Speedier Simulations with Quasi-Monte Carlo Methods

Final Presentation

Luke Bielawski[1] Brooke Feinberg[2] Aiwen Li[3] Richard Varela[4]

[1]University of Illinois Urbana-Champaign [2]Scripps College [3]University of Pennsylvania
[4]California State University, Sacramento

August 2, 2024

Overview

Overview

Purpose of Quasi-Monte Carlo

Many problems can be written in the form of a multidimensional integral taken over the d-dimensional unit cube, which can be approximated by point evaluations:

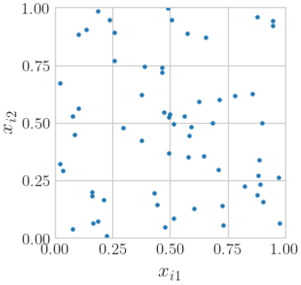
$$\int_{[0,1]^d} f(\mathbf{x})\, d\mathbf{x} \approx \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i) \tag{1}$$

Quasi-Monte Carlo (QMC) methods use point sets $P_n = \{\mathbf{x}_i\}_{i=1}^{n} \subset [0,1]^d$ with ***low discrepancy***, while traditional Monte Carlo methods use IID (independent and identically distributed) points.

Low Discrepancy

Discrepancy = a measure of the difference between the empirical distribution of a set of points and the uniform distribution [5].

- In other words, how "evenly" a set of points fills the domain.
- Lower discrepancy = faster convergence and lower-variance estimates
- **Sobol'** points = a type of LD sequence



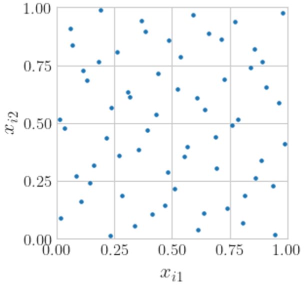


Figure 1: Left: IID, vs. right: low discrepancy (Sobol')

Overview

Kernel Density Estimation

Definition

Kernel Density Estimation is a non-parametric method for estimating probability distributions using kernels as weights

$$\hat{\rho}_h(y) = \frac{1}{n}\sum_{i=1}^{n} k_h(y - f(\boldsymbol{x}_i)) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h} k\left(\frac{y - f(\boldsymbol{x}_i)}{h}\right)$$

KDE's rely on the **placement** and **distribution** of the data points. Thus, using low discrepancy (LD) sequences allows for density estimates with **reduced sampling bias** and **variance**, especially in higher dimensions [5].

Bandwidth Parameter

Bandwidth, denoted h, determines the amount of smoothing applied when estimating $\hat{\rho}(y)$ [6].

- Smaller h = less smoothing; larger h more smoothing.

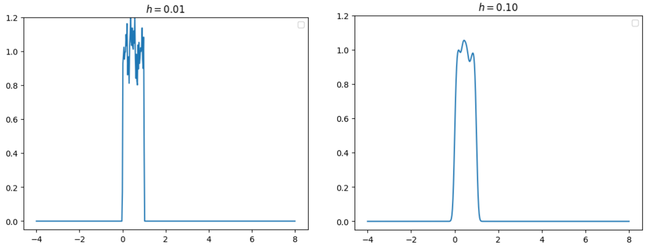


Figure 2: KDE for standard uniform distribution with smaller bandwidth (left) vs. larger bandwidth (right).

Objective of Error Analysis

- Study the effects of various factors on the accuracy of kernel density estimators.
- Factors/parameters include:
  - IID vs. Sobol' (LD) points
  - K = kernel function
  - n = sample size
  - h = bandwidth
  - d = dimension
  - $f(X)$ = function that transforms a set of IID or LD points X into a random variable with a known distribution

Parameters and Other Factors

- **Kernels**: Epanechnikov and Gaussian [5]
- **Bandwidths**: 100 different values in the range (0, 1], defined on a power scale: $\{h_i\} = 2^{\{a_i\}}$ where $\{a_i\}$ is the set of 100 evenly-spaced points between -10 and 0, inclusive.
- **Sample sizes**: $\{2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}\}$
- **Distributions**:
  - **1-dimensional**: standard Uniform, standard Gaussian, exponential (with $\lambda = 1$), Laplace (with $\mu = 0$ and $b = 1$), and chi-squared (with $\mathrm{df} = 1$).
  - **Multi-dimensional**: triangular (sum of 2 standard uniforms), sum of 3 and 5 standard Gaussians, and chi-squared with $\mathrm{df} = 3$ and $\mathrm{df} = 5$ (i.e., sum of 3 and 5 squared Gaussians).

Definition of RMSE

Definition

The **root mean square error (RMSE)** measures the average difference between the expected values and predicted values of a probability density function:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{\rho}(y_i) - \rho(y_i))^2} \tag{2}$$

Finding Minimum RMSE and Optimal Bandwidth

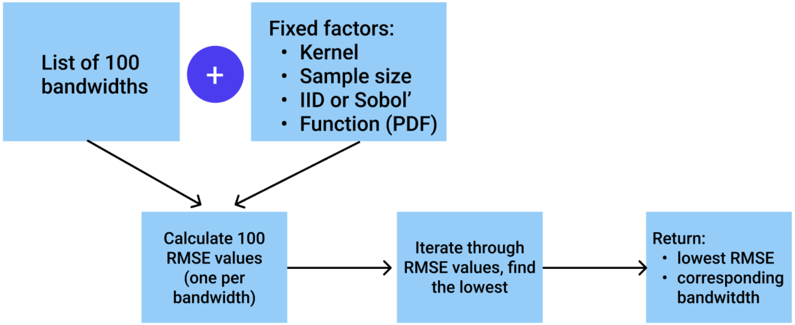

List of 100 bandwidths
+
Fixed factors:
• Kernel
• Sample size
• IID or Sobol'
• Function (PDF)
Calculate 100 RMSE values (one per bandwidth)
Iterate through RMSE values, find the lowest
Return:
• lowest RMSE
• corresponding bandwitdth

Figure 3: Flowchart to find optimal bandwidth using RMSE calculations. This procedure is performed for each set of factors and parameters that we test.

Results: RMSE vs. n

- We plotted four regression models of the form

$$-\log_{10}(RMSE) = b_0 \cdot \log_{10}(n) + \sum_{j=1}^{N} b_j \cdot x_j + \varepsilon_j, \tag{3}$$

where b_0 is the slope of $-\log_{10}(RMSE)$ vs. $\log_{10}(n)$, and x_j are indicator variables to separate the N different distributions we tested.

Table 1: Coefficients b_0 for linear regression models of log-scaled RMSE vs. log-scaled sample size.

	1-dim	**Multi-dim**
IID	0.8379	0.8811
Sobol'	1.0750	0.9465

Results: RMSE vs. n (con't)

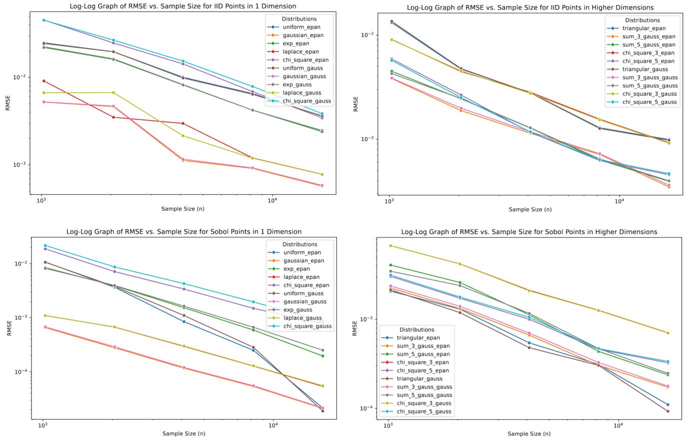


Figure 4: Log-log graphs of RMSE vs. sample size (n) based on Equation (3). For IID points, slope in 1 dim. is 0.8379 (top left) and in higher dims. is 0.8811 (top right). For Sobol' points, slope in 1 dim. is 1.0750 (bottom left) and in higher dims. is 0.9465 (bottom right).

Results: Optimal Bandwidth vs. n

- We plotted four regression models of the form

$$-\log_{10}(h) = b_0 \cdot \log_{10}(n) + \sum_{j=1}^{N} b_j \cdot x_j + \varepsilon_j, \tag{4}$$

where b_0 is the slope of $-\log_{10}(h)$ vs. $\log_{10}(n)$, and x_j are indicator variables to separate the N different distributions we tested.

Table 2: Coefficients b_0 for linear regression models of log-scaled optimal bandwidth vs. log-scaled sample size.

	1-dim	**Multi-dim**
IID	0.2889	0.2747
Sobol'	0.3374	0.2818

Results: Optimal Bandwidth vs. n (con't)

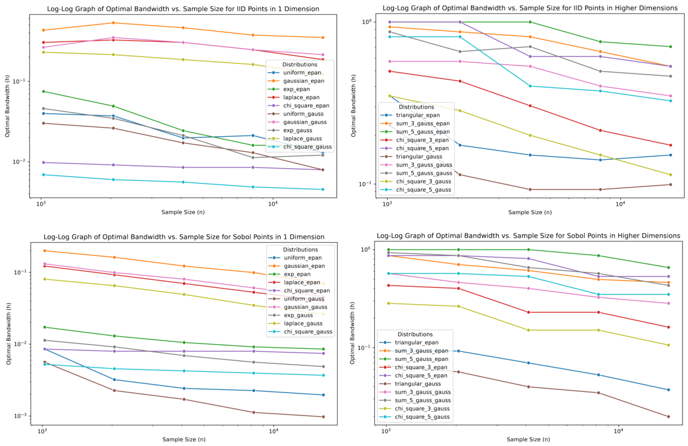
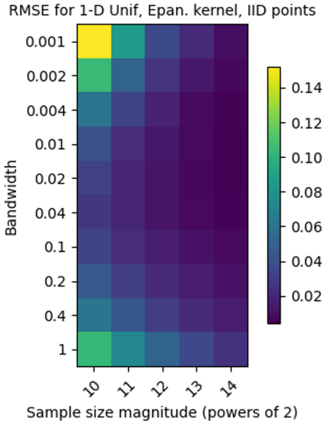


Figure 5: Log-log graphs of optimal h vs. sample size (n) based on Eq. (4). For IID points, slope in 1 dim. is 0.2889 (top left) and in higher dims. is 0.2747 (top right). For Sobol' points, slope in 1 dim. is 0.3374 (bottom left) and in higher dims. is 0.2818 (bottom right).

Results: Optimal Bandwidth vs. n (con't)



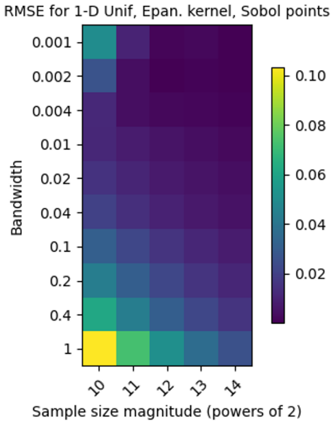


Figure 6: RMSE values (shaded) for different sample sizes and bandwidths when estimating the 1-dimensional standard uniform distribution using the Epanechnikov kernel.

Thoughts and Future Work

- Advantages of using Sobol' (LD) points vs. IID points
- Tailoring bandwidth specifically based on distribution & sample size
- *Future work*: testing different distributions, higher dimensions, different sets of bandwidths; theoretical work to illustrate relationships

Conditional Density Estimation

Let $Y = f(X_1, ..., X_d)$ such as

$$Y = X_1^2(X_2 + 3X_3)$$

If all but one of the variables were fixed, then this would be simple.

Conditional Density Estimation

Let $Y = f(X_1, ..., X_d)$ such as

$$Y = X_1^2(X_2 + 3X_3)$$

If all but one of the variables were fixed, then this would be simple. Conditioning allows us to do this by supposing that some of the information is known. Let $\mathcal{G}_{-k} = (X_1, ..., \hat{X}_k, ..., X_d)$

Conditional Density Estimation

Let $Y = f(X_1, \ldots, X_d)$ such as

$$Y = X_1^2(X_2 + 3X_3)$$

If all but one of the variables were fixed, then this would be simple. Conditioning allows us to do this by supposing that some of the information is known. Let $\mathcal{G}_{-k} = (X_1, \ldots, \hat{X}_k, \ldots, X_d)$
Let $f(X_1, \ldots, X_d \,|\, \mathcal{G}_{-k}) =: g(X_k; \mathcal{G}_{-k})$

$$F(y \,|\, \mathcal{G}_{-k}) = P(Y \leq y \,|\, \mathcal{G}_{-k}) = P(X_k \leq g^{-1}(y) \,|\, \mathcal{G}_{-k}) = F_k(g^{-1}(y))$$

This gives a conditional density $\rho(y \,|\, \mathcal{G}_{-k}) = \frac{d}{dy} F_k(g^{-1}(y))$

With QMC

Theorem

Suppose we have $\rho(y|\mathcal{G}_{-k})$ that satisfies certain technical assumptions. Then

$$\mathrm{E}[\rho(y \mid \mathcal{G}_{-k})]_{\mathcal{G}_{-k}} = \rho(y) \tag{5}$$

with

$$\mathrm{var}[\rho(y \mid \mathcal{G}_{-k})]_{\mathcal{G}_{-k}} < \infty$$

The improvement lies in eliminating one variable's variance.

With QMC

Theorem

Suppose we have $\rho(y|\mathcal{G}_{-k})$ that satisfies certain technical assumptions. Then

$$\mathrm{E}[\rho(y \mid \mathcal{G}_{-k})]_{\mathcal{G}_{-k}} = \rho(y) \tag{5}$$

with

$$\mathrm{var}[\rho(y \mid \mathcal{G}_{-k})]_{\mathcal{G}_{-k}} < \infty$$

The improvement lies in eliminating one variable's variance.
Under slight modification, we can use QMC to help approximate this expectation. The dimension is one less than the dimension of a standard KDE since we only need to realize $d - 1$ variables!

Experimental Methodology

The usual measure of error is the MISE $= \int_a^b \mathrm{E}[(\rho(y) - \hat{\rho}_n(y))^2]\, dy$

Experimental Methodology

The usual measure of error is the MISE $= \int_a^b \mathrm{E}[(\rho(y) - \hat{\rho}_n(y))^2]\, dy$
If we have the true density ρ, then we can approximate the 1-dimensional integral with n_e evenly spaced points: $\widehat{\text{MISE}} = \frac{b-a}{n_e} \sum_{i=1}^{n_e} (\rho(y_i) - \hat{\rho}(y_i))^2$

Experimental Methodology

The usual measure of error is the MISE $= \int_a^b \mathrm{E}[(\rho(y) - \hat{\rho}_n(y))^2]\, dy$
If we have the true density ρ, then we can approximate the 1-dimensional integral with n_e evenly spaced points: $\widehat{\mathrm{MISE}} = \frac{b-a}{n_e} \sum_{i=1}^{n_e} (\rho(y_i) - \hat{\rho}(y_i))^2$
If we don't have the true density, we can generate the conditional density estimator n_r times and use the unbiased estimator [2]

$$\widehat{\mathrm{MISE}} = \frac{b-a}{n_e} \sum_{i=1}^{n_e} \mathrm{var}[\hat{\rho}_n(y_i)]_{n_r}$$

Experimental Methodology

The usual measure of error is the MISE $= \int_a^b \mathrm{E}[(\rho(y) - \hat{\rho}_n(y))^2]\, dy$
If we have the true density ρ, then we can approximate the 1-dimensional integral with n_e evenly spaced points: $\widehat{\text{MISE}} = \frac{b-a}{n_e} \sum_{i=1}^{n_e} (\rho(y_i) - \hat{\rho}(y_i))^2$
If we don't have the true density, we can generate the conditional density estimator n_r times and use the unbiased estimator [2]

$$\widehat{\text{MISE}} = \frac{b-a}{n_e} \sum_{i=1}^{n_e} \text{var}[\hat{\rho}_n(y_i)]_{n_r}$$

Thus, as we vary n, we can get $\widehat{\text{MISE}}(n)$ and fit it to a power law

$$\widehat{\text{MISE}}(n) = Kn^{-\nu}$$

Example 1: Weighted Sum Uniform

$Y = X_1 + X_2 + X_3$ for $X_j \sim \mathcal{U}([0, 2^{j-1}])$

$$\rho(y \mid \mathcal{G}_{-k}) = \begin{cases} 2^{1-k} & y - \sum_{i \neq k} X_i \in [0, 2^{k-1}] \\ 0 & \text{else} \end{cases}$$

The true PDF is piecewise constant, linear, and quadratic.

Example 1: Weighted Sum Uniform

$Y = X_1 + X_2 + X_3$ for $X_j \sim \mathcal{U}([0, 2^{j-1}])$

$$\rho(y \mid \mathcal{G}_{-k}) = \begin{cases} 2^{1-k} & y - \sum_{i \neq k} X_i \in [0, 2^{k-1}] \\ 0 & \text{else} \end{cases}$$

The true PDF is piecewise constant, linear, and quadratic.

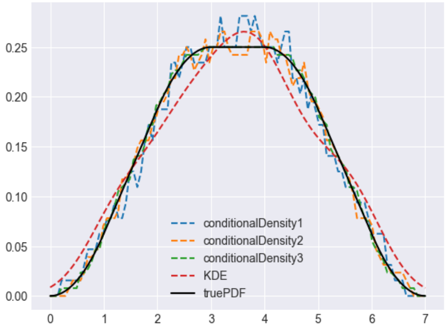


Figure 7: Plotting the conditional densities with Sobol' for $n = 32$ with a KDE and true PDF

Error graph

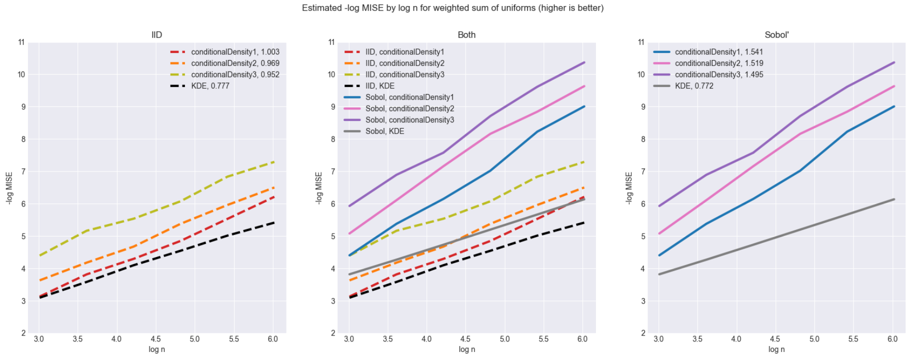


Figure 8: -log, log plot. The slopes of IID hover around 1, KDE around 0.8, and Sobol' around 1.5. Best conditional density is conditionalDensity3.

Example 2: Cantilevered Beam

$Y = \frac{\kappa}{X_1}\sqrt{\frac{X_2^2}{256} + \frac{X_3^2}{16}}$ for $X \sim \mathcal{N}(\mu_j, \sigma_j^2)$ with
$\kappa = 5 \cdot 10^5, \mu = (2.9 \cdot 10^7, 500, 1000)$ and $\sigma = (1.45 \cdot 10^6, 100, 100)$
No true PDF, and the conditional densities have complicated closed-form expressions

Example 2: Cantilevered Beam

$Y = \frac{\kappa}{X_1}\sqrt{\frac{X_2^2}{256} + \frac{X_3^2}{16}}$ for $X \sim \mathcal{N}(\mu_j, \sigma_j^2)$ with
$\kappa = 5 \cdot 10^5, \mu = (2.9 \cdot 10^7, 500, 1000)$ and $\sigma = (1.45 \cdot 10^6, 100, 100)$
No true PDF, and the conditional densities have complicated closed-form expressions

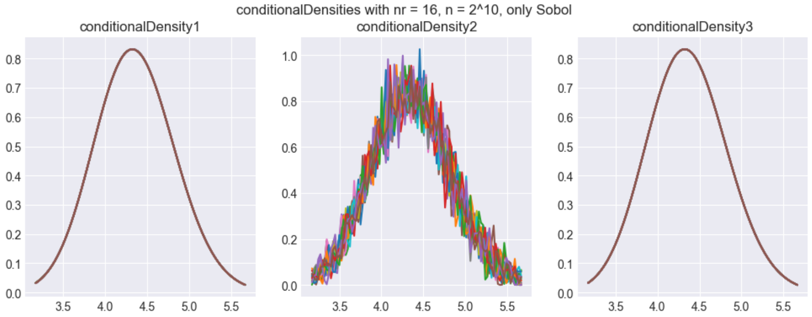


Figure 9: Plotting the conditional densities with Sobol' for $n = 1024$ over $nr = 16$ repetitions

Error graph

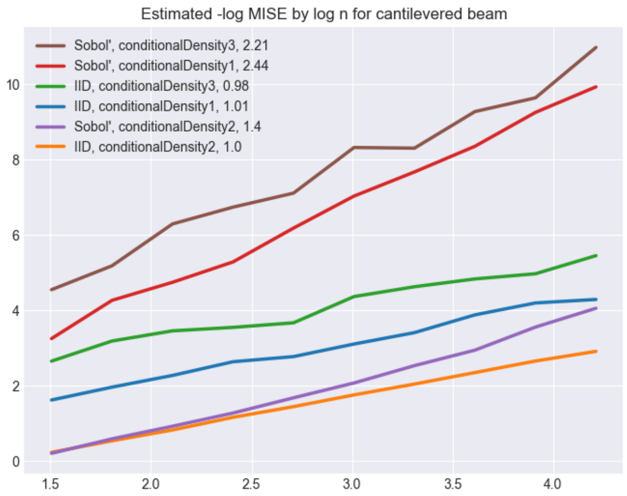


Figure 10: -log, log plot. Slope of IID hover around 1. conditionalDensities vary a lot. Sobol' better than IID

Thoughts

- Conditional density estimation, when applicable, is much less cumbersome and much better than KDE
- Most of these are toy examples; certain real examples (like options pricing) require much more care and attention
- The effect is most noticeable when certain variables contribute to a large portion of the variance
- Interesting work will continue in applying conditional densities estimates to different classes of problems

Overview

Kernel Defined on Combinatorial Objects

- **Combinatorial Objects** refer to structured data that can be represented in terms of discrete structures like graphs, trees, or sets. These objects are more complex than simple numerical data points because they encapsulate relationships and interactions between components

Kernel Defined on Combinatorial Objects

- **Combinatorial Objects** refer to structured data that can be represented in terms of discrete structures like graphs, trees, or sets. These objects are more complex than simple numerical data points because they encapsulate relationships and interactions between components
- The **Kernel Trick** is a powerful technique for modeling non-linear relationships using linear learning algorithms by implicitly mapping data into a higher-dimensional feature space using a kernel function.

Kernel Defined on Combinatorial Objects

- **Combinatorial Objects** refer to structured data that can be represented in terms of discrete structures like graphs, trees, or sets. These objects are more complex than simple numerical data points because they encapsulate relationships and interactions between components
- The **Kernel Trick** is a powerful technique for modeling non-linear relationships using linear learning algorithms by implicitly mapping data into a higher-dimensional feature space using a kernel function.
- Why are Kernels Defined on Discrete input Spaces Useful?
  - Various real-world applications: bio-informatics, community detection, manifold learning for deep generative modeling, etc... [3]

Applying Kernels to Combinatorial Objects

- **Kernel Function**: For discrete objects, like graphs, we define a *graph kernel* (or kernel function) as $K : \mathcal{X} x \mathcal{X} \rightarrow \mathbb{R}$ which returns a real number representing the similarity measure between these structures

Applying Kernels to Combinatorial Objects

- **Kernel Function**: For discrete objects, like graphs, we define a *graph kernel* (or kernel function) as $K : \mathcal{X} x \mathcal{X} \rightarrow \mathbb{R}$ which returns a real number representing the similarity measure between these structures
- **Mapping to a Higher-Dimensional Space**: Kernel function implicitly maps combinatorial object to a higher-dimension feature space where linear relationships are used to analyze objects

Applying Kernels to Combinatorial Objects

- **Kernel Function**: For discrete objects, like graphs, we define a *graph kernel* (or kernel function) as $K : \mathcal{X} x \mathcal{X} \to \mathbb{R}$ which returns a real number representing the similarity measure between these structures
- **Mapping to a Higher-Dimensional Space**: Kernel function implicitly maps combinatorial object to a higher-dimension feature space where linear relationships are used to analyze objects
- **Similarity Measure**: Kernel function replaces the Euclidean dot product $x_i^T x_j$ with $K(x_i, x_j)$ (the similarity measure)

Applying Kernels to Combinatorial Objects

- **Kernel Function**: For discrete objects, like graphs, we define a *graph kernel* (or kernel function) as $K : \mathcal{X} x \mathcal{X} \rightarrow \mathbb{R}$ which returns a real number representing the similarity measure between these structures
- **Mapping to a Higher-Dimensional Space**: Kernel function implicitly maps combinatorial object to a higher-dimension feature space where linear relationships are used to analyze objects
- **Similarity Measure**: Kernel function replaces the Euclidean dot product $x_i^T x_j$ with $K(x_i, x_j)$ (the similarity measure)
- **Gram (Kernel) Matrix**: Kernel evaluations are stored in a Gram Matrix $\mathbf{K} \in \mathbb{R}^{NxN}$ whose entries enumerate the pairwise kernel evaluations $\mathbf{K} := [K(x_i, x_j)]_{i,j=1}^{N}$
  - Entries of this matrix represent similarities between each pair of data points

The Problem

- **The Problem**: The time-complexity of constructing and inverting the Gram Matrix $\mathbf{K}$ becomes $\mathcal{O}(N^3)$. This is extremely expensive and inefficient for large datasets

The Problem

- **The Problem**: The time-complexity of constructing and inverting the Gram Matrix $\mathbf{K}$ becomes $\mathcal{O}(N^3)$. This is extremely expensive and inefficient for large datasets
- How Do We efficiently approximate $\mathbf{K}$?
    - We employ **Random Features**: a Monte-Carlo approach that allows one to construct a low-rank decomposition of $\mathbf{K}$, improving scalability

The Problem

- **The Problem**: The time-complexity of constructing and inverting the Gram Matrix $\mathbf{K}$ becomes $\mathcal{O}(N^3)$. This is extremely expensive and inefficient for large datasets
- How Do We efficiently approximate $\mathbf{K}$?
  - We employ **Random Features**: a Monte-Carlo approach that allows one to construct a low-rank decomposition of $\mathbf{K}$, improving scalability
    - **Randomised Function**: $\phi : \mathbb{R}^d \to \mathbb{R}^s$ maps the original data points into a lower dimensional space
    - **Low-Dimensional Feature Vector**: whose dot product approximates the kernel function: $\mathbf{K}_{ij} = \mathbb{E}[\phi(x_i)^\intercal \phi(x_j)]$ meaning the dot product of the transformed vectors is an unbiased estimator of the kernel function

Graph Random Features

- A recently viable **Graph Random Features** (GRFs) mechanism employs an ensemble of random walkers that deposit a "load" at every vertex they pass, depending on
  - i) the product of weights of edges traversed by the walker, and
  - ii) the marginal probability of the subwalk [3]

Graph Random Features

- A recently viable **Graph Random Features** (GRFs) mechanism employs an ensemble of random walkers that deposit a "load" at every vertex they pass, depending on
  - i) the product of weights of edges traversed by the walker, and
  - ii) the marginal probability of the subwalk [3]
- The algorithm constructs random features $\phi(i)_{i=1}^{N} \subset \mathbb{R}^N$ that $\phi_i^\top \phi(j)$ give unbiased approximation of the *ij*-th matrix element of a 2-regularised Laplacian kernel

Graph Random Features

- A recently viable **Graph Random Features** (GRFs) mechanism employs an ensemble of random walkers that deposit a "load" at every vertex they pass, depending on
  - i) the product of weights of edges traversed by the walker, and
  - ii) the marginal probability of the subwalk [3]
- The algorithm constructs random features $\phi(i)_{i=1}^{N} \subset \mathbb{R}^N$ that $\phi_i^\top \phi(j)$ give unbiased approximation of the *ij*-th matrix element of a 2-regularised Laplacian kernel
- **Key Limitation**: Only addresses a niche family of graph kernels

General-Graph Random Features

- **General-Graph Random Features** (g-GRFs) generalizes the algorithm to arbitrary functions of a weighted adjacency matrix, allowing for an efficient and accurate estimation of a much broader class of kernel

General-Graph Random Features

- **General-Graph Random Features** (g-GRFs) generalizes the algorithm to arbitrary functions of a weighted adjacency matrix, allowing for an efficient and accurate estimation of a much broader class of kernel [3]
- The key contribution to g-GRFs is the **Modulation Function**, f, that controls the weight of the load deposited by random walkers as they traverse the graph based on the length of the walk

General-Graph Random Features

- **General-Graph Random Features** (g-GRFs) generalizes the algorithm to arbitrary functions of a weighted adjacency matrix, allowing for an efficient and accurate estimation of a much broader class of kernel [3]
- The key contribution to g-GRFs is the **Modulation Function**, f, that controls the weight of the load deposited by random walkers as they traverse the graph based on the length of the walk
- By parameterizing this modulation function on a **Neural Network**, the algorithm provides an efficient, unbiased approximation of the desired graph kernel

Quasi-Monte Carlo Graph Random Features

Can we improve the accuracy and efficiency of g-GRFs?

- **Quasi-Monte Carlo Graph Random Features** (q-GRFs) introduces correlated ensembles, or antithetic walkers, to correlate the lengths of random walks with some stopping criterion [4]

Quasi-Monte Carlo Graph Random Features

Can we improve the accuracy and efficiency of g-GRFs?

- **Quasi-Monte Carlo Graph Random Features** (q-GRFs) introduces correlated ensembles, or antithetic walkers, to correlate the lengths of random walks with some stopping criterion [4]
  - In the IID implementation (g-GRFs) two termination random variables (TRVs) are sampled independently from from a uniform distribution $t_{1,2}$ $Unif(0,1)$. Each walker terminates if their respective TRV $t_{1,2} < p$
  - **Antithetic Walkers**: A pair of walkers are antithetic if their TRVs are marginally distributed as $t_{1,2} \sim \mathcal{U}(0,1)$ but are offset by $\frac{1}{2}$,

$$t_2 = \text{mod}_1\left(t_1 + \frac{1}{2}\right), \tag{6}$$

Quasi-Monte Carlo Graph Random Features

Can we improve the accuracy and efficiency of g-GRFs?

- **Quasi-Monte Carlo Graph Random Features** (q-GRFs) introduces correlated ensembles, or antithetic walkers, to correlate the lengths of random walks with some stopping criterion [4]
  - In the IID implementation (g-GRFs) two termination random variables (TRVs) are sampled independently from from a uniform distribution $t_{1,2}$ $\mathit{Unif}(0,1)$. Each walker terminates if their respective TRV $t_{1,2} < p$
  - **Antithetic Walkers**: A pair of walkers are antithetic if their TRVs are marginally distributed as $t_{1,2} \sim \mathcal{U}(0,1)$ but are offset by $\frac{1}{2}$,

$$t_2 = \mathrm{mod}_1 \left(t_1 + \frac{1}{2} \right), \tag{6}$$

  - Diversifying the lengths of random walks prevents clustering; therefore, Choromanski, Reid, and Weller assert that antithetic walkers (q-GRFs) yield lower variance estimators of the 2-regularised Laplacian kernel

Our Investigation

- **Motivating Question**: Can we identify an alternative kernel function or graph type where q-GRFs achieve a lower variance estimator like the previously introduced 2-Regularised Laplacian kernel?
    - Diffusion (or Heat), Matérn, Inverse Cosine Kernels
    - Erdős-Rényi and Barabási-Albert random graph models, Binary Trees, and Ladder graphs

Diffusion (or Heat) Kernel

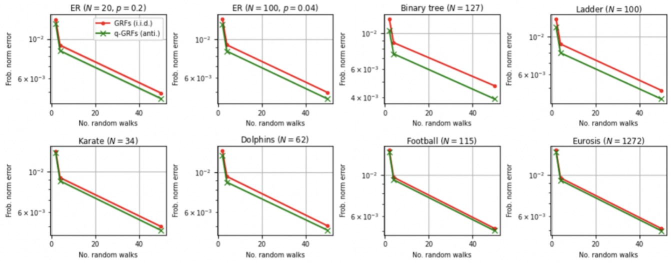
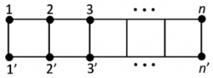


Figure 11: Relative Frobenius norm error of estimators of the diffusion (or heat) kernel with $t = 0.5$ using g-GRFs (red circles) and q-GRFs (green crosses). **Lower is better**. N gives the number of nodes and p is the edge-generation probability for the Erdös-Rényi graphs. One standard deviation is shaded.

Ladder Graphs



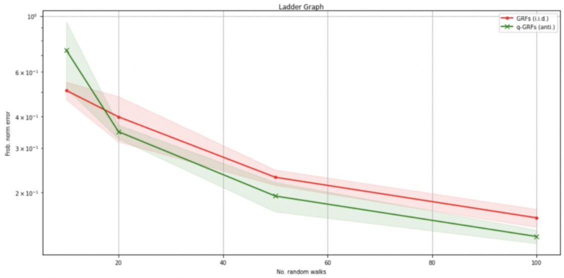


Figure 12: LEFT: Standard ladder graph. RIGHT: Ladder graph with number of random walks set to (10, 20, 50, 100) and number of rungs set to 9. Q-GRFs yield lower variance estimators of the diffusion kernel than general g-GRFs.

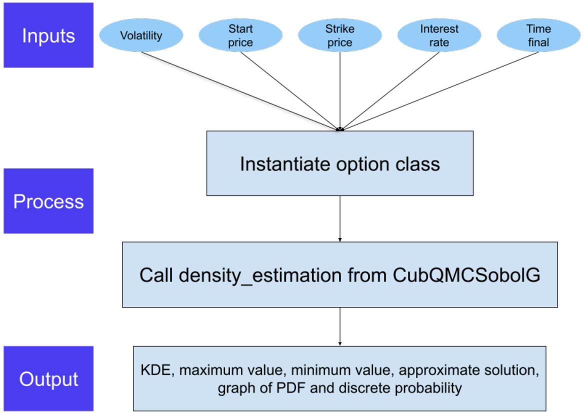
Results and Future Work

- Despite the inherent randomness in the antithetic procedure, there is strong evidence to suggest that q-GRFs have the potential to yield lower variance estimators of the diffusion kernel – though further experimentation and theoretical work is necessary to make final conjectures
- We assert that the **number of rungs** on a ladder graph impacts q-GRFs ability to achieve a lower variance estimator of the diffusion kernel
- Further theoretical results investigating these phenomena are forthcoming

QMC Application for Finance Options

- Use QMC to estimate the PDF of the payoff of an option.
- QMC allows faster computation than regular MC.
- QMCPy allows someone to compute such estimations.

QMC Application for Finance Options



Inputs
Volatility
Start price
Strike price
Interest rate
Time final
Process
Instantiate option class
Call density_estimation from CubQMCSobolG
Output
KDE, maximum value, minimum value, approximate solution, graph of PDF and discrete probability

Figure 13: Flowchart of using an option class.

European Density Estimation

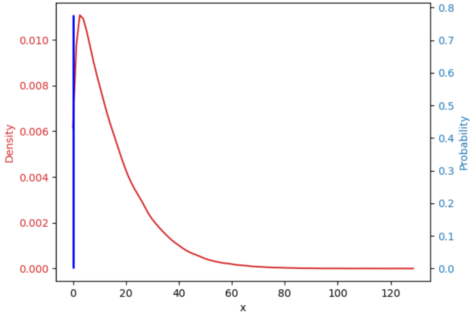


Figure 14: PDF of the payoff for a European option with starting price of \$100 and strike price of \$120, with interest rate of %0.5, and volatility of 0.2.

Pull Request

- Finish code that was in development by a graduate student.
- Review code before merging into the main development branch.
- Satisfy requested changes to the code before finally merging.
- Begin merging said code into the main development.

Refactoring

- **Issue:** Code among the option classes have the same code.
- **Solution:** Refactored all option classes to remove duplicate code.

Refactoring

What does refactoring mean?

Definition

Refactor: a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.[1]

Refactoring

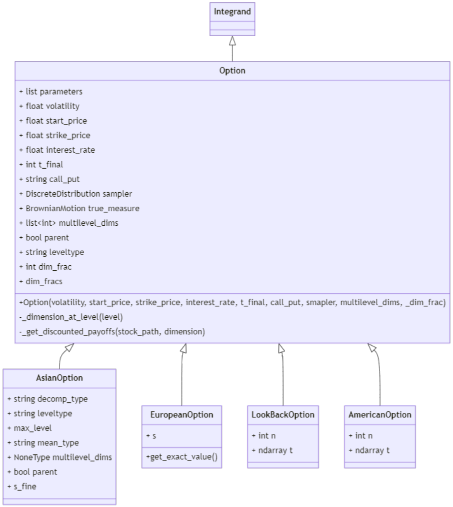


Figure 15: UML Class diagram of all option classes.

Conclusion

Important things learned and encountered when developing robust scientific software:

- Use software engineering principles.
- Structure your code and write everything to be clear.
- Use the documentation of the language and libraries to help build the software.

Overview

Conclusion and Final Thoughts

- Low discrepancy points can provide much lower error than IID uniform points in a variety of situations:
  - Density estimations (using kernels & conditioning)
  - Analyzing combinatoral objects
  - Modeling financial options
- Theoretical analysis, further applications, and robust implementation are key to maximizing the potential of low discrepancy points

Thank you!

References I

M. Fowler.
Refactoring.

P. L'Ecuyer, F. Puchhammer, and A. Ben Abdellah.
Monte carlo and quasi–monte carlo density estimation via conditioning.
34(3):1729–1748.
Publisher: INFORMS.

I. Reid, K. Choromanski, E. Berger, and A. Weller.
General graph random features.
(arXiv:2310.04859).

I. Reid, K. Choromanski, and A. Weller.
Quasi-monte carlo graph random features.
(arXiv:2305.12470).

References II

Y. Soh, Y. Hae, A. Mehmood, R. Hadi Ashraf, and I. Kim.
Performance evaluation of various functions for kernel density estimation.
03(1):58–64.

W. Zucchini.
APPLIED SMOOTHING TECHNIQUES part 1: Kernel density estimation.
2003.

Appendix A: Links

- KDE Error Analysis: Final report and Python notebook
- https://qmcpy.org/
- https://github.com/QMCSoftware/QMCSoftware
- Conditional QMC: Report and Python notebooks 1 and 2
- QMC with combinatorial objects: Final report and Python notebook

Appendix B: Epanechnikov vs. Gaussian Kernel

Definition

The **Gaussian** kernel is defined with the weighting function

$$w(y,h) = \frac{1}{2\pi h} e^{-\frac{y^2}{2h^2}}, \qquad -\infty < y < \infty \tag{7}$$

Definition

The **Epanechnikov** kernel is defined with the weighting function

$$w(y,h) = \begin{cases} \frac{3}{4h}(1 - \frac{y^2}{5h^2})/\sqrt{5} & \text{for } |y| < \sqrt{h}, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

- The above weighting functions, $w(y,h)$, are all of the form $w(y,h) = \frac{1}{h}K(\frac{y}{h})$, where K is the *kernel* function [6].

Appendix B: Epanechnikov vs. Gaussian Kernel (con't)

The Epanechnikov kernel has an efficiency of 100%, while the Gaussian kernel has an efficiency of 95.12%, where the *efficiency* of a kernel is defined as:

$$\text{Efficiency}(K) = \left(\frac{RMSE_{opt}(\hat{\rho}) \text{ using } K_{EP}}{RMSE_{opt}(\hat{\rho}) \text{ using } K} \right)^{\frac{5}{4}} \tag{9}$$

This means that the RMSE obtained using an Epanechnikov kernel with $n \approx 95$ is approximately equal to the RMSE obtained using a Gaussian kernel with $n = 100$ [5].

Appendix B: Epanechnikov vs. Gaussian Kernel (con't)

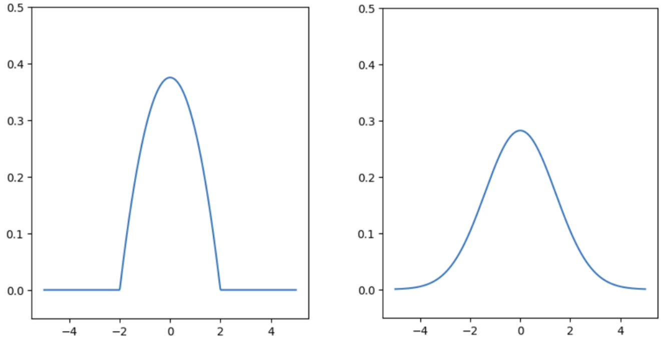


Figure 16: Shape of the Epanechnikov (left) vs. the Gaussian kernel (right). The kernel determines the shape of the weighting function.