`QMC_sampler` – A Quasi Monte Carlo Sampling Library

Documentation

April 12, 2025

1 Introduction

This document provides an overview of the `QMC_sampler` Python library, which generates Quasi Monte Carlo (QMC) point sets from a high-dimensional generating vector. These points are used in Multilevel Monte Carlo (MLMC) simulations, particularly for solving McKean-Vlasov type Stochastic Differential Equations (SDEs) like the Kuramoto oscillator model.

2 The Kuramoto Oscillator McKean-Vlasov SDE

The Kuramoto model describes a system of coupled oscillators, where each oscillator's phase evolves according to an SDE incorporating mean-field interactions. The McKean-Vlasov formulation extends the classical Kuramoto model by including stochastic perturbations. The governing equation is given by:

$$dX_t^i = \nu^i dt + \frac{K}{P}\sum_{j=1}^{P} \sin(X_t^j - X_t^i)dt + \sigma dW_t^i, \quad i = 1, \ldots, P, \tag{1}$$

where:

- X_t^i is the phase of the i-th oscillator at time t.
- ν^i is the frequency of the i-th oscillator.
- K is the coupling strength between oscillators.
- σ is the noise intensity.
- W_t^i is a Wiener process modeling stochastic fluctuations.

This SDE is simulated numerically using Euler-Maruyama and Richardson extrapolation methods described in later sections.

3 Class QMC_sampler

The QMC_sampler class constructs hierarchical quasi-Monte Carlo (QMC) point sets using lattice rules from a high-dimensional generating vector. It is designed for use in multilevel algorithms that require recursive refinement of both the number of samples and the discretization level in time. The class supports dynamic level construction by mixing previously used vectors with new components from the global generating vector.

3.1 Attributes

- p0 (int): Base-2 logarithm of the number of samples on the coarsest level.
- n0 (int): Base-2 logarithm of the number of time steps on the coarsest level.
- points (list of numpy arrays): Stores generated QMC point sets at each level. Each matrix is of shape $(2^{\texttt{max\_level}} \cdot 2^{\texttt{p0}}, d)$ where d varies with level.
- max_level (int): Current refinement level; incremented with each call to add_level().
- generating_vector (numpy array): High-dimensional vector loaded from file; provides components for constructing level-specific vectors.
- gen_vecs (list of numpy arrays): Stores the generating vectors used to construct the QMC point sets at each level.

3.2 Methods

3.2.1 initialize_from_file(filename)

- Loads the global generating vector from a text file with two columns: a dimension index and its corresponding value.
- Sorts entries by index and stores only the values as the generating_vector.
- Resets max_level to 0 and leaves points empty.
- **Arguments:**
  - filename (str): Path to the generating vector file.

3.2.2 add_level()

- Constructs a new generating vector for the current level using a recursive mixing scheme:
  - On level 0: selects the first $2+2^{\texttt{n0}}$ components of the global generating vector.

  - On higher levels: mixes new components from the global generating vector with the previous level's vector, alternating entries to retain low-discrepancy properties.

- Computes a QMC lattice point set of size $2^{\texttt{max\_level}} \cdot 2^{\texttt{p0}}$ using the most recent vector.
- Stores both the generating vector and the corresponding point set.
- Increments `max_level`.

4 Function Reference

4.1 `bridge_map(point, t, T)`

- Implements Brownian bridge construction.
- Uses midpoint recursion to generate Wiener process paths.
- **Arguments:**
  - `point` (numpy array of shape (M, P)): Input QMC points.
  - `t` (float): Initial time.
  - `T` (float): Final time.

4.2 `pts_to_paths(pts, shifts, T, incr=True)`

- Maps QMC points to Wiener paths and model parameters for Kuramoto oscillators.
- Computes a probability measure transform based on Gaussian quantiles.
- Applies the `bridge_map` function for path generation.
- **Arguments:**
  - `pts` (numpy array of shape (M, d)): Input QMC points.
  - `shifts` (numpy array of shape (m, d)): Shift vectors for each level.
  - `T` (float): Final time for simulation.
  - `incr` (bool, default=True): Whether to compute increments or absolute paths.
- **Returns:**
  - `p_measure_pts` (numpy array of shape (M, d)): Transformed QMC points representing Wiener paths and parameters.

4.3 `euler_maruyama_kuramoto(M_matrix, T, K, sigma, P)`

- Implements the Euler-Maruyama discretization for the McKean-Vlasov Kuramoto model.
- Simulates the evolution of interacting oscillators with stochastic perturbations and mean-field coupling, using consolidated input data in a flattened matrix format.
- **Arguments:**
  - `M_matrix` (numpy array of shape $(M \cdot P, 2 + N)$): Input matrix containing simulation data for M samples, each with P oscillators, over N time steps.
    - Column 0: Initial phase angles (flattened).
    - Column 1: Natural frequencies (flattened).
    - Columns 2 to $N + 1$: Wiener path increments (flattened).
  - `T` (float): Final simulation time.
  - `K` (float): Coupling strength between oscillators.
  - `sigma` (float): Noise intensity of the stochastic perturbation.
  - `P` (int): Number of oscillators per sample.
- **Returns:**
  - `X` (numpy array of shape (M, P)): Final phase angles of the oscillators at time T for all M samples.
- **Notes:**
  - Internally reshapes `M_matrix` to separate initial conditions, natural frequencies, and noise realizations.
  - Computes the mean-field interaction as the average of sine phase differences at each time step.
  - Updates oscillator states using a time-stepping scheme that includes deterministic and stochastic contributions.

4.4 `obj_fctn(x, P)`

- Evaluates the observable function for the output of the Kuramoto simulation.
- Applies a nonlinear transformation (here, the Gaussian function e^{-x^2}) to each oscillator phase, then averages over oscillators within each sample.
- **Arguments:**

  - `x` (numpy array of shape $(M \cdot P,)$ or (M, P)): A flattened or reshaped array containing the final phase values from the simulation, where M is the number of samples and P is the number of oscillators per sample.
  - `P` (int): Number of oscillators per sample.
- **Returns:**
  - `obs` (numpy array of shape $(M,)$): Observable values computed as the mean of e^{-x^2} over the P oscillators, for each of the M samples.
- **Notes:**
  - The input vector is reshaped to shape (M, P) if provided as a flat array.
  - This function is typically used to post-process simulation outputs into scalar values per sample for further statistical evaluation (e.g., mean, variance).
  - The Gaussian transformation e^{-x^2} serves as a smooth observable that decays for large phase values.

4.5 `richardson_euler_maruyama_kuramoto(M_fine, T, K, sigma, P)`

- Computes a Richardson-extrapolated estimate of the observable for the McKean-Vlasov Kuramoto system using the Euler-Maruyama method.
- Applies the Euler-Maruyama method at two resolutions (fine and coarse), evaluates the observable function, and combines the results to improve the accuracy via Richardson extrapolation.
- **Arguments:**
  - `M_fine` (numpy array of shape $(M \cdot P, 2 + N)$): Input matrix for the fine discretization containing initial conditions, natural frequencies, and fine-resolution Wiener increments.
  - `T` (float): Final time of the simulation.
  - `K` (float): Coupling strength in the Kuramoto system.
  - `sigma` (float): Noise intensity of the stochastic perturbation.
  - `P` (int): Number of oscillators per sample.
- **Returns:**
  - `X_rich` (numpy array of shape $(M,)$): Richardson-extrapolated observable values for each of the M samples.
- **Notes:**

  - Internally constructs the coarse-resolution input matrix by subsampling the fine input via the helper function `cut_incr`.
  - The observable is evaluated after each Euler-Maruyama integration using the `obj_fctn` function.
  - Richardson extrapolation is computed as $2 \cdot$ `X_fine` $-$ `X_coarse` to cancel leading-order bias in the time discretization.
  - This method increases accuracy without significantly increasing computational cost.

4.6 `cut_incr(M)`

- Constructs a coarser-resolution input matrix by aggregating adjacent columns of time increments from a fine-resolution matrix.
- Used to reduce the time resolution (e.g., halving the number of time steps) while preserving the same initial conditions and frequencies.
- **Arguments:**
  - `M` (numpy array of shape $(M \cdot P, 2 + N)$): Input matrix with each row representing a single oscillator and containing:
    * Column 0: initial phase angle
    * Column 1: natural frequency
    * Columns 2 onward: fine-resolution Wiener increments (must be even in number)
- **Returns:**
  - `M_coarse` (numpy array of shape $(M \cdot P, 2 + N/2)$): Coarsened matrix with the same first two columns and paired sums of every two adjacent Wiener increments starting from column 2.
- **Notes:**
  - Raises an error if the input does not have at least two columns or if the number of columns from index 2 onward is not even.
  - Ensures compatibility with the Richardson extrapolation routine by aligning coarse and fine time resolutions.

4.7 `cut_paths(M)`

- Removes every other row starting from the third row (i.e., rows with 0-based indices 2, 4, 6, . . .) from the input matrix.
- Typically used to reduce the number of simulation paths or rows in a structured way, preserving even-indexed rows.

- **Arguments:**
  - `M` (numpy array): A 2D input matrix from which rows will be selectively removed. The matrix can have arbitrary shape and content, as long as it has at least 3 rows to apply the row-cutting logic.
- **Returns:**
  - `M_cut` (numpy array): A matrix with the rows at indices 2, 4, 6, ... removed, preserving the structure of the remaining data.

4.8 `cut_pts(M)`

- Splits the input matrix `M` into two submatrices by alternating rows.
- **Arguments:**
  - `M` (numpy array): A 2D input matrix of arbitrary shape. Must contain at least one row.
- **Returns:**
  - `M1` (numpy array): Submatrix consisting of the even-indexed rows of `M` (rows 0, 2, 4, ...).
  - `M2` (numpy array): Submatrix consisting of the odd-indexed rows of `M` (rows 1, 3, 5, ...).