Gonzalez_Claudio_Project

May 31, 2018

1 Numerical Solutions Of American Options With Dividends Using Finite Difference Methods

1.1 Recretation of this Method in Python by Claudio Gonzalez

1.1.1 Introduction

This paper studies the Black Scholes Model for American Options with dividens using the finited difference methods. The Black Scholes model proposes a valuation of options, which are contracts that allow the holder the right to buy or sell a stock. As we know the difference between American Options and European Options is that American Options can be exercised whenever the owner of the option wants to. In this case by using the finite difference method, we will solve the problem as a free-boundary problem for heat equations and use transformations to rewrite the problem in a linear complementarity form. This is done this way due to the fact that the explicit formula for finite difference method no longer works for American Options, so we need to use a projected Succesive Over-Relaxation (SOR) algorithm and use the explicit, implicit and crank-nicholson method. The purpose of this paper is to study these American Call/Put options with constant dividends and also analyze them with different dividends.

1.1.2 Mathematical Details

Free Boundary Problem An American option value must be greater than or equal to the payoff function $C(S,T) = \max(S-E,0)$ where C denotes the value of a Call option. The condition follows from the assumption of no arbitrage opportunity. In the case of an option with dividend we have that the $C(S,T) \to Se^{-d(T-t)}$, where d is the constant dividend rate. Assume $d > 0$, and for large values of S, the value of the European call option is less than its intrinsic value, namely $C(S,t) < \max(S-E,0)$. If theres early exercising, there is a straightforward arbitrage opportunity. We can purchase the call option and if we immediately exercise the option by paying E (the strike price) to the seller, we earn a risk-free profit $S-E-C$. However, under the efficient market hypothesis, arbitrage opportunity would not last long because a large number of arbitragers would react instantaneously and push up the option value back to the normal level. So when early exercise is permitted, $C(S,t) \geq \max(S-E,0)$.

The second constraint deals with the values of S (the stock price) that are optimal from the holder's point of view to exercise an American option. When holding the option, the option would have the same value as an European option (In this case American) and the Black-Scholes equation would hold for all S. When having the option of early exercising, at each time we need to determine not only the option value, but also for each value of S, whether or not it should

be exercised. This is known as a free boundary problem mentioned above. At time t there is a particular value of S which distinguishes two regions: one suggesting holding the option and the other one saying the holder should exercise it. We call this value $S_f(t)$, the optimal exercise price. Recall the second-order parabolic differential equation used to evaluate European option is given by

$$\sigma^2 S^2 \frac{\partial^2 C(S,t)}{2\partial S^2} + (r-d)S\frac{\partial C(S,t)}{\partial S} - rC(S,t) + \frac{\partial C(S,t)}{\partial t} = 0$$

where $C(S,t)$ is the value of a call option, $t \in [0,T]$.

The Obstacle Problem We view the obstacle problem the with $x = +-1$ as, the free boundary is the set of points $P(x = x_p)$ and $Q(x = x_Q)$, the points that define the contact region. Since the contact region concaves down, $u = f$ and $u < 0$, while $u > f$ and $u = 0$ when the string is above the obstacle. We also assume that

$$f(\pm 1) < 0, f(x) > 0 \text{ for some } -1 < x < 1, f'' < 0$$

to ensure there exists only one contact region. We then find $u(x)$ and the points P, Q such that

$$\begin{aligned} u(-1) = u(1) &= 0 \\ u'' &= 0, \ -1 < x < x_p, \ x_Q < x < 1 \\ u(x_p) = f(x_p), &\ u'(x_p) = f'(x_p) \\ u(x_Q) = f(x_Q), &\ u'(x_Q) = f'(x_Q) \\ u(x) = f(x), &\ x_p < x < x_Q \end{aligned}$$

We can then write:

$$u'' \cdot (u-f) = 0, \quad -u'' \geq 0, \quad (u-f) \geq 0$$

where $u(-1) = u(1) = 0$, and u, u' are continuous.

Linear complementarity form for American option valuation Now we reduce the valuation problem of American call option with dividends to a linear complementarity problem. First, we transform the American call with dividends problem from the original (S,t) variables to (x,τ), where x and τ refer to the following transformation:

$$\begin{aligned} x &= ln(\tfrac{S}{E}) \\ \tau &= \tfrac{\sigma^2}{2}(T-t) \\ v(x,\tau) &= \tfrac{C}{E} \\ v &= e^{\alpha x + \beta \tau} u(x,\tau) \end{aligned}$$

for $-\infty < x < \infty, \tau > 0$, where $\alpha = -0.5(k'-1), \beta = -0.25(k'-1)^2 - k, k' = \frac{r-d}{0.5\sigma^2}$ and $k = \frac{r}{0.5\sigma^2}$

Then we multiply $v(x,\tau)$ by E (strike price), gives us the value of the option with respect to variables S and t. Through this set of transformation, the Black-Scholes equation becomes

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}, \quad -\infty < x < \infty, \quad \tau > 0$$

which is a diffusion equation with initial condition

$$u(x,0) = v(x,0)e^{-\alpha x} = \max\left(e^{0.5(k'+1)x} - e^{0.5(k'-1)x}, 0\right)$$

We know that the obstacle problem for American call options with dividends is equivalent to finding $u(x,\tau)$ and the unknown optimal exercise boundary $x_f(\tau)$ such that

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}$$

for $x \leq x_f(\tau)$ and

$$u(x,\tau) = g(x,\tau)$$

for $x > x_f(\tau)$ with boundary conditions

$$\begin{aligned} u(x,0) &= g(x,0) \\ \lim_{x\to\infty} u(x,\tau) &= g(x,\tau) \\ \lim_{x\to-\infty} u(x,\tau) &= 0 \end{aligned}$$

where

$$g(x,\tau) = e^{(\frac{1}{4}(k'-1)^2+k)\tau} \max\left(e^{\frac{1}{2}(k'+1)x} - e^{\frac{1}{2}(k'-1)x}, 0\right)$$

We also have:

$$u(x,\tau) \geq g(x,\tau).$$

Since we will be focusing on numerical solutions using finite difference methods, we will restrict the problem to a finite interval. Therefore, we consider the problem only for x in the interval $[x, x^+]$, where x is a large negative number and x^+ is a large positive number. Having the boundary condition as"

$$u(x^-,\tau) = 0, \quad u(x^+,\tau) = g(x^+,\tau).$$

Now, rewriting in a linear complementarity form, we obtain

$$\left(\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2}\right) \cdot (u(x,\tau) - g(x,\tau)) = 0$$

with two constraints

$$\left(\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2}\right) \geq 0, \quad u(x,\tau) - g(x,\tau) \geq 0,$$

with the initial condition $u(x,0) = g(x,0)$ and the boundary conditions

$$\begin{aligned} u(x^-,\tau) &= g(x^-,\tau) = 0 \\ u(x^+,\tau) &= g(x^+,\tau) \end{aligned}$$

Now to obtain the numerical result of American option value, our approach is to solve

$$(\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2}) = 0$$

and make sure

$$u(x,\tau) - g(x,\tau) \geq 0.$$

Finite Difference Methods The Finite Difference Method procedure is as follows: We divide the (x,τ) plane into a regular finite mesh, and take finite-difference approximations of the linear complementarity form problem. We approximate the terms $\frac{\partial u}{\partial \tau} - \frac{\partial^2 u}{\partial x^2}$ by the finite differences on a regular mesh with step sizes $\partial\tau$ and ∂x, and we truncate so that x lies between $N\delta x$ and $N^+\delta x$, where N is a large negative number and N^+ is a large positive number. We divide the non-dimensional time to expiry of the option, $0.5\sigma^2 T$ into M equal time-steps so that $\delta\tau = \frac{\sigma^2 T}{2M}$

We divide the x-axis into equally spaced nodes a distance δx apart, and the τ-axis into equally spaced nodes a distance $\delta\tau$ apart. This divides the (x,τ) plane into a mesh, where the mesh points have the form $(n\delta x, m\delta)$,$N \leq x \leq N^+$, and $0 < m \leq M$. We write $u_n^m = u(n\delta x, m\delta\tau)$ as the value of $u(x,\tau)$ at the mesh point $(n\delta x, m\delta\tau)$,and $g_n^m = g(n\delta x, m\delta\tau)$.

Explicit Method Using a forward difference to approximate $\frac{\partial u}{\partial \tau}$, and a second-order central difference for $\frac{\partial^2 u}{\partial x^2}$, we can rewrite the diffusion equation as

$$\frac{u_n^{m+1} - u_n^m}{\delta\tau} + O(\delta\tau) = \frac{u_{n+1}^m - 2u_n^m + u_{n-1}^m}{(\delta x)^2} + O((\delta x)^2)$$

If we ignore terms of $O(\delta\tau)$ and $O((\delta x)^2)$, we can rearrange the same equation to give the difference equation

$$u_n^{m+1} = \alpha u_{n+1}^m + (1-2\alpha)u_n^m + \alpha u_{n-1}^m$$

where $\alpha = \frac{\delta\tau}{(\delta x)^2}$

At time step $m+1$, we already know the values of u_n^m for all n so we can explicitly calculate u_n^{m+1}. After we calculate u_n^{m+1} using the difference equation, we also need the equation to satisfy the constraint $u(x,\tau)g(x,\tau) \geq 0$. The scheme for this explicit method can be expressed as:

$$\begin{aligned} y_n^{m+1} &= \alpha u_{n+1}^m + (1-2\alpha)u_n^m + \alpha u_{n-1}^m \\ u_n^{m+1} &= \max\,(y_n^{m+1}, g_n^{m+1}) \end{aligned}$$

With this approach, the stability question arises. The difference equation system is stable if $0 < \alpha \leq \frac{1}{2}$ and unstable if $\alpha > \frac{1}{2}$

Fully-Implicit Method This method is different from the original Implicit Method. We use the backward-difference approximation for $\frac{\partial u}{\partial \tau}$, and the central difference for $\frac{\partial^2 u}{\partial x^2}$. The diffusion equation for American option valuation is:

$$\frac{u_n^{m+1} - u_n^m}{\delta\tau} + O(\delta\tau) = \frac{u_{n+1}^{m+1} - 2u_n^{m+1} + u_{n-1}^{m+1}}{(\delta x)^2} + O((\delta x)^2)$$

Dropping the error terms, we obtain

$$-\alpha u_{n-1}^{m+1} + (1+2\alpha)u_n^{m+1} - \alpha u_{n+1}^{m+1} = u_n^m$$

The linear complementarity problem is then approximated by

$$(-\alpha u_{n-1}^{m+1} + (1+2\alpha)u_n^{m+1} - \alpha u_{n+1}^{m+1} - u_n^m) \cdot (u_n^{m+1} - g_n^{m+1}) = 0$$

at time-step $m+1$. Now, let

$$u^m = \begin{pmatrix} u_{N^-+1}^m \\ u_{N^-+2}^m \\ \vdots \\ u_{N^+-2}^m \\ u_{N^+-1}^m \end{pmatrix} \tag{1}$$

$$g^m = \begin{pmatrix} g_{N^-+1}^m \\ g_{N^-+2}^m \\ \vdots \\ g_{N^+-2}^m \\ g_{N^+-1}^m \end{pmatrix} \tag{2}$$

$$b^m = \begin{pmatrix} b_{N^-+1}^m \\ b_{N^-+2}^m \\ \vdots \\ b_{N^+-2}^m \\ b_{N^+-1}^m \end{pmatrix} = \begin{pmatrix} u_{N^-+1}^m \\ u_{N^-+2}^m \\ \vdots \\ u_{N^+-2}^m \\ u_{N^+-1}^m \end{pmatrix} + \alpha \begin{pmatrix} g_{N^-}^m \\ 0 \\ \vdots \\ 0 \\ g_{N^+}^m \end{pmatrix} \tag{3}$$

and the coefficient matrix

$$C = \begin{pmatrix} (1+2\alpha) & -\alpha & 0 & \dots & 0 \\ -\alpha & (1+2\alpha) & -\alpha & \ddots & \vdots \\ 0 & -\alpha & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\alpha \\ 0 & \dots & 0 & -\alpha & (1+2\alpha) \end{pmatrix} \tag{4}$$

We want to solve the following constrained matrix problem

$$Cu^{m+1} = b^{m+1}$$

and check if the $u \geq g$ constraint is satisfied.

Let $A = (a_{i,j})$ be an arbitrary $n \times n$ complex matrix, and let

$$\Lambda_i \equiv \sum_{j=1, j\neq i} \mid a_{i,j} \mid, \quad 1 \leq i \leq n$$

then all the eigenvalues λ of A lie in the union of the disks

$$\mid z - a_{i,j} \mid \leq \Lambda_i, \quad 1 \leq i \leq n$$

Then, let $a_{i,j} = 1 + 2\alpha$. For $i = 1$ and n, $\Lambda_i = \alpha$. Then $z \geq a_{i,j} - \Lambda_i = 1 + \alpha$ so for $\alpha \geq 0, z$ is always positive. For $2 \leq i \leq n-1, \Lambda_i = (\alpha) + (\alpha) = 2\alpha$. Then $z \geq a_{i,j} - \Lambda_i = 1$. Thus, for $\alpha \geq 0$, all the eigenvalues of C are positive real numbers, suggesting C is invertible and we can rewrite the constrained matrix problem as

$$u^{m+1} = C^{-1} b^{m+1}$$

We can first form b^{m+1} from u^m and the boundary conditions. Using the known initial condition u^0, we can obtain u^{m+1} sequentially using an iterative method such as Projected SOR.

Projected SOR Algorithm The SOR method is used to speed up convergence of iterations. The projected SOR algorithm involves five steps. The first step we rewrite

$$-\alpha u_{n-1}^{m+1} + (1 + 2\alpha) u_n^{m+1} - \alpha u_{n+1}^{m+1} = u_n^m$$

from the Fully-implicit method as

$$u_n^{m+1} = \frac{1}{1 + 2\alpha} (b_n^{m+1} + \alpha (u_{n-1}^{m+1} + u_{n+1}^{m+1}))$$

where $N^- + 1 \leq n \leq N^+ - 1$.

Let $u_n^{m,k}$ be the k-th iterate for u_n^m. Let us denote the initial guess by $u_n^{m,0}$. As $k \to \infty$, we expect $u_n^{m,k} \to u_n^m$. We update the values as soon as they are available and add a correction term $u_n^{m,k+1} - u_n^{m,k}$ to the original $u_n^{m,k+1}$. We also incorporate an over-correction parameter ω, which has optimal values between 1 and 2, and finally, we take the maximum between the estimated $u_n^{m+1,k}$ and the payoff g_n^{m+1}. The fully-implicit scheme is thus expressed as

$$y_n^{m+1,k+1} = \tfrac{1}{1+2\alpha} (b_n^{m+1} - \alpha (u_{n-1}^{m+1,k+1} + u_{n+1}^{m+1,k}))$$

$$u_n^{m+1,k+1} = \max\left(u_n^{m+1,k} + \omega (y_n^{m+1,k+1} - u_n^{m+1,k}), g_n^{m+1}\right)$$

until the difference between $u_n^{m+1,k+1}$ and $u_n^{m+1,k}$ is small enough to be ignored, we set $u_n^{m+1} = u_n^{m+1,k+1}$.

Crank Nicholson Method One improvement the Crank-Nicolson Method has over the fully-implicit method is that it increases the temporal convergence rate from $O(\delta\tau)$ to $O((\delta\tau)^2)$. The equation of the scheme is

$$u_n^{m+1} - 0.5\alpha (u_{n-1}^{m+1} - 2u_n^{m+1} + u_{n+1}^{m+1}) = u_n^m + 0.5\alpha (u_{n-1}^m - 2u_n^m + u_{n+1}^m)$$

where $\alpha = \frac{\delta\tau}{(\delta x)^2}$

Let $Z_n^m = (1 - \alpha) u_n^m + 0.5\alpha (u_{n+1}^m + u_{n-1}^m)$, then

$$(1 + \alpha) u_n^{m+1} - 0.5\alpha (u_{n+1}^{m+1} + u_{n-1}^{m+1}) = Z_n^m$$

Define a new b matrix by

$$b^m = \begin{pmatrix} b_{N^-+1}^m \\ b_{N^-+2}^m \\ \vdots \\ b_{N^+-2}^m \\ b_{N^+-1}^m \end{pmatrix} = \begin{pmatrix} Z_{N^-+1}^m \\ Z_{N^-+2}^m \\ \vdots \\ Z_{N^+-2}^m \\ Z_{N^+-1}^m \end{pmatrix} + 0.5\alpha \begin{pmatrix} g_{N^-}^{m+1} \\ 0 \\ \vdots \\ 0 \\ g_{N^+}^{m+1} \end{pmatrix} \tag{5}$$

The tri-diagonal symmetric matrix C becomes

$$C = \begin{pmatrix} (1+\alpha) & -0.5\alpha & 0 & \dots & 0 \\ -0.5\alpha & (1+\alpha) & -0.5\alpha & \ddots & \vdots \\ 0 & -0.5\alpha & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -0.5\alpha \\ 0 & \dots & 0 & -0.5\alpha & (1+\alpha) \end{pmatrix} \tag{6}$$

Now, using Gerschgorin theorem we see that the matrix C is invertible. Similar to the fully-implicit method we adopt the projected SOR algoritm and obtain

$$y_n^{m+1,k+1} = \tfrac{1}{1+\alpha}(b_n^m + 0.5\alpha(u_{n-1}^{m+1,k+1} + u_{n+1}^{m+1,k}))$$

$$u_n^{m+1,k+1} = \max\left(u_n^{m+1,k} + \omega(y_n^{m+1,k+1} - u_n^{m+1,k}), g_n^{m+1}\right)$$

until the difference between $u_n^{m+1,k+1}$ and $u_n^{m+1,k}$ is small enoguh to be ignored, we set $u_n^{m+1} = u_n^{m+1,k+1}$

1.1.3 Python Implementation

In order to recreate these 3 methods we used python to recrete the tables. As seen below we decided to uses a class called European Option (the calculations are for the american option but I decided to maintain the name of the class that we used in the homework) that contained the three methods we used. For the explicit method first we defined all the variables and matrices that were going to be used in this method. After that we created a payoff matrix "g" which had rows of size 0 to N and columns from 0 to M. We then also inserted the boundary conditions into the payoff matrix. After that we defined our solution_mesh to show the nodes of the payoff matrix, and then follow the mathematical structure by doind another for loop. Finally we interpolated the solution_mesh with the range of X that we chose and X itself, providing us with uresult which finally gave us the explicit value through Black Scholes.

In terms of the implicit and crank nicholson the approach was different because we had to use the Projected SOR algorithm. So I first calculated the payoff matrices the same way as the explicit method. But after that I had to create a for loop to incorporate the SOR algorithm, where first we defined our xmatrix and had a while loop to follow the condition of union of disks of the fully implicit method. After that we defined our z value which then was used to calculate the final x matrix using a maximum formula. Finally we put x = u for our final u matrix, which then we interpolated and found the value for implicit or crank nicholson.

After all the calculations were performed, data frames for each table (df_table and df_table2) were created in order to show the specific value of the option for different stock prices and dividend yields. In order to do that various functions were created calling the functions of the class but with different values of stock price and dividend yield.

In terms of python functionality the main library used was numpy for things like maximum, log function, euller etc. Other than that basic for loops and while loops were used to calculated the vector and matrices needed like any other programming language.

The following tables shows our recreation of the results of this methodology

Table 4.1

In [5]:
```
df_table
```

Out[5]:

	Explicit	Implicit	Crank Nicholson
S			
4	0.0472844	0.0232534	0.037945
6	0.435506	0.319519	0.383974
8	1.38308	1.17096	1.26476
11	3.62624	3.32146	3.39759
12	4.51174	4.18567	4.2473
15	7.40486	7.03152	7.03611

Table 4.2

In [6]:
```
df_table2
```

Out[6]:

	d=0.03	d=0.05	d=0.06	d=0.08	d=0.11
S					
4	0.0315291	0.0279409	0.0262852	0.0232534	0.0193398
6	0.393693	0.362	0.347091	0.319519	0.282843
8	1.35921	1.27756	1.2397	1.17096	1.08034
11	3.63564	3.48747	3.42508	3.32146	3.20265
12	4.51214	4.35041	4.28613	4.18567	4.08522

Conclusion

In this project we calculated option pricing using finited difference methods for American Options. We studied how the American Option behaves with differente calculations approaches and how they change as we change the variables (in this case Stock Price and Dividend Yield). This paper performs a solid option value calculation based on the linear complementarity form which has not been performedd many times. Using the advantages of a programming language such as python the computation was done pretty smoothly and the numerical results are satisfactory as well. The difference between the explicit form and implicit/crank-nicholson is that the explicit method suffers limitations due to the number of steps performed in comparison to the implicit/crank-nicholson which adopt the SOR algorithm providing more accurate results.

From the tables that contain the numerical results, first of all we notice that in comparison to the actual paper the values are almost the same with very slight differences in the values. This tells us that the recreation of the paper was done properly and the values achieved are pretty accurate as well. In terms of the explicit form in table 4.1 the values are basically the same with the actual paper due to what we explain above about the numer of iterations. In terms of implicit/crank nicholson in table 4.1 and 4.2 theres a small difference in values, due to possible difference in iterations and how matlab and python behave slightly different as well. Although the values are slightly different the conclusion is the same. As the stock price increases so does the value of the option for any of the three methods. In terms of the dividend yield, we can see a negative correlation where as the dividend yield increases, the value of the option decreases as well. Due to all this, the paper overall was recreated almost identical even though we used python instead of matlab, making our numerical results almost the same. Overall the results are satisfactory with what the paper tries to achieve.

Appendix

In [1]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 12 16:29:46 2018

@author: Claudio
"""
import numpy as np
import pandas as pd
from scipy.stats import norm
import scipy.sparse
np.seterr(divide='ignore', invalid='ignore')
import math
from numpy.linalg import inv
from scipy.linalg import solve
from matplotlib.ticker import MultipleLocator, FormatStrFormatter
import scipy.sparse
import math
from scipy.stats.mstats import gmean


class EuropeanOption(object):
    def __init__(self,stock_price,strike_price):
        self.stock_price = stock_price
        self.strike_price= strike_price
        self.strike_price_1 = strike_price[0]
        self.strike_price_2 = strike_price[1]
        self.strike_price_3 = strike_price[2]

    def European_Call_Payoff(self,strike_price):
        european_call_payoff = np.maximum(self.stock_price-strike_price,
0)
        return european_call_payoff

    def European_Put_Payoff(self,strike_price):
        european_put_payoff = np.maximum(strike_price-self.stock_price,0
)
        return european_put_payoff
    def Black_Scholes_Explicit_FD_AO(self,t,maturity_date,M,Nminus,Nplus
,interest_rate,dividend_yield,volatility,strike_price,stock_max,stock_mi
n ):

        X = np.log(self.stock_price/strike_price)
        k = interest_rate/(0.5*volatility**2)
        kd = (interest_rate - dividend_yield)/(0.5*volatility**2)
        N = Nplus - Nminus
        dt = (0.5*(volatility**2)*maturity_date)/M
        Xzero = np.log(stock_min/strike_price)
        Xmax = np.log(stock_max/strike_price)
        dX = (Xmax-Xzero)/N
        alpha = dt/(dX*dX)
        solution_mesh = np.zeros((N+1,M+1))
        g = np.zeros((N+1,M+1))
        Xmesh = np.arange(Xzero,Xmax,dX)
        Xmesh2 = np.append(Xmesh,Xmax)
```

```
        #payoff matrix
        for j in range(0,N+1):
            for i in range(1,M+1):
                g[j,i] = np.exp((0.25*(kd-1)**2 + k)*((i)*dt))*np.maximu
m(np.exp(0.5*(kd+1)*(j+Nminus)*dX)-
                            np.exp(0.5*(kd-1)*(j+Nminus)*dX),0)
            g[j,0] = np.maximum(np.exp(0.5*(kd+1)*(j+Nminus)*dX)-np.exp(
0.5*(kd-1)*(j+Nminus)*dX),0)
        g[0,:]=0

        #Boundary conditions of the u matrix
        solution_mesh[:,0] = g[:,0]
        solution_mesh[0,:] = g[0,:]
        solution_mesh[N,:] = g[N,:]

        a = alpha
        b = 1 - 2*alpha
        c = alpha

        y = np.zeros((N+1,M+1))

        for p in range(0,M):
            y[1:N,p+1] = b*solution_mesh[1:N,p]+a*solution_mesh[2:N+1,p]
+c*solution_mesh[0:N-1,p]
            solution_mesh[1:N,p+1] = np.maximum(y[1:N,p+1],g[1:N,p+1])

        uresult = np.interp(X,Xmesh2,solution_mesh[:,M])

        explicit_value = strike_price*strike_price**(0.5*(kd-1))*self.st
ock_price**(-0.5*(kd-1))*np.exp((-0.25*(kd-1)**2 - k)*0.5*volatility**2*
(maturity_date-t))*uresult
        return explicit_value

    def Black_Scholes_Implicit_FD_AO(self,t,maturity_date,M,Nminus,Nplus
,interest_rate,dividend_yield,volatility,strike_price,stock_max,stock_mi
n,omega,tol ):
        X = np.log(self.stock_price/strike_price)
        tau = (0.5*volatility**2)*(maturity_date-t)
        k = interest_rate/(0.5*volatility**2)
        kd = (interest_rate - dividend_yield)/(0.5*volatility**2)
        N = Nplus - Nminus
        u = np.zeros((N+1,M+1))
        dt = (0.5*(volatility**2)*maturity_date)/M
        Xzero = np.log(stock_min/strike_price)
        Xmax = np.log(stock_max/strike_price)
        dX = (Xmax-Xzero)/N
        alpha = dt/(dX*dX)

        g = np.zeros((N+1,M+1))
        Xmesh = np.arange(Xzero,Xmax,dX)
        Xmesh2=np.linspace(Xzero,Xmax,N+1)
        Tmesh = np.arange(0,(0.5*volatility**2*maturity_date),dt)
        Tmesh2 = np.append(Tmesh,(0.5*volatility**2*maturity_date))

        #payoff matrix
        for j in range(0,N+1):
            for i in range(1,M+1):
```

```
                g[j,i] = np.exp((0.25*(kd-1)**2 + k)*((i-1)*dt))*np.maxi
mum(np.exp(0.5*(kd+1)*(j+Nminus-1)*dX)-
                np.exp(0.5*(kd-1)*(j+Nminus-1)*dX),0)
            g[j,0] = np.maximum(np.exp(0.5*(kd+1)*(j+Nminus-1)*dX)-np.ex
p(0.5*(kd-1)*(j+Nminus-1)*dX),0)
        g[0,:]=0


        #Boundary conditions of the u matrix
        u[:,0] = g[:,0]
        u[0,:] = g[0,:]
        u[N,:] = g[N,:]

        a = -alpha
        b = 1 + 2*alpha
        c = -alpha
        z = np.linspace(0,0,N-1)
        for p in range(0,M):
            x = np.maximum(u[1:N,p],g[1:N,p+1])
            temp = np.zeros((N-1,))
            temp[0] = a*g[0,p+1]
            temp[-1] = c*g[N,p+1]
            bt = u[1:N,p]-temp
            #b = b_one[0,:]
            xold = 1000*x
            n = len(x)
            while np.linalg.norm(xold-x)>tol:
                xold = x
                for i in range(0,n):
                    if i==0:
                        z = (bt[i]+alpha*x[i+1])/(1+2*alpha)
                        x[i] = np.maximum(omega*z + (1-omega)*xold[i],g[
i,p])
                    elif i==n-1:
                        z = (bt[i]+alpha*x[i-1])/(1 + 2*alpha)
                        x[i] = np.maximum(omega*z + (1-omega)*xold[i],g[
i,p])
                    else:
                        z = (bt[i]+alpha*(x[i-1]+x[i+1]))/(1 + 2*alpha)
                        x[i] = np.maximum(omega*z + (1-omega)*xold[i],g[
i,p])
                u[1:N,p+1] = x

        uresult = np.interp(X,Xmesh2,u[:,M])
        implicit_value = strike_price*strike_price**(0.5*(kd-1))*self.st
ock_price**(-0.5*(kd-1))*np.exp((-0.25*(kd-1)**2 - k)*0.5*volatility**2*
(maturity_date-t))*uresult
        return implicit_value

    def Black_Scholes_Crank_Nicholson_FD_AO(self,t,maturity_date,M,Nminu
s,Nplus,interest_rate,dividend_yield,volatility,strike_price,stock_max,s
tock_min,omega,tol ):

        X = np.log(self.stock_price/strike_price)
        k = interest_rate/(0.5*volatility**2)
        kd = (interest_rate - dividend_yield)/(0.5*volatility**2)
        N = Nplus - Nminus
```

```
        u = np.zeros((N+1,M+1))
        dt = (0.5*(volatility**2)*maturity_date)/M
        Xzero = np.log(stock_min/strike_price)
        Xmax = np.log(stock_max/strike_price)
        dX = (Xmax-Xzero)/N
        alpha = dt/(dX*dX)
        g = np.zeros((N+1,M+1))
        Xmesh2=np.linspace(Xzero,Xmax,N+1)
        Tmesh = np.arange(0,(0.5*volatility**2*maturity_date),dt)
        Tmesh2 = np.append(Tmesh,(0.5*volatility**2*maturity_date))

        #payoff matrix
        for j in range(0,N+1):
            for i in range(1,M+1):
                g[j,i] = np.exp((0.25*(kd-1)**2 + k)*((i-1)*dt))*np.maxi
mum(np.exp(0.5*(kd+1)*(j+Nminus-1)*dX)-
                np.exp(0.5*(kd-1)*(j+Nminus-1)*dX),0)
            g[j,0] = np.maximum(np.exp(0.5*(kd+1)*(j+Nminus-1)*dX)-np.ex
p(0.5*(kd-1)*(j+Nminus-1)*dX),0)
        g[0,:]=0

        #Boundary conditions of the u matrix
        u[:,0] = g[:,0]
        u[0,:] = g[0,:]
        u[N,:] = g[N,:]

        a= -alpha
        b = 1 + 2*alpha
        c = -alpha
        zmatrix = np.zeros((N+1,M+1))
        for p in range(0,M):
            temp = np.zeros((N-1,))
            temp[0] = a*g[0,p+1]
            temp[-1] = c*g[N,p+1]
            zmatrix[1:N,p] = (1-alpha)*u[1:N,p]+0.5*alpha*(u[2:N+1,p]+u[
0:N-1,p])
            bt = zmatrix[1:N,p]-temp
            x = np.maximum(u[1:N,p],g[1:N,p+1])
            #b = b_one[0,:]
            xold = 1000*x
            n = len(x)
            while np.linalg.norm(xold-x)>tol:
                xold = x
                for i in range(0,n):
                    if i==0:
                        z = (bt[i]+0.5*alpha*x[i+1])/(1+ alpha)
                        x[i] = np.maximum(omega*z + (1-omega)*xold[i],g[
i,p])
                    elif i==n-1:
                        z = (bt[i]+0.5*alpha*x[i-1])/(1 + alpha)
                        x[i] = np.maximum(omega*z + (1-omega)*xold[i],g[
i,p])
                    else:
                        z = (bt[i]+0.5*alpha*(x[i-1]+x[i+1]))/(1 + alpha
)
                        x[i] = np.maximum(omega*z + (1-omega)*xold[i],g[
i,p])
```

```
                u[1:N,p+1] = x

        uresult = np.interp(X,Xmesh2,u[:,M])
        crank_value = strike_price*strike_price**(0.5*(kd-1))*self.stock
_price**(-0.5*(kd-1))*np.exp((-0.25*(kd-1)**2 - k)*0.5*volatility**2*(ma
turity_date-t))*uresult
        return crank_value
```

In [2]:
```
strike_price = [8,8,8]
maturity_date = 1
stock_price4 = 4
stock_price6 = 6
stock_price8 = 8
stock_price11 = 11
stock_price12 = 12
stock_price15 = 15
stock_max = 150
stock_min = 0.4
t = 0
interest_rate = 0.1
dividend_yield = 0.08
volatility = 0.4
omega = 1.2
tol = 0.001
M =200
Nplus = 100
Nminus = -100
##Table 4.1
payoff_S4 = EuropeanOption(stock_price4,strike_price)
payoff_S6 = EuropeanOption(stock_price6,strike_price)
payoff_S8 = EuropeanOption(stock_price8,strike_price)
payoff_S11 = EuropeanOption(stock_price11,strike_price)
payoff_S12 = EuropeanOption(stock_price12,strike_price)
payoff_S15 = EuropeanOption(stock_price15,strike_price)

explicit_S4 = payoff_S4.Black_Scholes_Explicit_FD_AO(t,maturity_date,M,N
minus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],stoc
k_max,stock_min )
implicit_S4 = payoff_S4.Black_Scholes_Implicit_FD_AO(t,maturity_date,M,N
minus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],stoc
k_max,stock_min,omega,tol )
crank_S4 = payoff_S4.Black_Scholes_Crank_Nicholson_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],
stock_max,stock_min,omega,tol )

explicit_S6 = payoff_S6.Black_Scholes_Explicit_FD_AO(t,maturity_date,M,N
minus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],stoc
k_max,stock_min )
implicit_S6 = payoff_S6.Black_Scholes_Implicit_FD_AO(t,maturity_date,M,N
minus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],stoc
k_max,stock_min,omega,tol )
crank_S6 = payoff_S6.Black_Scholes_Crank_Nicholson_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],
stock_max,stock_min,omega,tol )

explicit_S8 = payoff_S8.Black_Scholes_Explicit_FD_AO(t,maturity_date,M,N
minus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],stoc
k_max,stock_min )
implicit_S8 = payoff_S8.Black_Scholes_Implicit_FD_AO(t,maturity_date,M,N
minus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],stoc
k_max,stock_min,omega,tol )
crank_S8 = payoff_S8.Black_Scholes_Crank_Nicholson_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],
stock_max,stock_min,omega,tol )
```

```
explicit_S11 = payoff_S11.Black_Scholes_Explicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],st
ock_max,stock_min )
implicit_S11 = payoff_S11.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],st
ock_max,stock_min,omega,tol )
crank_S11 = payoff_S11.Black_Scholes_Crank_Nicholson_FD_AO(t,maturity_da
te,M,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0
],stock_max,stock_min,omega,tol )

explicit_S12 = payoff_S12.Black_Scholes_Explicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],st
ock_max,stock_min )
implicit_S12 = payoff_S12.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],st
ock_max,stock_min,omega,tol )
crank_S12 = payoff_S12.Black_Scholes_Crank_Nicholson_FD_AO(t,maturity_da
te,M,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0
],stock_max,stock_min,omega,tol )

explicit_S15 = payoff_S15.Black_Scholes_Explicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],st
ock_max,stock_min )
implicit_S15 = payoff_S15.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0],st
ock_max,stock_min,omega,tol )
crank_S15 = payoff_S15.Black_Scholes_Crank_Nicholson_FD_AO(t,maturity_da
te,M,Nminus,Nplus,interest_rate,dividend_yield,volatility,strike_price[0
],stock_max,stock_min,omega,tol )



"Make the dataframe for the Error values"
df_table = pd.DataFrame(columns=['Explicit','Implicit','Crank Nicholson'
], index=['4','6','8','11','12','15'])
df_table.loc['15'] = pd.Series({'Explicit':explicit_S15, 'Implicit':impl
icit_S15, 'Crank Nicholson':crank_S15})
df_table.loc['12'] = pd.Series({'Explicit':explicit_S12, 'Implicit':impl
icit_S12, 'Crank Nicholson':crank_S12})
df_table.loc['11'] = pd.Series({'Explicit':explicit_S11, 'Implicit':impl
icit_S11, 'Crank Nicholson':crank_S11})
df_table.loc['8'] = pd.Series({'Explicit':explicit_S8, 'Implicit':implic
it_S8, 'Crank Nicholson':crank_S8})
df_table.loc['6'] = pd.Series({'Explicit':explicit_S6, 'Implicit':implic
it_S6, 'Crank Nicholson':crank_S6})
df_table.loc['4'] = pd.Series({'Explicit':explicit_S4, 'Implicit':implic
it_S4, 'Crank Nicholson':crank_S4})
df_table.index.name = 'S'
```

In [4]:
```
dividend_yield3 = 0.03
dividend_yield5 = 0.05
dividend_yield6 = 0.06
dividend_yield8 = 0.08
dividend_yield11 = 0.11

payoff_S4 = EuropeanOption(stock_price4,strike_price)
payoff_S6 = EuropeanOption(stock_price6,strike_price)
payoff_S8 = EuropeanOption(stock_price8,strike_price)
payoff_S11 = EuropeanOption(stock_price11,strike_price)
payoff_S12 = EuropeanOption(stock_price12,strike_price)
payoff_S15 = EuropeanOption(stock_price15,strike_price)

implicit_S4D3 = payoff_S4.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield3,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S6D3 = payoff_S6.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield3,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S8D3 = payoff_S8.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield3,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S11D3 = payoff_S11.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield3,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S12D3 = payoff_S12.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield3,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S15D3 = payoff_S15.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield3,volatility,strike_price[0
],stock_max,stock_min,omega,tol )

implicit_S4D5 = payoff_S4.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield5,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S6D5 = payoff_S6.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield5,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S8D5 = payoff_S8.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield5,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S11D5 = payoff_S11.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield5,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S12D5 = payoff_S12.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield5,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S15D5 = payoff_S15.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield5,volatility,strike_price[0
],stock_max,stock_min,omega,tol )

implicit_S4D6 = payoff_S4.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield6,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S6D6 = payoff_S6.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield6,volatility,strike_price[0],s
```

```
tock_max,stock_min,omega,tol )
implicit_S8D6 = payoff_S8.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield6,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S11D6 = payoff_S11.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield6,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S12D6 = payoff_S12.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield6,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S15D6 = payoff_S15.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield6,volatility,strike_price[0
],stock_max,stock_min,omega,tol )

implicit_S4D8 = payoff_S4.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield8,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S6D8 = payoff_S6.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield8,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S8D8 = payoff_S8.Black_Scholes_Implicit_FD_AO(t,maturity_date,M
,Nminus,Nplus,interest_rate,dividend_yield8,volatility,strike_price[0],s
tock_max,stock_min,omega,tol )
implicit_S11D8 = payoff_S11.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield8,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S12D8 = payoff_S12.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield8,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S15D8 = payoff_S15.Black_Scholes_Implicit_FD_AO(t,maturity_date
,M,Nminus,Nplus,interest_rate,dividend_yield8,volatility,strike_price[0
],stock_max,stock_min,omega,tol )

implicit_S4D11 = payoff_S4.Black_Scholes_Implicit_FD_AO(t,maturity_date,
M,Nminus,Nplus,interest_rate,dividend_yield11,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S6D11 = payoff_S6.Black_Scholes_Implicit_FD_AO(t,maturity_date,
M,Nminus,Nplus,interest_rate,dividend_yield11,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S8D11 = payoff_S8.Black_Scholes_Implicit_FD_AO(t,maturity_date,
M,Nminus,Nplus,interest_rate,dividend_yield11,volatility,strike_price[0
],stock_max,stock_min,omega,tol )
implicit_S11D11 = payoff_S11.Black_Scholes_Implicit_FD_AO(t,maturity_dat
e,M,Nminus,Nplus,interest_rate,dividend_yield11,volatility,strike_price[
0],stock_max,stock_min,omega,tol )
implicit_S12D11 = payoff_S12.Black_Scholes_Implicit_FD_AO(t,maturity_dat
e,M,Nminus,Nplus,interest_rate,dividend_yield11,volatility,strike_price[
0],stock_max,stock_min,omega,tol )
implicit_S15D11 = payoff_S15.Black_Scholes_Implicit_FD_AO(t,maturity_dat
e,M,Nminus,Nplus,interest_rate,dividend_yield11,volatility,strike_price[
0],stock_max,stock_min,omega,tol )


"Make the dataframe for the Error values"
df_table2 = pd.DataFrame(columns=['d=0.03','d=0.05','d=0.06','d=0.08','d
=0.11'], index=['4','6','8','11','12'])
#df_table.loc['15'] = pd.Series({'d=0.03':implicit_S15D3, 'd=0.05':impli
```

```
cit_S15D5, 'd=0.06':implicit_S15D6, 'd=0.08':implicit_S15D8, 'd=0.11':im
plicit_S15D11})
df_table2.loc['12'] = pd.Series({'d=0.03':implicit_S12D3, 'd=0.05':impli
cit_S12D5, 'd=0.06':implicit_S12D6, 'd=0.08':implicit_S12D8, 'd=0.11':im
plicit_S12D11})
df_table2.loc['11'] = pd.Series({'d=0.03':implicit_S11D3, 'd=0.05':impli
cit_S11D5, 'd=0.06':implicit_S11D6, 'd=0.08':implicit_S11D8, 'd=0.11':im
plicit_S11D11})
df_table2.loc['8'] = pd.Series({'d=0.03':implicit_S8D3, 'd=0.05':implici
t_S8D5, 'd=0.06':implicit_S8D6, 'd=0.08':implicit_S8D8, 'd=0.11':implici
t_S8D11})
df_table2.loc['6'] = pd.Series({'d=0.03':implicit_S6D3, 'd=0.05':implici
t_S6D5, 'd=0.06':implicit_S6D6, 'd=0.08':implicit_S6D8, 'd=0.11':implici
t_S6D11})
df_table2.loc['4'] = pd.Series({'d=0.03':implicit_S4D3, 'd=0.05':implici
t_S4D5, 'd=0.06':implicit_S4D6, 'd=0.08':implicit_S4D8, 'd=0.11':implici
t_S4D11})
df_table2.index.name = 'S'
```

Bibliography

"Numerical Solutions Of American Options With Dividends Using Finite Difference Methods" by Nsoki Mavinga