Numerical Methods for Black-Scholes Model Under European and American Options

by

Yongzhen Huang

April 10, 2019

Contents

1 Abstract

Black-Scholes Model is one that is widely used in finance to model option pricing. In this paper, we use different methods to numerically approximate the values of call and put options for both European and American options. Furthermore, we compare the methods to identify advantages of one over the others and to extend this, Matlab code (which is used for most implementations) is compared with C implementation to test efficiency which is relevant in the field of finance.

2 Background

We will now give background materials into options and how Black-Scholes model and equation are derived and used. The background materials are mostly referenced from [1]. Some variations based on European options will be discussed in Section 3; variations on American options are not discussed but they follow similarly.

2.1 Options

We will start with **European Option** which is the simplest form of options and it has the following conditions where one can

- At a prescribed time (**Expiry Date**)
- purchase a **prescribed asset** for a
- prescribed amount (**exercise / strike price**)

In particular, there are two types of options, **call** and **put** options where

- **Call option** option to buy the asset
- **Put option** option to sell the asset

It is worth noting that for those who purchase the options, it is their *right* to buy or sell the option but they are not obligated to do so. European option is one of the ones that will be considered here while the other one is **American Option**. The difference between these two options is that European options can only be exercised at the expiry date while the American options can be exercised at *any time*. Call or put options that have no unusual or special features are often referred to as **vanilla options**.

When dealing with assets (currency or financial products, etc.), it is often a good idea to consider the present value (PV) and the idea of discounting. In essence, discounting is when the "money today does not worth the same in the future" due to inflation and other factors. Therefore, it is better to think of the future value in terms of the present value. Let E denote the future value of some asset at time T and r be the interest that can be accrued if money was deposited. Also, let M be the present value. Then we have a simple differential relationship as the following,

$$\frac{dM}{M} = rdt \Longrightarrow M = ce^{rt}$$

Since $M(T) = E$ by definition, then $M = Ee^{-r(T-t)}$ if r is constant, or $M = Ee^{-\int_t^T r(s)ds}$ if $r = r(t)$

2.2 Notations

For an asset, we will denote the following by the following letters for the rest of the paper,

- S: the price of the asset
- r: the deposit interest rate, in percentage
- σ: the volatility of the asset, in percentage

- T: the expiry date of the option, in years
- V: the value of the option (as we will see later, $V = V(S,t)$)
- $C(S,t)$ is the value of the call option while $P(S,t)$ is the value of put option

2.3 Asset Price Random Walk

Most likely, it is impossible to predict the future value of an asset and therefore a random walk is considered. First, let us consider the **Efficient Market Hypothesis**

1. present price fully reflects past history and nothing else
2. Markets respond *immediately* to any new information

This ensures that unanticipated changes in asset prices are **Markov Processes**. We will now define μ as the **drift**, measure of average rate of growth of the asset; so we have $\mu = \mu(S,t)$. Let us further define volatility σ as mentioned above where the volatility measures the standard deviation of return for samples drawn from a normal distribution dX. Then, we have the following **stochastic differential equation** (SDE)

$$\frac{dS}{S} = \sigma dX + \mu dt \tag{1}$$

where dX is the **Weiner Process** with the property that $dX \sim \mathcal{N}(0, dt)$ so that $dX = \Phi\sqrt{dt}$ where $\Phi \sim \mathcal{N}(0,1)$. Note that from equation (1) that only the term involving dX is random; so if we only consider the other terms, then it is a simple ODE and the result is entirely deterministic. Thus, it is dX that give rise to the random walk for the price. The intuition behind (1) follows clearly. We will now consider some properties of (1)

1. (1) does not depend on the past. It only depend on the current values. This is the Markov property
2. $E[dS] = E[\sigma S dX + \mu S dt] = \mu S dt$ since $E[dX] = 0 \Longrightarrow$ on average, S_{t+1} is $\mu S dt$ higher than S_t
3. $var[dS] = E[dS^2] - E^2[dS] = E[\sigma^2 S^2 dX^2] = \sigma^2 S^2 dt \Longrightarrow$ standard deviation is proportional to σ

2.4 Ito's Lemma

Lemma 2.1 (Ito's Lemma)**.** *For any random variable G with SDE of $dG = A(G,t)dX + B(G,t)dt$, then given $f(G)$,*

$$df = A\frac{df}{dG}dX + (B\frac{df}{dG} + \frac{1}{2}A^2\frac{d^2f}{dG^2})dt$$

We will not prove this lemma but rather show a derivation of how we arrive at the form given by the lemma in the context of our asset random walk.

First, one fact (which is not proved) is that with probability 1, $dX^2 \to dt$ as $dt \to 0$. Now, suppose function $f(S)$ is smooth and ignore that it is stochastic, then if we perturb f due to perturbation of S by dS, we get that

$$df = \frac{df}{dS}dS + \frac{1}{2}\frac{d^2f}{dS^2}dS^2 + ... \tag{2}$$

also, we have that

$$dS^2 = \sigma^2 S^2 dX^2 + 2\sigma\mu S^2 dt dX + \mu^2 S^2 dt^2 \longrightarrow \sigma^2 S^2 dt + 2\sigma\mu S^2 (dt)^{3/2} + \mu^2 S^2 dt^2 \tag{3}$$

So we have that the first term dominates and therefore $dS^2 \to \sigma^2 S^2 dt$. Now if we plug this result back into (2) we get the following

$$df = \sigma S\frac{df}{dS}dX + (\mu S\frac{df}{dS} + \frac{1}{2}\sigma^2 S^2\frac{d^2f}{dS^2})dt \tag{4}$$

Now, if we expand $f(S + dS)$ into $f(S + dS, t + dt)$, we obtain

$$df = \sigma S\frac{\partial f}{\partial S}dX + (\mu S\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t})dt \tag{5}$$

2.5 Black-Scholes Model and Analysis

We will follow closely to the variables that we defined before and E is the future value at time $t = T$. Note that at time $t = T$, possibility of arbitrage tells us that if $S > E$, then one would exercise call option and the profit is $S - E$. If, however, $S < E$, then the call option is not exercised. Therefore, we can generalize the value of the call option as

$$C(S, t) = \max(S - E, 0) \tag{6}$$

Similarly for put option, we have its value of expiry being

$$P(S, t) = \max(E - S, 0) \tag{7}$$

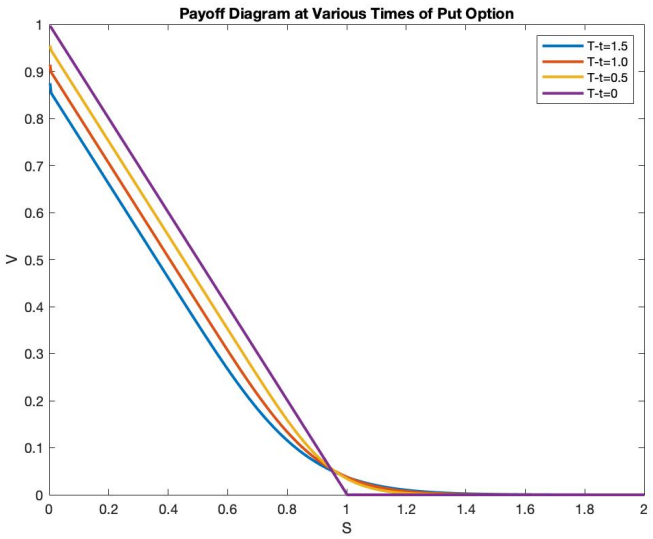

Figure 1. Payoff Diagram of Put Option $r = 0.1, \sigma = 0.2, E = 1, T = 2$

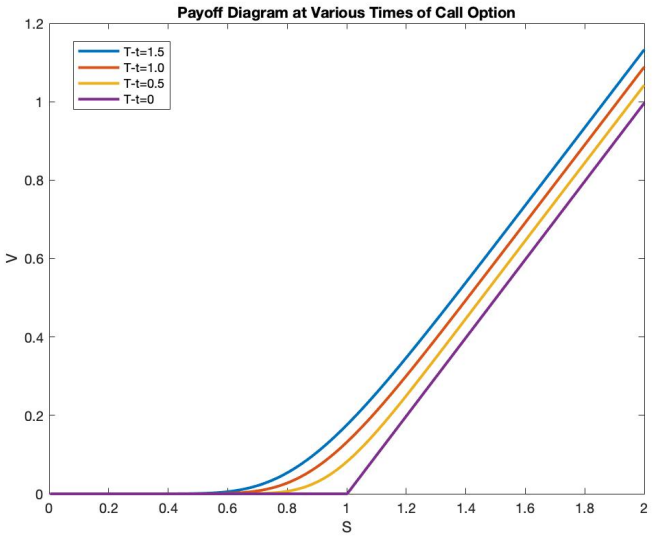

Figure 2. Payoff Diagram of Call Option $r = 0.1, \sigma = 0.2, E = 1, T = 2$

Figure 1, 2 shows these two *Payoff Diagrams* where the $T - t = 0$ line shows the payoff at expiry date. There are more general payoffs but they will not be considered here due to time limit. Examples include the *Heaveside Function*, i.e. cash-or-nothing, or combination of different payoffs.

One relation between put and call options is the **Put-Call Parity**. Suppose there is a portfolio of different assets with call and put with some expiry date T and exercise price E; in particular, long one asset, long one put, and short one call. The value of the portfolio is, then, $\Pi = S + P - C$. Then the pay off at time T is

$$S + \max(E - S, 0) - \max(S - E, 0)$$

So, it can be easily shown that no matter $S \geq E$ or $E \geq S$, the total payoff is E; so that the payoff is E with certainty and so $S + P - C = E$. Now, if we consider discounting as previously discussed, then

$$S + P - C = Ee^{-r(T-t)} \tag{8}$$

We will now proceed to the derivation of the Black-Scholes equation and analysis. We first make the following assumptions

- Price follows log-normal random walk as (1)
- Risk free rate r and volatility σ known functions of time
- There is no transaction
- Asset pays no dividend
- There is no possibility of arbitrage
- Trading can take place continuously
- Short selling permitted and asset divisible

For value $V(S, t)$, using Ito's Lemma, we get from (5) that

$$dV = \sigma S \frac{\partial V}{\partial S} dX + (\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t})dt \tag{9}$$

Suppose we have a number $-\Delta$ of an asset, then the value of the portfolio is $\Pi = V - \Delta S$. Furthermore, $d\Pi = dV - \Delta dS$. Now we combine this with (1) and (9) to get

$$\begin{aligned} d\Pi &= \sigma S \frac{\partial V}{\partial S} dX + (\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t})dt - \Delta \sigma S dX - \Delta \mu S dt \\ d\Pi &= \sigma S (\frac{\partial V}{\partial S} - \Delta) dX + (\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} - \mu \Delta S)dt \end{aligned}$$

It is clear that to eliminate randomness in our problem, we choose $\Delta = \frac{\partial V}{\partial S}$. The following is then obtained

$$d\Pi = (\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})dt$$

From the arbitrage assumption, note that return of Π is $r\Pi dt$ and

$$r\Pi dt = (\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2})dt$$

Combining with the fact that $\Pi = V - \Delta S$, we get the Black-Scholes PDE

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{10}$$

Before we proceed any further, there are a few remarks to be made

1. The value V depends completely on S (current value) and t only
2. $\Delta = \frac{\partial V}{\partial S}$, rate of change of option w.r.t. current price S

3. $\mathcal{L}_{BS} = (\frac{\partial}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2}{\partial S^2}) + (rS\frac{\partial}{\partial S} - r)$
   The first part is return on hedge while the second part is potential return on bank deposit. This interpretation makes intuitive sense.

4. The equation is homogeneous for European option to avoid arbitrage. This is different for American option which will be discussed later

5. Note that μ which is the average rate of growth does not matter in this case.

Let us now examine the boundary conditions of the equation. We will examine the call option and put option follows similarly. At $t = T$, the expiry date, it has already been discussed that $V(S,T) = \max(S-E, 0)$. Also, $S = 0$ is a equilibrium point so that $C(0,t) = 0$. As $S \to \infty$, the exercise price E becomes insignificant compare to S and so $C(S,t) \sim S$ as $S \to \infty$. However, it can actually shown that $C(S,t) \sim S - Ee^{-r(T-t)}$ as $S \to \infty$. For put option, we have $P(0,t) = Ee^{-r(T-t)}$ and $P(S,t) \to 0$ as $S \to \infty$.

2.6 Explicit Solution to Black-Scholes Equation

Consider the call option $\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC = 0$ with boundary conditions $C(0,t) = 0, C(S,t) \sim S$ as $S \to \infty$, and $C(S,T) = \max(S-E, 0)$.

Let $S = Ee^x, t = T - \frac{\tau}{\frac{1}{2}\sigma^2}, C = Ev(x,\tau)$. Then, equation becomes (by chain rule)

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + (k-1)\frac{\partial v}{\partial x} - kv \qquad \text{(where } k = \tfrac{r}{\frac{1}{2}\sigma^2}\text{)}$$

and $v(x,0) = \max(e^x - 1, 0)$ since $Ev(x,0) = C(S,T)$. Now let $v = e^{\alpha x + \beta\tau}u(x,\tau)$

$$\begin{aligned}
\implies \beta u + \frac{\partial u}{\partial \tau} &= \alpha^2 u + 2\alpha\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + (k-1)(\alpha u + \frac{\partial u}{\partial x}) - ku \\
\frac{\partial u}{\partial \tau} &= [\alpha^2 - \beta - k + (k-1)\alpha]u + (2\alpha + k - 1)\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2}
\end{aligned}$$

Choose $\beta = \alpha^2 - k + (k-1)\alpha$ and $\alpha = \frac{1-k}{2}$ so that we get $\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}$ which is the heat equation with initial condition $u(x,0) = \max(e^{-\frac{1}{2}(k+1)x} - e^{\frac{1}{2}(k-1)x}, 0) = u_0(x)$ and $v = e^{-\frac{1}{2}(k-1)x - \frac{1}{4}(k+1)^2\tau}u(x,\tau)$.

Solving heat equation, we get $u(x,t) = \frac{1}{2\sqrt{\pi t}}\int_{-\infty}^{\infty} u_0(x)e^{-\frac{(x-s)^2}{4\tau}}ds$. Then,

$$u(x,t) = e^{-\frac{1}{2}(k+1)x - \frac{1}{4}(k+1)^2\tau}N(d_1) - e^{-\frac{1}{2}(k-1)x - \frac{1}{4}(k-1)^2\tau}N(d_2)$$

where $d_1 = \frac{x}{\sqrt{2\tau}} + \frac{1}{2}(k+1)\sqrt{2\tau}$ and $d_2 = \frac{x}{\sqrt{2\tau}} + \frac{1}{2}(k-1)\sqrt{2\tau}$. Retracing, we get that

$$\begin{aligned}
C(S,t) &= SN(d_1) - Ee^{-r(T-t)}N(d_2) \\
d_1 &= \frac{\ln(\frac{S}{E}) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \\
d_2 &= \frac{\ln(\frac{S}{E}) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}
\end{aligned}$$

Finally, from Put-Call Parity, $C - P = S - Ee^{-r(T-t)} \implies P(S,t) = Ee^{-r(T-t)}N(-d_2) - SN(-d_1)$ where $N(d) + N(-d) = 1$. Further note that $\Delta = N(d_1)$ from differentiation.

2.7 American Option

As aforementioned, the difference between American option and European option is that it can be exercised at any time. The opportunity for one to exercise it at any time creates an arbitrage. Therefore, we must impose the constraint that $P(S,t) \geq \max(E-S, 0)$ if one were to choose to exercise earlier. Similarly,

$C(S,t) \geq \max(S-E,0)$. Thus, there must be an S such that it is optimal for the holder to exercise the American option. This is known as a **free boundary problem** (a boundary is not known). There are, however, some constraints to this problem, and they are,

- The option value must be greater or equal to the payoff function
- Black-Scholes equation is inequality instead of equality
- $V(S,t)$ must be a continuous function in S
- Slope of V must be continuous

First condition follows from the fact that arbitrage should generate a profit of at least 0. This also leads to the second condition as explained from the intuition behind Black-Scholes equation in European options. The third condition also follows from arbitrage; one should not be allowed to make risk-free profit off options only should there be any jump in the option value. The reasoning for fourth condition is quite cumbersome and [1] provides a nice financially based argument. Therefore, the equation for American (put) option, is

$$\frac{\partial P}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} + rS\frac{\partial P}{\partial S} - rP \leq 0 \tag{11}$$

In summary, for each time t, the S-axis must be divided into two distinct regions where one is optimal if exercised early while one is not. Further details can be found in [1].

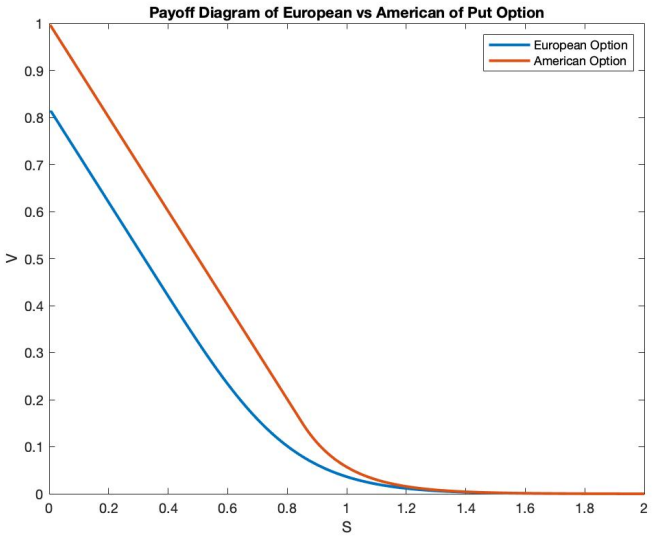

Figure 3 European vs American Put Option Payoff with $\sigma = 0.2, r = 10\%, t = 0, E = \$1$

The difference between European and American options are displayed in **Figure 3**. Note that the value of American option is higher than European option at almost everywhere. The reasoning is intuitive; since one can exercise an American option at any time, American options admit less risk and therefore have higher value when one tries to purchase the option.

However, it is quite different for American call option. First of all, it is noted in [9] that for a call option, American is lower than European since one can exercise at any time. The boundary condition is, therefore, $S - E$ instead of $S - Ee^{-r(T-t)}$. Also, one should never exercise American call option early on non-dividend paying stock. The following argument is adapted online. Suppose there are two possible portfolios A, B. A consists of American call option on non-dividend paying stock with price C and bond worth $\frac{E}{1+r}$ (because of present value concept). On the other hand, portfolio B has only the same stock. So at expiration date, portfolio A has value $(S_T - E) + E = S_T$ if $S_T > E$ and only E if $S_T < E$; portfolio B has value S_T regardless. So overall, portfolio A has higher payoff and therefore at $t = 0$, it has higher value

$\Rightarrow C_0 + \frac{E}{1+r} > S_0$. So, $C_0 > S_0 - \frac{E}{1+r} > S_0 - E$. Therefore, the call option always have higher value than the payoff generated by exercising early. Thus, we have implemented a version with constant dividend as shown in later section.

Let us first consider the American put option. **Figure 4** shows the payoff of American option with $\sigma = 0.2, r = 0.1, E = 1$ with $t = 0.5, 1.0, 1.5, 2.0$. The figure has been magnified around the strike price $E = 1$ to emphasize the difference. Note the difference of American put option with European put option; the payoffs only differ near the strike price while the other places are essentially the same. The reasoning is quite intuitive. Since one is allowed to exercise at any time, it is obvious why the value only differ around the strike price. Furthermore, the difference between various times follow the trend from European put options. The reasoning behind this follows similar with that of European options.

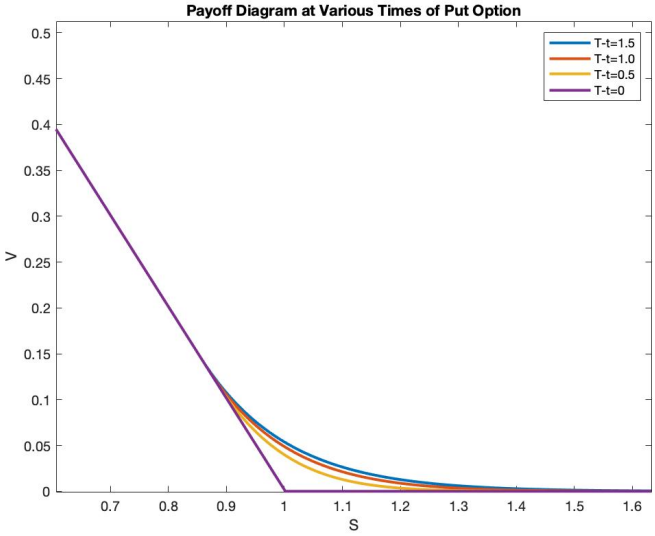

Figure 4 American Put Option Payoff with $\sigma = 0.2, r = 10\%, E = \$1$ at Various Times

3 Implementations and Results

In this section we will discuss the two methods that was used to implement a solver for Black-Scholes equations. In particular, both Implicit FD and Crank-Nicolson will be discussed. Explicit FD could be used as well but it generally has worse stability than these two and will be briefly mentioned only. Finally, the implementations for European option and American option will be compared.

3.1 European Options

European option is the simplest form of options. To implement European Options, the methods used will be described below. Note that only implicit FD and Crank-Nicolson are implemented with Crank-Nicolson using SOR method. A comparison between method based on reference [2] and using SOR method can be found in below. Example code can be found in Appendix A-B.

Before discussing the different methods, we will first present the results based on implicit FD and Crank-Nicolson with both using the Matlab operation for solving $Ax = b$. **Table 1** and **Table 2** shows the comparison of the actual results and the results obtained by the different methods. Note that with bounds of $S = 20$ shows much better accuracy than bounds of $S = 50$. The example below is based on one of the data presented from [1]; it uses volatility $\sigma = 0.2$, interest rate $r = 0.05$, exercise price of \$10 and 6 months to expiry (i.e. t=0.5) on a grid with $M = N = 599$. **Figure 5** shows the surface obtained by solving the equation with maturity being 1 year, $T = 1$.

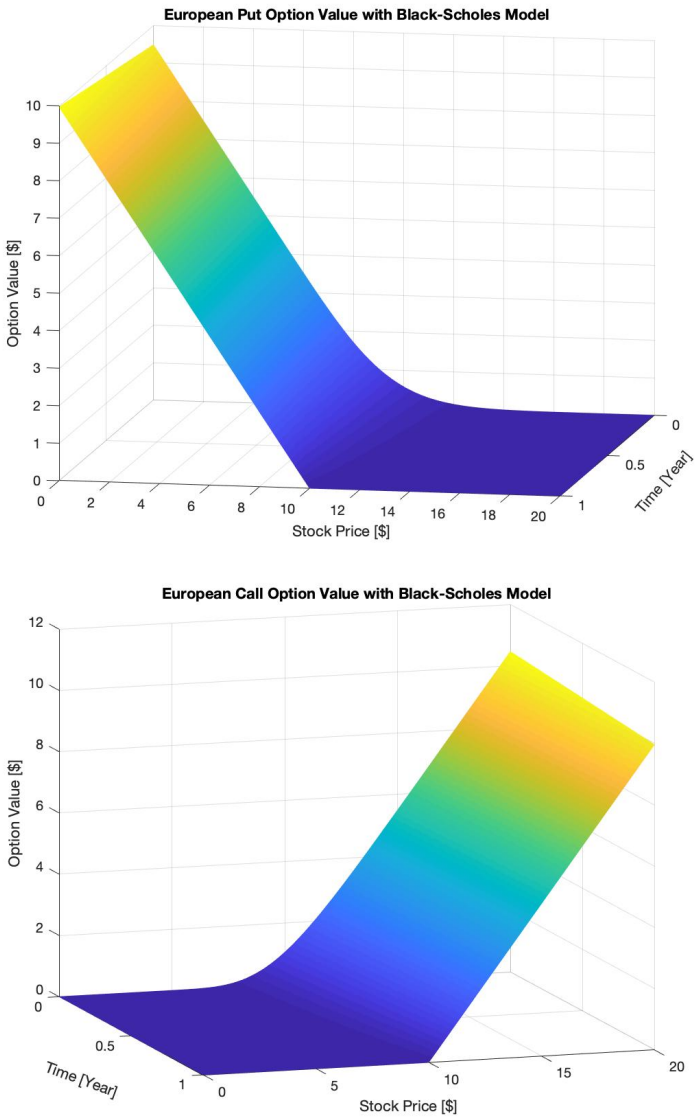

Figure 5 Eur. Put (left) Call (right) Option on grid $M = N = 599$ with $\sigma = 0.2, r = 5\%, t = 0.5, E = \$10$

Asset Price	Actual	Crank-N.	Error	Impl. FD	Error
2	7.753099	7.753109	1.015892e-05	7.753109	1.015892e-05
4	5.753099	5.753109	1.015894e-05	5.753109	1.015897e-05
6	3.753181	3.753194	1.340977e-05	3.753201	2.010072e-05
7	2.756835	2.756870	3.483143e-05	2.756948	1.122710e-04
8	1.798715	1.798717	2.780358e-06	1.798878	1.637144e-04
9	0.988042	0.987876	1.660190e-04	0.987847	1.946634e-04
10	0.441972	0.441721	2.504792e-04	0.441499	4.729344e-04
11	0.160638	0.160478	1.597890e-04	0.160350	2.877355e-04
12	0.048344	0.048293	5.156883e-05	0.048313	3.109809e-05
13	0.012381	0.012376	4.734544e-06	0.012437	5.584781e-05
14	0.002775	0.002779	3.834842e-06	0.002819	4.383409e-05
15	0.000558	0.000561	2.487326e-06	0.000578	1.963852e-05
16	0.000103	0.000104	9.258992e-07	0.000110	6.644314e-06

Table 1 Comparison of Crank-Nicolson and Implicit FD for European Put Option ($0 \leq S \leq 50$)

Asset Price	Actual	Crank-N.	Error	Impl. FD	Error
2	7.753099	7.753109	1.015892e-05	7.753109	1.015892e-05
4	5.753099	5.753109	1.015892e-05	5.753109	1.015894e-05
6	3.753181	3.753191	1.066823e-05	3.753198	1.730737e-05
7	2.756835	2.756849	1.354789e-05	2.756926	9.113136e-05
8	1.798715	1.798720	4.988764e-06	1.798880	1.658605e-04
9	0.988042	0.988014	2.814300e-05	0.987985	5.693210e-05
10	0.441972	0.441926	4.556038e-05	0.441704	2.677606e-04
11	0.160638	0.160607	3.076528e-05	0.160479	1.587763e-04
12	0.048344	0.048333	1.127462e-05	0.048353	9.045997e-06
13	0.012381	0.012379	2.003267e-06	0.012440	5.859318e-05
14	0.002775	0.002775	2.268712e-07	0.002815	4.028095e-05
15	0.000558	0.000559	3.051110e-07	0.000576	1.747680e-05
16	0.000103	0.000103	1.312189e-07	0.000109	5.848440e-06

Table 2 Comparison of Crank-N. and Implicit FD for European Put Option ($0 \leq S \leq 20$) $M = N = 599$

3.2 Explicit Finite Difference

Explicit FD essentially solves the system in a iterative way forward in time. From the initial boundary value, it can iterate forward to solve the equation. It uses similar numerical approximation as described in Implicit FD below. Furthermore, from [1], it is shown that Explicit FD is numerically unstable when $\alpha > \frac{1}{2}$ where $\alpha = \frac{dt}{(dx)^2}$. In the unstable case, the values turn out to be quite bogus and examples can be found from [1].

3.3 Implicit Finite Difference

The implementation of this method was guided by reference [1] and [2].
In general, Implicit FD uses a "backward" method for solving the Black-Scholes Equation

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial^2 S} = rf$$

Essentially, given the boundary condition from the payoff at maturity, we have the payoff of $\max(E - S, 0)$ as shown in the theory. From this boundary condition at $t = T$, we work backwards in time to calculate the solution since there is time component in the equation. Therefore, we will be solving a number of systems of equations. At each step, the system of equation can be obtained from numerical approximation to the equation.
The numerical approximation will be a four point stencil. In particular, we will do the following,

$$\begin{aligned}
\frac{\partial f}{\partial S} &\approx \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta S} \\
\frac{\partial f}{\partial t} &\approx \frac{f_{i+1,j} - f_{i,j}}{\Delta t} \\
\frac{\partial^2 f}{\partial^2 S} &\approx \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{\Delta S^2}
\end{aligned}$$

These equations yields that $a_j f_{i,j-1} + b_j f_{i,j} + c_j f_{i,j+1} = f_{i+1,j}$ for $j = 1, 2, ..., M-1, i = 1, 2, ..., N-1$ where

$$\begin{aligned}
a_j &= \frac{1}{2}rj\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t \\
b_j &= 1 + \sigma^2 j^2 \Delta t + r\Delta t \\
c_j &= -\frac{1}{2}rj\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t
\end{aligned}$$

We also get the following system of equations at each time step

$$\begin{bmatrix} b_1 & c_1 & & \\ a_2 & b_2 & c_2 & \\ & & \cdots\cdots & \\ & & a_{M-1} & b_{M-1} \end{bmatrix} \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ \vdots \\ f_{i,M-1} \end{bmatrix} = \begin{bmatrix} f_{i+1,1} - a_1 f_{i,0} \\ f_{i+1,2} \\ \vdots \\ f_{i+1,M-1} - c_{M-1} f_{i,M} \end{bmatrix}$$

As for the boundary conditions, we have the following,

1. $f_{N,j} = \max(E - j\Delta S, 0), j = 0, 1, ..., M$
2. $f_{i,0} = E, i = 0, 1, ..., N$
3. Since $f \to 0$ as $S \to \infty, f_{i,M} = 0, i = 0, 1, ..., N$

By solving this system iteratively, we obtain all f_{ij} which is the solution to the Black-Scholes equation. Note Equations (1) and (3) are both second order while (2) is first order. It is possible to obtain a better approximation to (2) similar to (1) but it is against the purpose of Implicit FD. Thus, the overall rate of convergence is linear as we will discuss later.

3.4 Crank-Nicolson Finite Difference

Crank-Nicolson scheme combines both Explicit and Implicit FD by taking the average of both. This leads to a rate of convergence of $\mathcal{O}(\Delta t^2)$. Similar to Implicit FD, the final boundary condition is first obtained and iterated backwards to get the solution[1]. Furthermore, comparing to Explicit and Implicit FD which have rate of convergence of $\mathcal{O}(\Delta t)$, Crank-Nicolson is better; comparing to Explicit FD, Crank-Nicolson is numerically stabler.

So for Crank-Nicolson, we obtain the following after averaging,

$$(\frac{\sigma^2 j^2 - rj}{2}) f_{i,j-1} + (\frac{-2}{\Delta t} - \sigma^2 j^2 - 2r) f_{i,j} + (\frac{\sigma^2 j^2 + rj}{2}) f_{i,j+1} = (\frac{rj - \sigma^2 j^2}{2}) f_{i+1,j-1} + (\frac{-2}{\Delta t} + \sigma^2 j^2) f_{i+1,j} - (\frac{\sigma^2 j^2 + rj}{2}) f_{i+1,j+1}$$

The rest of the procedure follows very similarly to the Implicit FD where we start from the final boundary condition and iterate backwards. The only difference is how we set up the matrix at each iteration and it is essentially as follows

$$\begin{bmatrix} b_1 & c_1 & & \\ a_2 & b_2 & c_2 & \\ & & \cdots\cdots & \\ & & a_{M-1} & b_{M-1} \end{bmatrix} \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ \vdots \\ f_{i,M-1} \end{bmatrix} = \begin{bmatrix} b'_1 & c'_1 & & \\ a'_2 & b'_2 & c'_2 & \\ & & \cdots\cdots & \\ & & a'_{M-1} & b'_{M-1} \end{bmatrix} \begin{bmatrix} f_{i+1,1} \\ f_{i+1,2} \\ \vdots \\ f_{i+1,M-1} \end{bmatrix} + \begin{bmatrix} a'_1 f_{i+1,0} - a_1 f_{i,0} \\ 0 \\ \vdots \\ c'_{M-1} f_{i+1,M} - c_{M-1} f_{i,M} \end{bmatrix}$$

$$\begin{aligned} a_j &= \frac{\sigma^2 j^2 - rj}{2} & a'_j &= -a_j \\ b_j &= \frac{-2}{\Delta t} - \sigma^2 j^2 - 2r & b'_j &= \frac{-2}{\Delta t} + \sigma^2 j^2 \\ c_j &= \frac{\sigma^2 j^2 + rj}{2} & c'_j &= -c_j \end{aligned}$$

By solving this iteration at each time step, we will obtain the value of the option at each time. Again, this method works for both put and call option and the only differences occur at the boundary of $f_{i,0}$ and $f_{i,M}$ Both European options and American options implementations can be found in the Appendix.

3.5 SOR Iterative Methods

SOR (Successive Over-Relaxation) is a type of iterative methods for solving $Ax = b$ that is based on Gauss-Seidel which is in turn based on Jacobi method. It will be briefly introduced here and details can be found from reference [1] and [8]. First of all, Gauss-Seidel is based on $A = L + U + D$ to obtain iteration at k^{th} step,

$$(L + D)x^{(k+1)} + Ux^{(k)} = b \tag{12}$$

for solving $Ax = b$. Iteration (12) can be modified in the following way[6],

$$\begin{aligned} Dx^{(k+1)} &= b - Lx^{(k+1)} - Ux^{(k)} \\ x^{(k+1)} &= D^{-1}[b - Lx^{(k+1)} - Ux^{(k)}] \end{aligned}$$

Now, subtracting $x^{(k)}$ from both sides yields

$$x^{(k+1)} - x^{(k)} = D^{-1}[b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}] \tag{13}$$

Note that if the method converges, then $x^{(k+1)} - x^{(k)} \longrightarrow 0$. We will not discuss the conditions for SOR method here but the method will converge for $Ax = b$ in the context of Black-Scholes model finite difference. In essence, the iteration becomes

$$x^{(k+1)} = x^{(k)} + \omega(x^{(k+1)} - x^{(k)})$$

where ω is the relaxation parameter. Now we substitute (13) in to obtain

$$x^{(k+1)} = x^{(k)} + \omega D^{-1}[b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}] \tag{14}$$

This is the SOR method iteration. In algorithm, (14) becomes

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right) \tag{15}$$

which is how the code is written out using SOR. Notice that for the matrix in interest (both implicit and Crank-Nicolson), the matrix is tridiagonal which allows us to simplify (15) to

$$x_i^{(k+1)} = (1 - \omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - a_{i,i-1} x_{i-1}^{(k+1)} - a_{i,i+1} x_{i+1}^{(k)} \right)$$

This way, total iterations are reduced to a small ratio of original iteration, especially if the matrix becomes large. The next natural question to be addressed is how to choose $\omega \in [1, 2]$. The optimal relaxation parameter, in theory, is given by $\omega = \frac{2}{1+\sqrt{1-\rho^2}}$ where ρ is the spectral radius of C_k [7] (This came from minimizing the error which can be reformulated from (13) and is related to the eigenvalue).

The implementation of SOR is included in Appendix. It is trivial to incorporate it in the Crank-Nicolson scheme; in particular, the $x^{(0)}$ used was the value from previous step and ω was determined experimentally which is discussed below. We use this method to compare with Matlab's '\' operation. The initial hypothesis is that Matlab's operation would lead to the best result. However, the goal here is to compare it with standard Matlab operation and try assess the quality of the method. We compare based on the Crank-Nicolson method. The script to compare SOR and standard Matlab solver is included in Appendix. There are a few things to be discussed.

First of all, the cost of (in terms of time) the two approaches differ significantly. That is, SOR method turns out to be a lot slower than the standard Matlab method. In particular, the standard method takes approximately $0.7 \sim 0.9$ seconds while the SOR takes $30 \sim 40$ seconds. Note that this SOR already took advantage of the fact that A is tridiagonal matrix. This result was tested with error tolerance of 10^{-6}. Clearly, Matlab's operation is much faster.

It is natural, of course, to discuss the accuracy of the SOR method. First note that there are a few ways to evaluate accuracy. First of all, the most rapid change occurs at the price of interest, namely, the strike price E. Since this point is perhaps the most important of all points, it becomes natural to prioritize the accuracy in this neighborhood. It was observed that,

1. With higher tolerance, the error around the price of interest is lower at the cost of the error elsewhere gets higher (though not significant). Overall, tolerance of 10^{-6} presented better result with optimal ω.
2. Comparing ω, the relaxation parameter, it is observed that with lower values (0.5, 0.75), the end of put option is better approximated (i.e. at $S = 14, 16$). Furthermore, it was observed that $\omega = 1.35$ is the experimental optimal value and the comparison of SOR with this value versus the standard Matlab operation is presented in **Table 3** with tolerance 10^{-6}.
3. The optimal relaxation parameter for SOR with $\omega = 1.35$ turned out to be slightly better than using Matlab operation. Although it can be observed that the error is slightly higher when S is small; near the price of interest, however, SOR with $\omega = 1.35$ turned out to be better.

Asset Price	Actual	A\b	Error	SOR	Error
2	7.753099	7.753109	1.015892e-05	7.753114	1.525084e-05
4	5.753099	5.753109	1.015894e-05	5.753112	1.258115e-05
6	3.753181	3.753194	1.340977e-05	3.753195	1.414182e-05
7	2.756835	2.756870	3.483143e-05	2.756870	3.517177e-05
8	1.798715	1.798717	2.780358e-06	1.798718	2.919464e-06
9	0.988042	0.987876	1.660190e-04	0.987876	1.659714e-04
10	0.441972	0.441721	2.504792e-04	0.441722	2.504663e-04
11	0.160638	0.160478	1.597890e-04	0.160478	1.597864e-04
12	0.048344	0.048293	5.156883e-05	0.048293	5.156860e-05
13	0.012381	0.012376	4.734544e-06	0.012376	4.734658e-06
14	0.002775	0.002779	3.834842e-06	0.002779	3.834755e-06
15	0.000558	0.000561	2.487326e-06	0.000561	2.487288e-06
16	0.000103	0.000104	9.258992e-07	0.000104	9.258857e-07

Table 3 Comparison of CN using Matlab A\b and SOR with $\omega = 1.35$ for European Put Option

3.6 American Options

The surface for American option is very similar to European options. However, American option is slightly trickier than European option. We have two methods for implementing this. The first one is based on Crank-Nicolson and reference [2]. For this approach, it essentially evaluates, at each time step, if it is more beneficial to exercise the option at that moment. The value, then, is the maximum of the computed value and the payoff of the option if exercised at that moment. The example code can be found in Appendix. **Table 4** shows a comparison between European option and American option at various asset price with the following conditions $E = 10, r = 0.1, \sigma = 0.4$ with three months and six months to expiry (i.e. $t = 0.75$ and $t = 0.5$). This comparison, based on similar comparison found on page 176 of [1], uses the American option computed using method from reference [2]. By comparing the results presented in [1] and this implementation, it can be observed that the values are quite close. Furthermore, it can be observed that overall, American options give higher value than European options. Intuitively, American options have higher value due to the freedom to exercise it at any time whereas European option refrains one from doing so; European options, therefore, generally present slightly more risk.

As aforementioned, we will now look at American call option with constant dividend payout. The implementation was not too different from any other so it is omitted. We will, however, note the differences here. In brief, the coefficient of $\frac{\partial C}{\partial S}$ was modified from only rS to $(r - d_0)S$ where d_0 is the amount payout. Therefore, the only thing that is different in implementation is the entries in the matrix. Referring to the code in Appendix for American option, we only need to change $r * j$ to $(r - d_0) * j$. **Figure 6** shows an American call option with $r = 0.1, d_0 = 0.05, \sigma = 0.2$ and $E = 1$.

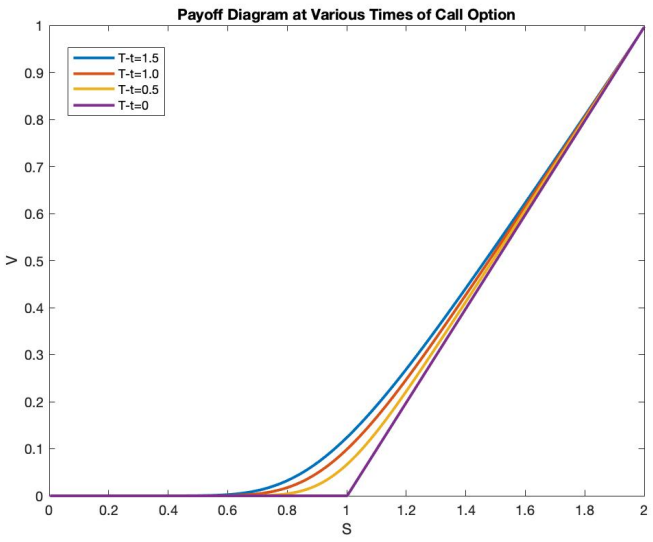

Figure 6 American Call Option with Constant Dividend Payout of $d_0 = 0.05$ and $r = 0.1, \sigma = 0.2, E = 1$

Asset Price	Payoff Value	3 months		6 months	
		American	European	American	European
2	8.00	8.000000	7.753119	8.000000	7.512334
4	6.00	6.000000	5.753121	6.000000	5.512867
6	4.00	4.000000	3.756929	4.000000	3.558326
8	2.00	2.019721	1.902373	2.094915	1.918019
10	0.00	0.691884	0.669199	0.921523	0.870190
12	0.00	0.171058	0.167417	0.362259	0.347590
14	0.00	0.033118	0.032610	0.132048	0.127879
16	0.00	0.005453	0.005388	0.046015	0.044825

Table 4 Comparison of American and European Put Options using Crank-Nicolson

The next implementation of American options uses PSOR methods which is a modification of SOR methods. PSOR tries to solve a constrained matrix for a linear complimentary problem. The implementation is quite simple; during a normal SOR step, we simply set

$$u_n^{k+1} = \max(u_n^{k+1}, g_n^{k+1})$$

The u_n^{k+1} on the right is the original one calculated in a normal SOR iteration while the one on the left is the new value. g_n^{k+1} is the potential payoff which is S at that specific time in this case. More detail on this method can be found in [1]. An implementation of PSOR can be found in Appendix. Using PSOR in the implementation of Crank-Nicolson is quite trivial and is similar to SOR. Note that in terms of time, PSOR also took much longer than Matlab operation. Since American option does not have an explicit solution, we will simply choose the relaxation parameter $\omega = 1$ since this value gives solution closest to the values presented in [1]. **Table 5** shows a comparison between the two methods, Matlab operation and PSOR.

Asset Price	Book[1]	A\b	SOR
2	8.0000	8.000000000	8.000000000
4	6.0000	6.000000000	6.000000000
6	4.0000	4.000000000	4.000000000
8	2.0951	2.094915393	2.095244853
10	0.9211	0.921522533	0.921696858
12	0.3622	0.362259317	0.362324821
14	0.1320	0.132047685	0.132052033
16	0.0460	0.046015400	0.045989414

Table 5 Comparison of CN using Matlab A\b and PSOR with $\omega = 1$ for American Put Option

It can be seen that the results are quite close and with difference in the order of 10^{-3} at least. Therefore, it might be preferable to use whichever one that is more efficient.

3.7 Variations of Black-Scholes Model

- We can consider (constant) dividend payouts where the asset pays out $D_0 S dt$. Then we have

$$dS = \sigma S dX + (\mu - D_0) S dt$$

  The rest follows closely to above. The numerical approximation can be easily modified by modifying the constants in the formulas presented for the matrices. Such constant dividend payout works for American options just as well as for European options as shown in American call options.

- Next we consider bullish vertical cash/put spread. This is comprised of simultaneous purchase of either call or put option that has the same underlying asset and expiration date at different strike prices. For call options, suppose we purchase a call option with strike price E_2 while writing a call option with strike price E_1 where $E_2 > E_1$, we aim to achieve a of $E_2 - E_1$. Then, the payoff function at expiration time T is

$$\max(S - E_1, 0) - \max(S - E_2, 0)$$

  Figure 6 shows both examples of bullish vertical call and put spread payoff diagrams with strike prices $E_1 = 1, E_2 = 1.5$ and $r = 0.1, \sigma = 0.2$. The only difference between the implementation of bullish vertical spread and vanilla options is that the boundary values are slightly different. For simplicity, implicit FD was modified for this model and is included in Appendix.

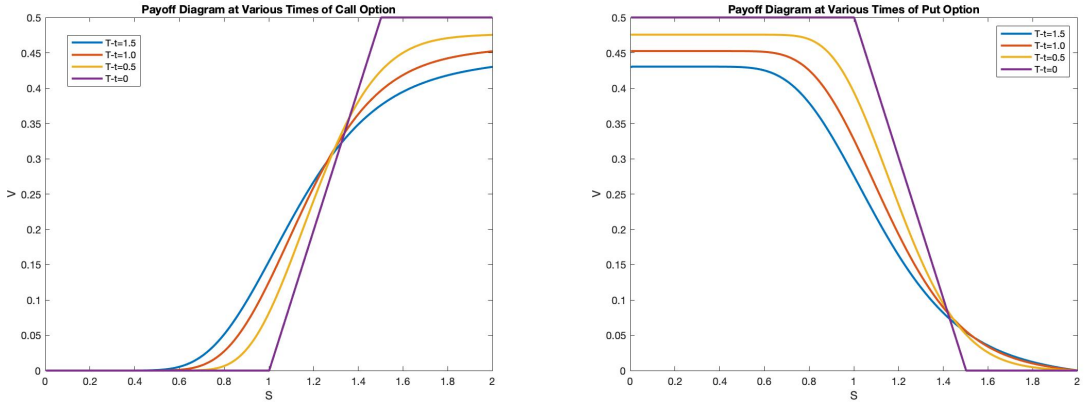

Figure 7 Bullish Vertical Call (left) and Put (right) Spread

- Finally, another small variation is binary call or put options (aka Cash-or-Nothing). This form of payoff was briefly mentioned above as being the **Heavside** function. Essentially, the payoff is constant amount A if $S > E$ whereas it is 0 otherwise. The implementation is not included since it was not quite complete. Furthermore, binary options seems to be tied closely to fraud which resulted in being banned in multiple countries.

4 Discussion

In this section, we will briefly discuss the expected convergence of the two methods that we have used as well as efficiency of our solver.

4.1 Rate of Convergence for Implementations

In theory, implicit FD has error of $\mathcal{O}(\Delta t)$ while Crank-Nicolson has error of $\mathcal{O}(\Delta t^2)$. To test this, we present the following data on European put option with $0 \leq S \leq 20, r = 5\%, \sigma = 0.2$ at 6 months ($t = 0.5$) to expiration around asset price $S = E = 10$.

ns = nt	Crank-Nicolson error	Implicit FD error
39	9.052809e-03	1.261389e-02
79	2.246572e-03	3.941365e-03
159	5.718560e-04	1.409037e-03
319	1.491384e-04	5.662355e-04
639	4.043827e-05	2.487368e-04
1279	1.169066e-05	1.157925e-04

Table 6 Error Comparison of European Put Option at $S = E = 10, \Delta t = \frac{1}{nt+1}$ (Same for ΔS)

It can be seen that the experimental results agree with theory quite well, as Δt was halved each time, the error for Crank-Nicolson decreased to approximately a factor of $\frac{1}{4}$ while implicit FD only decreased to about $\frac{1}{2}$. One technical point is that we also changed ΔS along with Δt since the error actually depends on ΔS as well. With a bit of Taylor expansion and algebra, it can be shown that the error for implicit FD is actually $\mathcal{O}(\Delta S^2) + \mathcal{O}(\Delta t)$ while for Crank-Nicolson is $\mathcal{O}(\Delta S^2) + \mathcal{O}(\Delta t^2)$. This explains why Implicit FD shows convergence slightly better than linear convergence.

4.2 Matlab vs C

In this final section on implementation, we quickly discuss the difference between implementing European put option in Matlab versus C. The code for C (Appendix J) only covers European put option for simplicity since the purpose was not to re-implement everything but rather for speed comparison. It was noted that Matlab SOR method take considerable amount of time while Crank-Nicolson using "\" operation was much faster. We, therefore, will compare Crank-Nicolson of European put option using Matlab "\" and SOR in C.

C implementation uses `clock()` to measure the CPU time while Matlab uses `cputime`. The model parameters for both programs are same. For Matlab, time taken was 0.8100 seconds while for C it was 0.119453 seconds without compiler optimization and 0.068819 seconds with `-O2` optimization on 2015 Macbook Pro with 2.7 GHz Intel Core i5 and 8GB DDR3 RAM. It can be seen that C takes much shorter time which is always preferable. However, the only issue, which was not addressed, was the accuracy the C program presented. We used `double` for floating point operations in C and the result was not nearly as accurate as in Matlab. It is possible to use better precision operation such as using `gmp.h` but this approach was not tested due to time constraint.

Therefore, although Matlab is easier implement, it might be preferable to use C for higher performance (although the implementation is not much harder) as long as the accuracy issue is addressed appropriately.

5 Conclusion

From the result of European call option between Implicit FD and Crank-Nicolson method, it can be observed that Crank-Nicolson presents smaller errors than the Implicit FD method. Overall, however, both Crank-Nicolson and Implicit FD present results that are accurate to at least 2 decimals. Furthermore, for the Crank-Nicolson method, Matlab solver for $Ax = b$ and SOR present similar results while the SOR method with (experimentally) optimal relaxation parameter ω gives the most accurate result. The only disadvantage for SOR is, however, that it takes much longer than the Matlab solver. In practice, such slow down may be detrimental and therefore faster implementations would be preferred, especially with small deviations.

It is also worth noting that it is important to choose the correct bound for S since it would lead to

accuracy difference as shown in **Table 1-2**. Most often, the bound should be chosen close to the strike price E to obtain higher accuracy.

The difference between American and European options are clearly shown with the payoffs they generated. It was observed that American options generally give higher payoffs than European options. Using PSOR and Matlab operation with Crank-Nicolson method both generated results that only differ in very small amount. While there is no explicit solution for American options, it is therefore more difficult to quantify which method gave better results.

Finally, although most data presented here were about put options, call options can be as easily quantified. The difference between the two are not large; in fact, they only differ by the boundary conditions most of the time.

6 Appendix

Appendix A

Matlab Code for Implicit FD for European Option

```
function values = BS_eur_impl_fd(Sa, Sb, E, r, sigma, T, type, ns, nt)

hs = (Sb-Sa) / (ns+1);
ht = T / (nt+1);

values = zeros(ns*nt, 1);
if type == "put"
    for i = 1:ns, values((nt-1)*ns+i,1) = max(E-i*hs,0); end
elseif type == "call"
    for i = 1:ns, values((nt-1)*ns+i,1) = max(i*hs-E,0); end
end

% calculates for put option
for i = nt-1:-1:1
    Aj = sparse(ns, ns);
    bj = zeros(ns, 1);
    for j = 1:ns
        center = 1 + sigma^2*j^2*ht + r*ht;
        left = 1/2*r*j*ht - 1/2*sigma^2*j^2*ht;
        right = -1/2*r*j*ht - 1/2*sigma^2*j^2*ht;

        Aj(j,j) = center;
        if j < ns, Aj(j, j+1) = right; end
        if j > 1,  Aj(j, j-1) = left;  end

        % bj vector
        bj(j, 1) = values(i*ns+j, 1);
        if j == 1 && type == "put"
            bj(j, 1) = bj(j, 1) - left*E;
        elseif j == ns && type == "call"
            bj(j, 1) = bj(j, 1) - right*(j*hs-E*exp(-r*(T-i*ht)));
        end
    end
    uj = Aj\bj;
    values((i-1)*ns+1 : i*ns, 1) = uj;
end

graph_surface(values, ns, nt, T, Sa, Sb, type);
end
```

Appendix B

Matlab Code for Crank-Nicolson Method for European Option

```
function values = BS_eur_cn(Sa, Sb, E, r, sigma, T, type, ns, nt)
dt = T / (nt+1);
ds = (Sb-Sa) / (ns+1);
sig2 = sigma^2;

values = zeros(ns*nt, 1);
if type == "put"
    for i = 1:ns, values((nt-1)*ns+i,1) = max(E-i*ds, 0); end
elseif type == "call"
    for i = 1:ns, values((nt-1)*ns+i,1) = max(i*ds-E, 0); end
end

for i = nt-1:-1:1
    Aj = sparse(ns, ns);
    bj = zeros(ns, 1);

    for j = 1:ns
        center = -2/dt-sig2*j^2-2*r;
        left   = 1/2*(sig2*j^2-r*j);
        right  = 1/2*(sig2*j^2+r*j);

        Aj(j,j) = center;
        if j > 1,  Aj(j, j-1) = left;  end
        if j < ns, Aj(j, j+1) = right; end

        % bj vector for the system
        index = i*ns+j;
        bj(j,1) = (-2/dt+sig2*j^2) * values(index,1);
        if j == 1
            bj(j,1) = bj(j,1) - right * values(index+1, 1);
            if type == "put", bj(j,1) = bj(j,1) - 2*left*E; end
        elseif j == ns
            bj(j,1) = bj(j,1) - left*values(index-1,1);
            if type == "call"
                bj(j, 1) = bj(j, 1) - 2*right*(Sb-E*exp(-r*(T-i*dt)));
            end
        else
            bj(j,1) = bj(j,1) - left * values(index-1, 1)...
                        -right * values(index+1, 1);
        end
    end
    uj = Aj\bj;
    values((i-1)*ns+1:i*ns, 1) = uj;
end

graph_surface(values, ns, nt, T, Sa, Sb, type);
end
```

Appendix C

Matlab Script for Generating Table 1

```
a = 0; b = 20;
T = 1; E = 10;
sigma = 0.20; r = 0.05;
ns = 599; nt = 599;
type = "call";

values_im = BS_eur_impl_fd(a, b, E, r, sigma, T, type, ns, nt);
values_cn = BS_eur_cn(a, b, E, r, sigma, T, type, ns, nt);
num_tests = {2,4,6,7,8,9,10,11,12,13,14,15,16};

disp("Price(S)  Actual      Crank-N.    Error          Impl. FD    Error");
for i = 1:length(num_tests)
    si = fix(num_tests{i} / (b-a) * (ns+1));
    ti = fix(0.5 / T * (nt+1));

    cn = values_cn((ti-2) * ns + si);
    im = values_im((ti-2) * ns + si);
    actual = BS_eur_actual(num_tests{i}, E, r, sigma, T, 0.5, type);

    fprintf("%4d       %8.6f    %8.6f    %8.6e    %8.6f    %8.6e \n", ...
                num_tests{i}, actual, cn, abs(actual-cn), im, abs(actual-im));
end
```

Appendix D

Matlab Script for Generating European Option Payoff Diagrams

```
r   = 0.1; sig  = 0.2;
E1  = 1;   E2   = 1.5;
T   = 2;   type = "call";
a = 0; b = 2;
ns = 599; nt = 599;
ts  = {0.5, 1.0, 1.5, 2.0};

values = BS_eur_impl_fd(a, b, E1, r, sig, T, type, ns, nt);
% values = BS_eur_spread(a, b, E1, E2, r, sig, T, type, ns, nt);

plots = zeros(ns, 4);
for i = 1:4
    ti = fix(ts{i} / T * (nt+1));
    plots(:,i) = values((ti-2)*ns+1 : (ti-1)*ns);
end

s = b/ns:b/ns:b;
figure
p1 = plot(s, plots(:,1)', "LineWidth", 2); hold on
p2 = plot(s, plots(:,2)', "LineWidth", 2); hold on
p3 = plot(s, plots(:,3)', "LineWidth", 2); hold on
p4 = plot(s, plots(:,4)', "LineWidth", 2);
legend([p1,p2,p3,p4], "T-t=1.5", "T-t=1.0", "T-t=0.5", "T-t=0");
ylabel("V"); xlabel("S");
if type == "call"
    title("Payoff Diagram at Various Times of Call Option");
elseif type == "put"
    title("Payoff Diagram at Various Times of Put Option");
end
```

Appendix E

Matlab Code for SOR Method

```
function x = SOR(A, b, x0, omega)

nIter = 0; maxIter = 5000;
error = 1; epi = 1E-6;
x = zeros(size(x0)); n = length(b);

while ((error > epi) && (nIter < maxIter))
    nIter = nIter + 1;
    for i = 1:n
        x(i) = b(i);
        if i > 1, x(i) = x(i) - A(i, i-1) * x(i-1);  end
        if i < n, x(i) = x(i) - A(i, i+1) * x0(i+1); end

        if (abs(A(i, i)) > 1E-10)
            x(i) = x(i)/A(i, i);
            x(i) = omega * x(i) + (1-omega)*x0(i);
        end
    end
    error = norm(x - x0);
    x0 = x;
end
end
```

Appendix F

Matlab Code for PSOR Method

```
function x = PSOR(A, b, x0, omega, payoffs)

nIter = 0; maxIter = 5000;
error = 1; epi = 1E-6;
x = zeros(size(x0)); n = length(b);

while ((error > epi) && (nIter < maxIter))
    nIter = nIter + 1;
    for i = 1:n
        x(i) = b(i);
        if i > 1, x(i) = x(i) - A(i, i-1) * x(i-1);  end
        if i < n, x(i) = x(i) - A(i, i+1) * x0(i+1); end

        if (abs(A(i, i)) > 1E-10)
            x(i) = x(i)/A(i, i);
            x(i) = omega * x(i) + (1-omega)*x0(i);
        end

        x(i) = max(x(i), payoffs(i));
    end
    error = norm(x - x0);
    x0 = x;
end

end
```

Appendix G

Matlab Script for Generating Table 4

```
format compact
a = 0;  b = 50;
sigma = 0.40;
E = 10; T = 1;
r = 0.1;
ns = 599; nt = 599;
type = 'put';

values_eu = BS_eur_cn(a, b, E, r, sigma, T, type, ns, nt);
values_am = BS_am_cn(a, b, E, r, sigma, T, type, ns, nt);

num_tests = {2,4,6,8,10,12,14,16};

disp('Asset   Payoff   3m  Amer.       Euro.       6m  Amer.       Euro.');
for i = 1:length(num_tests)
    si = fix(num_tests{i} / (b-a) * (ns+1));
    ti3 = fix(0.75 / T * (nt+1));
    ti6 = fix(0.5 / T * (nt+1));

    eu3 = values_eu((ti3-2) * ns + si);
    am3 = values_am((ti3-2) * ns + si);

    eu6 = values_eu((ti6-2) * ns + si);
    am6 = values_am((ti6-2) * ns + si);

    fprintf('%4d    %4.2f         %8.6f    %8.6f       %8.6f    %8.6f \n', ...
                num_tests{i}, max(10-num_tests{i},0), am3, eu3, am6, eu6);
end
```

Appendix H

Matlab Code for Crank-Nicolson Method for American Option

```
function values = BS_am_cn(Sa, Sb, E, r, sigma, T, type, ns, nt)

dt = T / (nt+1); ds = (Sb-Sa) / (ns+1);
sig2 = sigma^2;
values = zeros(ns*nt, 1);

if type == "put"
    for i = 1:ns, values((nt-1)*ns+i,1) = max(E-i*ds, 0); end
elseif type == "call"
    for i = 1:ns, values((nt-1)*ns+i,1) = max(i*ds-E, 0); end
end

for i = nt-1:-1:1
    Aj = sparse(ns, ns); bj = zeros(ns, 1);

    for j = 1:ns
        center = -2/dt-sig2*j^2-2*r;
        left   = 1/2*(sig2*j^2-r*j);
        right  = 1/2*(sig2*j^2+r*j);

        Aj(j,j) = center;
        if j > 1,  Aj(j, j-1) = left;  end
        if j < ns, Aj(j, j+1) = right; end

        % bj vector for the system
        index = i*ns+j;
        bj(j,1) = (-2/dt+sig2*j^2) * values(index,1);
        if j == 1
            bj(j,1) = bj(j,1) - right*values(index+1, 1);
            if type == "put", bj(j,1) = bj(j,1) - 2*left*E; end
        elseif j == ns
            bj(j,1) = bj(j,1) - left * values(index-1,1);
            if type == "call"
                bj(j,1) = bj(j,1) - 2*right*(j*ds-E);
            end
        else
            bj(j,1) = bj(j,1) - left * values(index-1, 1)...
                        - right * values(index+1, 1);
        end
    end
    uj = Aj\bj;
    if type == "put"
        for j = 1:ns, uj(j, 1) = max(uj(j,1), E-j*ds); end
    elseif type == "call"
        for j = 1:ns, uj(j, 1) = max(uj(j,1), j*ds-E); end
    end

    values((i-1)*ns+1 : i*ns, 1) = uj;
end

graph_surface(values, ns, nt, T, Sa, Sb, type);
end
```

Appendix I

Matlab Code for Bullish Call/Put Vertical Spread

```
function values = BS_eur_spread(Sa, Sb, E1, E2, r, sigma, T, type, ns, nt)

hs = (Sb-Sa) / (ns+1);
ht = T / (nt+1);

values = zeros(ns*nt, 1);
if type == "put"
    for i = 1:ns
        values((nt-1)*ns+i,1) = max(E2-i*hs,0)-max(E1-i*hs,0);
    end
elseif type == "call"
    for i = 1:ns
        values((nt-1)*ns+i,1) = max(i*hs-E1,0)-max(i*hs-E2,0);
    end
end

% calculates for put option
for i = nt-1:-1:1
    Aj = sparse(ns, ns);
    bj = zeros(ns, 1);
    for j = 1:ns
        center = 1 + sigma^2*j^2*ht + r*ht;
        left = 1/2*r*j*ht - 1/2*sigma^2*j^2*ht;
        right = -1/2*r*j*ht - 1/2*sigma^2*j^2*ht;

        Aj(j,j) = center;
        if j < ns, Aj(j, j+1) = right; end
        if j > 1,  Aj(j, j-1) = left;  end

        % bj vector
        bj(j, 1) = values(i*ns+j, 1);
        if j == 1 && type == "put"
            bj(j, 1) = bj(j, 1) - left*(E2-E1);
        elseif j == ns && type == "call"
            bj(j, 1) = bj(j, 1) + right* ...
                        (E1*exp(-r*(T-i*ht)) - E2*exp(-r*(T-i*ht)));
        end
    end
    uj = Aj\bj;
    values((i-1)*ns+1 : i*ns, 1) = uj;
end

graph_surface(values, ns, nt, T, Sa, Sb, type);
end
```

Appendix J

C Code for European Put Option

Note that code was modified by removing some unnecessary components (e.g. main function) to fit

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <assert.h>
#include <time.h>

#define MYMAX(x,y) ((x > y) ? x : y)

void eur() {
    double sig2 = 0.04; double E = 10; double r = 0.05;
    double T = 1.0; double b = 30.0;
    int nt = 599; double dt = (T / (double)(nt+1));
    int ns = 599; double ds = (b / (double)(ns+1));

    double eps = 1E-8, omega = 1.35; int maxIter = 5000;

    double *Aj = malloc(sizeof(double) * ns*ns);
    double *bj = malloc(sizeof(double) * ns);
    double *uj = malloc(sizeof(double) * ns);

    double *values = malloc(sizeof(double) * ns*nt);
    for(int i = 0; i < ns; i++)
        values[(nt-1)*ns+i] = MYMAX(E-i*ds, 0);

    double center, left, right;

    for(int i = nt-1; i >= 1; i--) {
        for(int j = 0; j < ns; j++) {
            double j2 = (j+1)*(j+1); double c = (sig2*j2-2/dt);
            center = -sig2*j2-2*r-2/dt;
            left   = (sig2*j2-r*(j+1))/2;
            right  = (sig2*j2+r*(j+1))/2;

            int main_diag = j*ns+j;
            Aj[main_diag] = center;
            if(j > 0)    Aj[main_diag-1] = left;
            if(j < ns-1) Aj[main_diag+1] = right;

            int index = i*ns + j;
            if(j == 0)
                bj[j] = c * values[index] - right * values[index+1]
                        - 2 * left * E;
            else if(j == ns-1)
                bj[j] = c * values[index] - left * values[index-1];
            else
                bj[j] = c * values[index] - right * values[index+1]
                        - left * values[index-1];
        }

        double error = 1.0; int nIter = 0;

        double temp[ns];
        memcpy(temp, &values[i*ns], sizeof(double)*ns);
```

```
55
56          while(error > eps && nIter < maxIter) {
57              nIter += 1;
58              for(int j = 0; j < ns; j++) {
59                  int m_center = j*ns+j;
60                  uj[j] = bj[j];
61                  if(j > 0)     uj[j] -= Aj[m_center-1] * uj[j-1];
62                  if(j < ns-1) uj[j] -= Aj[m_center+1] * temp[j+1];
63
64                  if(fabs(Aj[m_center]) > 1E-8) {
65                      uj[j] /= Aj[m_center];
66                      uj[j] = omega*uj[j] + (1-omega)*temp[j];
67                  }
68              }
69              double norm = 0;
70              for(int j = 0; j < ns; j++)
71                  if(fabs(uj[j]-temp[j]) > norm)
72                      norm = fabs(uj[j]-temp[j]);
73
74              error = norm;
75              memcpy(temp, uj, sizeof(double)*ns);
76          }
77
78          memcpy(&values[(i-1)*ns], uj, sizeof(double)*ns);
79      }
80
81      free(Aj); free(bj); free(uj); free(values);
82  }
```

7 References

[1] Wilmott P., Howison S. and Dewynne J. *Mathematics of Financial Derivatives: A Student Introduction.* The Press Syndicate of the University of Cambridge, third edition, 1995.

[2] Cornell University. (1998). Numerical Solution of Black-Scholes Equation [pdf]. Retrieved from http://www.cs.cornell.edu/info/Courses/Spring-98/CS522/content/lecture2.math.pdf.

[3] Yuwei Chen. (2017). *Numerical Methods for Pricing Multi-Asset Options* (Master Thesis). University of Toronto, Ontario.

[4] Ziqun Ye. (2013). *The Black-Scholes and Heston Model for Option Pricing* (Master Thesis). University of Waterloo, Ontario.

[5] York University. (2008). Finite Difference Methods: Dealing with American Options [pdf]. Retrieved from http://www.math.yorku.ca/∼hmzhu/Math-6911/lectures/Lecture6/6_BlkSch_FDM_Amer.pdf.

[6] Strong D.M. (2015). *Iterative Methods for Solving $Ax = b$ - The SOR Method.* Retrieved from https://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-the-sor-method.

[7] Yang S., Gobbert M.K. The Optimal Relaxation Parameter Applied to the Poisson Equation in Any Space Dimension. University of Maryland, Baltimore County, 2007.

[8] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations.* Cambridge Texts in Applied Mathematics. Cambridge University Press, 1996.

[9] Palczewski A. (n.d.). Numerical Methods for Solving American Options [pdf]. Retrieved from https://www.mimuw.edu.pl/ apalczew/CFP_lecture9.pdf.