Sorbonne Université

Calcul haute performance : programmation et algorithmique avancées

Projects: Merge Sort, Monte Carlo and PDE simulations

The presentation of the results on December 12th, 2024

/!\General introduction common to all projects

Students are required to submit the slides and the source code for their project to their instructor (by an email to L. ABBAS TURKI) by December 11th. The source code must be readable and, therefore, adequately commented to ensure clarity. In addition to evaluating the source code, grading will also be based on a presentation delivered during the session on December 12th. The entire presentation, including questions, should not exceed 15 minutes and should be structured as follows: 6 to 8 minutes for presenting results, 2 to 4 minutes for explaining the code, and 4 to 5 minutes for questions. Accordingly, the number of slides dedicated to presenting results should be limited to 8 maximum, and the duration of the presentation (excluding the question period) should fall between 10 and 11 minutes. During the 10 to 11 minutes of the presentation, speaking time must be evenly divided among students within the same group. If this balance is not maintained, all questions will be directed solely to the student who has spoken the least.

1 Monte Carlo simulation of Heston model

The Heston model for asset pricing has been widely examined in the literature. Under this model, the dynamics of the asset price S_t and the variance v_t are governed by the following system of stochastic differential equations:

$$dS_t = rS_t dt + \sqrt{v_t} S_t d\hat{Z}_t \tag{1}$$

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t \tag{2}$$

$$\hat{Z}_t = \rho W_t + \sqrt{1-\rho^2} Z_t \tag{3}$$

where:

- the spot values $S_0 = 1$ and $v_0 = 0.1$,
- r is the risk-free interest rate, we assume $r = 0$,
- κ is the mean reversion rate of the volatility,
- θ is the long-term volatility,
- σ is the volatility of volatility,
- W_t and Z_t are independent Brownian motions

In this project, we aim to compare two distinct methods for simulating an at-the-money call option (where "at-the-money" here means $K = S_0 = 1$) at maturity $T = 1$ under the Heston model. The option has a payoff given by $f(x) = (x - K)_+$, and, thus, we want to simulate with Monte Carlo the expectation $E[f(S_T)] = E[(S_1 - 1)_+]$. This comparison will focus on the efficiency and accuracy of each simulation method in pricing the call option within the stochastic volatility framework of the Heston model.

We begin with the Euler discretization scheme, which updates the asset price S_t and the volatility v_t at each time step as follows:

$$S_{t+\Delta t} = S_t + rS_t\Delta t + \sqrt{v_t}S_t\sqrt{\Delta t}(\rho G_1 + \sqrt{1-\rho^2}G_2) \tag{4}$$

$$v_{t+\Delta t} = g\left(v_t + \kappa(\theta - v_t)\Delta t + \sigma\sqrt{v_t}\sqrt{\Delta t}G_1\right) \tag{5}$$

where G_1 and G_2 are independent standard normal random variables, and the function g is either taken to be equal to $(\cdot)_+$ or to $|\cdot|$.

1. Assuming $\kappa = 0.5$, $\theta = 0.1$, $\sigma = 0.3$ and using a discretization $\Delta t = 1/1000$, write down a Monte Carlo simulation code using Euler discretization to approximate $E[(S_1 - 1)_+]$. (/8.5 MAIN5, /7 other M2)
2. specified soon (/6.5 MAIN5, /5 other M2)
3. specified soon (/5 MAIN5, /4 other M2)

2 Nested Monte Carlo for exponential Ornstein-Uhlenbeck model

Numerous mean-reverting stochastic volatility models are utilized in financial mathematics to describe asset price dynamics. Here, we focus on the exponential Ornstein-Uhlenbeck (OU) volatility model, in which the volatility process follows an Ornstein-Uhlenbeck-type dynamics. Specifically, this model is characterized by the following equations:

$$dS_t = S_t\left(rdt + \exp(Y_t)dW_t\right)$$

$$dY_t = \alpha(m - Y_t)dt + \beta d\hat{Z}_t$$

$$\hat{Z}_t = \rho W_t + \sqrt{1-\rho^2}Z_t$$

where:

- the spot values $S_0 = 1$ and $Y_0 = \log(0.1)$,
- r is the risk-free interest rate, we assume $r = 0$,
- α is the mean reversion rate of the volatility,
- m is the long-term log-volatility,
- β is the volatility of the log-volatility,
- W_t and Z_t are independent Brownian motions.

For further details on this model, we refer the reader to [2]. In this project, our goal is to train a neural network that can accurately predict the price of a call option for any maturity, strike, and a range of model parameters α, β, m, ρ, and Y_0.

To achieve this, starting from question 2, we will generate a large number (nested) of Monte Carlo simulated prices on GPUs, storing the results in .csv files to create a comprehensive dataset of call option prices. This dataset will then serve as the training set for the neural network, enabling it to learn and generalize the relationship between option prices and the various model parameters across different maturities and strikes.

Before nested Monte Carlo of question 2, we start in question 1 performing a regular Monte Carlo simulation. This Monte Carlo simulation requires an Euler discretization scheme, which updates the asset price S_t and the log-volatility Y_t at each time step as follows:

$$S_{t+\Delta t} = S_t + rS_t\Delta t + \exp(Y_t)S_t\sqrt{\Delta t}G_1 \tag{6}$$

$$Y_{t+\Delta t} = Y_t + \alpha(m - Y_t)\Delta t + \beta\sqrt{\Delta t}(\rho G_1 + \sqrt{1-\rho^2}G_2) \tag{7}$$

where G_1 and G_2 are independent standard normal random variables.

1. Assuming $\alpha = 1$, $m = 0.1$, $\beta = 0.1$ and using a discretization $\Delta t = 1/1000$, write down a Monte Carlo simulation code using Euler discretization to approximate $E[(S_1 - 1)_+]$ which is the price of a call option of maturity $T = 1$ and strike $K = S_0 = 1$. (/8.5 MAIN5, /7 other M2)
2. specified soon (/6.5 MAIN5, /5 other M2)
3. specified soon (/5 MAIN5, /4 other M2)

3 Nested Monte Carlo for bullet options

In mathematical finance, the Black & Scholes (B&S) model is commonly used to describe the dynamics of asset prices over a time interval $[0, T]$. For time increments s and t with $0 \leq s < t \leq T$, the asset price process under the B&S model can be simulated using the following time induction formula:

$$S_t = S_s \exp\left((r - \sigma^2/2)(t-s) + \sigma\sqrt{t-s}G\right) \text{ and } S_0 = x_0 \text{ where}$$

- σ is the volatility assumed here equal to 0.2,
- r is the risk-free rate assumed here equal to 0.1,
- x_0 is the initial spot price of S at time 0, assumed here equal to 100
- G is independent from S_s and has a standard Normal distribution $\mathcal{N}(0, 1)$.

This formula allows for simulating the path of the asset price from time s to t. This approach can be iteratively applied to generate an entire price path S for any time schedule $T_0 = 0 < T_1 < ... < T_M = T$.

Assuming the B&S model, we will begin by using Monte Carlo simulation, in question 1, to approximate the price of a bullet option at the initial time $t = 0$. In questions 2 and 3, we will employ two different nested Monte Carlo simulations to approximate the price of the bullet option across multiple increments of time and space.

The price of a bullet option $F(t, x, j) = e^{-r(T-t)}E(X \big| S_t = x, I_t = j)$, $X = (S_T - K)_+ 1_{\{I_T \in [P_1, P_2]\}}$ with $I_t = \sum_{T_i \leq t} 1_{\{S_{T_i} < B\}}$ and

- K is the contract's strike assumed here equal to $x_0 = 100$,
- T is the contract's maturity assumed here equal to 1
- barrier B should be bigger than S I_T times $\in \{P_1, ..., P_2\} \subset \{0, ..., M\}$ where P_1 and P_2 are two integers.

First, we want to use MC to approximate $F(0, x_0, 0) = e^{-rT}E(X)$ with $\{X_i\}_{i \leq n}$ being independent random variables that have the same distribution as X.

When $P_1 = P_2 = B = 0$, convince yourself that $E(X) = E((S_T - K)_+)$. In this case, use MC to approximate $F(0, x_0, 0)$ and check that you get a value that is close enough to

$$N(d_1)x_0 - N(d_2)Ke^{-rT} \text{ with } N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy,\ d_1 = \frac{\ln(x_0/K) + (r + \sigma^2/2)(T)}{\sigma\sqrt{T}}, \tag{8}$$

and $d_2 = d_1 - \sigma\sqrt{T}$. N is `NP` function already given in this course.

1. Assuming $B = 120$, $M = 100$, $P1 = 10$, $P2 = 50$ and $T_i = i/M$ for $i = 0, ..., M$, write down a Monte Carlo simulation code to approximate $F(0, x_0, 0)$. (/8.5 MAIN5, /7 other M2)
2. specified soon (/6.5 MAIN5, /5 other M2)
3. specified soon (/5 MAIN5, /4 other M2)

4 Merge large

This subject starts with the same algorithm of merge path, presented in [3], implemented on only one block. The students are then asked to translate it into the merge of large arrays.

We start then with the merge path algorithm. Let A and B be two ordered arrays (increasing order), we want to merge them in an M sorted array. The merge of A and B is based on a path that starts at the top-left corner of the $|A| \times |B|$ grid and arrives at the down-right corner. The Sequential Merge Path is given by Algorithm 1 and an example is provided in Figure 1.

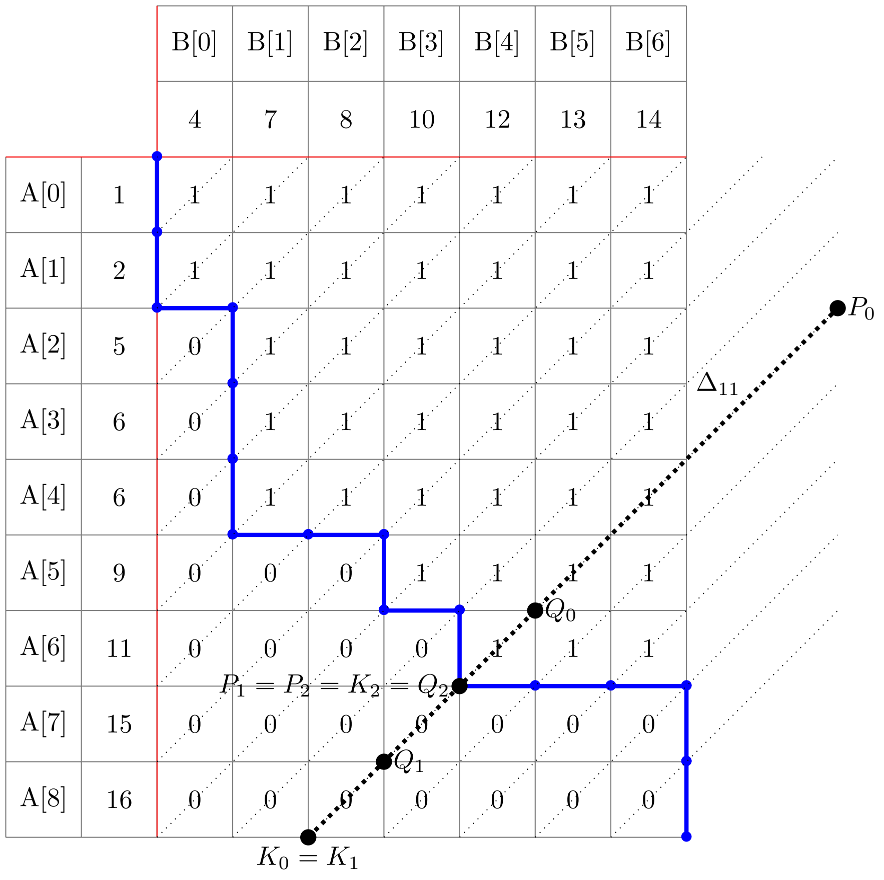

Figure 1: An example of Merge Path procedure

Algorithm 1 Sequential Merge Path

```
Require: $A$ and $B$ are two sorted arrays
Ensure: $M$ is the merged array of $A$ and $B$ with $|M| = |A| + |B|$
  procedure MergePath $(A, B, M)$
    $j = 0$ and $i = 0$
    while $i + j < |M|$ do
      if $i \geq |A|$ then
        M[i+j]=B[j]
        $j = j + 1$                                    ▷ The path goes right
      else if $j \geq |B|$ or $A[i] < B[j]$ then
        M[i+j]=A[i]                                    ▷ The path goes down
        $i = i + 1$
      else
        M[i+j]=B[j]
        $j = j + 1$                                    ▷ The path goes right
      end if
    end while
  end procedure
```

Algorithm 2 Merge Path (Indices of n threads are integers from 0 to $n-1$)

```
Require: $A$ and $B$ are two sorted arrays
Ensure: $M$ is the merged array of $A$ and $B$ with $|M| = |A| + |B|$
  for each thread in parallel do
    i=index of the thread
    if $i > |A|$ then
      $K = (i - |A|, |A|)$                             ▷ Low point of diagonal
      $P = (|A|, i - |A|)$                             ▷ High point of diagonal
    else
      $K = (0, i)$
      $P = (i, 0)$
    end if
    while True do
      $offset = abs(K_y - P_y)/2$
      $Q = (K_x + offset, K_y - offset)$
      if $Q_y \geq 0$ and $Q_x \leq B$ and
        $(Q_y = |A|$ or $Q_x = 0$ or $A[Q_y] > B[Q_x - 1])$ then
        if $Q_x = |B|$ or $Q_y = 0$ or $A[Q_y - 1] \leq B[Q_x]$ then
          if $Q_y < |A|$ and $(Q_x = |B|$ or $A[Q_y] \leq B[Q_x])$ then
            $M[i] = A[Q_y]$                            ▷ Merge in $M$
          else
            $M[i] = B[Q_x]$
          end if
          Break
        else
          $K = (Q_x + 1, Q_y - 1)$
        end if
      else
        $P = (Q_x - 1, Q_y + 1)$
      end if
    end while
  end for
```

Each point of the grid has a coordinate $(i, j) \in [\![0, |A|]\!] \times [\![0, |B|]\!]$. The merge path starts from the point $(i,j) = (0,0)$ on the left top corner of the grid. If $A[i] < B[j]$ the path goes down else it goes right. The array $[\![0, |A|-1]\!] \times [\![0, |B|-1]\!]$ of boolean values $A[i] < B[j]$ is not important in the algorithm. However, it shows clearly that the merge path is a frontier between ones and zeros.

To parallelize the algorithm, the grid has to be extended to the maximum size equal to $\max(|A|,|B|) \times \max(|A|,|B|)$. We denote K_0 and P_0 respectively the low point and the high point of the ascending diagonals Δ_k. On GPU, each thread $k \in [\![0, |A|+|B|-1]\!]$ is responsible of one diagonal. It finds the intersection of the merge path and the diagonal Δ_k with a binary search described in Algorithm 2.

1. For $|A| + |B| \leq 1024$, write a kernel `mergeSmall_k` that merges A and B using only one block of threads. (/8.5 MAIN5, /7 other M2)
2. specified soon (/6.5 MAIN5, /5 other M2)
3. specified soon (/5 MAIN5, /4 other M2)

5 Batch merge small and batch sort small

This subject starts with the same algorithm of merge path, presented in [3], implemented on only one block. The students are then asked to translate it into a batch merge and a batch sort. **Consequently, question 1 is common with question 1 of subject 4**

1. For $|A| + |B| \leq 1024$, write a kernel `mergeSmall_k` that merges A and B using only one block of threads. (/8.5 MAIN5, /7 other M2)
2. specified soon (/6.5 MAIN5, /5 other M2)
3. specified soon (/5 MAIN5, /4 other M2)

6 PDE simulation of bullet options

The students have to implement and compare Thomas algorithm to PCR for tridiagonal systems in question 1. Then, they have to simulate a PDE of a bullet option using Crank-Nicolson scheme based on either Thomas or PCR in questions 2 and 3

We refer to [6] for a fair description of PCR. Regarding Thomas algorithm, as described in [5], it allows to solve tridiagonal systems:

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{9}$$

using a forward phase

$$c'_1 = \frac{c_1}{b_1},\ y'_1 = \frac{y_1}{b_1},\ c'_i = \frac{c_i}{b_i - a_i c'_{i-1}},\ y'_i = \frac{y_i - a_i y'_{i-1}}{b_i - a_i c'_{i-1}} \text{ when } i = 2, ..., n \tag{10}$$

then a backward one

$$z_n = y'_n,\ z_i = y'_i - c'_i z_{i+1} \text{ when } i = n-1, ..., 1. \tag{11}$$

1. Using Thomas method, write a kernel that solves various tridiagonal systems ($d \leq 1024$) at the same time, one system per block. Do the same thing for PCR then compare both methods for $d \leq 1024$. To compare these two methods, generate strictly diagonally dominant tridiagonal matrices i.e. $|b_i| > |a_i| + |c_i|$ (/8.5 MAIN5, /7 other M2)

2. specified soon (/6.5 MAIN5, /5 other M2)

3. specified soon (/5 MAIN5, /4 other M2)

References

[1] Mark Broadie and Özgür Kaya. Exact simulation of stochastic volatility and other affine jump diffusion processes. *Operations research*, 54(2):217–231, 2006.

[2] Jean-Pierre Fouque, George Papanicolaou, and K Ronnie Sircar. Mean-reverting stochastic volatility. *International Journal of theoretical and applied finance*, 3(01):101–142, 2000.

[3] O. Green, R. McColl and D. A. Bader GPU Merge Path - A GPU Merging Algorithm. *26th ACM International Conference on Supercomputing (ICS),* San Servolo Island, Venice, Italy, June 25-29, 2012.

[4] G. Marsaglia and T. Wai-Wan. A simple method for generating gamma variables. *ACM Transactions on Mathematical Software*, 26(3):363–372, 2000.

[5] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (2002): *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press.

[6] Y. Zhang, J. Cohen and J. D. Owens (2010): Fast Tridiagonal Solvers on the GPU. *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 127–136.