qmcpy

Release 0.1

Fred J. Hickernell, Aleksei Sorokin, Sou-Cheng T. Choi

Dec 23, 2019

CONTENTS

CHAPTER

ONE

ABOUT OUR QMC SOFTWARE COMMUNITY

Contents

1.1 Quasi-Monte Carlo Community Software

Quasi-Monte Carlo (QMC) methods are used to approximate multivariate integrals. They have four main components: an integrand, a discrete distribution, summary output data, and stopping criterion. Information about the integrand is obtained as a sequence of values of the function sampled at the data-sites of the discrete distribution. The stopping criterion tells the algorithm when the user-specified error tolerance has been satisfied. We are developing a framework that allows collaborators in the QMC community to develop plug-and-play modules in an effort to produce more efficient and portable QMC software. Each of the above four components is an abstract class. Abstract classes specify the common properties and methods of all subclasses. The ways in which the four kinds of classes interact with each other are also specified. Subclasses then flesh out different integrands, sampling schemes, and stopping criteria. Besides providing developers a way to link their new ideas with those implemented by the rest of the QMC community, we also aim to provide practitioners with state-of-the-art QMC software for their applications.

1.1.1 Citation

If you find QMCPy helpful in your work, please support us by citing the following work:

Fred J. Hickernell, Sou-Cheng T. Choi, and Aleksei Sorokin, "QMC Community Software." Python software, 2019. Work in progress. Available from https://github.com/QMCSoftware/QMCSoftware

1.1.2 Developers

- Sou-Cheng T. Choi
- Fred J. Hickernell
- Aleksei Sorokin

1.1.3 Contributors

- Michael McCourt

1.1.4 Acknowledgment

We thank Dirk Nuyens for fruitful discussions related to Magic Point Shop.

1.1.5 References

[1] F.Y. Kuo & D. Nuyens. "Application of quasi-Monte Carlo methods to elliptic PDEs with random diffusion coefficients - a survey of analysis and implementation",Foundations of Computational Mathematics, 16(6):1631-1696, 2016. (springer link, arxiv link)

[2] Fred J. Hickernell, Lan Jiang, Yuewei Liu, and Art B. Owen, "Guaranteed conservative fixed width confidence intervals via Monte Carlo sampling," Monte Carlo and Quasi-Monte Carlo Methods 2012 (J. Dick, F.Y. Kuo, G. W. Peters, and I. H. Sloan, eds.), pp. 105-128, Springer-Verlag, Berlin, 2014. DOI: 10.1007/978-3-642-41095-6_5

[3] Sou-Cheng T. Choi, Yuhan Ding, Fred J. Hickernell, Lan Jiang, Lluis Antoni Jimenez Rugama, Da Li, Jagadeeswaran Rathinavel, Xin Tong, Kan Zhang, Yizhi Zhang, and Xuan Zhou, GAIL: Guaranteed Automatic Integration Library (Version 2.3) [MATLAB Software], 2019. Available from http://gailgithub.github.io/GAIL_Dev/

[4] Sou-Cheng T. Choi, "MINRES-QLP Pack and Reliable Reproducible Research via Supportable Scientific Software," Journal of Open Research Software, Volume 2, Number 1, e22, pp. 1-7, 2014.

[5] Sou-Cheng T. Choi and Fred J. Hickernell, "IIT MATH-573 Reliable Mathematical Software" [Course Slides], Illinois Institute of Technology, Chicago, IL, 2013. Available from http://gailgithub.github.io/GAIL_Dev/

[6] Daniel S. Katz, Sou-Cheng T. Choi, Hilmar Lapp, Ketan Maheshwari, Frank Loffler, Matthew Turk, Marcus D. Hanwell, Nancy Wilkins-Diehr, James Hetherington, James Howison, Shel Swenson, Gabrielle D. Allen, Anne C. Elster, Bruce Berriman, Colin Venters, "Summary of the First Workshop On Sustainable Software for Science: Practice and Experiences (WSSSPE1)," Journal of Open Research Software, Volume 2, Number 1, e6, pp. 1-21, 2014.

[7] Fang, K.-T., & Wang, Y. (1994). Number-theoretic Methods in Statistics. London, UK: CHAPMAN & HALL

[8] Lan Jiang, Guaranteed Adaptive Monte Carlo Methods for Estimating Means of Random Variables, PhD Thesis, Illinois Institute of Technology, 2016.

[9] Lluis Antoni Jimenez Rugama and Fred J. Hickernell, "Adaptive multidimensional integration based on rank-1 lattices," Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014 (R. Cools and D. Nuyens, eds.), Springer Proceedings in Mathematics and Statistics, vol. 163, Springer-Verlag, Berlin, 2016, arXiv:1411.1966, pp. 407-422.

[10] Kai-Tai Fang and Yuan Wang, Number-theoretic Methods in Statistics, Chapman & Hall, London, 1994.

[11] Fred J. Hickernell and Lluis Antoni Jimenez Rugama, "Reliable adaptive cubature using digital sequences", Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014 (R. Cools and D. Nuyens, eds.), Springer Proceedings in Mathematics and Statistics, vol. 163, Springer-Verlag, Berlin, 2016, arXiv:1410.8615 [math.NA], pp. 367-383.

Contents

1.2 Python 3 Library of QMC Software

1.2.1 QMCPy

Package of main components

- Integrand classes
- True Measure classes
- Discrete Distribution classes
- Stopping Criterion classes
- Accumulate Data classes
- Third Party contributed classes
- integrate function

1.2.2 workouts

Example uses of QMCPy package

1.2.3 test

Sets of long and short unittests

1.2.4 outputs

Logs and figures generated by workouts

1.2.5 demos

Example use of QMCPy as an independent package

1.2.6 sphinx

Automated project documentation

1.2.7 Installation

```
pip install qmcpy
```

A virtual environment is recommended for developers/contributors Ensure *.../python_prototypes/* is in your path Install dependencies with

```
pip install requirements.txt
```

Contents

1.3 QMCPy

1.3.1 Integrand

The function to integrate *Abstract class with concrete implementations*

- Linear: $y_i = \sum_{j=0}^{d-1}(x_{ij})$
- Keister: $y_i = \pi^{d/2} * \cos(||x_i||_2)$
- Asian Call
  - $S_i(t_j) = S(0)e^{(r-\frac{\sigma^2}{2})t_j + \sigma\mathcal{B}(t_j)}$
  - discounted call payoff $= max(\frac{1}{d}\sum_{j=0}^{d-1} S(jT/d) - K)\ ,\ 0)$
  - discounted put payoff $= max(K - \frac{1}{d}\sum_{j=0}^{d-1} S(jT/d))\ ,\ 0)$

1.3.2 True Measure

General measure used to define the integrand *Abstract class with concrete implementations*

- Uniform: $\mathcal{U}(a, b)$
- Gaussian: $\mathcal{N}(\mu, \sigma^2)$
- Brownian Motion: $\mathcal{B}(t_j) = B(t_{j-1}) + Z_j\sqrt{t_j - t_{j-1}}$ for $Z_j \sim \mathcal{N}(0, 1)$

1.3.3 Discrete Distribution

Sampling nodes iid or lds (low-discrepancy sequence) *Abstract class with concrete implementations*

- IID Standard Uniform: $x_j \overset{iid}{\sim} \mathcal{U}(0, 1)$
- IID Standard Gaussian: $x_j \overset{iid}{\sim} \mathcal{N}(0, 1)$
- Lattice (base 2): $x_j \overset{lds}{\sim} \mathcal{U}(0, 1)$
- Sobol (base 2): $x_j \overset{lds}{\sim} \mathcal{U}(0, 1)$

1.3.4 Stopping Criterion

The stopping criterion to determine sufficient approximation *Abstract class with concrete implementations* Central Limit Theorem (CLT) $\hat{\mu}_n = \overline{Y}_n \approx \mathcal{N}(\mu, \frac{\sigma^2}{n})$ $\mathbb{P}[\hat{\mu}_n - \frac{\mathcal{Z}_{\alpha/2}\hat{\sigma}_n}{\sqrt{n}} \leq \mu \leq \hat{\mu}_n + \frac{\mathcal{Z}_{\alpha/2}\hat{\sigma}_n}{\sqrt{n}}] \approx 1 - \alpha$

- CLT for $x_i \sim$ iid
- CLT Repeated for $\{x_{r,i}\}_{r=1}^{R} \sim$ lds

1.3.5 Accumulate Data Class

Stores data values of corresponding stopping criterion procedure *Abstract class with concrete implementations*

- Mean Variance Data (Controlled by CLT)
- Mean Variance Repeated Data (Controlled by CLT Repeated)

1.3.6 Integrate Method

Repeatedly samples the integrand at nodes generated by the discrete distribution and transformed to mimic the integrand's true measure until the Stopping Criterion is met *Function with arguments:*

- Integrand object
- True Measure object
- Discrete Distribution object
- Stopping Criterion object

Contents

1.4 Tests

1.4.1 Fast unittests

Quickly check functionality Run all in < 1 second

```
python -m unittest discover -s test/fasttests
```

1.4.2 Long unittests

Call workout functions Runs all in < 10 seconds

```
python -m unittest discover -s test/longtests
```

CHAPTER TWO

LICENSE

CHAPTER

THREE

QMCPY DOCUMENTATION

3.1 Integration Method

Main driver function for QMCPy.

qmcpy.integrate.**integrate**(*integrand*, *true_measure*, *discrete_distrib=None*, *stopping_criterion=None*)

Specify and compute integral of $f(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{X}$.

Parameters

- **integrand** (*Integrand*) – an object from class Integrand. If None (default), sum of two variables defined on unit square is used.
- **true_measure** (*TrueMeasure*) – an object from class TrueMeasure. If None (default), standard uniform distribution is used.
- **discrete_distrib** (*DiscreteDistribution*) – an object from class DiscreteDistribution. If None (default), IID standard uniform distribution is used.
- **stopping_criterion** (*StoppingCriterion*) – an object from class StoppingCriterion. If None (default), criterion based on central limit theorem with absolute tolerance equal to 0.01 is used.

Returns

tuple containing:

solution (float): estimated value of the integral

data (AccumData): input data and information such as number of sampling points and run time used to obtain solution

Return type tuple

3.2 Integrand Class

3.2.1 Asican Call Option Payoff

Definition for class AsianCall, a concrete implementation of Integrand

class qmcpy.integrand.asian_call.**AsianCall**(*bm_measure*, *volatility=0.5*, *start_price=30*, *strike_price=25*, *interest_rate=0*, *mean_type='arithmetic'*)

Specify and generate payoff values of an Asian Call option

__init__(*bm_measure*, *volatility=0.5*, *start_price=30*, *strike_price=25*, *interest_rate=0*, *mean_type='arithmetic'*)

Initialize AsianCall Integrand's'

Parameters

- **bm_measure** (*TrueMeasure*) – A BrownianMotion Measure object
- **volatility** (*float*) – sigma, the volatility of the asset
- **start_price** (*float*) – S(0), the asset value at t=0
- **strike_price** (*float*) – strike_price, the call/put offer
- **interest_rate** (*float*) – r, the annual interest rate
- **mean_type** (*string*) – 'arithmetic' or 'geometric' mean

g(*x*)

Original integrand to be integrated

Parameters x – nodes, $\boldsymbol{x}_{\mathfrak{u},i} = i^{\text{th}}$ row of an $n \cdot |\mathfrak{u}|$ matrix

Returns $n \cdot p$ matrix with values $f(\boldsymbol{x}_{\mathfrak{u},i}, \mathbf{c})$ where if $\boldsymbol{x}_i' = (x_{i,\mathfrak{u}}, \mathbf{c})_j$, then $x'_{ij} = x_{ij}$ for $j \in \mathfrak{u}$, and $x'_{ij} = c$ otherwise

get_discounted_payoffs(*stock_path*, *dimension*)

Calculate the discounted payoff from the stock path

stock_path (ndarray): option prices at monitoring times dimension (int): number of dimensions

3.2.2 Keister Function

Definition for class Keister, a concrete implementation of Integrand

class qmcpy.integrand.keister.**Keister**(*dimension*)

Specify and generate values $f(\boldsymbol{x}) = \pi^{d/2} \cos(\|\boldsymbol{x}\|)$ for $\boldsymbol{x} \in \mathbb{R}^d$.

The standard example integrates the Keister integrand with respect to an IID Gaussian distribution with variance 1/2.

Reference:

B. D. Keister, Multidimensional Quadrature Algorithms, *Computers in Physics*, *10*, pp. 119-122, 1996.

__init__(*dimension*)

Parameters dimension (*ndarray*) – dimension(s) of the integrand(s)

g(*x*)

Original integrand to be integrated

Parameters x – nodes, $\boldsymbol{x}_{\mathfrak{u},i} = i^{\text{th}}$ row of an $n \cdot |\mathfrak{u}|$ matrix

Returns $n \cdot p$ matrix with values $f(\boldsymbol{x}_{\mathfrak{u},i}, \mathbf{c})$ where if $\boldsymbol{x}_i' = (x_{i,\mathfrak{u}}, \mathbf{c})_j$, then $x'_{ij} = x_{ij}$ for $j \in \mathfrak{u}$, and $x'_{ij} = c$ otherwise

3.2.3 A Linear Function

Definition for class Linear, a concrete implementation of Integrand

class qmcpy.integrand.linear.**Linear**(*dimension*)

Specify and generate values $f(\boldsymbol{x}) = \sum_{i=1}^{d} x_i$ for $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$

__init__ (*dimension*)

Parameters **dimension** (*ndarray*) – dimension(s) of the integrand(s)

g (x)

Original integrand to be integrated

Parameters **x** – nodes, $\boldsymbol{x}_{\mathfrak{u},i} = i^{\text{th}}$ row of an $n \cdot |\mathfrak{u}|$ matrix

Returns $n \cdot p$ matrix with values $f(\boldsymbol{x}_{\mathfrak{u},i}, \mathbf{c})$ where if $\boldsymbol{x}'_i = (x_{i,\mathfrak{u}}, \mathbf{c})_j$, then $x'_{ij} = x_{ij}$ for $j \in \mathfrak{u}$, and $x'_{ij} = c$ otherwise

3.2.4 Quick Construct for Function

Definition for class QuickConstruct, a concrete implementation of Integrand

class qmcpy.integrand.quick_construct.**QuickConstruct** (*dimension*, *custom_fun*)

Specify and generate values of a user-defined function

__init__ (*dimension*, *custom_fun*)

Initialize custom Integrand

Parameters

- **dimension** (*ndarray*) – dimension(s) of the integrand(s)
- **custom_fun** (*int*) – a callable univariable or multivariate Python function that returns a real number.

Note: Input of the function:

x: nodes, $\boldsymbol{x}_{\mathfrak{u},i} = i^{\text{th}}$ row of an $n \cdot |\mathfrak{u}|$ matrix

g (x)

Original integrand to be integrated

Parameters **x** – nodes, $\boldsymbol{x}_{\mathfrak{u},i} = i^{\text{th}}$ row of an $n \cdot |\mathfrak{u}|$ matrix

Returns $n \cdot p$ matrix with values $f(\boldsymbol{x}_{\mathfrak{u},i}, \mathbf{c})$ where if $\boldsymbol{x}'_i = (x_{i,\mathfrak{u}}, \mathbf{c})_j$, then $x'_{ij} = x_{ij}$ for $j \in \mathfrak{u}$, and $x'_{ij} = c$ otherwise

3.3 Measure Class

Definitions of TrueMeasure Concrete Classes

class qmcpy.true_measure.measures.**BrownianMotion** (*dimension*, *time_vector=[array([0.250, 0.500, 0.750, 1.000])]*)

Brownian Motion Measure

__init__ (*dimension*, *time_vector=[array([0.250, 0.500, 0.750, 1.000])]*)

Parameters

- **dimension** (*ndarray*) – dimension’s’ of the integrand’s’
- **time_vector** (*list of ndarrays*) – monitoring times for the Integrand’s’

class qmcpy.true_measure.measures.**Gaussian** (*dimension*, *mean=0*, *variance=1*)

Gaussian (Normal) Measure

__init__(*dimension*, *mean=0*, *variance=1*)

Parameters

- **dimension** (*ndarray*) – dimension's' of the integrand's'
- **mean** (*float*) – mu for Normal(mu,sigma^2)
- **variance** (*float*) – sigma^2 for Normal(mu,sigma^2)

class qmcpy.true_measure.measures.**Lebesgue**(*dimension*, *lower_bound=0.0*, *upper_bound=1*)

Lebesgue Uniform Measure

__init__(*dimension*, *lower_bound=0.0*, *upper_bound=1*)

Parameters **dimension** (*ndarray*) – dimension's' of the integrand's'

class qmcpy.true_measure.measures.**Uniform**(*dimension*, *lower_bound=0.0*, *upper_bound=1.0*)

Uniform Measure

__init__(*dimension*, *lower_bound=0.0*, *upper_bound=1.0*)

Parameters

- **dimension** (*ndarray*) – dimension's' of the integrand's'
- **lower_bound** (*float*) – a for Uniform(a,b)
- **upper_bound** (*float*) – b for Uniform(a,b)

3.4 Discrete Distribution Class

This module implements mutiple subclasses of DiscreteDistribution.

class qmcpy.discrete_distribution.iid_generators.**IIDStdGaussian**(*rng_seed=None*)

Standard Gaussian

__init__(*rng_seed=None*)

Parameters **rng_seed** (*int*) – seed the random number generator for reproducibility

gen_dd_samples(*replications*, *n_samples*, *dimensions*)

Generate r nxd IID Standard Gaussian samples

Parameters

- **replications** (*int*) – Number of nxd matrices to generate (sample.size()[0])
- **n_samples** (*int*) – Number of observations (sample.size()[1])
- **dimensions** (*int*) – Number of dimensions (sample.size()[2])

Returns replications x n_samples x dimensions (numpy array)

class qmcpy.discrete_distribution.iid_generators.**IIDStdUniform**(*rng_seed=None*)

IID Standard Uniform

__init__(*rng_seed=None*)

Parameters **rng_seed** (*int*) – seed the random number generator for reproducibility

gen_dd_samples(*replications*, *n_samples*, *dimensions*)

Generate r nxd IID Standard Uniform samples

Parameters

- **replications** (*int*) – Number of nxd matrices to generate (sample.size()[0])
- **n_samples** (*int*) – Number of observations (sample.size()[1])
- **dimensions** (*int*) – Number of dimensions (sample.size()[2])

Returns replications x n_samples x dimensions (numpy array)

This module implements mutiple subclasses of DiscreteDistribution.

class qmcpy.discrete_distribution.lds_generators.**Lattice**(*rng_seed=None*)
Quasi-Random Lattice low discrepancy sequence (Base 2)

__init__(*rng_seed=None*)

Parameters **rng_seed** (*int*) – seed the random number generator for reproducibility

gen_dd_samples(*replications*, *n_samples*, *dimensions*, *scramble=True*)
Generate r nxd Lattice samples

Parameters

- **replications** (*int*) – Number of nxd matrices to generate (sample.size()[0])
- **n_samples** (*int*) – Number of observations (sample.size()[1])
- **dimensions** (*int*) – Number of dimensions (sample.size()[2])
- **scramble** (*bool*) – If true, random numbers are in unit cube, otherwise they are non-negative integers

Returns replications x n_samples x dimensions (numpy array)

class qmcpy.discrete_distribution.lds_generators.**Sobol**(*rng_seed=None*, *backend='Pytorch'*)
Quasi-Random Sobol low discrepancy sequence (Base 2)

__init__(*rng_seed=None*, *backend='Pytorch'*)

Parameters **rng_seed** (*int*) – seed the random number generator for reproducibility

gen_dd_samples(*replications*, *n_samples*, *dimensions*, *scramble=True*)
Generate r nxd Sobol samples

Parameters

- **replications** (*int*) – Number of nxd matrices to generate (sample.size()[0])
- **n_samples** (*int*) – Number of observations (sample.size()[1])
- **dimensions** (*int*) – Number of dimensions (sample.size()[2])
- **scramble** (*bool*) – If true, random numbers are in unit cube, otherwise they are non-negative integers

Returns replications x n_samples x dimensions (numpy array)

qmcpy.discrete_distribution.lds_generators.**randint**(*low*, *high=None*, *size=None*, *dtype='l'*)
Return random integers from *low* (inclusive) to *high* (exclusive).

Return random integers from the "discrete uniform" distribution of the specified dtype in the "half-open" interval [*low*, *high*). If *high* is None (the default), then results are from [0, *low*).

low [int or array-like of ints] Lowest (signed) integers to be drawn from the distribution (unless `high=None`, in which case this parameter is one above the *highest* such integer).

high [int or array-like of ints, optional] If provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if `high=None`). If array-like, must contain integer values

size [int or tuple of ints, optional] Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.

dtype [dtype, optional] Desired dtype of the result. All dtypes are determined by their name, i.e., 'int64', 'int', etc, so byteorder is not available and a specific precision may have different C types depending on the platform. The default value is 'np.int'.

New in version 1.11.0.

out [int or ndarray of ints] *size*-shaped array of random integers from the appropriate distribution, or a single such random int if *size* not provided.

random.random_integers [similar to *randint*, only for the closed] interval [*low*, *high*], and 1 is the lowest value if *high* is omitted.

```
>>> np.random.randint(2, size=10)
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0]) # random
>>> np.random.randint(1, size=10)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

```
>>> np.random.randint(5, size=(2, 4))
array([[4, 0, 2, 1], # random
       [3, 2, 2, 0]])
```

Generate a 1 x 3 array with 3 different upper bounds

```
>>> np.random.randint(1, [3, 5, 10])
array([2, 2, 9]) # random
```

Generate a 1 by 3 array with 3 different lower bounds

```
>>> np.random.randint([1, 5, 7], 10)
array([9, 8, 7]) # random
```

Generate a 2 by 4 array using broadcasting with dtype of uint8

```
>>> np.random.randint([1, 3, 5, 7], [[10], [20]], dtype=np.uint8)
array([[ 8,  6,  9,  7], # random
       [ 1, 16,  9, 12]], dtype=uint8)
```

3.5 Data Class

Definition of MeanVarData, a concrete implementation of AccumData

class `qmcpy.accum_data.mean_var_data.`**`MeanVarData`**(*levels*, *n_init*)

Accumulated data for IIDDistribution calculations, and store the sample mean and variance of integrand values

`__init__`(*levels*, *n_init*)

Initialize data instance

Parameters

- **levels** (*int*) – number of integrands
- **n_init** (*int*) – initial number of samples

update_data (*integrand*, *true_measure*)
Update data

Parameters

- **integrand** (*Integrand*) – an instance of Integrand
- **true_measure** (*TrueMeasure*) – an instance of TrueMeasure

Returns None

Definition for MeanVarDataRep, a concrete implementation of AccumData

class qmcpy.accum_data.mean_var_data_rep.**MeanVarDataRep** (*levels*, *n_init*, *replications*)
Accumulated data Repeated Central Limit Stopping Criterion (CLTRep) calculations.

__init__ (*levels*, *n_init*, *replications*)
Initialize data instance

Parameters

- **levels** (*int*) – number of integrands
- **n_init** (*int*) – initial number of samples
- **replications** (*int*) – number of random nxm matrices to generate

update_data (*integrand*, *true_measure*)
Update data

Parameters

- **integrand** (*Integrand*) – an instance of Integrand
- **true_measure** (*TrueMeasure*) – an instance of TrueMeasure

Returns None

3.6 Stopping Criterion Class

Definition for CLT, a concrete implementation of StoppingCriterion

class qmcpy.stopping_criterion.clt.**CLT** (*discrete_distrib*, *true_measure*, *inflate=1.2*, *alpha=0.01*, *abs_tol=0.01*, *rel_tol=0*, *n_init=1024*, *n_max=10000000000.0*)
Stopping criterion based on the Central Limit Theorem (CLT)

__init__ (*discrete_distrib*, *true_measure*, *inflate=1.2*, *alpha=0.01*, *abs_tol=0.01*, *rel_tol=0*, *n_init=1024*, *n_max=10000000000.0*)

Parameters

- **discrete_distrib** –
- **true_measure** – an instance of DiscreteDistribution
- **inflate** – inflation factor when estimating variance
- **alpha** – significance level for confidence interval
- **abs_tol** – absolute error tolerance

- **rel_tol** – relative error tolerance
- **n_max** – maximum number of samples

stop_yet()
Determine when to stop

Definition for CLTRep, a concrete implementation of StoppingCriterion

class qmcpy.stopping_criterion.clt_rep.**CLTRep**(*discrete_distrib*, *true_measure*, *replications=16*, *inflate=1.2*, *alpha=0.01*, *abs_tol=0.01*, *rel_tol=0*, *n_init=32*, *n_max=1073741824*)
Stopping criterion based on var(stream_1_estimate, . . . , stream_16_estimate) < errorTol

__init__(*discrete_distrib*, *true_measure*, *replications=16*, *inflate=1.2*, *alpha=0.01*, *abs_tol=0.01*, *rel_tol=0*, *n_init=32*, *n_max=1073741824*)

Parameters

- **discrete_distrib** –
- **true_measure** (*DiscreteDistribution*) – an instance of DiscreteDistribution
- **replications** (*int*) – number of random nxm matrices to generate
- **inflate** (*float*) – inflation factor when estimating variance
- **alpha** (*float*) – significance level for confidence interval
- **abs_tol** (*float*) – absolute error tolerance
- **rel_tol** (*float*) – relative error tolerance
- **n_max** (*int*) – maximum number of samples

stop_yet()
Determine when to stop

3.7 Utilities

Meta-data and public utilities for qmcpy

Exceptions and Warnings thrown by qmcpy

exception qmcpy._util._exceptions_warnings.**DimensionError**
Class for raising error about dimension

exception qmcpy._util._exceptions_warnings.**DistributionCompatibilityError**
Class for raising error about incompatible distribution

exception qmcpy._util._exceptions_warnings.**DistributionGenerationError**
Class for raising error about parameter inputs to gen_dd_samples (method of a DiscreteDistribution)

exception qmcpy._util._exceptions_warnings.**DistributionGenerationWarnings**
Class for issuing warningssabout parameter inputs to gen_dd_samples (method of a DiscreteDistribution)

exception qmcpy._util._exceptions_warnings.**MaxSamplesWarning**
Class for issuing warning about using maximum number of data samples

exception qmcpy._util._exceptions_warnings.**MeasureCompatibilityError**
Class for raising error of incompatible measures

exception qmcpy._util._exceptions_warnings.**NotYetImplemented**
Class for raising error when a component has been implemented yet

exception qmcpy._util._exceptions_warnings.**ParameterError**
Class for raising error about input parameters

exception qmcpy._util._exceptions_warnings.**ParameterWarning**
Class for issuing warnings about unacceptable parameters

exception qmcpy._util._exceptions_warnings.**TransformError**
Class for raising error about transforming function to accommodate distribution

CHAPTER

FOUR

DEMOS

4.1 QMCPy Intro

4.2 Integration Examples

4.3 Sampling Points Visualization

4.4 MC and QMC Comparison

4.5 Quasi-Random Sequence Generators

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

q

INDEX

Symbols

A

B

C

D

G

I

K

L

M

N

P

Q

R

S

T

U