-Monte Carlo Framework

Aleksei G. Sorokin[1], Sou-Cheng T. Choi[1,2], Fred J. Hickernell[1], Mike McCourt[3], Jagadeeswaran Rathinavel[4]

[1]Illinois Institute of Technology (IIT), Department of Applied Mathematics
[2]Kamakura Corporation
[3]SigOpt, an Intel company
[4]Wi-Tronix LLC

August 17, 2021

# Rewrite an Integral as an Expectation

Applications in applied statistics, finance, computer graphics, ...

$$\mu = \int_{\mathcal{T}} g(\boldsymbol{t})\lambda(\boldsymbol{t})\,\mathrm{d}\boldsymbol{t} = \int_{[0,1]^d} g(\boldsymbol{\Psi}(\boldsymbol{x}))\lambda(\boldsymbol{\Psi}(\boldsymbol{x}))|\boldsymbol{\Psi}'(\boldsymbol{x})|\,\mathrm{d}\boldsymbol{x} = \int_{[0,1]^d} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \mathbb{E}[f(\boldsymbol{X})]$$

$$\boldsymbol{X} \sim \mathcal{U}[0,1]^d$$

Original Integrand $g : \mathcal{T} \to \mathbb{R}$

True Measure $\lambda : \mathcal{T} \to \mathbb{R}^+$ e.g. probability density or 1 for Lebesgue measure

Transformation $\boldsymbol{\Psi} : [0,1]^d \to \mathcal{T}$ with Jacobian $|\boldsymbol{\Psi}'(\boldsymbol{x})|$

Transformed Integrand $f : [0,1]^d \to \mathbb{R}$

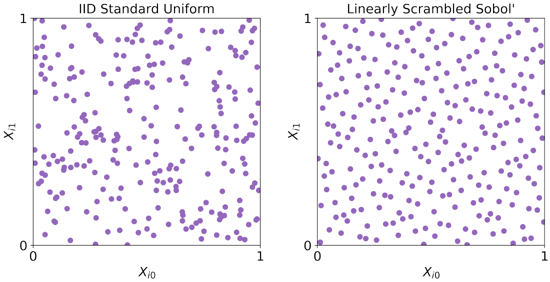
QMCPy automatically approximates integrals

# Approximate the Integral by Sampling Well

$$\text{sample mean} = \hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}_i) \approx \int_{[0,1]^d} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \mu = \text{mean}$$

Simple Monte Carlo: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \overset{\text{IID}}{\sim} \mathcal{U}[0,1]^d$

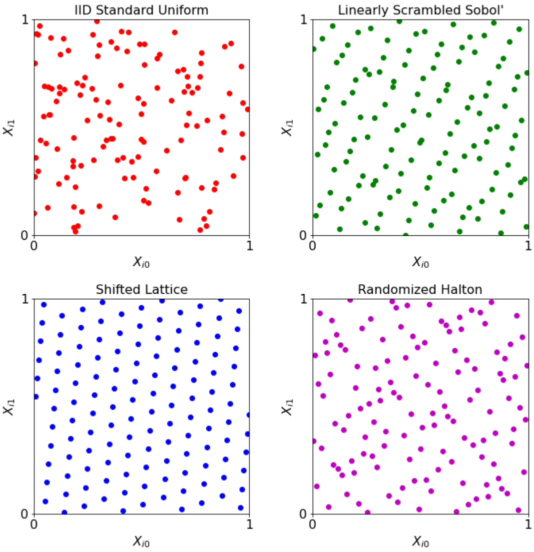
Quasi-Monte Carlo: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \overset{\text{LD}}{\sim} \mathcal{U}[0,1]^d$ (Low-Discrepancy)



# Sample Generators

## Sobol' Example

```
>>> import qmcpy as qp
>>> sobol = qp.Sobol(2)
>>> sobol.gen_samples(2**3)
array([[0.387, 0.146],
       [0.552, 0.506],
       [0.169, 0.901],
       [0.771, 0.258],
       [0.303, 0.724],
       [0.639, 0.116],
       [0.023, 0.48 ],
       [0.922, 0.867]])
```



# Custom Digital Nets in Base 2

Niederreiter Sequence supporting $20,000$ dimensions and $2^{32}$ points

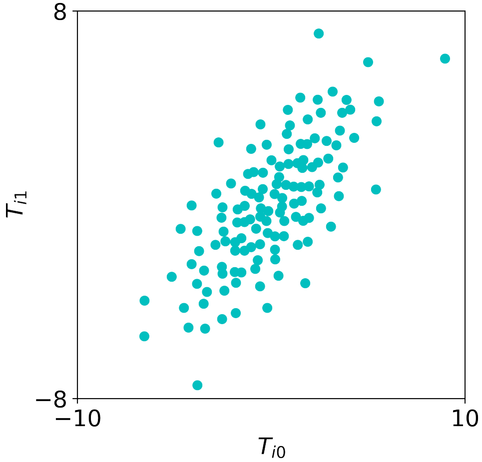
```
>>> nied = qp.DigitalNet(
...       dimension = 5,
...       z_path = "niederreiter_mat.20000.32.msb.npy",
...       randomize = False)
>>> nied.gen_samples(n_min=8, n_max=16)
array([[0.0625, 0.9375, 0.375 , 0.3906, 0.7656],
       [0.5625, 0.4375, 0.125 , 0.2656, 0.8906],
       [0.3125, 0.1875, 0.625 , 0.1406, 0.6406],
       [0.8125, 0.6875, 0.875 , 0.0156, 0.5156],
       [0.1875, 0.3125, 0.9375, 0.7656, 0.1406],
       [0.6875, 0.8125, 0.6875, 0.8906, 0.0156],
       [0.4375, 0.5625, 0.1875, 0.5156, 0.2656],
       [0.9375, 0.0625, 0.4375, 0.6406, 0.3906]])
```

# True Measure Transforms: Apply change of variables

## Gaussian Example

$$\boldsymbol{\Psi}(\boldsymbol{X}) = \boldsymbol{a} + \mathsf{A}\boldsymbol{\Phi}^{-1}(\boldsymbol{X}) \sim \mathcal{N}(\boldsymbol{a}, \Sigma = \mathsf{A}\mathsf{A}^T)$$

```
>>> gauss = qp.Gaussian(sobol,
...     mean =          [0,0],
...     covariance = [[7,5],
...                   [5,7]])
>>> gauss.gen_samples(2**2)
array([[ 0.352, -1.754],
       [ 0.302,  0.333],
       [-3.633, -1.054],
       [ 2.462,  1.166]])
```



# Integrand Examples: Define the original integrand

## Keister Example [1]

$$\mu = \int_{\mathbb{R}^d} \cos(\|\boldsymbol{t}\|) \exp(-\|\boldsymbol{t}\|^2) \, \mathrm{d}\boldsymbol{t}$$

$$= \int_{\mathbb{R}^d} \underbrace{\pi^{d/2} \cos(\|\boldsymbol{t}\|)}_{g(\boldsymbol{t})} \underbrace{\mathcal{N}(\boldsymbol{t}|\boldsymbol{0},\mathsf{I}/2)}_{\lambda(\boldsymbol{t})} \, \mathrm{d}\boldsymbol{t}$$

$$= \int_{[0,1]^d} \pi^{d/2} \cos(\|\boldsymbol{\Psi}(\boldsymbol{x})\|) \, \mathrm{d}\boldsymbol{x}$$

$$= \int_{[0,1]^d} \underbrace{g(\boldsymbol{\Psi}(\boldsymbol{x}))}_{f(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x}$$

```
>>> from numpy import sqrt,pi,cos
>>> def my_keister(t):
...     d = t.shape[1]
...     norm = sqrt((t**2).sum(1))
...     k = pi**(d/2)*cos(norm)
...     return k
>>> sob5 = qp.Sobol(5)
>>> gauss_sob = qp.Gaussian(sob5,
...     mean = 0, covariance = 1/2)
>>> keister = qp.CustomFun(
...     true_measure = gauss_sob,
...     g = my_keister)
>>> x = sob5.gen_samples(2**20)
>>> y = keister.f(x)
>>> mu_hat = y.mean()
>>> mu_hat
1.1353362571289711
```

# Stopping Criterion: Determine $n$ so $|\mu - \hat{\mu}_n| < \epsilon$

Samples $n$ required for

$$\text{Monte Carlo: } \mathcal{O}(\epsilon^{-2})$$

$$\text{Quasi-Monte Carlo: } \mathcal{O}(\epsilon^{-1})$$

QMC is significantly more efficient!

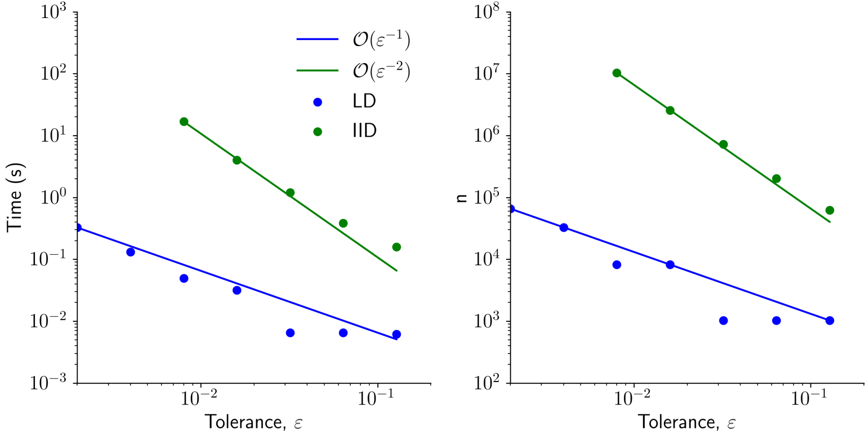
**Sobol' Cubature Example [2]**

```
>>> sc = qp.CubQMCSobolG(
...       integrand = keister,
...       abs_tol = 1e-4)
>>> sol,data = sc.integrate()
```

```
>>> data
LDTransformData
    solution          1.135
    error_bound       9.69e-05
    n_total           2^(20)
    time_integrate    0.611
CubQMCSobolG
    abs_tol           1.00e-04
    rel_tol           0
CustomFun
Gaussian
    mean              0
    covariance        2^(-1)
Sobol
    d                 5
    randomize         1
```

# QMC Beats MC

## Standard Keister Integrand in 5 Dimensions



# Vectored Stopping Criterion: Box Integral Example [3]

$$B_d(s) = \int_{[0,1]^d} \underbrace{(t_1^2 + \cdots + t_d^2)^{s/2}}_{g_s(\boldsymbol{t})} \mathrm{d}\boldsymbol{t}$$

```
>>> B = qp.BoxIntegral(qp.Lattice(3), s=[-1,1,7])
>>> solution,data = qp.CubQMCCLT(B, abs_tol=1e-4).integrate()
>>> data
MeanVarDataRep (AccumulateData Object)
    solution        [1.19   0.961  2.329]
    error_bound     [8.263e-05 9.038e-05 8.922e-05]
    n_total         2^(25)
    n               [2097152.     8192.   262144.]
    replications    2^(4)
    time_integrate  8.241
```

# Future Work

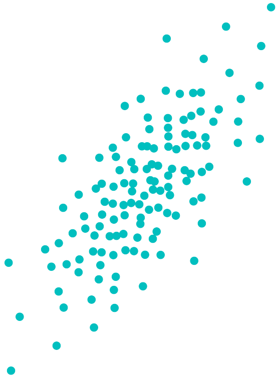
- Continue vectorizing stopping criteria
- Enable stopping criteria to support ratio of integrals

$$\mu = \frac{\int_{[0,1]^d} f_1(\boldsymbol{x})\mathrm{d}\boldsymbol{x}}{\int_{[0,1]^d} f_2(\boldsymbol{x})\mathrm{d}\boldsymbol{x}}$$

- Support higher order digital net generating vectors
- Add Latin Hypercube Sampling
- Add more use cases
- Refactor code for speed and efficiency

# QMCPy Resources

- PyPI: pypi.org/project/qmcpy/
- GitHub: github.com/QMCSoftware/QMCSoftware
- Documentation: qmcpy.readthedocs.io
- Blogs: qmcpy.org
- MCQMC2020 Tutorial
  - Slides: qmcpy.org/mcqmc-2020-tutorial/
  - Notebook: tinyurl.com/QMCPyTutorial
  - "Quasi-Monte Carlo Software" Article [4]



# References

1. Keister, B. D. Multidimensional Quadrature Algorithms. *Computers in Physics* **10,** 119–122 (1996).
2. Hickernell, F. J. & Jiménez Rugama, L. A. *Reliable Adaptive Cubature Using Digital Sequences.* 2014. arXiv: 1410.8615 [math.NA].
3. Bailey, D., Borwein, J. & Crandall, R. Box integrals. *Journal of Computational and Applied Mathematics* **206,** 196–208. ISSN: 0377-0427. https://www.sciencedirect.com/science/article/pii/S0377042706004250 (2007).
4. Choi, S.-C. T., Hickernell, F. J., Jagadeeswaran, R., McCourt, M. J. & Sorokin, A. G. *Quasi-Monte Carlo Software.* arXiv:2102.07833 [cs.MS]. 2021. arXiv: 2102.07833 [cs.MS].