QMC integration of non-smooth functions: application to pricing exotic options

Konstantin Medyanikov

February 16, 2024

Abstract

Focus of this report consist in investigation of different approaches, used to numerically approximate multiple integrals of the form 1, and detect one with highest convergence rate :

$$\int_{\mathbb{R}^m} f(x)\rho(x)\mathrm{d}x = \int_{-\infty}^{\infty} \cdots \int_{\infty}^{\infty} f\left(x_1, \ldots, x_m\right) \rho_1\left(x_1\right) \ldots \rho_m\left(x_m\right) \mathrm{d}x_1 \ldots \mathrm{d}x_m. \tag{1}$$

Where, $\rho_i(x_i)$- probability distribution function of variable X_i, and $m \gg 1$.
In this work, we consider special case functions $f(x)$, with discontinuities in their derivatives, due to which Koksma-Hlawka inequality fails in quasi Monte-Carlo settings.
We will introduce new technique called "Pre-Integration", which use QMC in order to numerically estimate integral 1. Next step would be to compare proposed algorithm to standard CMQ and QMC methods in terms of error with respect to number of samples- $N = 2^7, \ldots, 2^{13}$.
Since such type of function often appears in finance, we focus our self on two special cases of optional pricing payoffs: Ψ_1 - Asian call option and Ψ_2- binary digital Asian option.

0.1 introduction

In order to approximate numerically integral of the form:

$$\int_{[0,1]^m} f(x)\mathrm{d}x = \int_0^1 \cdots \int_0^1 f\left(x_1, \ldots, x_m\right) \mathrm{d}x_1 \ldots \mathrm{d}x_m \tag{2}$$

One can apply quasi-Monte Carlo (QMC), with set of points: $P_N = \{x_1, \ldots, x_N\}$, $x_i \in [0,1]^m$. And approximate $\int_{[0,1]^m} f(x)\mathrm{d}x$ by qmc estimator $\mu_{qmc} = \frac{1}{N}\sum_{i=1}^N f\left(x_i\right)$.
QMC became quite popular approach, especially due to high convergence rate, obtained by famous Koksma-Hlawka inequality where error bound as follow:

$$\left|\int_{[0,1]^m} f(x)\mathrm{d}x - \frac{1}{N}\sum_{i=1}^N f\left(x_i\right)\right| \leq \left(\sum_{\mathbf{k}\subset\{1,\ldots,s\}} \int_{[0,1]^{|\mathbf{k}|}} \left|\frac{\partial^{\mathbf{k}|}}{\partial x_{\mathbf{k}}} f\left(x_{\mathbf{k}}, \mathbf{1}\right)\right| \mathrm{d}x_{\mathbf{k}}\right) \mathcal{D}_N^*\left(P_N\right),$$

$x_{\mathbf{k}} = \left(x_{k_1}, x_{k_2}, \ldots, x_{k_p}\right)$, if $|\mathbf{k}| = p \leq s$. This inequality shows that error is propositional to $\mathcal{D}_N^*\left(P_N\right)$ also called star-discrepancy. In the case of well chosen point set P_N, star-discrepancy has convergence rate of $O(\frac{\log(N)^{m-1}}{N})$.

For function-f having "kinks" (discontinuities in the gradients) and "jumps" (discontinuities in the function), Koksma-Hlawka inequality fails, since it requires integrable first order mixed derivatives. And presence of "kinks" often leads to unbounded total variation norm in the sense of Hardy and Krause. One one strategy would be just to ignore the presence of "kinks" and "jumps" and apply Crude Monte Carlo (CMC) or QMC. Another way, that will be presented in following chapter,consist in the integration among one of the dimension x_i, that will result in obtaining smooth function without any "kinks" and "jumps", that depends on x_{-i}. Finally, we apply classical approaches over resting $[0,1]^{m-1}$ space .
We will be particularly interested in studding functions $f : \mathbb{R}^m \mapsto \mathbb{R}$ of the form:

$$f(x) = \theta(x)\mathbb{I}_{\{\phi(x)\}}$$

Often such functions have "kinks" around manifold given by x —. $\phi(x) = 0$.
That type of function often appears in finance settings,which will be discussed in chapter 3 0.3.
One can notice, that f is not only defined in unit cube but over R^m, then another point that will be addressed, is how we can generalise QMC approach for bigger class of random variables not only for $X\ uniform([0,1]^m)$. And apply QMC on:

$$\int_{\mathbb{R}^m} f(x)\rho(x)\mathrm{d}x$$

Where $\rho(x)$- probability distribution function.

0.2 Preintegration Trick

The problem is to approximate:

$$\int_{\mathbb{R}^m} f(x)\rho(x)\mathrm{d}x = \int_{-\infty}^{\infty} \cdots \int_{\infty}^{\infty} f\left(x_1, \ldots, x_m\right) \rho_1\left(x_1\right) \ldots \rho_m\left(x_m\right) \mathrm{d}x_1 \ldots \mathrm{d}x_m.$$

Where integration occurs over $\mathbb{R}^m$ with respect to some distribution $\rho := \rho_1\left(x_1\right)\rho_2\left(x_2\right)\ldots\rho_m\left(x_m\right)$. And we assume f be of form:

$$f(x) = \theta(x)\mathbb{I}_{\{\phi(x)\}},$$

The key, of the approach would be in assumption that, $\exists j \in \{1, \ldots, m\}$ such that:

$$\frac{\partial\phi}{\partial x_j}(x) > 0 \tag{3}$$

and:

$$\phi(x) \to \pm\infty \textbf{ as } x_j \to \pm\infty \tag{4}$$

Then, we can rewrite integral as repeated integral using Fubini theorem:

$$\int_{\mathbb{R}^m} f(x)\rho(x)\mathrm{d}x = \int_{\mathbb{R}^{m-1}} \underbrace{\left(\int_{\mathbb{R}} f\left(x_j, x_{-j}\right)\rho_j\left(x_j\right)\mathrm{d}x_j\right)}_{:=p(x_{-j})} \rho_{-j}\left(x_{-j}\right)\mathrm{d}x_{-j}.$$

We first will integrate the inner integral among x_j, while fixing $x_{-j} = (x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_m)$. Which is considered as "preintegration" step. Then under assumptions 3 and 4 the resulting function p depending on $m-1$ variables x_{-j}, doesn't have any "kinks" and often deferential:

$$p(x_{-j}) := \int_{\mathbb{R}} f\left(x_j, x_{-j}\right) \rho_j\left(x_j\right) \mathrm{d}x_j$$

In some cases, we hope to get closed form solution for $p(x_j)$. But we can always apply numeric methods: simply use CMC/QMC or one can also notice that f has "jump" around $\phi(x) = 0$,where under our conditions 3, 4, $\phi(x)$ is monotone (non-decreasing) function with respect to x_j, then, by Implicit function theorem, for each x_{-j}, there exists a unique value $x_j^* := \psi\left(x_{-j}\right)$ for which $\phi\left(x_j, x_{-j}\right) < 0$ if $x_j < x_j^*$ and $\phi\left(x_j, x_{-j}\right) > 0$ if $x_j > x_j^*$. Which let us to rewrite:

$$p\left(x_{-j}\right) := \int_{\mathbb{R}} f\left(x_j, x_{-j}\right) \rho_j\left(x_j\right) \mathrm{d}x_j = \int_{\psi\left(x_{-j}\right)}^{+\infty} \theta\left(x_j, x_{-j}\right) \rho_j\left(x_j\right) \mathrm{d}x_j,$$

Then semi-infinite integral can be calculated by any 1-dimensional integral methods, for example: Gaussian quadrature. Our algorithm can be summarize as follow:

- Consider x_j respecting assumption
- Generate N samples of $x_{-j}^{(i)}$ by QMC or CMC
- For each $x_{-j}^{(i)}$ calculate $p(x_{-j}^{(i)})$ by any of approach mentioned above
- Finally approximate $\int_{\mathbb{R}^m} f(x)\rho(x)\mathrm{d}x$ by $\frac{1}{N}\sum_{i=1}^{N} p(x_{-j}^{(i)})$

0.2.1 Quasi-MC, for normal random variables

In this subsection, we introduce the approach to apply QMC, to approximate:

$$\int_{\mathbb{R}^m} f(x)\rho(x)\mathrm{d}x$$

with, $\rho(x) = \rho_1\left(x_1\right) \ldots \rho_m\left(x_m\right)$, where $\rho_i(x_i)$-pdf of normal distribution, $X_i \sim \mathcal{N}(0,\,1)$ and X_i are independent.
We can consider equivalent integral over $[0,1]^m$:

$$\int_{\mathbb{R}^m} f\left(x_1, \ldots, x_m\right) \rho(x)\mathrm{d}x = \int_{[0,1]^m} f\left(\Phi(t_1)^{-1}, \ldots, \Phi(t_m)^{-1}\right) \mathrm{d}t$$

Then, we simply generate low-discrepancy points set P_N for $[0,1]^m$, for example by Sobol points set.

0.3 Application to option pricing

We are going to consider problem of optional pricing. where the the stock price S_t at time t given by solution of following SDE:

$$dS = rSdt + \sigma Sdw_t, \quad S(0) = S_0.$$

S_0 represents the initial value of the underlying asset, K is the so-called strike price, r is the interest rate, σ is the volatility and w_t denotes a standard one-dimensional Wiener process. One can show that $S_t = S_0 \exp\left(\left(r - \sigma^2/2\right)t + \sigma t w_t\right)$.For $m \in \mathbb{N}$, let $t_i = i\Delta t$ with $\Delta t = T/m$ denote the discrete observation times.In particular, we will be interested in pricing Asian options with the following payoffs:

$$\Psi_1\left(w_{t_1}, \ldots, w_{t_m}\right) := \left(\frac{1}{m}\sum_{i=1}^{m} S_{t_i}\left(w_{t_i}\right) - K\right) \mathbb{I}_{\left\{\frac{1}{m}\sum_{i=1}^{m} S_{t_i}\left(w_{t_i}\right) - K\right\}}$$

$$\Psi_2\left(w_{t_1}, \ldots, w_{t_m}\right) := \mathbb{I}_{\left\{\frac{1}{m}\sum_{i=1}^{m} S_{t_i}\left(w_{t_i}\right) - K\right\}}.$$

Ψ_1 corresponds to the payoff of an Asian call option and Ψ_2 corresponds to the payoff of a binary digital Asian option. Notice that the value $V_i, i = 1, 2$, of such options is given by

$$V_i := e^{-rT}\mathbb{E}\left[\Psi_i\left(w_{t_1}, \ldots, w_{t_m}\right)\right] = \frac{e^{-rT}}{(2\pi)^{m/2}\sqrt{\det(C)}} \int_{\mathbb{R}^m} \Psi_i\left(w_{t_1}, \ldots, w_{t_m}\right) e^{-\frac{1}{2}w^T C^{-1} w}\ \mathrm{d}w$$

In following four sub-chapters, each chapter corresponds to one of the method, we are going implement and compare algorithms in terms of convergence rate and time cost.
In order to evaluate convergence rate, we look how error depends on number of samples N:

$$error = |V_{actual} - V_{approximated}| \approx N^{\alpha}$$

Take logarithm in both sides:

$$\log(error) \approx \alpha \log(N)$$

We fit linear regression in order to detect α, then convergence of algorithm: $0(N^{\alpha})$.
Fixing parameters: $K = 100, S_0 = 100, r = 0.1, \sigma = 0.1, T = 1$, and discretization parameter $m = 32$

0.3.1 Standard Monte Carlo

We are going to apply CMC algorithm in order to estimate V_i, that is simply consist in simulation of N independent replicas 1 $S^i = (S^i_{t_j})^N_{i=1}$, $t_j = j\frac{T}{m}$ Then:

$$V_k = \frac{1}{N}\sum_{i=1}^{N} \psi_k(S^i_1, \ldots, S^i_m)$$

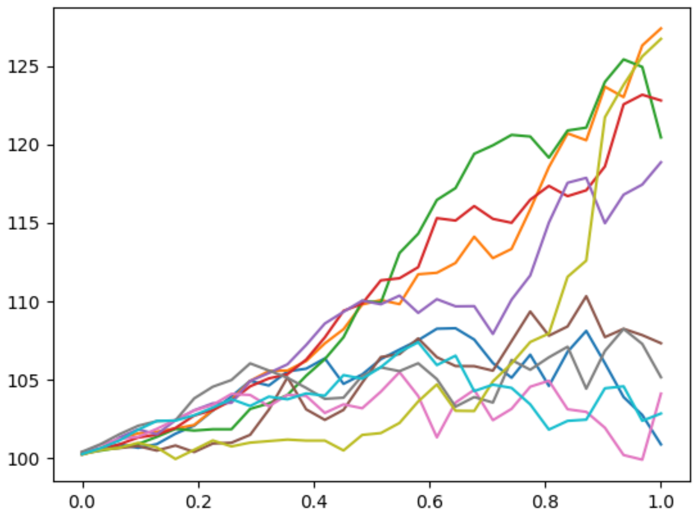

Figure 1: Simulation of Stock price S_{t_i}

(code for simulation: 0.6.1)
In order to calculate error, we use asymptotic result (central limit theorem):

$$|error| < 2C_{1-\frac{\lambda}{2}} \frac{std_{approximated}}{\sqrt{N}}$$

On the left figure 2 we get convergence rate for Φ_2 and on the right for Φ_1. Where stars corresponds to actual measurements, we do several measurements for each $N = 2^i$ and then we fit regression line.
(code for CMC convergence rate and time estimation 0.6.2)
Finally, we obtain:

- convergence rate of $V_1 \approx O(N^{-0.5})$
- convergence rate of $V_2 \approx O(N^{-0.5})$
- 0.07 sec. per 1000 samples

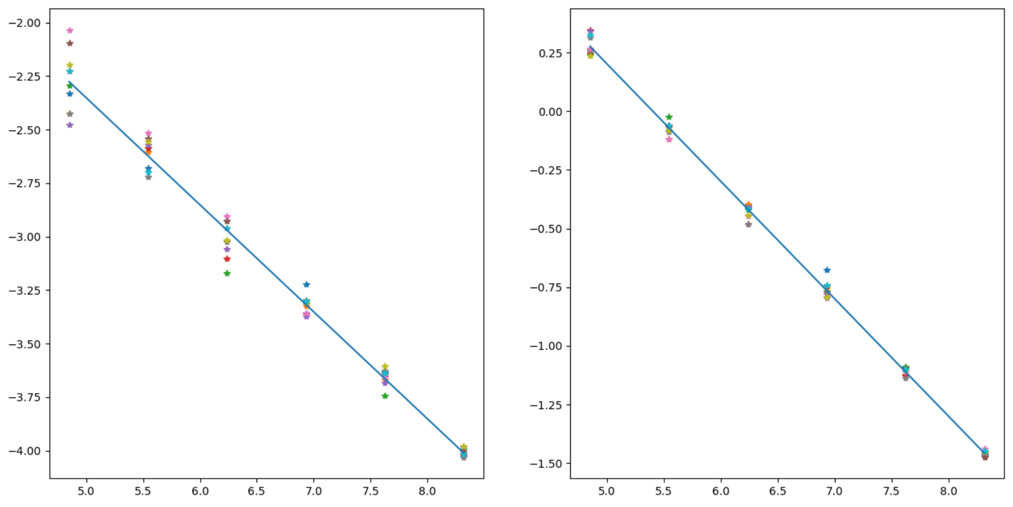

Figure 2: Each point correspond to $(\log error(N_i), \log N_i)$ produced by CMC. Where on the left we have payoff Φ_2, on the right Φ_1. With fitted regression line for 10 independent tries

0.3.2 Quasi-Monte Carlo

In order to apply the trick we have seen in section: "Quasi-MC, for normal random variables" 0.2.1. We first have to make appears $X = (X_1, \dots, X_m)$ independent normal RVs.
Actually, one can think of Brownian motion as Gaussian process with mean and covariance function defined as:

$$\mu(t) = 0$$

$$C(s,t) = min(s,t)$$

then:

$$(w_1, \dots, w_m) = 0 + A(X_1, \dots, X_m)^T$$

where A, can be obtained by svd or Cholesky decomposition:

$$C = AA^T$$

We can rewrite our problem as follow:

$$V_i := \frac{e^{-rT}}{(2\pi)^{m/2}\sqrt{\det(C)}} \int_{\mathbb{R}^m} \Psi_i\left(w_{t_1}, \dots, w_{t_m}\right) e^{-\frac{1}{2}w^T C^{-1} w} \, \mathrm{d}w = \frac{e^{-rT}}{(2\pi)^{m/2}} \int_{\mathbb{R}^m} \Psi_i(A\boldsymbol{x}) e^{-\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{x}} \mathrm{d}\boldsymbol{x}$$

Finally, we use trick 0.2.1, to obtain:

$$\frac{e^{-rT}}{(2\pi)^{m/2}} \int_{\mathbb{R}^m} \Psi_i(A\boldsymbol{x}) e^{-\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{x}} \mathrm{d}\boldsymbol{x} = e^{-rT} \int_{[0,1]^m} \Psi_i(A(\Phi(t_1)^{-1}, \dots, \Phi(t_m)^{-1}))\mathrm{d}t$$

Therefor, we come to our usual quasi-Monte-Carlo setting, where we just have to compute:

$$V_i = \int_{[0,1]^m} \Psi_i(A(\Phi(t_1)^{-1}, \dots, \Phi(t_m)^{-1}))\mathrm{d}t$$

The procedure consist in:

- generate points set $P_N = (t^{(i)})$, $t^{(i)} \in [0,1]^m$, for example Sobol sequence
- compute

$$V_k = \frac{1}{N} \sum_{i=1}^{N} \Psi_k(A(\Phi^{-1}(t^{(i)})))$$

How we calculate error?
We use Randomize QMC approach. We generate m low-discrepancy points set, by adding to initial point set P_N a vector $U^i = (U^i_1, \dots, U^i_m)$, where U^i_j-uniform random variable, then we simply take fractional part of $P^i = (P_N + U^i)$, for $i = 1, \dots, m$. Where each point set P^i contribute to one estimation of V^i_k. As in a case of CMC, we apply Central limit theorem:

$$|error| < 2C_{1-\frac{\lambda}{2}} \frac{std_{approximated}}{\sqrt{m}}$$

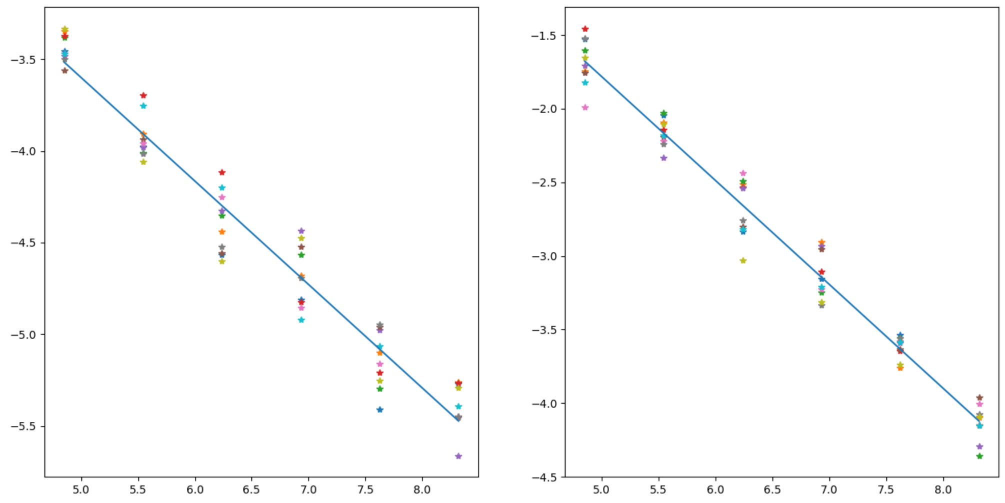

Figure 3: Each point correspond to $(\log error(N_i), \log N_i)$ produced by QMC. Where on the left we have payoff Φ_2, on the right Φ_1. With fitted regression line for 10 independent tries.

(code for computing qmc and randomized qmc 0.6.3)
Results:

- convergence rate of $V_1 \approx O(N^{-0.56})$
- convergence rate of $V_2 \approx O(N^{-0.7})$
- 0.011 sec. per 1000 samples

We see that QMC perform better then CMC.

0.3.3 Standard-Monte Carlo with pre-integration step

it's time to combine together CMC and pre-integration step. We look at:

$$V_i = \frac{e^{-rT}}{(2\pi)^{m/2}} \int_{\mathbb{R}^m} \Psi_i(A\boldsymbol{x}) e^{-\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{x}} \mathrm{d}\boldsymbol{x}$$

With pre-integration among x_1, we obtain:

$$V_k = \frac{1}{N} \sum_{i=1}^{N} p_k(x^{(i)}_{-1})$$

Where

$$x^{(i)}_{-1} = (x^{(i)}_2, \dots, x^{(i)}_m) \textit{ where } x^{(i)}_j \textit{-normally distributed}$$

$$p_k\left(x_{-1}\right) := \int_{\psi_k(x_{-1})}^{+\infty} \theta_k\left(x_1, x_{-1}\right) \rho_1\left(x_1\right) \mathrm{d}x_1,$$

With:

$$\psi\left(x_{-1}\right) = x^*, \textit{ such that: } \frac{1}{m} \sum_{\ell=1}^{m} S_{\ell_\ell}\left(\left(A\left(x^*, \boldsymbol{x}^{(i)}_{-1}\right)^\top\right)_\ell\right) - K = 0$$

$$\rho(x) = \frac{1}{\sqrt{2\pi}} exp(-1/2x^2)$$

$$\theta_2 = 1$$

$$\theta_1 = \frac{1}{m}\sum_{\ell=1}^{m} S_{\ell_\ell}\left(\left(A\left(x, \boldsymbol{x}_{-1}^{(i)}\right)^\top\right)_\ell\right) - K$$

Algorithm:

- generate N replicas $X^{(i)} = (X_1^{(i)}, \ldots, X_{m-1}^{(i)})$, independent and normally distributed
- for each $X^{(i)}$:
  - find root $x_*^{(i)} = \psi(X^{(i)})$(we use Newton approach to find root)
  - calculate $p_k(X^{(i)})$, integral from $x_*^{(i)}$ to $+\infty$ (we use Gaussian quadrature)
- compute:

$$V_k = \frac{1}{N}\sum_{i=1}^{N} p_k(X^{(i)})$$

(code: 0.6.4)

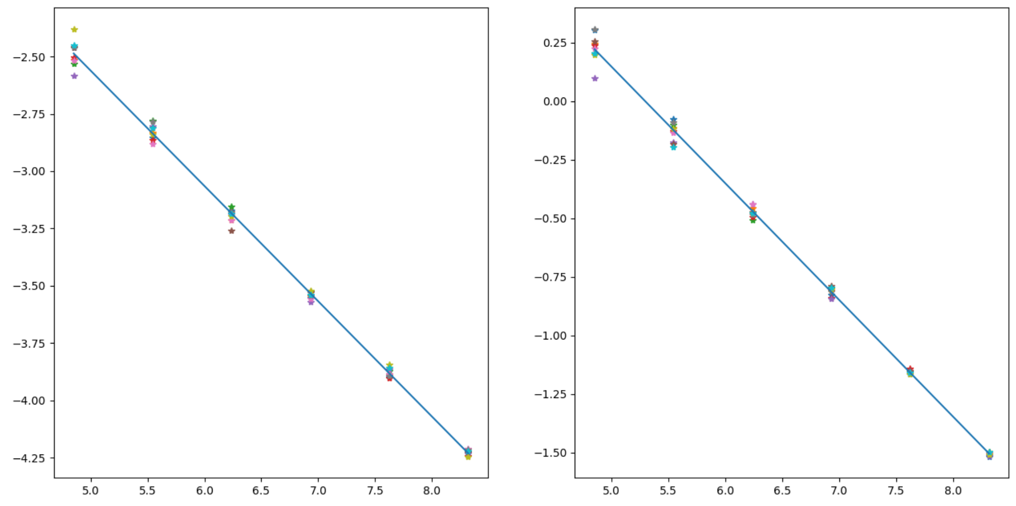

Figure 4: Each point correspond to ($\log error(N_i), \log N_i$) produced by pre-integrated cmc. Where on the left we have payoff Φ_2, on the right Φ_1. With fitted regression line for 10 independent tries

Results 4:

- convergence rate of $V_1 \approx O(N^{-0.5})$
- convergence rate of $V_2 \approx O(N^{-0.5})$
- 1.5 sec. per 1000 samples

If we compare convergence rate, of pre-integrated with standard, we see no-differences for both cases, but if we just look on error it self, pre-integrated shows better performance.
For now QMC is the best in convergence rate and error. but has slightly higher computational cost then crude Monte-Carlo.

0.3.4 Quasi-Monte Carlo with pre-integration step

Algorithm is essentially the same with "pre-integrated standard Monte Carlo" 0.3.3, the only difference consist in generating $X^{(i)} = (X_1^{(i)}, \dots, X_{m-1}^{(i)})$.
Algorithm:

- generate points set $P_N = (t^{(i)})$, $t^{(i)} \in [0,1]^{m-1}$ (Sobol sequence, $N = 2^{m-1}$)
- $X^{(i)} = \phi^{-1}(t^(i))$
- for each $X^{(i)}$:
  - find root $x_*^{(i)} = \psi(X^{(i)})$(we use Newton approach to find root)
  - calculate $p_k(X^{(i)})$, integral from $x_*^{(i)}$ to $+\infty$ (we use Gaussian quadrature)
- compute:

$$V_k = \frac{1}{N}\sum_{i=1}^{N} p_k(X^{(i)})$$

(code: 0.6.5)

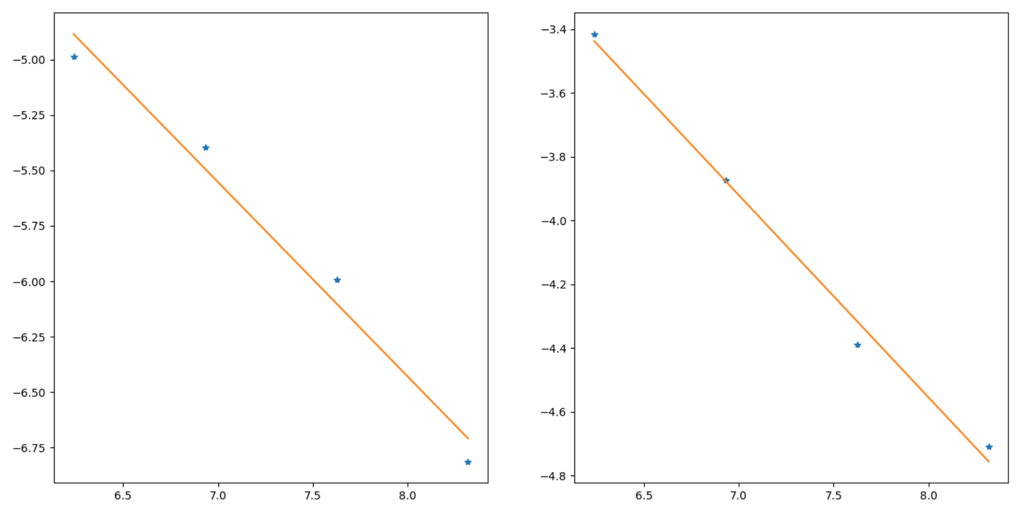

Figure 5: Each point correspond to ($\log error(N_i), \log N_i$) produced by pre-integrated qmc. Where on the left we have payoff Φ_2, on the right Φ_1. With fitted regression line for only 1 try, since computational cost is to high

Results :

- convergence rate of $V_1 \approx O(N^{-0.65})$
- convergence rate of $V_2 \approx O(N^{-0.85})$
- 1.3 sec. per 1000 samples

As we observe on the figure 5, "pre-integrated qmc" achieves the lowest error among all 4 methods, moreover has best convergence rates for both cases. Interesting the fact, that for Φ_2 convergence rate is close to the best possible $0(\frac{1}{N})$. The only issue is computational cost, which is way higher comparing to non pre-integrated approaches.
Why choosing x_1 for pre-integration ?
There are several reasons behind it:

- all x_i satisfying assumptions discussed in section 0.1:

$$\frac{\partial \phi}{\partial x_i}(x) > 0$$

$$\phi(x) \to \pm\infty \textbf{ as } x_i \to \pm\infty$$

  Since, $\phi(x) = \frac{1}{m}\sum_{i=1}^{m} S_{t_i}((Ax)_i) - K$, where coefficients of A are positive, that's lead to positive derivatives. The resting assumption is trivially satisfied.

- for implementation purposes
- if we look on parametrisation of wiener processes given by

$$(w_1, \dots, w_m) = A(X_1, \dots, X_m)^T$$

  With A-lower triangular matrix with all positive coefficients. Therefore, one pick x_1, since it will have highest contribution.

0.4 Variance reduction

In this section, we fix K-strike price to be equal 120, then payoffs are rare. Our goal will be to apply variance reeducation technique in order to get relative error ≈ 0.01 in square mean sense. In addition we interest to generalise it for QMC and "pre-integrated" QMC.
First, we apply crude Monte Carlo to check current relative error:

```
    N: 10000
V1: 0.036966789607178765 relative error: 0.11582301118628913
V2: 0.013120142561521414 relative error: 0.0824412016175039

    N: 100000
V1: 0.032579110531262286 relative error: 0.03616768099521913
V2: 0.013047755568078533 relative error: 0.026143474826532682

    N: 1000000
V1: 0.03356466990672407 relative error: 0.0115032212825799
V2: 0.012950937964348692 relative error: 0.00829858752319808
```

Only at $N \approx 10^6$ we get desire result.

0.4.1 Importance sampling

We use importance sampling to reduce variance.
Idea:
we sample from new stochastic process with greater drift, we fix new interest rate $r_{new} = 0.35$ (0.1 before). And multiply each by correcting factor:

$$W(S_{0:m}) = \prod_{n=0}^{m-1} \exp\left(\frac{1}{2}\frac{(r - r_{new})^2 \Delta t}{\sigma^2} + \frac{(r - r_{new})}{\sigma^2 S_n}(S_{n+1} - S_n(1 + r\Delta t)\right)$$

(description of the approach in Lecture notes, page 60, "Discretized stochastic differential equations")

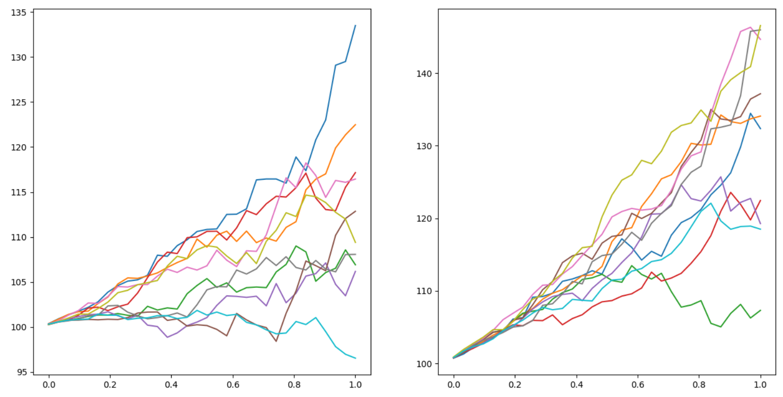

Figure 6: Simulation of Stock price S_t with interest rate: 0.1(left), 0.35(right)

Algorithm: (code for variance reduction for CMC: 0.6.6)

- Generate N Stock price $S^{(i)} = (S_1^{(i)}, \dots, S_m^{(i)})$ with new interest rate: r_{new}
- For each $S^{(i)}$, compute correcting factor: $W(S_{0:m}^{(i)})$
- estimate V_k:

$$V_k \approx \frac{1}{N}\sum_{i=1}^{N} \psi_k(\tilde{S}_{0:m}^{(i)}) W(\tilde{S}_{0:m}^{(i)})$$

Monte Carlo with importance sampling:
N: 10000
V1: 0.03504703851885694 relative error: 0.03260128851583022
V2: 0.013287825653481354 relative error: 0.042257193286885944

N: 100000
V1: 0.03645086121253862 relative error: 0.011606013168272878
V2: 0.013145205757424533 relative error: 0.01172103107931842

We see that already at $N \approx 10^5$ we reach tolerance of 0.01.

0.4.2 QMC-importance sampling

We first look on relative error applying QMC and pre-integrated QMC:
N: 512
V1: relative error: 0.08507682444; pre-int relative error: 0.063977138
V2: relative error: 0.0698077769; pre-int relative error: 0.07893459856

N: 1024
V1: relative error: 0.057239305; pre-int relative error: 0.077221294
V2: relative error: 0.03566248; pre-int relative error: 0.073733766395

N: 2048
V1: relative error: 0.03859433129; pre-int relative error: 0.050977380938
V2: relative error: 0.026949318660; pre-int relative error: 0.024879671348

N: 4096
V1: relative error: 0.0251349488819; pre-int relative error: 0.029210692259
V2: relative error: 0.01853459840; pre-int relative error: 0.0256492877629

Noticeable fact: QMC and pre-intagrated QMC (Using Sobol points set) requires way less samples to get relatively low error. Moreover, QMC is slightly better then it's upgraded version.

The following question is: How to apply importance sampling in QMC settings ?
In a fact, importance sampling can be directly generalized to QMC approach, since we can rewrite target integral as follow:

$$V_i = \frac{e^{-rT}}{(2\pi)^{m/2}} \int_{\mathbb{R}^m} \Psi_i(A\boldsymbol{x}) e^{-\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{x}} \mathrm{d}\boldsymbol{x} = \frac{e^{-rT}}{(2\pi)^{m/2}} \int_{\mathbb{R}^m} \tilde{\Psi}_i(A\boldsymbol{x}) W(A\boldsymbol{x}) e^{-\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{x}} \mathrm{d}\boldsymbol{x}$$

Where $\tilde{\Psi}_i$ corresponds to payoff with new rate: r_{new}
Algorithm: (code for importance sampling QMC:0.6.7)

- generate low-discrepancy points set $P_N = (t^{(1)}, \dots, t^{(N)})$
- $X^{(i)} = \Phi^{-1}(t^{(i)})$
- compute: $V_i = \sum_{i=1}^{N} \tilde{\Psi}_i(AX^{(i)}) W(AX^{(i)})$

Results:
N: 512
V1: 0.0359701931951792 relative error: 0.017907640752359132

V2: 0.013240618281473571 relative error: 0.025595348686068876

N: 1024
V1: 0.03639127876334638 relative error: 0.014969505590505095
V2: 0.012458128319422368 relative error: 0.021075755949775996

N: 2048
V1: 0.03462089915834737 relative error: 0.008414426872242973
V2: 0.013370482535482158 relative error: 0.012962666533517474

N: 4096
V1: 0.03449028254248328 relative error: 0.005530462794411522
V2: 0.013288694327379188 relative error: 0.011570691558232963

We observe significant improvement in a case of importance sampling, already for $N = 2048$ we get low relative error, which is even less then for $N = 4096$(without variance reduction).
Can we apply importance sampling to QMC with pre-integration step?
Yes, pre-integration approach, can be as simply adapted, as it was for QMC.
Pre-integration step:

$$\tilde{p}_k\left(x_{-1}\right) := \int_{\tilde{\psi}_k\left(x_{-1}\right)}^{+\infty} \tilde{\theta}_k\left(x_1, x_{-1}\right) W(A(x_1, x_{-1})^T)\rho_1\left(x_1\right) \mathrm{d} x_1,$$

Where $\tilde{\psi}_k$, $\tilde{\theta}_k$ defined as before but with new interest rate.
The Algorithm is similar to algorithm 0.3.4, the only difference is that we use $\tilde{p}_k$ instead p_k.
(code for importance sampling with pre-int. step: 0.6.8)
The only issue, is that it becomes very costly to implement, mostly due to Gauss quadrature step, since target function to integrate is to complicated.
Results:
N: 512
V1: 0.03533402691730898 relative error: 0.023495774278433627
V2: 0.0143135817378971 relative error: 0.04843299760839202

N: 1024
V1: 0.03684276096563001 relative error: 0.01260592758254299
V2: 0.012540628864593086 relative error: 0.017142694651974783

N: 2048
V1: 0.03756747848696616 relative error: 0.013985678459471905
V2: 0.013801420387836032 relative error: 0.014988091836570183

Again, we see effectiveness of variance reduction technique.
So, we can conclude, that importance sampling can be easily adapted to QMC and pre-integrated QMC, for both is very effective. That let us sample rare events with new improved approach.

0.4.3 Comments

Initially, it was demanded to fix K=500, but it seems to be irrelevant to exercise, since even for $N = 10^8$ I was getting only zero results. Even with $K = 150$ the picture is the same. Which is reasonable, since as we can see on figure 1, we get the highest point at $S_t = 127$, but the goal to be in average bigger then $K = 150$, not just at one point. Then, in order to show the improvement of using importance sampling, even in "pre-integrated" settings (which is costly), I had to lower K to 120.
Also I had to analyse convergence rate for different number m of discritizations. Which is not presented due to several reasons:

- for m=64 and m=128, convergence rate approximately the same as in a case of m=32. Example for m=128 :

$$\text{CMC: } V_2 \approx O(N^{-0.4940393})$$

$$\text{QMC: } V_2 \approx O(N^{-0.68969753})$$

$$\text{pre-int CMC: } V_2 \approx O(N^{-0.50376903})$$

pre-int QMC: $V_2 \approx O(N^{-0.747804613})$

Then it seems, that increase of dimension doesn't effect the order of performance.

- The cost of producing results, especially in case of pre-integrated qmc, getting to high with increase of dimension. from 1.3 sec. for $m = 32$, to 3.5 for $m = 128$ per 1000 samples. For higher m, it would take the whole day or more to get reasonable results.

0.5 Conclusion

We saw that "pre-integrated" trick combined with qmc , is very efficient tool, that achieve the lowest error, and have highest convergence rate. Moreover, when applied to standard monte-carlo, we don't have any improvement in convergence rate, but error it self is lower then in a case of simple MC. We also have seen, that variance reduction technique, such as importance sampling can be adapted for "pre-integrated" step, allowing us to sample rare events. On other hand , computational cost caused by searching root with Newton algorithm and Gaussian quadrature for semi-infinite integration, could be extremely high, especially in a case of large dimension m. Solution might be to use more appropriate algorithms or in the best scenario is to have closed form solution for "pre-integrated" step, which is a case for some option call problems.

0.6 Code

0.6.1 simulation

```
#setup parameters
T=1
sigma=0.1
r=0.1
S_0=100
K=100
m=32
C=norm.ppf(1-0.05/2)# 1-lamda/2 quantile of normal distribution

#simulation of wiener process W_t
def Weiner(m):
    delta_t=T/m
    W=np.sqrt(delta_t)*norm.rvs(size=m) # vector of independentes increments
    W=np.cumsum(W) # browian motion
    return W

#Stock price
def ST(m):
    W=Weiner(m)
    t=np.ones(m)/m
    t=np.cumsum(t)
    S = S_0 * np.exp( (r-0.5*sigma**2) * t + sigma * (t*W) )
    return S

#plotting
for k in range(0,10):
    plt.plot(np.linspace(0,1,m), ST(m))
plt.show()
```

0.6.2 CMC error and plotting regression line

Here we represent the code of CMC and production of regression line, and this structure will be the same for all four algorithms, therefore it won't be presented for rest of algorithms, only implantation of methods it's self will be shown

```
#Crude monte carlo
#define functions
#Phi_2
```

```
Phi2 = lambda St:  ( (1/len(St) ) * np.sum(St)-K > 0 )
#Phi_1
Phi1 = lambda St:  ( (1/len(St) ) * np.sum(St)-K > 0 ) * ( (1/len(St) ) * np.sum(St)-K )






#plotting
fig=plt.figure(figsize=(4,8))
ax1=fig.add_subplot(121)
ax2=fig.add_subplot(122)


E1=np.array([])#accimulate errors for diferent trials
E2=np.array([])
x=np.array([2**i for i in range(7,13)])
xx=np.array([])
#error with respect to N-number of samples
for try_ in range(10):
    Error_1=np.array([])
    Error_2=np.array([])
    V_1=np.array([])
    V_2=np.array([])
    Time=np.array([])
    for n in range(7,13):
        N=2**n

        t_start=time.time()
        phi2=np.array([ Phi2( ST(m) ) for i in range(0,N)])
        V_2 = np.append(V_2, np.mean(phi2))
        t_end=time.time()
        Time=np.append(Time,t_end-t_start)
        std=np.std(phi2)#standard deviation
        error=2*C*std/np.sqrt(N)
        Error_2=np.append(Error_2,error)




        phi1 = np.array([Phi1(ST(m)) for i in range(0, N)])
        std = np.std(phi1)  # standard deviation
        error = 2 * C * std / np.sqrt(N)
        Error_1 = np.append(Error_1, error)
        V_1 = np.append(V_1, np.mean(phi1))

    ax2.plot(np.log(x), np.log(Error_1), "*")
    ax1.plot(np.log(x), np.log(Error_2), "*")
    E1=np.append(E1,Error_1)
    E2=np.append(E2,Error_2)
    xx=np.append(xx,x)



#phi2 regression-line
X=np.log(xx).reshape(-1,1)
y=np.log(E2)
reg=LinearRegression().fit(X,y)
print(reg.coef_)
ax1.plot(np.log(x),reg.predict(np.log(x).reshape(-1,1)))
```

```
#phi1 regression-line
X=np.log(xx).reshape(-1,1)
y=np.log(E1)
reg=LinearRegression().fit(X,y)
print(reg.coef_)

ax2.plot(np.log(x),reg.predict(np.log(x).reshape(-1,1)))
plt.show()


#time
X=x.reshape(-1,1)
y=Time
reg=LinearRegression().fit(X,y)
print(1000*reg.coef_)
plt.plot(x,Time)
plt.show()
```

0.6.3 QMC

```
#covariance matrix
h=1/m
C=np.ones([m,m])
for i in range(0,m):
    for j in range(0,m):
        C[i,j]=min((i+1)*h,(j+1)*h)



#cholesky
A=scipy.linalg.cholesky(C)
A=np.transpose(A)



t=np.ones(m)/m
t=np.cumsum(t)
#Phi_2
Phi2 = lambda X:  ( (S_0/len(X)) * np.sum(np.exp( (r-0.5*sigma**2)*t + sigma*A@X)) - K > 0 )
#Phi_1
Phi1 = lambda X:  ( (S_0/len(X))* np.sum(np.exp( (r-0.5*sigma**2)*t + sigma* A@X)) -K > 0 ) * (


def QMC(function,m,p):
    #N=m^p
    t_start=time.time()
    sampler = qmc.Sobol(d=m, scramble=True)
    sample = sampler.random_base2(m=p)
    X=norm.ppf(sample)
    mu=np.array([function(X[i,:]) for i in range(len(X[:,0])) ])
    mean=np.mean(mu)
    t_end=time.time()
    #randomized qmc to evaluate error
    MU=np.array([])
    for i in range(30):
        sample_ = sample + uniform.rvs(size=m)
        sample_ = np.modf(sample_)[0]
        X=norm.ppf(sample_)
        F = np.array([function(X[i, :]) for i in range(len(X[:, 0]))])
        MU = np.append(MU, np.mean(F))

    C = norm.ppf(1 - 0.05 / 2)
    std = np.std(MU)
    error = 2 * C * std / np.sqrt(10)
    return mean, error, t_end-t_start
```

0.6.4 Pre-integration with CMC

```
f1 = lambda x, X: (1 / ((2 * np.pi) ** (1 / 2))) * ((S_0 / (len(X) + 1)) * np.sum(
    np.exp((r - 0.5 * sigma ** 2) * t + sigma * A @ np.append(x, X))) - K) * np.exp(-0.5 * x**2

f2 = lambda x, X: (1 / ((2 * np.pi) ** (1 / 2))) * np.exp(-0.5 * x**2) #thetha_2 * rho

f_0= lambda x, X: (S_0 / (len(X) + 1)) * np.sum(
    np.exp((r - 0.5 * sigma ** 2) * t + sigma * A @ np.append(x, X))) - K  #for finding root



def pre_integrate_CMC(f, N, m):
    mu=np.array([])
    t_start=time.time()
    for n in range(N):
        X = norm.rvs(size=m - 1)
        root = optimize.newton(f_0, 0, args=(X,))#finding root
        Q, error = integrate.quad(f, root + 0.01, np.inf , args=(X,))#Gaussian quadrature
        mu=np.append(mu,Q)
    C=norm.ppf(1 - 0.05 / 2)
    mean=np.mean(mu)
    t_end=time.time()
    std = np.std(mu)
    error = 2 * C * std / np.sqrt(N)
    return mean, error, t_end-t_start
```

0.6.5 Pre-integration with QMC

```
f1 = lambda x, X: (1 / ((2 * np.pi) ** (1 / 2))) * ((S_0 / (len(X) + 1)) * np.sum(
    np.exp((r - 0.5 * sigma ** 2) * t + sigma * A @ np.append(x, X))) - K) * np.exp(-0.5 * x**2

f2 = lambda x, X: (1 / ((2 * np.pi) ** (1 / 2))) * np.exp(-0.5 * x**2)

f_0= lambda x,X: (S_0 / (len(X) + 1)) * np.sum(
    np.exp((r - 0.5 * sigma ** 2) * t + sigma * A @ np.append(x, X))) - K  #for finding root

def pre_integrate_QMC(f, m, p):
    #N=m^p
    mu=np.array([])
    t_start=time.time()
    sampler = qmc.Sobol(d=m-1, scramble=True)
    sample = sampler.random_base2(m=p)
    Y = norm.ppf(sample)
    for i in range(len(Y[:,0])):
        X = Y[i,:]
        root = optimize.newton(f_0, 0, args=(X,))#finding root
        Q, error = integrate.quad(f, root + 0.01, np.inf , args=(X,))#Gaussian quadrature
        mu=np.append(mu,Q)
    mean = np.mean(mu)
    t_end = time.time()
    #randomized-QMC
    MU = np.array([])
    for i in range(10):
        mu=np.array([])
        sample_ = sample + uniform.rvs(size=m-1)
        sample_ = np.modf(sample_)[0]
        Y = norm.ppf(sample_)
        for i in range(len(Y[:, 0])):
            X = Y[i, :]
            root = optimize.newton(f_0, 0, args=(X,))  # finding root
            Q, error = integrate.quad(f, root + 0.01, np.inf, args=(X,))  # Gaussian quadrature
            mu = np.append(mu, Q)
        mean = np.mean(mu)
        MU = np.append(MU, mean)
    C = norm.ppf(1 - 0.05 / 2)
    std = np.std(MU)
    error = 2 * C * std / np.sqrt(10)
    return mean, error, t_end - t_start
```

0.6.6 Variance reduction for CMC

```
#variance reduction
#define new functions
r_new=0.34
Phi2 = lambda X:  np.exp(-r_new)*( (S_0/len(X)) * np.sum(np.exp( (r_new-0.5*sigma**2)*t + sigma
#Phi_1
Phi1 = lambda X:  np.exp(-r_new)*( (S_0/len(X))* np.sum(np.exp( (r_new-0.5*sigma**2)*t + sigma*

#computing ratio
def Weight(S):
    h=1/m
    ratio=1
    for i in range(m-1):
        ratio=ratio*np.exp(0.5*h*((r-r_new)*(r-r_new))/(sigma*sigma) + (r-r_new)*(S[i+1]-S[i]*(1

    return ratio


def CMC_var_red(function,m,N):
    #N=m^p
    mu=np.array([])
    for i in range(N):
        X=norm.rvs(size=m)
        S=S_0*np.exp( (r_new-0.5*sigma**2)*t + sigma*A@X)
        W=Weight(S)
        mu=np.append(mu,function(X)*W )
    mean=np.mean(mu)
    std = np.std(mu)
    error =  std / np.sqrt(N)/mean
    return mean, error

for n in range(3,6):
     N = 10**n
     print("N:",N)
     V1, error1 = CMC_var_red(Phi1, m, N)
     V2, error2 = CMC_var_red(Phi2, m, N)

     print("V1:",V1, "relative_error:", error1)
    print("V2:",V2, "relative error:", error2)
    print()
```

0.6.7 Variance reduction for QMC

```
#define new functions
r_new=0.35
Phi2_ = lambda X:  np.exp(-r_new)*( (S_0/len(X)) * np.sum(np.exp( (r_new-0.5*sigma**2)*t + sigma
#Phi_1
Phi1_ = lambda X:  np.exp(-r_new)*( (S_0/len(X))* np.sum(np.exp( (r_new-0.5*sigma**2)*t + sigma

#computing ratio
def Weight(S):
    h=1/m
    ratio=1
    for i in range(m-1):
        ratio=ratio*np.exp(0.5*h*((r-r_new)*(r-r_new))/(sigma*sigma) + (r-r_new)*(S[i+1]-S[i]*(1

    return ratio
#variance reduction version of qmc
def QMC_var_red(function,m,p):
    #N=m^p
    sampler = qmc.Sobol(d=m, scramble=True)
    sample = sampler.random_base2(m=p)
    X=norm.ppf(sample)
    mu=np.array([function(X[i,:])*Weight(S_0*np.exp( (r_new-0.5*sigma**2)*t + sigma*A@X[i,:]))
    mean=np.mean(mu)

    #randomized qmc to evaluate error
    MU=np.array([])
    for i in range(30):
        sample_ = sample + uniform.rvs(size=m)
        sample_ = np.modf(sample_)[0]
        X=norm.ppf(sample_)
        F = np.array([function(X[i, :])*Weight(S_0*np.exp( (r_new-0.5*sigma**2)*t + sigma*A@X[i
        MU = np.append(MU, np.exp(-r)*np.mean(F))

    std = np.std(MU)
    error = std / np.sqrt(30)/np.mean(MU)
    return mean, error
#checking reduction of relative error
for i in range(9,13):
    N = 2**i
    print("N:",N)
    V1, error1 = QMC_var_red(Phi1_, m, i)
    V2, error2 = QMC_var_red(Phi2_, m, i)
    print("V1:",V1,"    QMC relative error:", error1)
    print("V2: ",V2,"   QMC relative error:", error2)
    print()


#variance reduction with pre-integration
```

0.6.8 Variance reduction for QMC with pre-integrated step

```
#define new functions
h=1/m

f_0_= lambda x,X: (S_0 / (len(X) + 1)) * np.sum(
    np.exp((r_new - 0.5 * sigma ** 2) * t + sigma * A @ np.append(x, X))) - K  #for finding roo
```

```
W  = lambda x,X: np.prod( np.array([ np.exp(0.5*h*((r-r_new)*(r-r_new))/(sigma*sigma) + (r-r_new
 for i in range(m-1)       ])    )

 f1_ = lambda x, X: np.exp(-r_new)*(1 / ((2 * np.pi) ** (1 / 2))) * ((S_0 / (len(X) + 1)) * np.su
     np.exp((r_new - 0.5 * sigma ** 2) * t + sigma * A @ np.append(x, X))) - K) * np.exp(-0.5 * 

 f2_ = lambda x, X: np.exp(-r_new)*(1 / ((2 * np.pi) ** (1 / 2))) * np.exp(-0.5 * x**2) * W(x,X)

#reduction variance version
def pre_integrate_QMC_var_red(f, m, p):
    #N=m^p
    mu=np.array([])
    sampler = qmc.Sobol(d=m-1, scramble=True)
    sample = sampler.random_base2(m=p)
    Y = norm.ppf(sample)
    for i in range(len(Y[:,0])):
        X = Y[i,:]
        root = optimize.newton(f_0_, 0, args=(X,))#finding root
        Q, error = integrate.quad(f, root+0.001 , np.inf , args=(X,))#Gaussian quadrature
        mu=np.append(mu,Q)
    mean = np.mean(mu)
    #randomized-QMC
    MU = np.array([])
    for i in range(5):
        mu=np.array([])
        sample_ = sample + uniform.rvs(size=m-1)
        sample_ = np.modf(sample_)[0]
        Y = norm.ppf(sample_)
        for i in range(len(Y[:, 0])):
            X = Y[i, :]
            root = optimize.newton(f_0_, 0, args=(X,))  # finding root
            Q, error = integrate.quad(f, root + 0.001 , np.inf, args=(X,))  # Gaussian quadratu
            mu = np.append(mu, Q)
        mean = np.mean(mu)
        MU = np.append(MU, mean)
    C = norm.ppf(1 - 0.05 / 2)
    std = np.std(MU)
    error = std / np.sqrt(5)/np.mean(MU)
    return mean, error

#checking results
for i in range(9,12):
     N = 2**i
     print("N:",N)
     V1, error1 = pre_integrate_QMC_var_red(f1_, m, i)
     V2, error2 = pre_integrate_QMC_var_red(f2_, m, i)
     print("V1:",V1,"␣␣␣␣QMC␣relative␣error:", error1)
     print("V2:␣",V2,"␣␣␣QMC␣relative␣error:", error2)
     print()
```

0.7 References

[1] Andreas Griewank, Frances Y Kuo, Hernan Leövey, and Ian H Sloan. High dimensional integration of kinks and jumps - smoothing by preintegration. Journal of Computational and Applied Mathematics, 344:259-274, 2018.