Numerical Analysis Project

Arundhati Sinha, Ashutosh
Rishita Tyagi, Shaunak Gupta

4th December 2023

1 Introduction

1.1 Objectives of this report

- To discretize the Black-Scholes PDE using Forward and Backward Euler methods.
- To understand the difference between the use of discretization in American and European options.
- To find how the payoff of the option evolves over time and as the strike price changes.
- To find the optimal exercise point for an American put.
- To check how accurate our discretized solutions are by conducting an error analysis for the analytical solution versus the discretized solutions.

Throughout the course of this project, we found that within finite difference schemes, the Implicit method is a better choice due to lower errors and unconditional stability. We also concluded that the Black-Scholes PDE alone is not enough to realistically price an option.

2 Theoretical overview

Throughout this report, the following variables will always represent the following:

- S_t: Price of stock at time t,
- p: Payoff function,
- r: Interest rate,
- σ: Stock volatility,
- μ: Rate of return,
- K: Strike price.

The assumptions of the Black-Scholes model are as follows [1]:

- The interest rate of the market is known and constant through time.
- The stock price follows a random walk in continuous time. The stock price also follows a lognormal distribution. The variance of the rate of return on the stock is constant.
- The stock does not pay any dividends.
- The option is a European option.
- There are no transaction costs involved in the purchasing and selling of the option.
- Short selling is permitted and does not have penalties.

The Black-Scholes partial differential equation can be derived and written out as,

$$df = \left(\frac{\partial f}{\partial t} + \mu s_t \frac{\partial f}{\partial s} + \frac{1}{2}\sigma^2 s_t^2 \frac{\partial^2 f}{\partial s^2} \right) dt + \sigma s_t \frac{\partial f}{\partial s} dB_t.$$

Where B_t is a standard Brownian motion. The Black-Scholes equation can be represented by the following PDE:

$$\frac{\partial p}{\partial t} + rs\frac{\partial p}{\partial s} + \frac{\sigma^2 s^2}{2}\frac{\partial^2 p}{\partial s^2} - rp = 0. \tag{1}$$

$C(s,t)$ refers to a call option and with $P(s,t)$ refers to a put option.
The final time condition for a call option is:

$$C(s,T) = \max\{s-K, 0\}, \tag{2}$$

and the following boundary conditions

$$\begin{aligned} C(0,t) &= 0, \\ C(s_{\text{end}},t) &= s_{\text{end}} - Ke^{-r(T-t)}. \end{aligned} \tag{3}$$

The final time condition for a put option is:

$$P(s,T) = \max\{K-s, 0\}, \tag{4}$$

and the following boundary conditions

$$\begin{aligned} P(0,t) &= Ke^{-r(T-t)}, \\ P(s_{\text{end}},t) &= 0. \end{aligned} \tag{5}$$

Note that there is no closed form solution for American options and hence there is not a subsection for this.

2.1 Optimality of Early Exercise of Options

For this section, we have referred mainly to [2].

Let us assume a setup where: C designs a European Call, and c designs an American call. P designs a European Put, and p designs an American put. Let B(t,T) be the price at time t of a zero-coupon bond paying \$1 at time T. This means that the value KB(t,T) is the present value at time t of the exercise price K.

For a stock paying no dividends between now and expiration, we have:

$$\begin{aligned} c(S,t) \geq C(S,t) &\geq \max[0, S - KB(t,T)] \\ p(S,t) \geq P(S,t) &\geq \max[KB(t,T) - S, 0] \end{aligned} \tag{6}$$

Early exercise is never optimal for calls on non-dividend paying stocks. The call price equals the European call price in this case.

For puts on non-dividend stocks, early exercise can be optimal. There is a tradeoff between gaining time value of money on the strike price and losing the insurance value of the put.

For a stock paying dividends D at time π between now and expiration, we have:

$$\begin{aligned} c(S,t) \geq C(S,t) \geq S - [KB(t,T) + DB(t,\pi)] \\ p(S,t) \geq P(S,t) \geq [KB(t,T) + DB(t,\pi)] - S \end{aligned} \tag{7}$$

For dividend-paying stocks, early exercise of calls can be optimal right before a dividend date. Exercising early involves a tradeoff between gaining the dividend, losing time value of money on the strike price, and losing the call's insurance value.

For dividend-paying stocks, dividends reduce incentives for early put exercise since the put holder would not receive dividends. Early exercise involves trading off gaining time value on the strike price against losing insurance value and losing dividend payments.

There are cases where early exercise of American Put Options are favoured. We will be exploiting this theoretical case and determine an optimal early exercise strike.

2.2 Analytical solution for European options

In the case of European options, it is possible to find the exact solution of (1).
We define the following:

$$\begin{aligned} d_1 &= \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}, \\ d_2 &= \frac{\ln\left(\frac{S}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}. \end{aligned} \tag{8}$$

The exact analytical solution for the call option is then:

$$C(s,t) = SN(d_1) - Ke^{-r(T-t)}N(d_2). \tag{9}$$

Where $N(d)$ represents the cumulative normal density function.
The exact analytical solution for the put option is:

$$P(s,t) = -SN(-d_1) + Ke^{-rT}N(-d_2), \tag{10}$$

where, $N(-d) = 1 - N(d)$.

3 Method used

For this section we referred to [3, 4].
For this project we have decided to make use of the Forward and Backward Euler methods to discretize (1). Throughout the discretization, it is assumed that the grid points have equidistant spacing, i.e the distance between s_{i-1} and s_i is the same as the distance between s_i and s_{i+1} $\forall i = 1, ..., n-1$.

3.1 Discretization

Assume that f is a continuous function on a chosen interval. Using Taylor's expansion, we find that

$$\begin{aligned} f(x_i+\Delta x) &= f(x_i) + \Delta x \frac{df}{dx}(x_i) + \frac{\Delta x^2}{2}\frac{d^2 f}{dx^2}(x_i) + \dots \\ \Rightarrow \frac{df}{dx}(x_i) &= \frac{f(x_i+\Delta x) - f(x_i)}{\Delta x} + \mathcal{O}(\Delta x). \end{aligned} \tag{11}$$

$$\begin{aligned} f(x_i-\Delta x) &= f(x_i) - \Delta x \frac{df}{dx}(x_i) + \frac{\Delta x^2}{2}\frac{d^2 f}{dx^2}(x_i) + \dots \\ \Rightarrow \frac{df}{dx}(x_i) &= \frac{f(x_i) - f(x_i-\Delta x)}{\Delta x} + \mathcal{O}(\Delta x). \end{aligned} \tag{12}$$

(11) is known as the forward difference scheme, and (12) is known as the backwards difference scheme. Subtracting (12) from (11) we get a second order accurate discretization for the first derivative.

$$\frac{df}{dx}(x_i) = \frac{f(x_i+\Delta x) - f(x_i-\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2). \tag{13}$$

(13) is known as the central difference scheme.

3.1.1 Forward Euler Method

In mathematics and computational science, the Euler method (also called the forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge–Kutta method [5].

When given the values for t_0 and $y(t_0)$, and the derivative of y is a given function of t and y denoted as $y'(t) = f(t, y(t))$. Begin the process by setting $y_0 = y(t_0)$. Next choose a value h for the size of every step along t-axis, and set $t_n = t_0 + nh$ (or equivalently $t_{n+1} = t_n + h$). Now, the Euler method is used to find y_{n+1} from y_n and t_n.

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{14}$$

The value of y_n is an approximation of the solution at time t_n, i.e., $y_n \approx y(t_n)$. The Euler method is explicit.

P_j^n represents P observed at the j^{th} step on the x-axis and the n^{th} time step.

$$\frac{P_j^n - P_j^{n-1}}{\Delta t} = rP_j^n - r(j\Delta s)\frac{P_{j+1}^n - P_{j-1}^n}{2\Delta s} - \frac{\sigma^2 (j\Delta s)^2}{2}\frac{P_{j+1}^n - 2P_j^n + P_{j-1}^n}{\Delta s^2}.$$

$\Rightarrow$

$$\begin{aligned} P_j^{n-1} = P_{j+1}^n \left(\frac{rj\Delta t}{2} + \frac{\sigma^2 j^2 \Delta t}{2}\right) + P_j^n \left(1 - r\Delta t - \sigma^2 j^2 \Delta t\right) \\ + P_{j-1}^n \left(\frac{\sigma^2 j^2 \Delta t}{2} - \frac{rj\Delta t}{2}\right). \end{aligned} \tag{15}$$

Observe that if we replace $n-1$ with n and simultaneously replace n with $n+1$, we can write out our equation in a way that will be similar to the explicit method representation without altering the essence of our discretization. Then let,

$$\alpha_j = \left(\frac{rj\Delta t}{2} + \frac{\sigma^2 j^2 \Delta t}{2}\right), \tag{16}$$

$$\beta_j = \left(1 - r\Delta t - \sigma^2 j^2 \Delta t\right), \tag{17}$$

$$\gamma_j = \left(\frac{\sigma^2 j^2 \Delta t}{2} - \frac{rj\Delta t}{2}\right), \tag{18}$$

and let $\mathbf{P}^n$ be the same as (29). Using the above information, a matrix notation can be developed for the explicit discretization.

$$\mathbf{A} = \begin{bmatrix} \beta_1 & \alpha_1 & & & & & \\ \gamma_2 & \beta_2 & \alpha_2 & & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \gamma_{M-2} & \beta_{M-2} & \alpha_{M-2} \\ & & & & & \gamma_{M-1} & \beta_{M-1} \end{bmatrix}, \tag{19}$$

$$\mathbf{b}^{n+1} = \begin{bmatrix} \gamma_1 P_0^{n+1} \\ 0 \\ \vdots \\ 0 \\ \alpha_{M-1} P_M^n \end{bmatrix}. \tag{20}$$

$\mathbf{b}^n$ accounts for the boundary conditions of the model.
Using these notations, the explicit method discretization can now be written as:

$$\mathbf{P}^n = \mathbf{A}\mathbf{P}^{n+1} + \mathbf{b}^{n+1} \text{ for n = N-1, N-2, \ldots, 0.} \tag{21}$$

3.1.2 Backward Euler method

Consider the following ordinary differential equation

$$\frac{df}{dt} = f(t, y) \tag{22}$$

with initial value $y(t_0)$. Here the function f and the initial data t_0 and y_0 are known; the function y depends on the real variable t and is unknown. A numerical method produces sequence y_0, y_1. . . such that y_k approximates $y(t_0 + kh)$ where h is called the step size[5].

The backward Euler method computes the approximations using

$$y_{k+1} = y_k + hf(t_{k+1}, yi_{k+1}) \tag{23}$$

The backward Euler method is an implicit method: the new approximation ${y^0}_{k+1}$ appears on both sides of the equation, and thus the method needs to solve an algebraic equation for the unknown y_{k+1}.

We discretize (1) with respect to the Backward Euler method. This is an implicit method. All assumptions made with respect to grids and notation for the Forward Euler method also hold for this subsection.

$$\frac{P_j^{n+1}-P_j^n}{\Delta t} = rP_j^n - \frac{\sigma^2(j\Delta s)^2}{2}\frac{P_{j+1}^n - 2P_j^n + P_{j-1}^n}{\Delta s^2} - r(j\Delta s)\frac{P_{j+1}^n - P_{j-1}^n}{2\Delta s}.$$

$\Rightarrow$

$$\begin{aligned} P_j^{n+1} = P_{j+1}^n\left(-\frac{\sigma^2 j^2\Delta t}{2} - \frac{rj\Delta t}{2}\right) + P_j^n\left(1 + r\Delta t + \frac{\sigma^2 j^2 \Delta t}{2}\right) \\ + P_{j-1}^n\left(\frac{rj\Delta t}{2} - \frac{\sigma^2 j^2\Delta t}{2}\right) \end{aligned} \tag{24}$$

To simplify our notation and to be able to write an efficient code for this we need to represent our discretization in matrix form. Define the following:

$$a_j = \left(-\frac{\sigma^2 j^2\Delta t}{2} - \frac{rj\Delta t}{2}\right), \tag{25}$$

$$b_j = \left(1 + r\Delta t + \frac{\sigma^2 j^2\Delta t}{2}\right), \tag{26}$$

$$c_j = \left(\frac{rj\Delta t}{2} - \frac{\sigma^2 j^2\Delta t}{2}\right). \tag{27}$$

Using a_j, b_j, c_j and (24), the matrix representation is:

$$\mathbf{A} = \begin{bmatrix} b_1 & a_1 & & & & & \\ c_2 & b_2 & a_2 & & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & c_{M-2} & b_{M-2} & a_{M-2} \\ & & & & & c_{M-1} & b_{M-1} \end{bmatrix}, \tag{28}$$

$$\mathbf{P}^n = \begin{bmatrix} P_1^n \\ P_2^n \\ \vdots \\ P_{M-2}^n \\ P_{M-1}^n \end{bmatrix}. \tag{29}$$

We need to account for boundary conditions of our model. To do this, we define the following vector for the European options:

$$\mathbf{b}^n = \begin{bmatrix} c_1 P_0^n \\ 0 \\ \vdots \\ 0 \\ a_{M-1}P_M^n \end{bmatrix} \tag{30}$$

The Backward Euler discretization can be written as:

$$\mathbf{A}\mathbf{P}^n = \mathbf{P}^{n+1} - \mathbf{b}^n \text{ for n = N-1, N-2, \ldots, 0.} \tag{31}$$

For the American put option we create a boundary condition that updates itself throughout the discretization. If the value of the option at the i^{th} point on the space grid (asset price) is less than $(K - (i-1) \cdot \Delta s)$, we equate the option price to $(K - (i-1) \cdot \Delta s)$ instead.

4 Results

To aid with the computations and plotting of the graphs, the values of K, σ and time to expiry were found for SPDR (S&P500 ETF Trust) using yfinance on Python. These were used for the outputs provided in the enxt two subsections. We chose $S_{\min}$, $S_{\max}$ ourselves. In the 3-D plots, our three axes are: Strike price of the stock, Payoff, and time. The 2-D graphs do not have a time axis.

4.1 Backward Euler

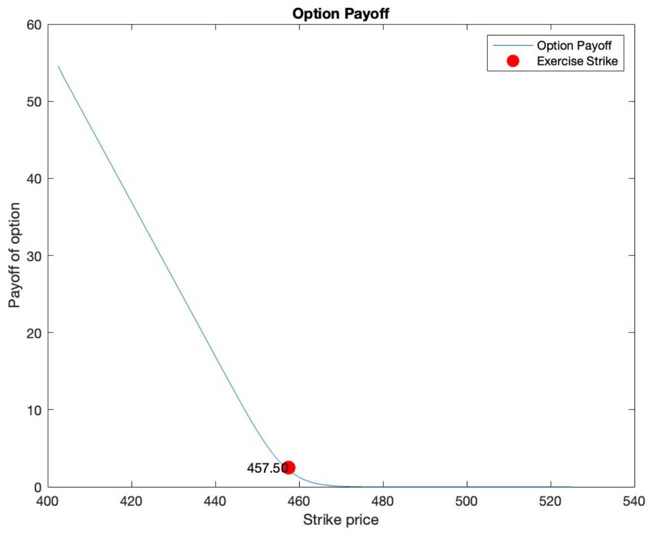

Figure 1: Payoff of an at-the-money put option when we use the Backward Euler method.

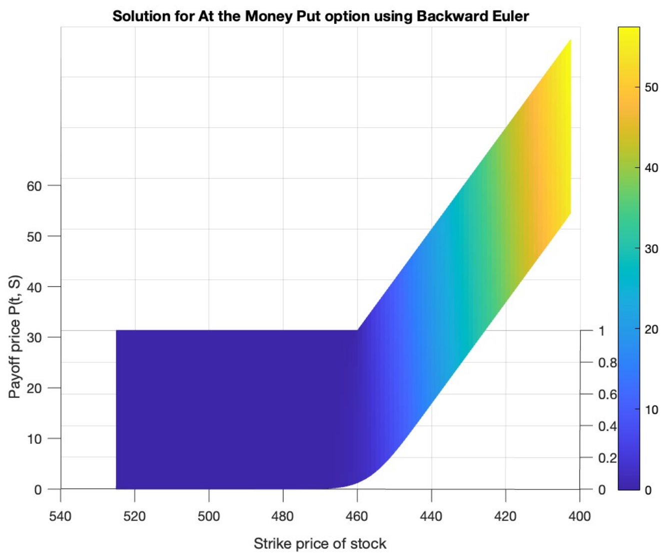

Figure 2: Payoff of an at-the-money put option when we use the Backward Euler method.

For the at-the-money put option our parameters for the code were as follows: $S_{\text{min}} = 400$, $S_{\text{max}} = 525$, K (strike price) $= 460$, T (time to expiry in months) $= 1$, r $= 5.25\%$, σ (volatility) $= 0.1072$.

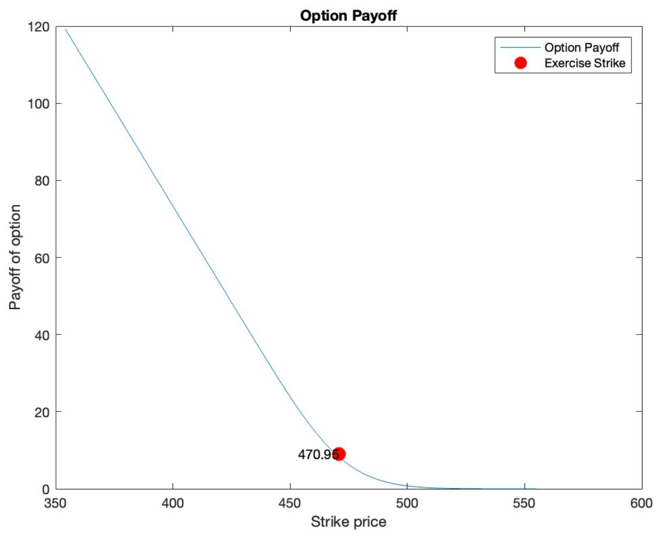

Figure 3: Payoff of an in-the-money put option when we use the Backward Euler method.

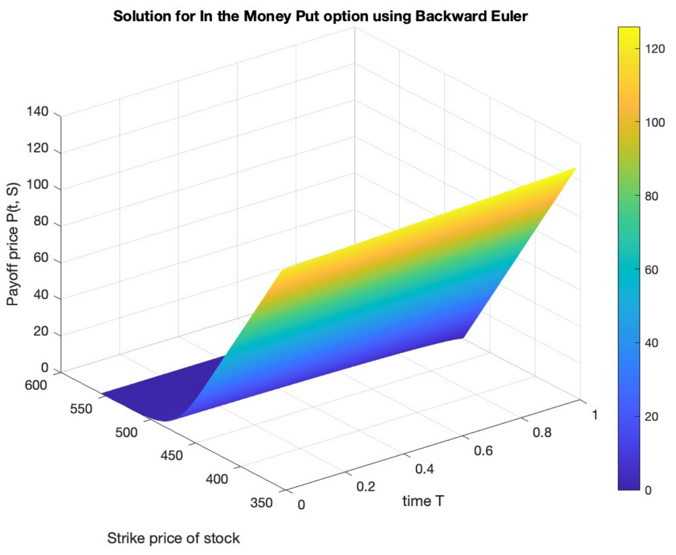

Figure 4: Payoff of an in-the-money put option when we use the Backward Euler method.

For the in-the-money put option our parameters for the code were as follows: $S_{\text{min}} = 350$, $S_{\text{max}} = 550$, K (strike price) $= 480$, T (time to expiry in months) $= 1$, r $= 5.25\%$, σ (volatility) $= 0.1425$.

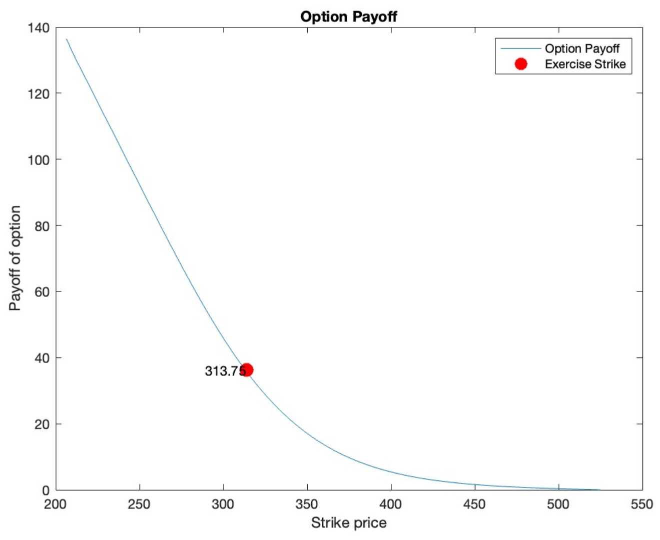

Figure 5: Payoff of an out-the-money put option when we use the Backward Euler method.

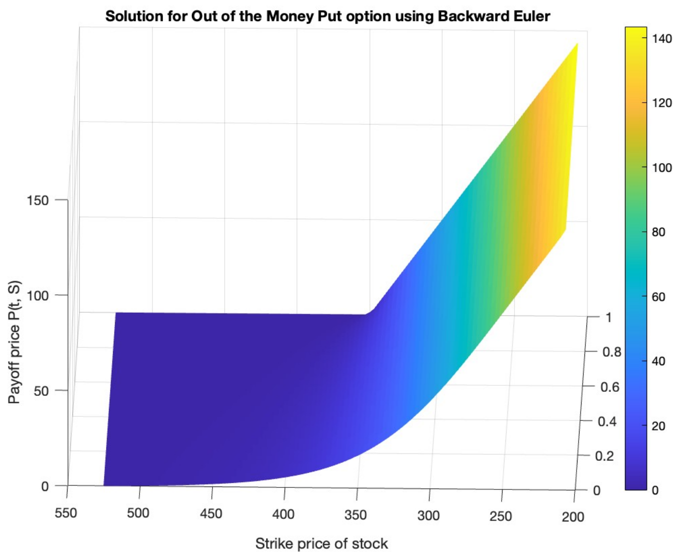

Figure 6: Payoff of an out-the-money put option when we use the Backward Euler method.

For the out-the-money put option our parameters for the code were as follows: $S_{\min} = 200$, $S_{\max} = 525$, K (strike price) $= 350$, T (time to expiry in months) $= 1$, r $= 5.25\%$, σ (volatility) $= 0.3569$.

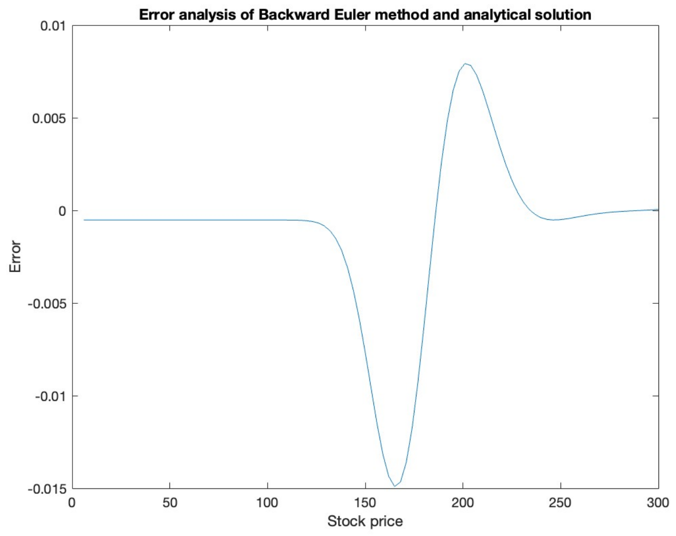

Figure 7: Error between discretization and analytical solution.

4.2 Forward Euler

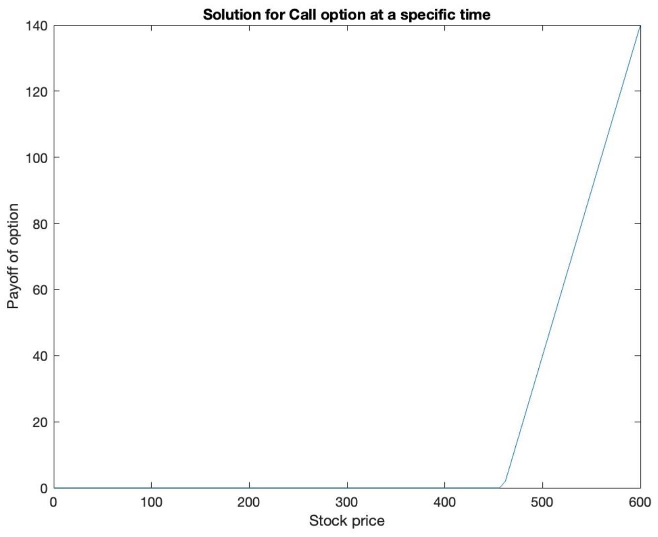

Figure 8: Payoff of an at-the-money call option when we use the Forward Euler method.

For the at-the-money call option our parameters for the code were as follows: $S_{\text{min}} = 0$, $S_{\text{max}} = 600$, K (strike price) = 460, T (time to expiry in months) = 1, r = 5.25%, σ (volatility) = 0.0855.

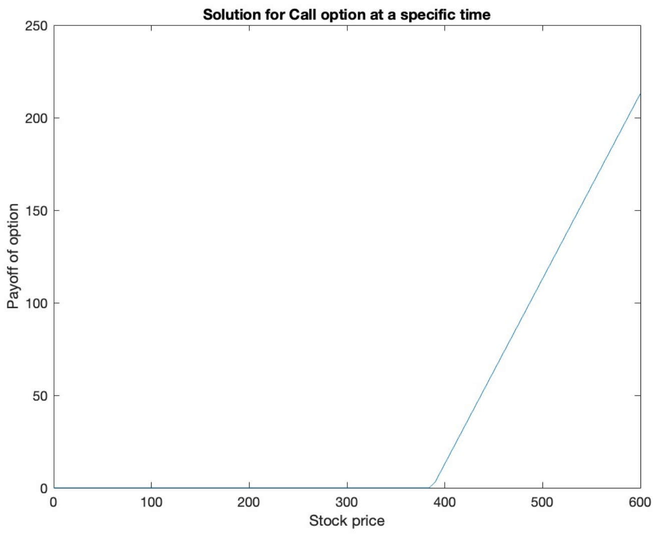

Figure 9: Payoff of an in-the-money call option when we use the Forward Euler method.

For the in-the-money call option our parameters for the code were as follows: $S_{\text{min}} = 0$, $S_{\text{max}} = 600$, K (strike price) = 387, T (time to expiry in months) = 1, r = 5.25%, σ (volatility) = 0.9590.

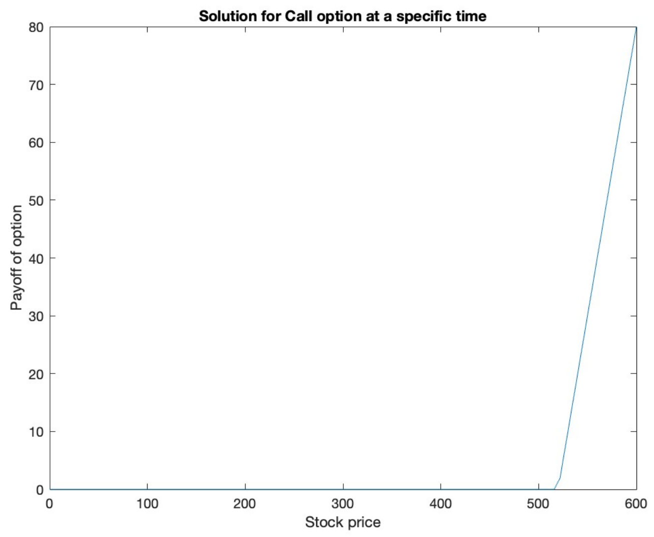

Figure 10: Payoff of an out-the-money call option when we use the Forward Euler method.

For the out-the-money call option our parameters for the code were as follows: $S_{\min} = 0$, $S_{\max} = 600$, K (strike price) = 520, T (time to expiry in months) = 1, r = 5.25%, σ (volatility) = 0.5313.

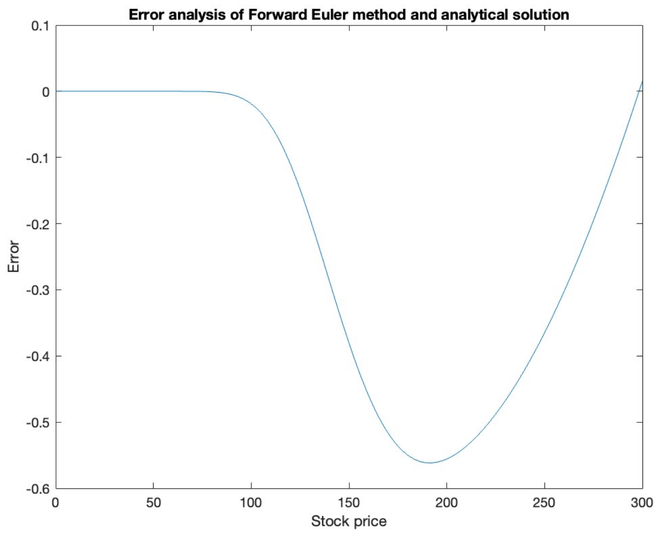

Figure 11: Error between discretization and analytical solution.

Note: The forward method is not a numerically stable method and hence changing the $S_{\min}$, $S_{\max}$, and the mesh grid can have an impact on the outputs and lead to a graph that makes no sense. To combat this it is possible to make the ds and dt values dependent on σ and each other.

4.3 Error analysis

We compared the payoff values obtained from our discretized models and the values obtained using the analytical solution: 10, 9. For the table below, we have chosen to keep r = 5.25%, $S_{\text{min}} = 0$, $S_{\text{max}} = 500$, T = 1, $\sigma = 0.3$, and vary the strike price.

	Error for FE results	Error for BE results
K = 50	-0.23329	0.0042102
K = 100	-0.25126	0.0042105
K = 150	-0.2692	0.0042303
K = 250	-0.2797	0.024934

Table 1: Errors between the two numerical methods and the analytical solution.

In the table below we will look at the errors when we vary σ for a constant K = 100. The remaining values stay the same.

	Error for FE results	Error for BE results
$\sigma = 0.1$	-0.19432	0.0041298
$\sigma = 0.2$	-0.21188	0.0041727
$\sigma = 0.35$	-0.28	0.0042112
$\sigma = 0.5$	-0.40626	0.005563

Table 2: Errors between the two numerical methods and the analytical solution.

We calculated the errors for the data that we used to creates our plots in the previous subsections. They are as follows:

	At-The-Money	Out-The-Money	In-The-Money
BE	0.050126	0.99266	0.23945
FE	-0.096826	1.6532	

Table 3: Errors between numerical methods and analytical solution.

Note: In order to keep our stability conditions for the Forward Euler methods we had to increase the value of S_{max} for the OTM and ITM case.

5 Discussion

Recall that it is never optimal to exercise a call option early and hence we did not alter our code to account for American calls. The red circle in our graphs represents the optimal exercise strike for an American put at a given time. From the figures under the results of the Backward Euler method we observe that for our chosen options it is usually optimal to exercise our puts early.

From the tables in the error analysis subsection we observe that the Backward Euler method appears to be more accurate than the Forward Euler scheme. In both cases, as we increase the strike price, while keeping other factors constant, the error increases in the absolute

sense. We observe a similar pattern for increasing volatility, while keeping other factors constant. Something to note is that as volatility increases, the payoff of our option also increases. This is because the risk associated with these stocks is higher and there must be higher compensation for this added risk.
In 3 we make the observation once again that the Backward Euler method appears to be more accurate than the Forward Euler scheme.

From our project, if we have the strike price, time to expiry, volatility, and risk free interest rate we can find the payoff of a call or put for either American or European options. $S_{\min}$, $S_{\max}$ are chosen user inputs. It is also possible to create a discretization scheme to obtain the prices of various options too by using interpolation at the final time step. However we found that for large values of K, this price was not always reliable which could be due to the limitations of the theoretical Black-Scholes formula. The line for interpolation is included in our code.

5.1 Extra theory and Limitations

For the following we referred to [6]. Euler's method in simple terms works by finding the slope at a known, traveling in a small amount h in the same direction, and then calculates a new slope and travel in the direction of the new slope and so on until the iterative exhaustion.

Proposition 1: *Euler's method provides answer with accuracy $\mathcal{O}(h)$*
Euler's method is designed in such a way that it takes $\mathcal{O}(|1/h|)$ steps to reach the desired value and the total error upon reaching that value is $\mathcal{O}(|h|)$. This creates a trade off between number of calculations and accuracy. To have an additional decimal in accuracy we must use new h which significantly increases the number of calculations. Which places Euler's method in a non-efficient computational category of methods to get high accuracy. This was seen as part of the project as well. Increasing the number of steps increased elapsed time to calculate answers.

For the following we referred to [7]. Just like any other mathematical models, the Black-Scholes model makes certain assumptions, which can lead to certain limitations.

1. Assumptions of Geometric Brownian Motion (GBM):
   The Black-Scholes model assumes that stock prices follow a geometric Brownian motion. This implies that the series of first differences of the log prices must be uncorrelated. However, empirical evidence shows small but statistically significant correlations in the differences of log prices at short time lags.

2. Normality Assumption: The model assumes that the returns (or innovations) are normally distributed. Empirical evidence, such as the kurtosis of returns on the S&P 500 series, suggests that returns are leptokurtic, indicating a tendency to exhibit outliers. The distribution of returns deviates significantly from a normal distribution.

3. Constant Volatility (σ): The model requires a constant volatility (σ) of the stock's continuously compounded rate of return over time. In practice, volatility may change, and periods of high volatility (volatility clustering) can persist for extended periods, leading to inadequacies in the model.

4. Volatility Clustering: Financial time series often exhibit volatility clustering, where periods of high volatility follow significant changes in the level of the original series. Autocorrelations of the squares of the differences of log prices may be significant for many lags, indicating that shocks to volatility persist for many periods.

5. Implied Volatility Variability: Implied volatility, inferred from observed option prices using the Black-Scholes formula in reverse, varies with strike price and maturity. In a perfect Black-Scholes world, implied volatility should be constant, but observed patterns such as the volatility smile and term structure indicate deviations from the model.

References

[1] F. Black and M. Scholes, “The pricing of options and corporate liabilities,” *Journal of political economy*, vol. 81, no. 3, p. 637, 1973.

[2] B. Dupoyet. Optimality of early exercise. [Online]. Available: https://faculty.fiu.edu/~dupoyetb/Options_seminar/lectures/lecture9.pdf

[3] Comparison of numerical methods. [Online]. Available: https://fse.studenttheses.ub.rug.nl/27951/1/bAppM_2022_SinhaA.pdf

[4] S. Mazumder, *Numerical methods for partial differential equations: Finite difference and finite volume methods.* Academic Press, 2016, p. 230–233.

[5] Euler method. [Online]. Available: https://en.wikipedia.org/wiki/Euler_method

[6] L. Evans. Limitations of euler’s method for numerical integration. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/55903/18-034Spring-2007/NR/rdonlyres/Mathematics/18-034Spring-2007/Projects/eulerl.pdf

[7] R. Chen. The black-scholes option pricing model. [Online]. Available: https://math.nyu.edu/~avellane/DSLecture5.pdf

Matlab code

```
clear all; close all;
%Creating a code to evaluate the value of an option using Backward Euler
%method

%%%%% VARIABLES USED %%%%%

S_min = 0;
S_max = 150;
K = 50; %strike price
T = 36; %time of expiry in months
r = 0.01; %rate of interest (risk-free)
sigma = 0.25; %volatility

%%%%% %%%%%
tic
%%% GRID MAKING %%%

Num_time = 5000; %time
dt = T/Num_time;
T_space = T:-dt:0;
T_points = (0:Num_time)*dt; %creating a vector with all the timesteps we
    consider

Num_S = 100; %Stock price grid points
ds = (S_max - S_min)/Num_S;
S_vec = S_min:ds:S_max; %creating a vector with all the timesteps we
    consider

i = 0:Num_S;

%%% %%%

p = zeros(Num_S+1, Num_time+1);

%put boundary
p(1,:) = K * exp(-r * (T_space));
p(:,Num_time+1) = max(K - S_vec, 0);
p(Num_S+1, :) = 0;

a = 0.5*(dt*r*i - dt*sigma^2*i.^2); %p(j-1) coeff
b = 1 + dt*r + dt*sigma^2*i.^2; %p(j) coeff
c = 0.5 * (-dt*sigma^2*i.^2 - dt*r*i); %p(j+1) coeff

%A = spdiags([a(2:end)' b(2:end)' c(2:end)'], -1:1, Num_S-1, Num_S-1);
A = diag(a(3:Num_S),-1) + diag(b(2:Num_S)) + diag(c(2:Num_S-1),1);
[L,U] = lu(A);

b = zeros(length(a)-2,1); %creating a vector for the boundary conditions

for j = Num_time:-1:1
    b(1) = a(2) * p(1,j); %we don't consider b(end) here as it is always 0
    p(2:Num_S, j) = A \ (p(2:Num_S,j+1) - b);
end
toc


```

```
price=interp1(S_vec,p(:, Num_time+1),S_min)


% good exercise point P(T) > K - S(T)
exercise_time = NaN;
exercise_strike = NaN;

for j = 2:Num_time
    exercise_point = find( p(j,1) >  (-S_vec(j) + K) );

    if ~isempty(exercise_point)
        exercise_time = Num_time*T_points(j);

        if exercise_point <= length(S_vec)
            exercise_strike = S_vec(exercise_time);
        else
            exercise_strike = NaN;
        end

        break;
    end
end


figure(1)
S_vec = S_vec(3:end);
p_1 = p(3:end,:);
plot(S_vec, p_1(:,1))
hold on
plot(exercise_strike, K - exercise_strike, 'ro', 'MarkerSize', 10, '
    MarkerFaceColor', 'r')
xlabel('Strike price')
ylabel('Payoff of option')
legend('Option Payoff' , 'Exercise Strike')
title('Option Payoff')
text(exercise_strike, K - exercise_strike, sprintf(' %.2f', exercise_strike)
    , 'HorizontalAlignment', 'right');
hold off

figure(2)
mesh(T_points, S_vec, p_1)
title('Solution for In the Money Put option using Backward Euler')
xlabel('time T')
ylabel('Strike price of stock')
zlabel('Payoff price P(t, S)')
colorbar
```

```
close all; clear all

% Parameters
S_min = 0;
S_max = 600;
K = 520; %strike price
T = 1; %time of expiry in months
r = 0.0525; %rate of interest (risk-free)
sigma = 0.5313; %volatility

% Grid
Num_time = 500; % time
```

```
dt = T/Num_time;
T_grid = (0:Num_time)*dt;

Num_S = 100; % Stock price grid points
ds = (S_max - S_min)/Num_S;
i = 0:Num_S;
S = S_min + (i)*ds;
S_vec = S_min:ds:S_max;
% Price value vector
p = zeros(Num_S+1, Num_time+1);

% Initial condition for a call option
p(:,1) = max(S - K, 0);

% Boundary conditions
p(1,:) = 0;
p(Num_S+1, :) = S_max - K*exp(-r*(T_grid));
a = 0.5*(dt*r*i - dt*sigma^2*i.^2); %p(j-1) coeff
b = 1 - dt*r - dt*sigma^2*i.^2; %p(j) coeff
c = 0.5 * (dt*sigma^2*i.^2 - dt*r*i); %p(j+1) coeff
% Discretization
for n = 1:Num_time
    for j = 2:Num_S
        %p(j, n+1) = p(j, n) + dt * (r * j * p(j, n) + 0.5 * sigma^2 * j^2 *
      (p(j+1, n) - 2 * p(j, n) + p(j-1, n)));
        p(j,n+1) = a(j)*p(j-1,n) + b(j)*p(j,n) + c(j)*p(j+1,n);
    end
end
price=interp1(S_vec,p(:, Num_time+1),S_min)

% Plotting
figure;
plot(S, p(:,1))
xlabel('Stock price')
ylabel('Payoff of option')
title('Solution for Call option at a specific time')
```

```
clear all; close all;
%Error analysis of the Backward Euler method

%%%%% VARIABLES USED %%%%%
S_min = 0;
S_max = 300;
K = 200; %strike price
T = 1; %time of expiry in months?
r = 0.0525; %rate of interest (risk-free)
sigma = 0.107; %volatility

%%%%% %%%%%
tic
%%% GRID MAKING %%%

Num_time = 500; %time
dt = T/Num_time;
T_grid = T:-dt:0;
T_vec = (0:Num_time)*dt;

Num_S = 100; %Stock price grid points
ds = (S_max - S_min)/Num_S;
```

```matlab
S_grid = S_min:ds:S_max;

i = 0:Num_S;
S = S_min + (i)*ds;

%%% %%%

p = zeros(Num_S+1, Num_time+1);

p(1,:) = K * exp(-r * (T_grid));
p(:,Num_time+1) = max(K - S_grid, 0);
p(Num_S+1, :) = 0;

a = 0.5*(dt*r*i - dt*sigma^2*i.^2); %p(j-1) coeff
b = 1 + dt*r + dt*sigma^2*i.^2; %p(j) coeff
c = 0.5 * (-dt*sigma^2*i.^2 - dt*r*i); %p(j+1) coeff

%A = spdiags([a(2:end)' b(2:end)' c(2:end)'], -1:1, Num_S-1, Num_S-1);
A = diag(a(3:Num_S),-1) + diag(b(2:Num_S)) + diag(c(2:Num_S-1),1);
[L,U] = lu(A);

b = zeros(size(A,2),1);
for j = Num_time:-1:1
    b(1) = a(2) * p(1,j); %we don't consider b(end) here as it is always 0
    p(2:Num_S, j) = A \ (p(2:Num_S,j+1) - b);
end
toc

ds_new = S_max/Num_S;
x=linspace(0,S_max,Num_S+1);

for t=0:Num_time
 for m=0:Num_S

      d1(m+1,t+1)=1/(sigma*sqrt(T-t*dt))*(log(m*ds_new/K)+(r+sigma^2/2)*(T-t*dt));
      d2(m+1,t+1)=d1(m+1,t+1)-sigma*sqrt(T-t*dt);

      exact_call(m+1,t+1)=normcdf(d1(m+1,t+1))*m*ds_new-normcdf(d2(m+1,t+1))*K*exp(-r*(T-t*dt));%exact value for call option
      exact_put(m+1,t+1)=K*exp(-r*(T-t*dt))-m*ds_new+exact_call(m+1,t+1); %exact value for put option
 end
end
S_gridnew = S_grid(3:end);
p_1 = p(3:end,:);
exact_put_new = exact_put(3:end, :);
error = exact_put_new(:,1) - p_1(:,1);

figure(1)
plot(S_gridnew, exact_put_new(:,1))
hold on
plot(S_gridnew, p_1(:,1))
hold off
title('Comparison of exact solution and discretized solution')
xlabel('Stock price')
ylabel('Payoff of option')
legend('Exact solution', 'Discretized solution')

```

```
figure(2)
plot(S_gridnew, error)
title('Error analysis of Backward Euler method and analytical solution')
xlabel('Stock price')
ylabel('Error')
```

```
close all; clear all;

%%%%% VARIABLES USED %%%%%
S_min = 0;
S_max = 300;
K = 200; %strike price
T = 1; %time of expiry in months?
r = 0.2; %rate of interest (risk-free)
sigma = 0.2; %volatility
%%%%% %%%%%

%%% GRID MAKING %%%
Num_time = 2300; %time
dt = T/Num_time;
T_grid = linspace(0, T, Num_time);

Num_S = 100; %Stock price grid points
ds = (S_max - S_min)/Num_S;

tau = (0:Num_time)*dt; %gives me an array of every value at a given timestep
i = 0:Num_S;
S = S_min + (i)*ds;

%%% %%%

%%% Price value vector %%%
p = zeros(Num_S+1, Num_time+1);

%Boundary conditions for a call option
p(:,1) = max(S - K, 0); %Initial condition when T-t=0
p(1,:) = 0;
p(Num_S+1, :) = S_max - K*exp(-r*(tau));


%%% %%%

%%%%% DISCRETIZATION %%%%%

for n = 1:Num_time
    for j = 2:Num_S
        p(j,n+1) = (0.5*(dt*(sigma^2)*(j^2) - dt*r*j))*p(j-1, n) + (1 - dt*r
     - dt*(sigma^2)*(j^2))*p(j,n) + (0.5 * (dt*sigma^2*j^2 + dt*r*j))*p(j+1,
    n);
    end
end
p = fliplr(p);

%%%%% %%%%%


plot(S, p(:,1))
%plot(S, p(:,1), '-', S, p(:,Num_time/2), '-', S, p(:, Num_time/4), '-')
xlabel('Stock price')
```

```
ylabel('Payoff of option')

h = S_max/Num_S;
dt = T/Num_time;
x=linspace(0,S_max,Num_S+1);

for t=0:Num_time
 for m=0:Num_S
     d1(m+1,t+1)=1/(sigma*sqrt(T-t*dt))*(log(m*h/K)+(r+sigma^2/2)*(T-t*dt));
     d2(m+1,t+1)=d1(m+1,t+1)-sigma*sqrt(T-t*dt);
     callex(m+1,t+1)=normcdf(d1(m+1,t+1))*m*h-normcdf(d2(m+1,t+1))*K*exp(-r
    *(T-t*dt));%exact value for call option
     %putex(m+1,t+1)=K*exp(-r*(T-t*dt))-m*h+callex(m+1,t+1); %from put-call
    parity %exact value for put option
 end
end

error = callex(:,1) - p(:,1);

figure(1)
plot(x, callex(:,1))
hold on
plot(x, p(:,1))
hold off
title('Comparison of exact solution and discretized solution')
xlabel('Stock price')
ylabel('Payoff of option')
legend('Exact solution', 'Discretized solution')

figure(2)
plot(x, error(:,1))
title('Error analysis of Forward Euler method and analytical solution')
xlabel('Stock price')
ylabel('Error')
```