Fast QMC matrix vector multiplication in option pricing

Josef Dick
joint work with F. Kuo, Q. T. Le Gia, Ch. Schwab

School of Mathematics and Statistics, UNSW, Sydney, Australia

We consider the pricing of options:

Consider the geometric Brownian motion

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t\,\mathrm{d}W_t, \quad t \geq 0,$$

where r is the risk-free interest rate, σ is the volatility, and W_t is a standard Brownian motion. The solution of this SDE is given by

$$S_t = S_0 \exp\left((r - \tfrac{1}{2}\sigma^2)t + \sigma W_t\right), \quad t \geq 0.$$

Let g be the payoff function of some option. Then the expected payoff is given by

$$\mathbb{E}(g) = \int_{\mathbb{R}^s} g(\boldsymbol{w}) \frac{\exp\left(-\frac{1}{2}\boldsymbol{w}^\top \Sigma^{-1} \boldsymbol{w}\right)}{\sqrt{(2\pi)^s \det(\Sigma)}} \, \mathrm{d}\boldsymbol{w},$$

where the covariance matrix $\Sigma = (\Sigma_{i,j})_{1 \leq i,j \leq s}$ is given by

$$\Sigma_{i,j} = \Delta t \min(i,j), \quad i,j = 1, \ldots, s.$$

Here we assume equally spaced times $t_j = j\Delta t$ for $j = 1, \ldots, s$, where $\Delta t = \frac{T}{s}$.

The option price is then $\mathrm{e}^{-rT}\mathbb{E}(g)$.

Generate normally distributed samples with a general covariance matrix.

Consider

$$\int_{\mathbb{R}^s} g(\boldsymbol{w}) \frac{\exp\left(-\frac{1}{2}\boldsymbol{w}\Sigma^{-1}\boldsymbol{w}^\top\right)}{\sqrt{(2\pi)^s \det(\Sigma)}} \,\mathrm{d}\boldsymbol{w}.$$

Generate normally distributed samples with a general covariance matrix.

Consider

$$\int_{\mathbb{R}^s} g(\boldsymbol{w}) \frac{\exp\left(-\frac{1}{2}\boldsymbol{w}\Sigma^{-1}\boldsymbol{w}^\top\right)}{\sqrt{(2\pi)^s \det(\Sigma)}}\,\mathrm{d}\boldsymbol{w}.$$

Use a factorization $\Sigma = A^\top A$ and $\boldsymbol{w} = \Phi^{-1}(\boldsymbol{y})A$, where Φ^{-1} is the inverse standard normal CDF, to get

$$\int_{[0,1]^s} f(\Phi^{-1}(\boldsymbol{y})A)\,\mathrm{d}\boldsymbol{y}.$$

Generate normally distributed samples with a general covariance matrix.

Consider

$$\int_{\mathbb{R}^s} g(\boldsymbol{w}) \frac{\exp\left(-\frac{1}{2}\boldsymbol{w}\Sigma^{-1}\boldsymbol{w}^\top\right)}{\sqrt{(2\pi)^s \det(\Sigma)}} \,\mathrm{d}\boldsymbol{w}.$$

Use a factorization $\Sigma = A^\top A$ and $\boldsymbol{w} = \Phi^{-1}(\boldsymbol{y})A$, where Φ^{-1} is the inverse standard normal CDF, to get

$$\int_{[0,1]^s} f(\Phi^{-1}(\boldsymbol{y})A) \,\mathrm{d}\boldsymbol{y}.$$

We approximate this integral by

$$\frac{1}{N}\sum_{n=0}^{N-1} f(\Phi^{-1}(\boldsymbol{y}_n)A).$$

Commonly used factorizations of Σ are: Cholesky factorization, Brownian bridge (Moskowitz and Caflisch), Principle component analysis (Acworth, Broadie and Glasserman).

All of these factorizations are known explicitely and in each case the matrix A has a 'nice' structure such that the matrix vector product $\Phi^{-1}(\boldsymbol{y})A$ can be computed fast (PCA: Scheicher).

Commonly used factorizations of Σ are: Cholesky factorization, Brownian bridge (Moskowitz and Caflisch), Principle component analysis (Acworth, Broadie and Glasserman).

All of these factorizations are known explicitely and in each case the matrix A has a 'nice' structure such that the matrix vector product $\Phi^{-1}(\boldsymbol{y})A$ can be computed fast (PCA: Scheicher).

For QMC the choice of A can make a difference in the approximation properties of the estimator (Papageorgiou).

The strategie is then to choose A "wisely" (Imai and Tan). For instance, Leobacher and Leobacher and Irrgeher use the Householder transform.

Now consider multi-asset options: Assume d stocks with constant volatilities σ_i and correlations $\rho_{i,j}$. The covariance between these stocks is given by the symmetric positive definite matrix $\Gamma = (\Gamma_{i,j})_{1\le i,j\le d}$ with entries

$$\Gamma_{i,j} = \sigma_i \rho_{i,j} \sigma_j, \quad i,j = 1,\ldots,d.$$

The expected payoff for a given payoff function g is then

$$\begin{aligned}\mathbb{E}(g) &= \int_{\mathbb{R}^{ds}} g(\boldsymbol{w}) \frac{\exp(-\frac{1}{2}\boldsymbol{w}^\top (\Sigma\otimes\Gamma)^{-1}\boldsymbol{w})}{\sqrt{(2\pi)^{ds}\det(\Sigma\otimes\Gamma)}}\,\mathrm{d}\boldsymbol{w} \\ &= \int_{[0,1]^{sd}} g(\Phi^{-1}(\boldsymbol{y})(A\otimes B))\,\mathrm{d}\boldsymbol{y},\end{aligned}$$

where $\Sigma = AA^\top$ and $\Gamma = BB^\top$, and $A\otimes B$ denotes the Kronecker product

$$A\otimes B = \begin{pmatrix} a_{1,1}B & \ldots & a_{1,s}B \\ \vdots & \ddots & \vdots \\ a_{s,1}B & \ldots & a_{d,d}B \end{pmatrix}.$$

We approximate the integral by

$$\int_{[0,1]^{sd}} g(\Phi^{-1}(\boldsymbol{y})(A \otimes B))\,\mathrm{d}\boldsymbol{y} \approx \frac{1}{N}\sum_{n=0}^{N-1} g(\Phi^{-1}(\boldsymbol{y}_n)(A \otimes B)).$$

In general, the matrix $A \otimes B$ does not have a nice structure anymore.

One way to speed up the computation of $\boldsymbol{y}_n(A \otimes B)$ is by rearranging this product to

$$A\Phi^{-1}(Y_n)B^{\top},$$

where Y_n is a suitable rearrangement of $\boldsymbol{y}_n$. The matrix A has a nice structure as before. However, the matrix B does not have a nice structure.

More generally, let us consider the approximation of an integral of the form

$$\int_{[0,1]^s} f(\Phi^{-1}(\boldsymbol{y})A)\,\mathrm{d}\boldsymbol{y} \approx \frac{1}{N}\sum_{n=0}^{N-1} f(\Phi^{-1}(\boldsymbol{y}_n)A),$$

where $\boldsymbol{y}$ is a row-vector of length s, the function Φ^{-1} is applied component-wise, and $A \in \mathbb{R}^{s\times t}$.

More generally, let us consider the approximation of an integral of the form

$$\int_{[0,1]^s} f(\Phi^{-1}(\boldsymbol{y})A)\,\mathrm{d}\boldsymbol{y} \approx \frac{1}{N}\sum_{n=0}^{N-1} f(\Phi^{-1}(\boldsymbol{y}_n)A),$$

where $\boldsymbol{y}$ is a row-vector of length s, the function Φ^{-1} is applied component-wise, and $A \in \mathbb{R}^{s\times t}$.

Let

$$X = \begin{pmatrix} \Phi^{-1}(\boldsymbol{y}_0) \\ \Phi^{-1}(\boldsymbol{y}_1) \\ \vdots \\ \Phi^{-1}(\boldsymbol{y}_{N-1}) \end{pmatrix} = \begin{pmatrix} \Phi^{-1}(y_{0,1}) & \dots & \Phi^{-1}(y_{0,s}) \\ \Phi^{-1}(y_{1,1}) & \dots & \Phi^{-1}(y_{1,s}) \\ \vdots & & \vdots \\ \Phi^{-1}(y_{N-1,1}) & \dots & \Phi^{-1}(y_{N-1,s}) \end{pmatrix}.$$

For $X = \Phi^{-1}(Y)$, we compute

$$XA = B = \begin{pmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_{N-1} \end{pmatrix} \in \mathbb{R}^{N \times t}.$$

Computing the matrix product XA costs $\mathcal{O}(Nst)$ operations.

We want to find a method to compute this product faster.

For $X = \Phi^{-1}(Y)$, we compute

$$XA = B = \begin{pmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_{N-1} \end{pmatrix} \in \mathbb{R}^{N \times t}.$$

Computing the matrix product XA costs $\mathcal{O}(Nst)$ operations.

We want to find a method to compute this product faster.

Idea: Find quadrature points $\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N-1}$ such that the matrix-vector product Xa_k can be computed fast, where $A = (\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_t)$.

Lattice rule:

Let N be a prime number and $\boldsymbol{g} = (g_1, g_2, \ldots, g_s) \in \{1, 2, \ldots, N-1\}^s$. Then

$$\boldsymbol{y}_n = \left(\left\{\frac{ng_1}{N}\right\}, \ldots, \left\{\frac{ng_s}{N}\right\}\right) \quad \text{for } n = 0, 1, \ldots, N-1,$$

where $\{x\} = x - \lfloor x \rfloor$ for $x \geq 0$.

Let β be a primitive element of the multiplicative group $\mathbb{Z}_N^*$, that is, $\{\beta^k \mod N : k = 0, 1, 2, \ldots, N-2\} = \{1, 2, \ldots, N-1\}$. Write

$$g_j = \beta^{c_j - 1} \pmod{N}, \quad c_j \in \{1, \ldots, N-1\}.$$

Conveniently, the fast component-by-component construction computes c_j.

Let β be a primitive element of the multiplicative group $\mathbb{Z}_N^*$, that is, $\{\beta^k \mod N : k = 0, 1, 2, \ldots, N-2\} = \{1, 2, \ldots, N-1\}$. Write

$$g_j = \beta^{c_j - 1} \pmod N, \quad c_j \in \{1, \ldots, N-1\}.$$

Conveniently, the fast component-by-component construction computes c_j.
Let

$$x_{n,j} = \Phi^{-1}\left(\left\{\frac{\beta^{c_j - 1 - (n-1)}}{N}\right\}\right) = \Phi^{-1}\left(\left\{\frac{\beta^{c_j - n}}{N}\right\}\right)$$

and

$$X' = \begin{pmatrix} x_{1,1} & \cdots & x_{1,s} \\ \vdots & & \vdots \\ x_{N-1,1} & \cdots & x_{N-1,s} \end{pmatrix}.$$

Since the ordering of the points is irrelevant, we can compute $X' a_k$ instead of $X a_k$.

Goal: Write $X' = ZP$.

For $k \in \mathbb{Z}$ let

$$z_k = \Phi^{-1}\left(\left\{\frac{\beta^k}{N}\right\}\right).$$

Then $z_k = z_{k+v(N-1)}$ for all $v \in \mathbb{Z}$. Let

$$Z = \begin{pmatrix} z_0 & z_1 & z_2 & \dots & z_{N-3} & z_{N-2} \\ z_{N-2} & z_0 & z_1 & \ddots & \ddots & z_{N-3} \\ z_{N-3} & z_{N-2} & z_0 & \ddots & \ddots & z_{N-4} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ z_2 & \ddots & \ddots & \ddots & z_0 & z_1 \\ z_1 & z_2 & \dots & \dots & z_{N-2} & z_0 \end{pmatrix}.$$

Let $P = (p_{k,j})_{1 \le k \le N-1, 1 \le j \le s} \in \{0,1\}^{(N-1)\times s}$ be given by

$$p_{k,j} = \begin{cases} 1 & \text{if } k = c_j, \\ 0 & \text{otherwise.} \end{cases}$$

Each column of P has at most one entry 1 with the remaining entries being 0.

Then

$$X' = ZP.$$

Computing $\boldsymbol{c}_k = P\boldsymbol{a}_k$ is just a re-ordering of the elements and $Z\boldsymbol{c}_k$ can be computed using the fast Fourier transform in $\mathcal{O}(N \log N)$ operations.

Then

$$X' = ZP.$$

Computing $\boldsymbol{c}_k = P\boldsymbol{a}_k$ is just a re-ordering of the elements and $Z\boldsymbol{c}_k$ can be computed using the fast Fourier transform in $\mathcal{O}(N \log N)$ operations.

The number of operations is now $\mathcal{O}(N(\log N)t)$ instead of $\mathcal{O}(Nst)$, i.e., we replace s by $\log N$.

Then

$$X' = ZP.$$

Computing $\boldsymbol{c}_k = P\boldsymbol{a}_k$ is just a re-ordering of the elements and $Z\boldsymbol{c}_k$ can be computed using the fast Fourier transform in $\mathcal{O}(N \log N)$ operations.

The number of operations is now $\mathcal{O}(N(\log N)t)$ instead of $\mathcal{O}(Nst)$, i.e., we replace s by $\log N$.

This method incurs a storage cost of $\mathcal{O}(Nt)$.

Apart from lattice point sets, the following point sets also work:

- Polynomial lattice point set

Apart from lattice point sets, the following point sets also work:

- Polynomial lattice point set
- Union of all (polynomial) lattice point sets

Apart from lattice point sets, the following point sets also work:

- Polynomial lattice point set
- Union of all (polynomial) lattice point sets
- One can apply a transformation $\phi : [0, 1] \to \mathbb{R}$: for instance an inverse (normal) cumulative distribution function or the tent-transform;

Union of all lattice point sets:

$$\boldsymbol{y}_{n,g} = \left(\left\{ \frac{ng^0}{N} \right\}, \ldots, \left\{ \frac{ng^{s-1}}{N} \right\} \right), \quad n, g = 1, 2, \ldots, N-1.$$

Quadrature rules based on this point set behave more like MC rather than QMC.

Union of all lattice point sets:

$$\boldsymbol{y}_{n,g} = \left(\left\{ \frac{ng^0}{N} \right\}, \ldots, \left\{ \frac{ng^{s-1}}{N} \right\} \right), \quad n, g = 1, 2, \ldots, N-1.$$

Quadrature rules based on this point set behave more like MC rather than QMC.

Apply transformation $\phi : [0, 1] \to \mathbb{R}$: Use point set

$$\boldsymbol{y}_n = \left(\phi\left(\left\{ \frac{ng_1}{N} \right\} \right), \ldots, \phi\left(\left\{ \frac{ng_s}{N} \right\} \right) \right), \quad n = 0, 1, \ldots, N-1.$$

Method works in the same way as before, one only needs to replace z_k by

$$z_k = \phi\left(\left\{ \frac{\beta^k}{N} \right\} \right).$$

Similarly as in the case when we apply the inverse normal cumulative distribution function.

Cases where (currently) it does not work:

- Randomizations such as random (digital) shift, scrambling;
- Interlaced polynomial lattice rules;
- Higher order polynomial lattice rules (works but is not effective);

Cases where (currently) it does not work:

- Randomizations such as random (digital) shift, scrambling;
- Interlaced polynomial lattice rules;
- Higher order polynomial lattice rules (works but is not effective);

Second order + fast QMC matrix vector multiplication possible. A special case from a result by (Goda, Suzuki, and Yoshiki) shows that applying the tent transform to polynomial lattice rules yields a second order QMC rule (no randomization needed).

Method	$N \setminus s$	200	400	600	800	1000
std.	16001	0.309	0.741	1.296	1.617	2.154
fast		0.164	0.301	0.450	0.589	0.741
std.	32003	0.589	1.468	2.435	3.063	4.238
fast		0.603	1.198	1.792	2.395	2.994
std.	64007	1.167	2.970	4.921	6.001	8.349
fast		1.804	3.853	5.551	7.582	9.827
std.	127997	2.579	5.889	9.490	11.891	16.818
fast		2.331	4.661	7.321	9.984	12.284
std.	256019	4.279	11.105	17.646	23.115	33.541
fast		5.401	10.933	16.174	24.147	26.898
std.	512009	8.885	23.368	31.942	48.059	66.378
fast		10.947	22.066	35.543	45.164	56.190

Table: Times (in seconds) to generate normally distributed points with random covariance matrix. The top row is the time required by using the standard approach, whereas the bottom row shows the time required using the fast QMC matrix-vector approach.

Another application: Partial differential equations with random coefficients

In PDE examples:

- $\mathcal{O}(MNs)$ operations for standard implementation;
- $\mathcal{O}(MN \log N)$ operations + $\mathcal{O}(MN)$ memory using fast QMC matrix-vector multiplication;

I.e., we can replace s by $\log N$ in the cost. If $s = N^{\kappa}$, then $\log N \asymp \log s$.

N	509	1021	2053	4001	8009	16001
std.	190	1346	10610	74550	$\approx$ 144 hrs	$\approx$ 1000 hrs
fast	0.462	1.562	5.591	19.678	87.246	342.615

Table: Times (in seconds) where $M = s = 2N$

N	509	1021	2053	4001	8009	16001
std.	1.272	3.570	10.813	30.127	89.42	273.873
fast	0.059	0.126	0.265	0.516	1.113	2.443

Table: Times (in seconds) where $M = s = \lceil\sqrt{N}\rceil$

N	67	127	257	509
std.	6	82	1699	27935
fast	0.243	1.385	11.268	107.042

Table: Times (in seconds) where $s = N$ and $M = N^2$

N	509	1021	2053	4001	8009	16001
std.	0.436	1.734	15.173	84.381	614.636	4391.2
fast	0.326	1.122	4.296	15.203	60.546	270.691

Table: Times (in seconds) where $M = s = 2N$

.

N	509	1021	2053	4001	8009	16001
std.	0.182	0.375	0.791	1.609	4.100	7.874
fast	0.106	0.228	0.480	0.940	2.670	4.597

Table: Times (in seconds) where $M = s = \lceil \sqrt{N} \rceil$

.

N	67	127	257	509	1021
std.	0.162	0.945	9.935	84.790	891.175
fast	0.204	1.084	10.154	83.861	746.907

Table: Times (in seconds) where $s = N$ and $M = N^2$

.

Thank You!