Exercise Sheet - Simulation of Fractional Brownian Motion

Mini-course on machine learning methods in finance @ KAUST

Paul P. Hager

18th May 2022

The ultimate goal of this exercise sheets is to produce an algorithm that can efficiently simulate sample paths of a fractional Brownian motion on finite time grids.

Readers that already sufficiently familiar with the simulation of multivariate Gaussian random variables might consider skipping Section 1.1. Who is also familiar with Gaussian processes may start at Section 1.3.

After completing Problem 1 and 2 you will be able to simulate the sample paths of a fractional Brownian motion very efficiently. Completing Exercise 1-12 will help you to develop the theoretical background of the algorithm.

In case none of this is new to you, you might still consider taking a look at Problem 3 at the end of the sheet, in which your are engaged to ponder about the problem of optimally stopping a fractional Brownian motion.

1 Simulation of Gaussian processes

1.1 Multivariate Gaussian distribution

Before turning towards Gaussian processes, let us quickly recall the definition of the multivariate Gaussian distribution.

Definition 1. *Let $X_1, \ldots, X_n$ be real-valued random variables defined on a common probability space. The vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ is said to have* multivariate Gaussian distribution *if for all $\boldsymbol{a} \in \mathbb{R}^n$ the scalar product $\langle \boldsymbol{a}, \boldsymbol{X} \rangle$ has a normal distribution.*

Exercise 1. Show that the covariance matrix $\Sigma = (\mathrm{Cov}(X_i, X_j))_{1 \leqslant i,j \leqslant n} \in \mathbb{R}^{n \times n}$ is positive semi-definite.

Exercise 2. Show that if the covariance matrix Σ is not positive definite, that at least one of the variates X_i can be represented as a linear combination of the other variates.

Let $Z_1, \ldots, Z_n$ be i.i.d. standard normal random variables. Since the some of two independent normal random variables is again normal, it is straight forward to verify that $\boldsymbol{Z} = (Z_1, \ldots, Z_n)$ has a multivariate Gaussian distribution.

Exercise 3. Let $A \in \mathbb{R}^{m \times n}$ be a matrix, $\mu \in \mathbb{R}^n$ a vector and define $\boldsymbol{X} = \mu + A\boldsymbol{Z}$. Show that $\boldsymbol{X}$ has a multivariate Gaussian distribution with covariance matrix $\Sigma = AA^T$ and mean $\mu = \mathbb{E}[\boldsymbol{X}]$.

Recall that a positive semi-definite matrix Σ has a *Cholesky decomposition* $\Sigma = LL^T$, where L is a lower triangular matrix. Using the previous exercise, we therefore can formulate the following algorithm for simulating a multivariate Gaussian.

Algorithm 1

```
Input:
–   $\Sigma \in \mathbb{R}^{n\times n}$ positive definite covariance matrix
–   $\mu \in \mathbb{R}^n$ mean vector
–   $m \in \mathbb{N}$ number of samples
Output:
–   $m$ samples of multivariate Gaussian with mean $\mu$ and covariance $\Sigma$.
1. Compute $L$ such that $\Sigma = LL^T$.
2. Repeat the following $m$ times:
    a) Simulate $n$ independent samples from the standard normal distribution $\boldsymbol{Z} = (Z_1, \dots, Z_n)$.
    b) Save new sample $\mu + L\boldsymbol{Z}$.
```

To implement this algorithm we further need to understand how to proceed with steps 1. and 2.a). The simulation of independent normal random variables is beyond the scope of this exercise sheet (look up *Box-Muller transform* for instance). In *Python* you can use the numpy package and the command `numpy.random.randn(n)`.

Regarding the computation of the Cholesky decomposition, one can use the fact that L is lower triangular to compute its entries iteratively from the top to the bottom. Here is a simple Python code that works for positive definite matrices:

```python
import numpy as np

def cholesky(A):
    L = np.zeros_like(A)
    n, _ = A.shape
    for i in range(n):
        for j in range(i+1):
            sum_ = np.sum(L[i] * L[j])
            if i == j:
                L[i, j] = np.sqrt(A[i, i] - sum_)
            else:
                L[i, j] = (A[i,j] - sum_) / L[j, j]
    return L
```

Note that the costs of this algorithm of the order $O(n^3)$ (a third inner loop is hidden in the numpy sum). Let us also note that there is - of course - a function in numpy that implements Algorithm 1: `numpy.random.multivariate_normal(mean, cov)`.

1.2 Gaussian processes

Definition 2. *Let $I \subset \mathbb{R}$ be an interval. A family of real valued random variables $X = (X_t)_{t\in I}$ defined on a common probability space is called a* Gaussian process *if for any $(t_1, \dots, t_n) \in I^n$ and $n \in \mathbb{N}$ the random variable*

$$(X_{t_1}, \dots, X_{t_n}) \tag{1}$$

has a multivariate Gaussian distribution.

Note that the finite-dimensional distributions of the process X, i.e., the distribution of all vectors (1), are characterized by the mean and the covariance function, which are respectively defined by

$$\mu(t) := \mathbb{E}[X_t] \qquad \text{and} \qquad C(s,t) = \mathbb{E}[X_s X_t] - \mathbb{E}[X_s]\mathbb{E}[X_t] \qquad s, t \in I.$$

Simulating the Gaussian process X on a time grid $\{t_0, ..., t_n\} \subset I$ for some $n \in \mathbb{N}$ with $t_0 < t_1 < \cdots < t_n$, then simply amounts to the simulation of the $(n+1)$-dimensional Gaussian random variable $\boldsymbol{X} := (X_{t_0}, ..., X_{t_n})$, which we already have discussed in the previous section. However, as described in the next section, in certain cases one can improve on the efficiency of the simulation.

1.3 Efficient simulation of certain stationary Gaussian processes

In Monte-Carlo and machine learning task it is often necessary to simulate a large amounts of samples on fine grids in order to decrease errors and improve accuracy. The computational effort of Algorithm 1 for generating m samples paths of a Gaussian process on a grid of size n is of the order $O(n^3) + m \times O(n^2)$, the first term appearing form the computation of the Cholesky $\Sigma = L^T L$ decomposition and the second term from the matrix multiplication LZ with the standard Gaussian vector Z. In particular, its is the factor in front of the m that can make simulations infeasible.

In this section, we are going to discuss an algorithm that allows to simulate paths from certain stationary Gaussian process with cost of the order $m \times O(n \log n)$. The algorithm is based on a matrix decomposition $\Pi = AA^T$ (where Σ is embedded in Π) that can be computed more efficiently, but more importantly also allows to evaluate the matrix multiplication $A\boldsymbol{Z}$ more efficiently.

Definition 3. *A process $X = (X_t)_{t \in \mathbb{R}}$ is called stationary if for any $(t_1, ..., t_n) \in \mathbb{R}^n$ and $\tau \in \mathbb{R}$ it holds*

$$(X_{t_1}, ..., X_{t_n}) \overset{d}{=} (X_{t_1+\tau}, ..., X_{t_n+\tau}),$$

where $\overset{d}{=}$ denotes the equality in distribution.

Exercise 4. Show that a Gaussian process X is stationary if and only if the mean is constant $\mathbb{E}(X_t) \equiv \mu \in \mathbb{R}$ and the covariance function satisfies $C(s,t) = C(0, t-s)$ for all $s, t \in \mathbb{R}$.

Assume that the grid $\{t_0, t_1, ..., t_n\} \subset \mathbb{R}$ is regular in the sense that there exists some Δ such that $t_k = t_0 + k\Delta$ for $k = 0, 1, ..., n$. Denote by Σ the covariance matrix of the multivariate Gaussian vector $\boldsymbol{X} := (X_{t_0}, ..., X_{t_n})$, where X is a stationary Gaussian process with covariance function C. Define $c_k := C(t_0, t_k)$ for $k = 0, 1, ..., n$. Then the covariance matrix is of the form

$$\Sigma = \begin{pmatrix} c_0 & c_1 & \cdots & c_n \\ c_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & c_1 \\ c_n & \cdots & c_1 & c_0 \end{pmatrix}.$$

A matrix of this form is called *Toeplitz*. This form is not yet what we need to obtain a more efficient decomposition. Consider the following matrix that is obtained from the reflecting and shifting the first row of Σ:

$$\Pi = \begin{pmatrix} c_0 & c_1 & \cdots & c_{n-1} & c_n & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_2 & \cdots & c_{n-2} & c_{n-1} & c_n & \cdots & c_3 & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & c_0 & c_1 & c_2 & \cdots & c_{n-1} & c_n \\ c_n & c_{n-1} & \cdots & c_1 & c_0 & c_1 & \cdots & c_{n-2} & c_{n-1} \\ c_{n-1} & c_n & \cdots & c_2 & c_1 & c_0 & \cdots & c_{n-3} & c_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_1 & c_2 & \cdots & c_n & c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{pmatrix} \in \mathbb{R}^{2n \times 2n}$$

This matrix satisfies the property that the ith row can be obtained by periodically shifting the first row i-entries towards the right. Such matrices are called *circulant*. Clearly, our original covariance matrix Σ is embedded in the upper-left block of Π.

Define the complex number $\omega_n = e^{-\pi i/n}$ and the vector $q_k = \left(\frac{1}{\sqrt{2n}}\omega_n^{jk}\right)_{0\leqslant j\leqslant 2n-1}$ for $k=0,\dots,2n-1$.

Exercise 5. Show for any $k=0,\dots,2n-1$ that q_k is an eigenvector of Π with eigenvalue $\lambda_k = \sum_{j=0}^{2n-1}\gamma_j\omega_n^{jk}$, where $\gamma = (c_0, c_1, \dots, c_{n-1}, c_n, c_{n-1}, \dots, c_2, c_1)$.

Exercise 6. Show that the matrix $Q = \left(\frac{1}{\sqrt{2n}}\omega_n^{jk}\right)_{0\leqslant j,k\leqslant 2n-1}$ is unitary, i.e., $Q.Q^* = I$.

From the above two exercises we see that

$$\Pi = Q\,\mathrm{diag}(\lambda_0, \dots, \lambda_N)\,Q^*. \tag{2}$$

Note that if the eigenvalues $\lambda_0, \dots, \lambda_N$ of Π are non-negative, then Π is again a positive semi-definite matrix and thus is a covariance matrix.

Exercise 7. Verify that if $\hat{\boldsymbol{X}}$ is a multivariate Gaussian vector with covariance matrix Π then first $n+1$ coordinates $\boldsymbol{X} = \hat{\boldsymbol{X}}_{\mathbf{1:n+1}}$ are Gaussian with covariance matrix Σ.

Assume now that Π is positive definite. The decomposition (2) is almost the type of decomposition that we need for the simulating from Π, however, the matrix Q is complex valued.

Exercise 8. Define $A = Q\,\mathrm{diag}(\sqrt{\lambda_0}, \dots, \sqrt{\lambda_N})\,Q^*$ and show that A is a real matrix and satisfies $\Pi = A\,A^T$.

Hence, we have arrived with a suitable decomposition of the matrix Π. However, at this point it is not clear why this decomposition allows more efficient computations. Note that the multiplication of a vector $\boldsymbol{x} \in \mathbb{R}^{2n}$ with the matrix Q corresponds to a discrete Fourier transform, i.e.,

$$(Q\,\boldsymbol{x})_k = \frac{1}{\sqrt{2n}}\sum_{j=0}^{2n-1}\boldsymbol{x}_j\omega_n^{jk} = \frac{1}{\sqrt{2n}}\sum_{j=0}^{2n-1}\boldsymbol{x}_j e^{-2\pi i\frac{jk}{2n}} = \frac{1}{\sqrt{2n}}\mathrm{DFT}(\boldsymbol{x})_k.$$

Analogously the multiplication with Q^* corresponds to an inverse discrete Fourier transform, i.e.,

$$(Q^*\,\boldsymbol{x})_k = \frac{1}{\sqrt{2n}}\sum_{j=0}^{2n-1}\boldsymbol{x}_j\overline{\omega_n^{jk}} = \frac{1}{\sqrt{2n}}\sum_{j=0}^{2n-1}\boldsymbol{x}_j e^{2\pi i\frac{jk}{2n}} = \sqrt{2n}\,\mathrm{IDFT}(\boldsymbol{x})_k.$$

These multiplications can therefore be algorithmically computed using the *fast Fourier transform* (fft). While the standard evaluation of $Q^*\boldsymbol{x}$ would cost $O(n^2)$, using the fft this product can be evaluated with costs $O(n\log n)$. The following exercise demonstrates how this possible.

Exercise 9. Let $\boldsymbol{x} \in \mathbb{R}^{2n}$ and show that for $k=0,\dots,n-1$ it holds

$$\mathrm{DFT}(\boldsymbol{x})_k - \mathrm{DFT}(\boldsymbol{x})_{k+n} = 2\sum_{j=0}^{n-1}\boldsymbol{x}_{2j}e^{-2\pi i\frac{jk}{n}}, \qquad \mathrm{DFT}(\boldsymbol{x})_k + \mathrm{DFT}(\boldsymbol{x})_{k+n} = 2\sum_{j=0}^{n-1}\boldsymbol{x}_{2j+1}e^{-2\pi i\frac{jk}{n}}.$$

Verify from the above identity that $\mathrm{DFT}(\boldsymbol{x})$ can be computed by using a divide-and-conquer routine.

We are finally ready to formulate an efficient algorithm for the simulation of the Gaussian process X on the gird $\{t_0 + k\Delta | k=0,\dots,n\}$ given that associated matrix Π is positive semi-definite.

Algorithm 2

Input:

- Mean $\mu \in \mathbb{R}$ and covariance function C of a stationary Gaussian process
- $\{t_0, \dots, t_n\}$ a regular time grid
- m the number of samples

```
Output:
– $m$ samples paths of the Gaussian process on the grid $\{t_0, ..., t_n\}$.
1. Define $\boldsymbol{\gamma} = (c_0, ..., c_{n-1}, c_n, c_{n-1}, ..., c_1) \in \mathbb{R}^{2n}$ with $c_k := C(t_0, t_k)$ for all $k = 0, ..., 2n-1$.
2. Compute $\boldsymbol{\lambda} = \mathrm{DFT}(\boldsymbol{c})$ and verify that $\boldsymbol{\lambda}_k \geqslant 0$ for all $k = 0, ..., 2n-1$.
3. Repeat the following $m$ times:
    a) Simulate $2n$ independent samples from the standard normal distribution
       $\boldsymbol{Z} = (Z_0, ..., Z_{2n-1})$.
    b) Compute $\hat{\boldsymbol{X}} = \mathrm{DFT}\big(\mathrm{diag}\big(\sqrt{\lambda_0}, ..., \sqrt{\lambda_{2n-1}}\big)\mathrm{IDFT}(\boldsymbol{Z})\big)$.
    c) Save new sample path $\mu + \hat{\boldsymbol{X}}_{0:n}$.
```

Problem 1. Implement Algorithm 2 as a Python function with input arguments m, μ and $\mathbf{c} = (c_0, ..., c_n)$ using the `scipy.fft` module.

Hint: Use the automatic broadcasting in numpy/scipy. More precisely, feeding `scipy.fft.iftt` with an array of shape $(m, 2n)$ will apply the fft to each row.

Hint: Note for a real vector $\boldsymbol{x}$ the vector $\mathrm{DFT}(\boldsymbol{x})$ will have an hermitian symmetry and vice versa if $\boldsymbol{y}$ is a vector with hermitian symmetry then $\mathrm{IDFT}(\boldsymbol{y})$ is a real vector. The `scipy.fft` module provides special commands for these symmetries. In particular the line 3.b) can be expressed using `scipy.fft.ihftt` and `scipy.fft.hftt`.

Hint: Due to numerical errors you may check whether $\boldsymbol{\lambda}$ is real valued with `np.allclose(lam, np.real(lam))`.

2 Simulation of Fractional Brownian Motion

A *fractional Brownian motion* (fBm) $X = (X_t)_{t\in\mathbb{R}}$ is a mean zero Gaussian process with a covariance function given by

$$\mathbb{E}[X_t X_s] = \frac{1}{2}(|t|^{2H} + |s|^{2H} + |t-s|^{2H}), \qquad s, t \in \mathbb{R},$$

where $H \in (0, 1]$ is called the *Hurst parameter*. In this section we will focus on the simulation of this Gaussian process and therefore are neither going to discuss its properties in detail nor going to elaborate on its use in applications. Nevertheless, let just mention that for $H = 0.5$ the process is a standard Brownian motion and for $H \neq 0.5$ the process is not a semimartingale and not a Markov process, which renders it as a typical example for methodologies that claim applicability top processes beyond these two classes. Furthermore, the sample paths of X are Hölder continuous for any coefficient strictly below H.

The following property will be important for simulation:

Exercise 10. The process is self similar, in the sense that for any $\lambda > 0$ the process $(\lambda^H X_{\lambda t})_{t\in\mathbb{R}}$ is also a fBm.

Indeed, using the above property, in order to simulate the fBm on a time grid $\{0, \Delta, 2\Delta, ..., n\Delta\}$ for some $\Delta > 0$ it suffices to simulate the process on the unit grid $\{0, 1, 2, ..., n\}$ and then rescale the resulting paths by Δ^H. For the latter, we want to use the efficient algorithm from the previous section, however, it is apparent that the fBm itself is not stationary. Nevertheless, we have the following

Exercise 11. Show that the increment process $\delta X := (X_{t+1} - X_t)_{t\geqslant 0}$ has the covariance structure

$$\mathbb{E}[\delta X_s \delta X_t] = \frac{1}{2}(|(t-s)+1|^{2H} + |1-(t-s)|^{2H}) - |t-s|^{2H}, \quad s, t \in \mathbb{R}$$

and is hence stationary.

Summarizing the two previous observations we then obtain:

Exercise 12. Let $\Delta > 0$ and define the mean zero Gaussian vector $\tilde{\boldsymbol{X}}$ by

$$\tilde{\boldsymbol{X}}_k := \Delta^H \sum_{j=1}^{k} \delta X_{j-1}, \qquad k = 0, \ldots, n.$$

Then it holds that $\mathbb{E}[\tilde{\boldsymbol{X}}_j \tilde{\boldsymbol{X}}_k] = \mathbb{E}[X_{\Delta j} X_{\Delta k}]$. Hence $\tilde{\boldsymbol{X}}$ represents an evaluation of an fBm on the time grid $\{0, \Delta, 2\Delta, \ldots, n\Delta\}$.

It can be shown that covariance matrix of the fBm increment process on a regular grid can always be embedded into a positive definite circulant matrix as described in the previous section. The proof of this result is rather difficult an is beyond the scope of this exercise sheet. Nevertheless, this fact grands us that we can apply the efficient Algorithm 2 to simulate increments of an fBm. Thus we are finally able propose our algorithm for the simulation of an fBm:

Algorithm 3

Input:

- H the Hurst parameter
- n the size of the time grid
- Δ the mesh of the time grid
- m the number of samples.

Output:

- m samples paths of the fractional Brownian motion on the grid $\{0, \Delta, 2\Delta, \ldots, n\Delta\}$.

1. Define $\boldsymbol{c} = \left(\frac{1}{2}(|k+1|^{2H} + |k-1|^{2H}) - |k|^{2H}\right)_{0 \leqslant k \leqslant n-1}$.
2. Use Algorithm 2 with $\boldsymbol{c}$ to simulate $(\delta \boldsymbol{X}^{(k)})_{1 \leqslant k \leqslant m}$, where each $\delta \boldsymbol{X}^{(k)}$ is a sample of the increment process on $\{0, \Delta, 2\Delta, \ldots, (n-1)\Delta\}$.
3. For each $k = 1, \ldots, m$ save a new sample $\mathbf{X}^{(k)} := \left(\Delta^H \sum_{l=1}^{j} \delta \boldsymbol{X}_{l-1}^{(k)}\right)_{0 \leqslant j \leqslant n}$.
4. Return $(\boldsymbol{X}^{(k)})_{1 \leqslant k \leqslant m}$.

Problem 2. Implement the Algorithm 3 using your solution of Problem 1. Test the final outcome by first simulating a the fBm on a small grid for a large number of samples and then calculating the empirical covariance matrix using `numpy.cov` for different values of H. Then sample the fBm for $n = 100$, $\Delta = \frac{1}{n}$ and $H \in \{0.1, 0.5, 0.9\}$ and plots the sample paths for comparison.

Optimal stopping of fractional Brownian motion

Let X^H be a fBm with Hurst parameter $H \in (0, 1]$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and consider the filtration $(\mathcal{F}_t)_{t \geqslant 0}$ generated by X^H on the positive halfline, i.e., $\mathcal{F}_t := \sigma(X_s^H | 0 \leqslant s \leqslant t)$. We consider the problem of optimally stopping X^H so that its expected value is maximized, i.e., the problem of calculating

$$\sup_{\tau \in \mathcal{S}} \mathbb{E}[X_\tau^H],$$

where $\mathcal{S}$ is the set of all stopping times with respect to the filtration $(\mathcal{F}_t)_{t \geqslant 0}$, i.e., $\mathcal{S}$ consists of all random variables $\tau : \Omega \to [0, \infty)$ such that $\{\tau \leqslant t\} \in \mathcal{F}_t$ for all $t \geqslant 0$. In simple words, this means that the decision on whether to stop the process at time t can only depend on the observed evolution of the process on the interval $[0, t]$.

Problem 3. Think about possible stopping times that approximate the optimal stopping value of the fBm. In particular consider the special case $H = 0.5$ and the extreme case $H = 1$. What is your guess for the dependence of the optimal stopping value on H and where does it tend to for $H \to 0$? You can also use your fBm simulator from try out some strategies.