Numerical methods for option pricing

MATH 221 Final Project

Raymond Tsao

1 Introduction

Options are financial contracts that provide the holder with the right, but not the obligation, to buy (call option) or sell (put option) an underlying asset at a predetermined price K (strike price) within a specified time period. They are versatile instruments used for hedging, speculation, and portfolio management in financial markets. Given the complexity of options and their prevalence in the financial world, it is crucial to understand how to accurately price these derivatives.

The Black-Scholes-Merton model, developed by economists Fischer Black, Myron Scholes, and Robert Merton in the early 1970s, provides a mathematical framework to calculate the theoretical value of European-style options. The model is given by a partial differential equation known as the Black-Scholes equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 V}{\partial s^2} + rs\frac{\partial V}{\partial s} - rV = 0$$

Where V denote the value of the option, s denote the price of the underlying stock, r denote the risk-free rate, and σ^2 denote the volatility. In this paper, it is assumed that V represents the value of a put option.

Numerous numerical methods exist for solving the Black-Scholes equation. The finite difference method is a commonly employed numerical technique. It involves two primary approaches: the explicit method and the implicit method. However, it has been reported that the explicit method is prone to numerical instability [1, 2]. As a result, in practical applications, the implicit method is more widely utilized. In the following section, we will formulate the mathematical problem for the implicit method.

2 The Problem

Our primary objective is to estimate the value of $V(s,t)$ on a grid $[0,S] \times [0,T]$. The implicit method begins by dividing the (s,t) space into meshes of size $\delta s \times \delta t$ (for future reference, let M and N represent the number of partitions on the s and t axes, respectively). Letting $V_{ij} = V(i\delta s, j\delta t)$, the implicit method approximates the derivatives as follows:

$$\begin{cases} \frac{\partial V}{\partial t} \approx \frac{V_{i,j+1} - V_{i,j}}{\delta t} \\ \frac{\partial V}{\partial s} \approx \frac{V_{i+1,j} - V_{i-1,j}}{\delta s} \\ \frac{\partial^2 V}{\partial s^2} \approx \frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{2\delta s} \end{cases}$$

When substituted into the PDE, the following recurrence relation is obtained:

$$V_{i,j+1} = \alpha_i V_{i-1,j} + \beta_i V_{i,j} + \gamma_i V_{i,j}$$

Here, the coefficients are defined as:

$$\begin{cases} \alpha_i = \frac{1}{2} r i \delta t - \frac{1}{2}\sigma^2 i^2 \delta t \\ \beta_i = 1 + \sigma^2 i^2 \delta t + r\delta t \\ \gamma_i = -\frac{1}{2} r i \delta t - \frac{1}{2}\sigma^2 i^2 \delta t \end{cases}$$

And the boundary condition is given by

$$\begin{cases} V_{i,N} = \max\{i\delta s - K, 0\} & \text{for } i = 0,1,2,...,M \\ V_{0,j} = 0 & \text{for } j = 0,1,2,...,N \\ V_{M,j} = M\delta s - Ke^{-r(N-j)\delta t} & \text{for } j = 0,1,2,...,N \end{cases}$$

The computation graph for the implicit method is depicted in figure 1.

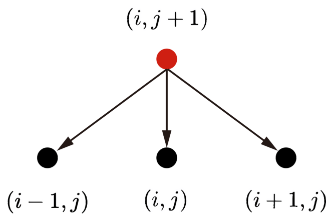

Figure 1: Computation graph of implicit method. The red note represents nodes with known value. In the implicit scheme, the information flows in reverse from larger t to smaller t.

The recurrence equation can be expressed compactly in matrix form

$$\underbrace{\begin{bmatrix} V_{1,j+1} \\ V_{2,j+1} \\ V_{3,j+1} \\ \vdots \\ V_{M-2,j+1} \\ V_{M-1,j+1} \end{bmatrix}}_{\mathbf{v}_{j+1}} = \underbrace{\begin{bmatrix} \beta_1 & \gamma_1 & & & & \\ \alpha_2 & \beta_2 & \gamma_2 & & & \\ & \alpha_3 & \beta_3 & \gamma_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{M-2} & \beta_{M-2} & \gamma_{M-2} \\ & & & & \alpha_{M-1} & \beta_{M-1} \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} V_{1,j} \\ V_{2,j} \\ V_{3,j} \\ \vdots \\ V_{M-2,j} \\ V_{M-1,j} \end{bmatrix}}_{\mathbf{v}_j} - \underbrace{\begin{bmatrix} \alpha_1 V_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \gamma_{M-1} V_{M,j} \end{bmatrix}}_{\mathbf{b}_j}$$

Since $\mathbf{v}_{j+1}$ is always known before $\mathbf{v}_j$, to solve the recurrence, we iterate from $j = N-1$ to 1 and solve the linear system.

$$\mathbf{C}\mathbf{v}_j = \mathbf{v}_{j+1} + \mathbf{b}_j$$

Hence, solving the Black-Scholes equations numerically is equivalent to solving N systems of equations. The most direct approach is to multiply both sides by $\mathbf{C}^{-1}$. However, in cases when N is large, computing $\mathbf{C}^{-1}$ can be computationally expensive. Therefore, the objective of this paper is to assess the performance of various algorithms in solving these equations. In particular, we will focus on

1. LU-decomposition
2. Jacobi method
3. Gauss-Seidel method
4. Successive overrelaxation method (SOR)
5. Multigrid method

We will justify the choices of these algorithms in the next section when we analyze the structure of the coefficient matrix $\mathbf{C}$.

3 Analysis of coefficient matrix

We begin by noting that $\mathbf{C}$ is tridiagonal, which means that the LU-factorization of $\mathbf{C}$ takes a nice form:

$$\mathbf{C} = \underbrace{\begin{bmatrix} 1 & & & & & \\ l_2 & 1 & & & & \\ & l_3 & 1 & & & \\ & & \ddots & \ddots & \ddots & \\ & & & l_{M-2} & 1 & \\ & & & & l_{M-1} & 1 \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} d_1 & u_1 & & & & \\ & d_2 & u_2 & & & \\ & & d_3 & u_3 & & \\ & & & \ddots & \ddots & \\ & & & & d_{M-2} & u_{M-2} \\ & & & & & d_{M-1} \end{bmatrix}}_{\mathbf{U}}$$

Since at each iteration, the coefficient matrix $\mathbf{C}$ is held constant, and the matrices $\mathbf{L}$ and $\mathbf{U}$ are both sparse, solving the N linear equations requires only $O(M)$ sparse matrix multiplications. This suggests that LU-decomposition may exhibit good performance.

To justify the use of iterative methods, we will analyze whether $\mathbf{C}$ satisfies the convergence criteria. We claim that in practice, the matrix $\mathbf{C}$ is strictly row diagonally dominant. To see this, note that

$$\begin{cases} |\alpha_i| + |\gamma_i| = \frac{1}{2}\delta t|\sigma^2 i^2 - ri| + \frac{1}{2}\delta t|\sigma^2 i^2 + ri| \\ |\beta_i| = 1 + \sigma^2 i^2 \delta t + r\delta t \end{cases}$$

If we impose a condition $\sigma^2 i^2 - ri > 0$, or equivalently

$$i > \frac{r}{\sigma^2} \tag{*}$$

We have

$$|\alpha_i| + |\gamma_i| = \sigma^2 i^2 \delta t < |\beta_i|$$

For $i > r/\sigma^2$. This shows that $\mathbf{C}$ is strictly row diagonally dominant when i is large enough. In practice, we can assume this holds for all $i \geq 1$ since the risk-free interest rate usually lies within the range $[0, 0.1]$, whereas the volatility for most stocks is greater than 1 [3].

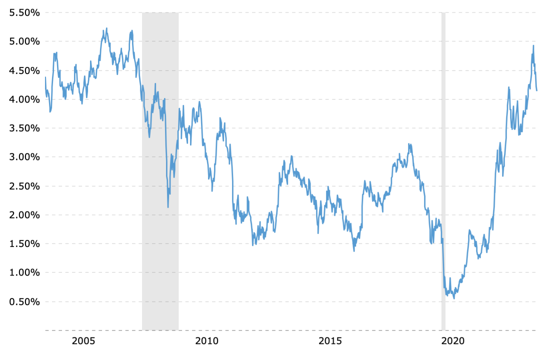

Figure 2: 10 year Treasury rate for the past ten years. From the plot, it can be seen that the risk free rate is strictly lower than 0.1 [3].

Since $\mathbf{C}$ is strictly row diagonally dominant, it follows that both the Jacobi and Gauss-Seidel methods converge. With the convergence of these methods established, it becomes natural to compare their performance with the SOR method and multigrid, which are reportedly faster than these two algorithms. In the next section, we will describe the methods used for testing.

4 Method

4.1 Code

The Black-Scholes solver utilizing LU decomposition, Jacobi, Gauss-Seidel, and SOR methods is implemented in MATLAB. Due to the complexity of implementing the multigrid method, the code for the multigrid linear system solver, authored by Benjamin Beaudry, is downloaded via the MATLAB File Exchange [4].

4.2 Parameters

For the testing of the algorithms, unless otherwise specified, the following parameters are used:

- Initial stock price: $S_0 = 50$
- Strike price: $K = 50$
- Risk-free rate: $r = 0.1$
- Time to maturity: $T = 5/12$ year
- Volatility: $\sigma = 0.4$
- Maximum stock price: $S_{\max} = 100$

In particular, for the SOR method, the relaxation parameter w is set to 1.2. This parameter is determined by testing the performance of the algorithm for different values of $w \in (1, 2)$.

4.3 Ground truth

The true option price, which will be used as a benchmark, is determined using the `blsprice` method in the MATLAB Finance Toolbox.

4.4 Evaluation method

In evaluating and comparing the performance of each algorithm, we will primarily focus on three dimensions:

- The convergence of the algorithm
- The accuracy of the algorithm
- The time complexity of the algorithm

To address each of these questions, various experiments will be conducted. For example, to test the time complexity of the algorithm, the number of mesh points M and N may be varied to determine the time complexity as a function of problem size.

5 Results and discussion

5.1 Sample output

We first present a sample output using the LU decomposition method (outputs from other algorithms will yield similar plots). By calling the LU-solver function with $M = N = 200$, we obtain the following estimate for V at each mesh point:

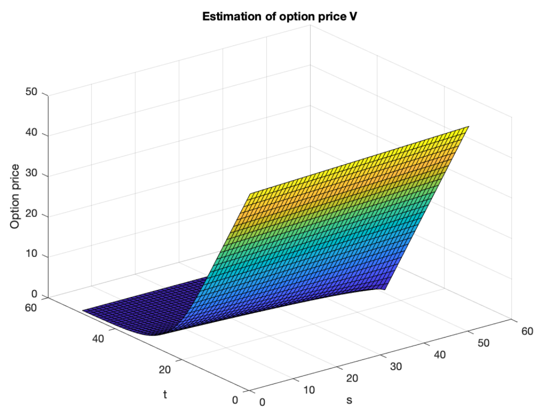

Figure 3: Sample plot of the estimated option price as a function of price s and time t. The number of meshes used is $M = 50, N = 50$.

In this example, the final price of the option is estimated to be 4.0480, whereas the actual value is given by 4.071, resulting in a 0.00568 relative error.

5.2 Convergence analysis

Before assessing the accuracy of the algorithm, we start by investigating its convergence to pinpoint the mesh size M, N at which the algorithm halts its progress. This is done by setting $M = N$ and varying the values from 50 to 200 with a step size of 10. The estimated option values at each mesh size are recorded and plotted in the figure below.

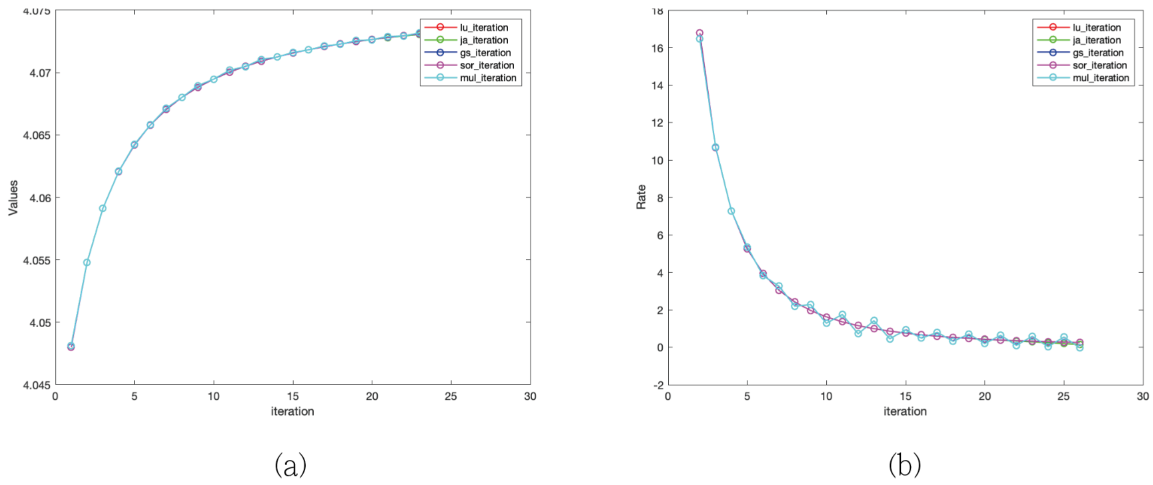

Figure 4: (a) The estimated option value as a function of problem size, ranging from 50 partitions on each side to 200 partitions on each side (from a matrix size of 2601 to 40401), the true option price is given by 4.073. (b) The relative convergence, computed by taking the quotient of $V_{i+1} - V_i$ and V_i.

The plot reveals that all five algorithms display a similar behavior, converging near the 200 partition size as the relative growth rate approaches 0. This indicates that any partition size greater than 200 should be appropriate for accuracy comparison, as the estimated option value stabilizes and ceases to change.

5.3 Accuracy analysis

To assess the accuracy, the parameters r, σ, T are sampled uniformly from $[0, 1]$. The true option price is determined using `blsprice` method in Matlab financial toolbox. This is repeated for 50 iterations. The mean error, computed by

$$\epsilon = \sqrt{\sum_{i=1}^{50} (y_{true} - y_{estimate})^2}$$

Is given by the table below

Algorithm	Mean error
LU-decomposition	0.00215
Jacobi	0.00221
Gauss Seidel	0.00297
SOR Method	0.00241
Multigrid Method	0.00198

From the results, we observe that all algorithms yield a comparable mean error on the test problem, hovering around 0.002. It's important to acknowledge that the algorithms are not the sole contributors to the observed errors in this case, since in the implementation of the PDE solver, linear interpolation is employed to determine the true option price. This introduces another potential source of error. Considering the similarity in results, the choice of the optimal algorithm now hinges on comparing their respective time complexities.

5.4 Time complexity

To test the time complexity of the algorithms, different number of partitions, ranging from 50 to 200 is is used, and the time spent for each algorithm is determined using the `tic` and `toc` function in Matlab. The result is plotted in figure 6.

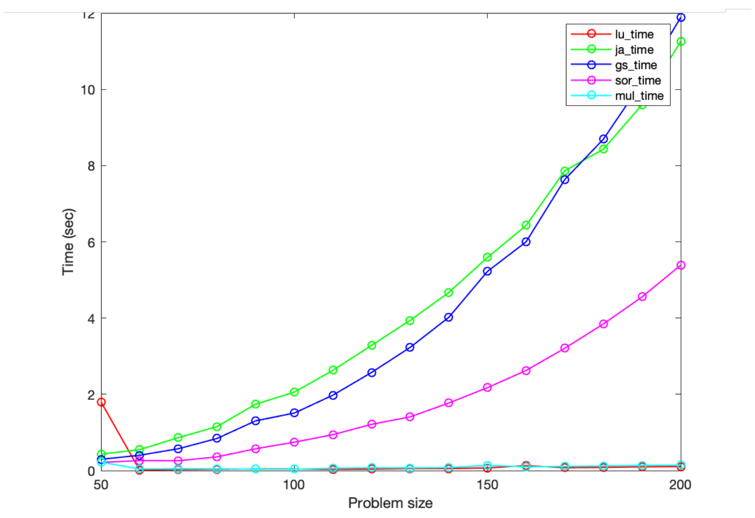

Figure 6: Plot of the time complexity of each algorithm as a function of problem size.

From the plot, it is evident that both LU decomposition and the multigrid method outperform the other iterative algorithms. One explanation for this result is that the LU decomposition of the coefficient matrix **C** is sparse. This implies that each iteration on j only involves two sparse matrix multiplications.

On the other hand, for iterative methods like Jacobi and Gauss-Seidel, each iteration for $\mathbf{v}_j$ requires performing multiple matrix-vector multiplication until the estimates converge, significantly slowing down the process. The time complexity of the SOR method is faster than the Jacobi and Gauss-Seidel methods by an order of magnitude. This demonstrates that the over-relaxation parameter indeed accelerates convergence.

6 Conclusions and practical considerations

In the previous section, we compared the convergence, accuracy and time complexity of selected algorithms. The results showed that despite all algorithm exhibit similar performances in terms of convergence and accuracy, the class of iterative algorithms such that Jacobi and Gauss Seidel significantly underperforms methods like LU-decomposition and multigrid method. This suggests that overall, LU-decomposition and multigrid method may be better suited for solving PDEs like black Scholes equation, which involves repeatedly solving linear systems with the same coefficient matrix.

However, it turns out that in practice, iterative methods are more commonly used. This is because sometimes the options are American-style. Unlike a European option, an American option allows early exercise. Mathematically, this means that at each node V_{ij}, we need to check whether

$$V_{ij} \geq \max\{K - S(t), 0\}$$

If not, then the put option can be exercised early. An iterative scheme becomes a natural way of implementing this additional requirement. By iteratively evaluating the option's value at each time step and comparing it to the immediate exercise payoff, these methods is able to capture the dynamic nature of American options, making them a practical choice in option pricing. This suggests that the multigrid method may be more extendable to other types of options compared to LU decomposition.

7 References

[1] Danho, S. (2017, September 5). Pricing financial derivatives with the finitedifference method. DIVA. http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1138545amp;dswid=4543

[2] Brandimarte, P. (2006). Numerical Methods in Finance and Economics: A MATLAB-based introduction. Wiley-Interscience.

[3] 10 year Treasury Rate - 54 Year historical chart. MacroTrends. (n.d.). https://www.macrotrends.net/2016/10-year-treasury-bond-rate-yield-chart

[4] Multigrid method for solving AX = b. MathWorks. (n.d.-a). https://www.mathworks.com/matlabcentral/fileexchange/123145-multigrid-method-for-solving-ax-b