FE 621-HW 3 Report

March 24, 2021

[1]:
```
from scipy.stats import norm
import scipy.linalg as linalg
import numpy as np
import datetime as dt
import pandas as pd
import yfinance as yf
from datetime import datetime
import math
import matplotlib.pyplot as plt
```

1 Problem 1

1.1 Finite Difference Class

[3]:
```
class FiniteDifferences(object):

    def __init__(self,S0,K,r,T,sigma,M,N,is_call=True):


        self.S0=S0
        self.K=K
        self.r=r
        self.T=T
        self.sigma=sigma
        #self.Smax=Smax
        self.M=int(M)
        self.N=int(N)
        self.Smax=self.S0*np.exp(self.N* 1/(self.M+2))
        Smax=self.S0*np.exp(self.N* 1/(self.M+2))
        self.is_call= is_call
        self.dS=Smax/float(self.M) # Dividing S into M number of intervals
        self.dt=T/float(self.N)    # Dividing T into N number of intervals
        self.i_values=np.arange(self.M)
        self.j_values=np.arange(self.N)
        self.grid=np.zeros(shape=(self.M+1,self.N+1))  # Constructing the grid
        self.boundary_conds= np.linspace(0,Smax,self.M+1)
```

```
    def _setup_boundary_conditions_(self):

        pass


    def _setup_coefficients_(self):

        pass



    def _traverse_grid_(self):

        pass



    def _interpolate_(self):


        return np.interp(self.S0, self.boundary_conds, self.grid[:,0])


    def price(self):


        self._setup_boundary_conditions_()

        self._setup_coefficients_()

        self._traverse_grid_()


        return self._interpolate_()
```

1.1.1 FDExplicitEu class

[6]:
```
class FDExplicitEu(FiniteDifferences):



    def _setup_boundary_conditions_(self):


        if self.is_call:

            self.grid[:,-1]= np.maximum(
                self.boundary_conds-self.K,0)
```

```
            self.grid[-1, :-1]= (self.Smax - self.K) * \
                                    np.exp(-self.r *

                                           self.dt *

                                           (self.N-self.j_values))


        else:
            self.grid[:, -1] = \
                np.maximum(self.K-self.boundary_conds, 0)
            self.grid[0, :-1] = (self.K - self.Smax) * \
                                    np.exp(-self.r *
                                            self.dt *
                                            (self.N-self.j_values))


    def _setup_coefficients_(self):
        self.a = 0.5*self.dt*((self.sigma**2) *
                              (self.i_values**2) -
                              self.r*self.i_values)
        self.b = 1 - self.dt*((self.sigma**2) *

                                (self.i_values**2) +
                                  self.r)
        self.c = 0.5*self.dt*((self.sigma**2) *
                              (self.i_values**2) +
                              self.r*self.i_values)


    def _traverse_grid_(self):
        for j in reversed(self.j_values):
            for i in range(self.M)[2:]:
                self.grid[i,j] = self.a[i]*self.grid[i-1,j+1] +\
                                self.b[i]*self.grid[i,j+1] + \
                                self.c[i]*self.grid[i+1,j+1]
```

European Call Option price using explicit difference method

[8]:
```
#### European Call Option price using explicit difference method

option = FDExplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
↪is_call=True)

print(option.price())
```

17.811600777587547

European Put Option price using explicit difference method

[9]:
```
option = FDExplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
 ↪is_call=False)

print(option.price())
```

0.14529840919604145

1.1.2 FDImplicitEu class

[10]:
```
class FDImplicitEu(FDExplicitEu):
    def _setup_coefficients_(self):
        self.a = 0.5*(self.r*self.dt*self.i_values -
            (self.sigma**2)*self.dt*(self.i_values**2))
        self.b = 1 + \
            (self.sigma**2)*self.dt*(self.i_values**2) + \
            self.r*self.dt
        self.c = -0.5*(self.r * self.dt*self.i_values +
            (self.sigma**2)*self.dt*(self.i_values**2))
        self.coeffs = np.diag(self.a[2:self.M], -1) + \
            np.diag(self.b[1:self.M]) + \
            np.diag(self.c[1:self.M-1], 1)



    def _traverse_grid_(self):
        """ Solve using linear systems of equations """
        P, L, U = linalg.lu(self.coeffs)
        aux = np.zeros(self.M-1)

        for j in reversed(range(self.N)):
            aux[0] = np.dot(-self.a[1], self.grid[0, j])
            x1 = linalg.solve(L, self.grid[1:self.M, j+1]+aux)
            x2 = linalg.solve(U, x1)
            self.grid[1:self.M, j] = x2
```

European Call Option price using implicit difference method

[11]:
```
option= FDImplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
 ↪is_call=True)

print (option.price())
```

11.542325827594295

European Put Option price using implicit difference method

[12]:
```
option= FDImplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
↪is_call=False)

print (option.price())
```

13.26916724241089

1.1.3 Crank-Nicolson Finite Difference method

[13]:
```
class FDCnEu(FDExplicitEu):
    def _setup_coefficients_(self):
        self.alpha = 0.25*self.dt*(
        (self.sigma**2)*(self.i_values**2) -
        self.r*self.i_values)
        self.beta = -self.dt*0.5*(
        (self.sigma**2)*(self.i_values**2) +
        self.r)
        self.gamma = 0.25*self.dt*(
        (self.sigma**2)*(self.i_values**2) +
        self.r*self.i_values)
        self.M1 = -np.diag(self.alpha[2:self.M], -1) + \
        np.diag(1-self.beta[1:self.M]) - \
        np.diag(self.gamma[1:self.M-1], 1)
        self.M2 = np.diag(self.alpha[2:self.M], -1) + \
        np.diag(1+self.beta[1:self.M]) + \
        np.diag(self.gamma[1:self.M-1], 1)
    def _traverse_grid_(self):
        """  We are solving the linear systems of equations """
        P, L, U = linalg.lu(self.M1)
        for j in reversed(range(self.N)):
            x1 = linalg.solve(L,
            np.dot(self.M2,
            self.grid[1:self.M, j+1]))
            x2 = linalg.solve(U, x1)
            self.grid[1:self.M, j] = x2
```

European Call Option price using Crank-Nicolson method

[14]:
```
option = FDCnEu(S0=50, K=50, r=0.1, T=1, sigma=0.4, M=3, N=3, is_call=True)

print(option.price())
```

3.6597266500731678

European Put Option price using Crank-Nicolson method

[15]:
```
option = FDCnEu(S0=50, K=50, r=0.1, T=1, sigma=0.4, M=3, N=3, is_call=False)

print(option.price())
```

7.143375995657307

1.1.4 Question e

Parameters: S0 = 100;K = 100, T = 1 year, sigma = 20%; r = 6%; div = 2%

Explicit EUCall

[16]:
```
option = FDExplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
 ↪is_call=True)

print(option.price())
```

17.811600777587547

Explicit EUPut

[17]:
```
option = FDExplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
 ↪is_call=False)

print(option.price())
```

0.14529840919604145

Implicit EUCall

[18]:
```
option= FDImplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
 ↪is_call=True)

print (option.price())
```

11.542325827594295

Implicit EUPut

[19]:
```
option= FDImplicitEu(S0=100, K=100, r=0.06, T=1, sigma=0.2, M=3, N=3,␣
 ↪is_call=False)

print (option.price())
```

13.26916724241089

Crank-Nicolson EUCall

[20]:
```
option = FDCnEu(S0=100, K=100, r=0.1, T=1, sigma=0.2, M=3, N=3, is_call=True)

print(option.price())
```

```
11.129968769236118
```

Crank-Nicolson EUPut

[21]:
```python
option = FDCnEu(S0=100, K=100, r=0.1, T=1, sigma=0.2,M=3, N=3, is_call=False)

print(option.price())
```

```
11.643816394845299
```

1.1.5 Question f

[22]:
```python
## Blackscholes function to calulate call option price

# S= Stock Price

# K= Strike Price

# t= Expiration Date

# sig= Volatility

# optype= Type

# r= risk free interest rate



def blackscholes_C(S,K,t,sig,r=0.0008):

    d1= (np.log(S/K)+(r+sig**2/2)*t)/(sig*np.sqrt(t))

    d2= d1-sig*np.sqrt(t)

    call_price=norm.cdf(d1,0,1)*S-  norm.cdf(d2,0,1)*K*np.exp(-r*t)

    return call_price
```

[23]:
```python
blackscholes_C(S=100,K=100,t=1,sig=0.20,r=0.06)
```

[23]:
```
10.98954915262599
```

2 Problem 2

2.1 Importing and Organising Data

[24]:
```
# Importing option chain from yahoo finance, and organizing the dataframe

def get_optionchain(inpt,exprdt):

# expiration date format should be like this "2020-03-12"


    stock=yf.Ticker(inpt)

    opt=stock.option_chain(exprdt)

    call=opt.calls

    put=opt.puts

    option_chain=call.append(put)




    a=option_chain.
 ↪drop(["lastTradeDate","change","percentChange","volume","openInterest","inTheMoney","contra

    a["Expiration Date"]=exprdt


    a.columns=['Option Name', 'Strike',"Last Price","Bid","Ask","Implied␣
 ↪Volatility","Expiration Date"]

    a.reset_index(drop=True,inplace=True)



    # Loop to assign P or C values depending on the type of the option
    for i,j in a.iterrows():

        if j["Option Name"][-9]=="P":


            a.loc[i,"Type"]="put"
```

```
        elif j["Option Name"][-9]=="C":

            a.loc[i,"Type"]="call"



    a = a[['Option Name',"Expiration Date","Type",'Strike',"Bid","Ask","Last␣
↪Price","Implied Volatility"]]

    a.sort_values(by=['Strike'], inplace=True, ascending=True)



    return a
```

[25]:
```
# example for the function above


a1=get_optionchain("AMZN",exprdt="2021-03-26")
a2=get_optionchain("AMZN",exprdt="2021-04-16")
a3=get_optionchain("AMZN",exprdt="2021-05-21")


AMZN_opt1=a1.append(a2).append(a3)

AMZN_opt1=AMZN_opt1.reset_index(drop=True)

AMZN_opt1
```

[25]:
```
            Option Name Expiration Date  Type  Strike     Bid     Ask  \
0     AMZN210326P01660000      2021-03-26   put  1660.0     0.0     0.0
1     AMZN210326P01680000      2021-03-26   put  1680.0     0.0     0.0
2     AMZN210326P01690000      2021-03-26   put  1690.0     0.0     0.0
3     AMZN210326C01700000      2021-03-26  call  1700.0  1432.5  1442.5
4     AMZN210326P01700000      2021-03-26   put  1700.0     0.0     0.0
…                     …           …    …     …      …      …
1133  AMZN210521C04700000      2021-05-21  call  4700.0     0.0     0.0
1134  AMZN210521C04800000      2021-05-21  call  4800.0     0.0     0.0
1135  AMZN210521C04900000      2021-05-21  call  4900.0     0.0     0.0
1136  AMZN210521C05000000      2021-05-21  call  5000.0     0.0     0.0
1137  AMZN210521P05000000      2021-05-21   put  5000.0     0.0     0.0

      Last Price  Implied Volatility
0           0.01            0.500005
```

```
1           0.03           0.500005
2           0.05           0.500005
3        1363.05           3.289553
4           0.03           0.500005
…            …               …
1133        1.78           0.125009
1134        1.60           0.250007
1135        1.33           0.250007
1136        1.25           0.250007
1137     1919.00           0.000010

[1138 rows x 8 columns]
```

[26]:

```python
# Subsetting only call options

AMZN_calls=AMZN_opt1.loc[AMZN_opt1["Type"]=="call"].reset_index(drop=True)
AMZN_calls
```

[26]:

```
             Option Name Expiration Date  Type  Strike     Bid     Ask  \
0    AMZN210326C01700000      2021-03-26  call  1700.0  1432.5  1442.5
1    AMZN210326C01710000      2021-03-26  call  1710.0     0.0     0.0
2    AMZN210326C01730000      2021-03-26  call  1730.0     0.0     0.0
3    AMZN210326C01740000      2021-03-26  call  1740.0     0.0     0.0
4    AMZN210326C01760000      2021-03-26  call  1760.0     0.0     0.0
..                     …             …   …       …       …
558  AMZN210521C04600000      2021-05-21  call  4600.0     0.0     0.0
559  AMZN210521C04700000      2021-05-21  call  4700.0     0.0     0.0
560  AMZN210521C04800000      2021-05-21  call  4800.0     0.0     0.0
561  AMZN210521C04900000      2021-05-21  call  4900.0     0.0     0.0
562  AMZN210521C05000000      2021-05-21  call  5000.0     0.0     0.0

     Last Price  Implied Volatility
0       1363.05            3.289553
1       1335.65            0.000010
2       1315.70            0.000010
3       1343.95            0.000010
4       1285.75            0.000010
..          …                 …
558        2.01            0.125009
559        1.78            0.125009
560        1.60            0.250007
561        1.33            0.250007
562        1.25            0.250007

[563 rows x 8 columns]
```

[27]:
```
# Subsetting only put options

AMZN_puts=AMZN_opt1.loc[AMZN_opt1["Type"]=="put"].reset_index(drop=True)
AMZN_puts
```

[27]:
```
              Option Name Expiration Date Type  Strike  Bid   Ask  Last Price  \
0    AMZN210326P01660000      2021-03-26  put  1660.0  0.0  0.00        0.01
1    AMZN210326P01680000      2021-03-26  put  1680.0  0.0  0.00        0.03
2    AMZN210326P01690000      2021-03-26  put  1690.0  0.0  0.00        0.05
3    AMZN210326P01700000      2021-03-26  put  1700.0  0.0  0.00        0.03
4    AMZN210326P01710000      2021-03-26  put  1710.0  0.0  0.28        0.22
..                   …             …   …     …   …    …         …
570  AMZN210521P04400000      2021-05-21  put  4400.0  0.0  0.00     1338.05
571  AMZN210521P04500000      2021-05-21  put  4500.0  0.0  0.00     1447.10
572  AMZN210521P04600000      2021-05-21  put  4600.0  0.0  0.00     1437.28
573  AMZN210521P04700000      2021-05-21  put  4700.0  0.0  0.00     1575.75
574  AMZN210521P05000000      2021-05-21  put  5000.0  0.0  0.00     1919.00

     Implied Volatility
0              0.500005
1              0.500005
2              0.500005
3              0.500005
4              2.187505
..                  …
570            0.000010
571            0.000010
572            0.000010
573            0.000010
574            0.000010

[575 rows x 8 columns]
```

Subsettin AMZN at the money calls for 3 different expiration date

[28]:
```
# AMZN at the money calls for 3 different expiration date
AMZN_ATM_calls=AMZN_calls[(AMZN_calls["Strike"]>1700) &␣
 ↪(AMZN_calls["Strike"]<2000)].reset_index(drop=True)
#AMZN_ATM_calls

AMZN_ATM_calls=AMZN_ATM_calls.sort_values("Strike",ascending=True).
 ↪reset_index(drop=True)
AMZN_ATM_calls
```

[28]:
```
            Option Name Expiration Date  Type  Strike     Bid     Ask  \
0   AMZN210326C01710000      2021-03-26  call  1710.0    0.00    0.00
1   AMZN210416C01710000      2021-04-16  call  1710.0    0.00    0.00
```

2	AMZN210416C01720000	2021-04-16	call	1720.0	0.00	0.00
3	AMZN210326C01730000	2021-03-26	call	1730.0	0.00	0.00
4	AMZN210416C01730000	2021-04-16	call	1730.0	0.00	0.00
5	AMZN210326C01740000	2021-03-26	call	1740.0	0.00	0.00
6	AMZN210416C01750000	2021-04-16	call	1750.0	0.00	0.00
7	AMZN210326C01760000	2021-03-26	call	1760.0	0.00	0.00
8	AMZN210416C01760000	2021-04-16	call	1760.0	0.00	0.00
9	AMZN210326C01770000	2021-03-26	call	1770.0	0.00	0.00
10	AMZN210416C01770000	2021-04-16	call	1770.0	0.00	0.00
11	AMZN210416C01780000	2021-04-16	call	1780.0	0.00	0.00
12	AMZN210326C01800000	2021-03-26	call	1800.0	0.00	0.00
13	AMZN210521C01800000	2021-05-21	call	1800.0	0.00	0.00
14	AMZN210416C01800000	2021-04-16	call	1800.0	1403.30	1416.65
15	AMZN210416C01810000	2021-04-16	call	1810.0	0.00	0.00
16	AMZN210326C01810000	2021-03-26	call	1810.0	0.00	0.00
17	AMZN210416C01820000	2021-04-16	call	1820.0	1323.25	1339.45
18	AMZN210416C01830000	2021-04-16	call	1830.0	0.00	0.00
19	AMZN210416C01850000	2021-04-16	call	1850.0	0.00	0.00
20	AMZN210326C01860000	2021-03-26	call	1860.0	0.00	0.00
21	AMZN210416C01860000	2021-04-16	call	1860.0	0.00	0.00
22	AMZN210521C01860000	2021-05-21	call	1860.0	1272.95	1287.15
23	AMZN210416C01880000	2021-04-16	call	1880.0	1118.60	1129.95
24	AMZN210326C01890000	2021-03-26	call	1890.0	0.00	0.00
25	AMZN210326C01900000	2021-03-26	call	1900.0	0.00	0.00
26	AMZN210521C01900000	2021-05-21	call	1900.0	0.00	0.00
27	AMZN210416C01900000	2021-04-16	call	1900.0	0.00	0.00
28	AMZN210416C01910000	2021-04-16	call	1910.0	0.00	0.00
29	AMZN210416C01920000	2021-04-16	call	1920.0	1214.00	1223.20
30	AMZN210326C01930000	2021-03-26	call	1930.0	0.00	0.00
31	AMZN210416C01930000	2021-04-16	call	1930.0	0.00	0.00
32	AMZN210521C01940000	2021-05-21	call	1940.0	1131.00	1146.25
33	AMZN210416C01940000	2021-04-16	call	1940.0	0.00	0.00
34	AMZN210416C01950000	2021-04-16	call	1950.0	0.00	0.00
35	AMZN210416C01960000	2021-04-16	call	1960.0	0.00	0.00
36	AMZN210521C01960000	2021-05-21	call	1960.0	0.00	0.00
37	AMZN210416C01970000	2021-04-16	call	1970.0	0.00	0.00
38	AMZN210326C01970000	2021-03-26	call	1970.0	0.00	0.00
39	AMZN210416C01980000	2021-04-16	call	1980.0	1227.45	1239.20
40	AMZN210521C01980000	2021-05-21	call	1980.0	0.00	0.00

	Last Price	Implied Volatility
0	1335.65	0.000010
1	1366.30	0.000010
2	1386.00	0.000010
3	1315.70	0.000010
4	1384.70	0.000010
5	1343.95	0.000010

```
6       1354.80             0.000010
7       1285.75             0.000010
8       1500.75             0.000010
9       1314.00             0.000010
10      1345.40             0.000010
11      1238.00             0.000010
12      1251.30             0.000010
13      1241.90             0.000010
14      1521.30             1.904816
15      1283.29             0.000010
16      1259.05             0.000010
17      1365.00             1.248814
18      1211.35             0.000010
19      1105.35             0.000010
20      1121.35             0.000010
21      1195.75             0.000010
22      1204.50             0.583012
23      1414.80             0.000010
24      1155.80             0.000010
25      1228.85             0.000010
26      1242.65             0.000010
27      1261.27             0.000010
28      1162.97             0.000010
29      1202.24             0.776980
30      1110.15             0.000010
31      1077.90             0.000010
32      1361.05             0.000010
33      1221.40             0.000010
34      1171.50             0.000010
35      1083.15             0.000010
36      1123.20             0.000010
37      1077.23             0.000010
38      1087.52             0.000010
39      1318.56             1.670091
40      1313.22             0.000010
```

Subsettin AMZN at the money put for 3 different expiration date

[29]:
```
# AMZN at the money puts for 3 different expiration date
AMZN_ATM_puts=AMZN_puts[(AMZN_puts["Strike"]>3950) &␣
↪(AMZN_puts["Strike"]<5000)].reset_index(drop=True)
#AMZN_ATM_put

AMZN_ATM_puts=AMZN_ATM_puts.sort_values("Strike",ascending=True).
↪reset_index(drop=True)
AMZN_ATM_puts
```

[29]:

	Option Name	Expiration Date	Type	Strike	Bid	Ask	\
0	AMZN210326P03960000	2021-03-26	put	3960.0	0.00	0.00	
1	AMZN210326P03980000	2021-03-26	put	3980.0	0.00	0.00	
2	AMZN210326P03995000	2021-03-26	put	3995.0	0.00	0.00	
3	AMZN210416P04000000	2021-04-16	put	4000.0	0.00	0.00	
4	AMZN210521P04000000	2021-05-21	put	4000.0	0.00	0.00	
5	AMZN210326P04000000	2021-03-26	put	4000.0	0.00	0.00	
6	AMZN210416P04050000	2021-04-16	put	4050.0	0.00	0.00	
7	AMZN210326P04050000	2021-03-26	put	4050.0	0.00	0.00	
8	AMZN210326P04100000	2021-03-26	put	4100.0	0.00	0.00	
9	AMZN210521P04100000	2021-05-21	put	4100.0	1023.05	1038.00	
10	AMZN210416P04100000	2021-04-16	put	4100.0	0.00	0.00	
11	AMZN210416P04150000	2021-04-16	put	4150.0	0.00	0.00	
12	AMZN210326P04200000	2021-03-26	put	4200.0	0.00	0.00	
13	AMZN210416P04200000	2021-04-16	put	4200.0	0.00	0.00	
14	AMZN210521P04200000	2021-05-21	put	4200.0	0.00	0.00	
15	AMZN210416P04250000	2021-04-16	put	4250.0	0.00	0.00	
16	AMZN210521P04300000	2021-05-21	put	4300.0	0.00	0.00	
17	AMZN210416P04300000	2021-04-16	put	4300.0	0.00	0.00	
18	AMZN210416P04350000	2021-04-16	put	4350.0	0.00	0.00	
19	AMZN210521P04400000	2021-05-21	put	4400.0	0.00	0.00	
20	AMZN210416P04400000	2021-04-16	put	4400.0	0.00	0.00	
21	AMZN210416P04450000	2021-04-16	put	4450.0	0.00	0.00	
22	AMZN210416P04500000	2021-04-16	put	4500.0	0.00	0.00	
23	AMZN210521P04500000	2021-05-21	put	4500.0	0.00	0.00	
24	AMZN210326P04500000	2021-03-26	put	4500.0	0.00	0.00	
25	AMZN210416P04550000	2021-04-16	put	4550.0	0.00	0.00	
26	AMZN210416P04600000	2021-04-16	put	4600.0	0.00	0.00	
27	AMZN210326P04600000	2021-03-26	put	4600.0	0.00	0.00	
28	AMZN210521P04600000	2021-05-21	put	4600.0	0.00	0.00	
29	AMZN210416P04650000	2021-04-16	put	4650.0	0.00	0.00	
30	AMZN210326P04700000	2021-03-26	put	4700.0	0.00	0.00	
31	AMZN210521P04700000	2021-05-21	put	4700.0	0.00	0.00	
32	AMZN210416P04700000	2021-04-16	put	4700.0	0.00	0.00	
33	AMZN210416P04750000	2021-04-16	put	4750.0	1671.30	1685.00	
34	AMZN210416P04800000	2021-04-16	put	4800.0	0.00	0.00	
35	AMZN210416P04850000	2021-04-16	put	4850.0	0.00	0.00	
36	AMZN210326P04900000	2021-03-26	put	4900.0	0.00	0.00	
37	AMZN210416P04900000	2021-04-16	put	4900.0	1821.30	1834.95	
38	AMZN210416P04950000	2021-04-16	put	4950.0	1944.50	1956.00	

	Last Price	Implied Volatility
0	877.28	0.000010
1	901.65	0.000010
2	970.94	0.000010
3	964.27	0.000010
4	945.64	0.000010

```
5      902.80            0.000010
6     1026.00            0.000010
7      988.60            0.000010
8     1047.55            0.000010
9      873.70            0.631603
10    1192.40            0.000010
11    1153.75            0.000010
12    1154.85            0.000010
13    1208.50            0.000010
14    1073.45            0.000010
15    1216.80            0.000010
16    1243.85            0.000010
17    1254.40            0.000010
18    1291.85            0.000010
19    1338.05            0.000010
20    1291.88            0.000010
21    1451.20            0.000010
22    1339.40            0.000010
23    1447.10            0.000010
24    1402.40            0.000010
25    1552.30            0.000010
26    1527.50            0.000010
27    1507.50            0.000010
28    1437.28            0.000010
29    1673.20            0.000010
30    1735.40            0.000010
31    1575.75            0.000010
32    1453.85            0.000010
33    1437.10            1.302181
34    1691.90            0.000010
35    1745.95            0.000010
36    1837.60            0.000010
37    1652.65            1.366611
38    1631.25            1.737909
```

2.1.1 Blackscholes

[30]:
```
## Blackscholes function to calulate option price

# S= Stock Price

# K= Strike Price

# t= Expiration Date

# sig= Volatility

```

```
# optype= Type

# r= risk free interest rate



def blackscholes(S,K,t,optype,sig,r=0.0030):

    d1= (np.log(S/K)+(r+sig**2/2)*t)/(sig*np.sqrt(t))

    d2= d1-sig*np.sqrt(t)

    call_price=norm.cdf(d1,0,1)*S-  norm.cdf(d2,0,1)*K*np.exp(-r*t)

    put_price = K* np.exp(-r*t)* norm.cdf(-d2,0,1) -  S* norm.cdf(-d1,0,1)

    if optype== "call":

        return call_price

    elif optype=="put":

        return put_price
```

2.1.2 Bisection

[31]:
```
# bisection function compatible with apply function

def bisection(row):

    S=3049
    K=row["Strike"]
    optype=row["Type"]

    today = datetime.today()
    exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
    t=(exp-today).days
    avr_price=(row["Bid"]+row["Ask"])/2



    a= 0.01
    b=1



    f_b=blackscholes(S,K,t,optype,b)-avr_price

```

```
f_a=blackscholes(S,K,t,optype,a)-avr_price

count=0



while b-a>0.01:


        count+=1

        if count>1000:

            break





        c=a+b/2

        f_c=blackscholes(S,K,t,optype,c)-avr_price

        f_b=f_b
        f_a=f_a


        #f_b=blackscholes(S,K,t,optype,b)-avr_price

        #f_a=blackscholes(S,K,t,optype,a)-avr_price



        if f_c<0.01:


            break


        if f_c*f_b<0:

            a=c


        elif f_c*f_a<0:

            b=c
```

```
    return c
```

[32]:
```
# example using bisection with apply function on ATM calls

AMZN_vol=AMZN_ATM_calls.apply(lambda row: bisection(row),axis=1)
AMZN_ATM_calls["bisection_implied"]=AMZN_vol
AMZN_ATM_calls
```

[32]:

	Option Name	Expiration Date	Type	Strike	Bid	Ask	\
0	AMZN210326C01710000	2021-03-26	call	1710.0	0.00	0.00	
1	AMZN210416C01710000	2021-04-16	call	1710.0	0.00	0.00	
2	AMZN210416C01720000	2021-04-16	call	1720.0	0.00	0.00	
3	AMZN210326C01730000	2021-03-26	call	1730.0	0.00	0.00	
4	AMZN210416C01730000	2021-04-16	call	1730.0	0.00	0.00	
5	AMZN210326C01740000	2021-03-26	call	1740.0	0.00	0.00	
6	AMZN210416C01750000	2021-04-16	call	1750.0	0.00	0.00	
7	AMZN210326C01760000	2021-03-26	call	1760.0	0.00	0.00	
8	AMZN210416C01760000	2021-04-16	call	1760.0	0.00	0.00	
9	AMZN210326C01770000	2021-03-26	call	1770.0	0.00	0.00	
10	AMZN210416C01770000	2021-04-16	call	1770.0	0.00	0.00	
11	AMZN210416C01780000	2021-04-16	call	1780.0	0.00	0.00	
12	AMZN210326C01800000	2021-03-26	call	1800.0	0.00	0.00	
13	AMZN210521C01800000	2021-05-21	call	1800.0	0.00	0.00	
14	AMZN210416C01800000	2021-04-16	call	1800.0	1403.30	1416.65	
15	AMZN210416C01810000	2021-04-16	call	1810.0	0.00	0.00	
16	AMZN210326C01810000	2021-03-26	call	1810.0	0.00	0.00	
17	AMZN210416C01820000	2021-04-16	call	1820.0	1323.25	1339.45	
18	AMZN210416C01830000	2021-04-16	call	1830.0	0.00	0.00	
19	AMZN210416C01850000	2021-04-16	call	1850.0	0.00	0.00	
20	AMZN210326C01860000	2021-03-26	call	1860.0	0.00	0.00	
21	AMZN210416C01860000	2021-04-16	call	1860.0	0.00	0.00	
22	AMZN210521C01860000	2021-05-21	call	1860.0	1272.95	1287.15	
23	AMZN210416C01880000	2021-04-16	call	1880.0	1118.60	1129.95	
24	AMZN210326C01890000	2021-03-26	call	1890.0	0.00	0.00	
25	AMZN210326C01900000	2021-03-26	call	1900.0	0.00	0.00	
26	AMZN210521C01900000	2021-05-21	call	1900.0	0.00	0.00	
27	AMZN210416C01900000	2021-04-16	call	1900.0	0.00	0.00	
28	AMZN210416C01910000	2021-04-16	call	1910.0	0.00	0.00	
29	AMZN210416C01920000	2021-04-16	call	1920.0	1214.00	1223.20	
30	AMZN210326C01930000	2021-03-26	call	1930.0	0.00	0.00	
31	AMZN210416C01930000	2021-04-16	call	1930.0	0.00	0.00	
32	AMZN210521C01940000	2021-05-21	call	1940.0	1131.00	1146.25	
33	AMZN210416C01940000	2021-04-16	call	1940.0	0.00	0.00	
34	AMZN210416C01950000	2021-04-16	call	1950.0	0.00	0.00	
35	AMZN210416C01960000	2021-04-16	call	1960.0	0.00	0.00	
36	AMZN210521C01960000	2021-05-21	call	1960.0	0.00	0.00	

37	AMZN210416C01970000	2021-04-16	call	1970.0	0.00	0.00
38	AMZN210326C01970000	2021-03-26	call	1970.0	0.00	0.00
39	AMZN210416C01980000	2021-04-16	call	1980.0	1227.45	1239.20
40	AMZN210521C01980000	2021-05-21	call	1980.0	0.00	0.00

	Last Price	Implied Volatility	bisection_implied
0	1335.65	0.000010	0.510000
1	1366.30	0.000010	0.510000
2	1386.00	0.000010	0.510000
3	1315.70	0.000010	0.510000
4	1384.70	0.000010	0.510000
5	1343.95	0.000010	0.510000
6	1354.80	0.000010	0.510000
7	1285.75	0.000010	0.510000
8	1500.75	0.000010	0.510000
9	1314.00	0.000010	0.510000
10	1345.40	0.000010	0.510000
11	1238.00	0.000010	0.510000
12	1251.30	0.000010	0.510000
13	1241.90	0.000010	0.510000
14	1521.30	1.904816	0.081250
15	1283.29	0.000010	0.510000
16	1259.05	0.000010	0.510000
17	1365.00	1.248814	0.510000
18	1211.35	0.000010	0.510000
19	1105.35	0.000010	0.510000
20	1121.35	0.000010	0.510000
21	1195.75	0.000010	0.510000
22	1204.50	0.583012	0.510000
23	1414.80	0.000010	0.510000
24	1155.80	0.000010	0.510000
25	1228.85	0.000010	0.510000
26	1242.65	0.000010	0.510000
27	1261.27	0.000010	0.510000
28	1162.97	0.000010	0.510000
29	1202.24	0.776980	0.510000
30	1110.15	0.000010	0.510000
31	1077.90	0.000010	0.510000
32	1361.05	0.000010	0.510000
33	1221.40	0.000010	0.510000
34	1171.50	0.000010	0.510000
35	1083.15	0.000010	0.510000
36	1123.20	0.000010	0.510000
37	1077.23	0.000010	0.510000
38	1087.52	0.000010	0.510000
39	1318.56	1.670091	0.050625
40	1313.22	0.000010	0.510000

```
[ ]:
```

2.2 Applying Explicit Finite Difference on AMZN

```
[33]: def getpricesEXPut(row):

          today = datetime.today()
          exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
          t=(exp-today).days


          option = FDExplicitEu(S0=3000, K=row['Strike'], r=0.06, T=t,␣
       ↪sigma=row["Implied Volatility"], M=3, N=3, is_call=False)
          return option.price()
```

```
[34]: def getpricesEXCall(row):

          today = datetime.today()
          exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
          t=(exp-today).days


          option = FDExplicitEu(S0=3000, K=row['Strike'], r=0.06, T=t,␣
       ↪sigma=row["Implied Volatility"], M=3, N=3, is_call=True)
          return option.price()
```

2.2.1 Applying Explicit Finite Difference on AMZN Calls

```
[35]: EXCall=AMZN_ATM_calls.apply(getpricesEXCall, axis=1)
      EXCall=pd.DataFrame(EXCall,columns=["Explicit Finite-Price"])
      EXCall
```

[35]:

	Explicit Finite-Price
0	1.315389e+03
1	1.373144e+03
2	1.369803e+03
3	1.302724e+03
4	1.366462e+03
5	1.296391e+03
6	1.359781e+03
7	1.283726e+03
8	1.356440e+03
9	1.277393e+03
10	1.353099e+03
11	1.349758e+03
12	1.258395e+03

```
13          2.047491e+02
14          1.048320e+07
15          1.339735e+03
16          1.252063e+03
17          1.888368e+06
18          1.332351e+03
19          1.323885e+03
20          1.219929e+03
21          1.319652e+03
22          1.628382e+06
23          1.311187e+03
24          1.200559e+03
25          1.194102e+03
26          1.995635e+02
27          1.302721e+03
28          1.298488e+03
29          4.184483e+05
30          1.174732e+03
31          1.290023e+03
32          1.973614e+02
33          1.285790e+03
34          1.281557e+03
35          1.277324e+03
36          1.962604e+02
37          1.273091e+03
38          1.148905e+03
39          3.325392e+07
40          1.951594e+02
```

2.2.2 Applying Explicit Finite Difference on AMZN Puts

[36]:
```
EXPut=AMZN_ATM_puts.apply(getpricesEXPut, axis=1)
EXPut=pd.DataFrame(EXPut,columns=["Explicit Finite-Price"])
EXPut
```

[36]:
```
    Explicit Finite-Price
0           1.655705e+02
1           1.774905e+02
2           1.864306e+02
3          -1.538743e+02
4          -3.208825e+01
5           1.894106e+02
6          -1.526579e+02
7           2.192107e+02
8           2.490109e+02
9           1.082575e+07
10         -1.514416e+02
```

```
11      -1.502253e+02
12       3.086111e+02
13      -1.490089e+02
14      -3.533180e+01
15      -1.477926e+02
16      -3.695358e+01
17      -1.465763e+02
18      -1.453599e+02
19      -3.857535e+01
20      -1.441436e+02
21      -1.429273e+02
22      -1.417109e+02
23      -4.019713e+01
24       4.874119e+02
25      -1.404946e+02
26      -1.392783e+02
27       5.470122e+02
28      -4.181891e+01
29      -1.380619e+02
30       6.066124e+02
31      -4.344068e+01
32      -1.368456e+02
33       2.680885e+07
34      -1.344129e+02
35      -1.331966e+02
36       7.258130e+02
37       2.824844e+07
38       1.113678e+08
```

2.3 Applying Implicit Finite Difference on AMZN

[37]:
```
def getpricesIMCall(row):

    today = datetime.today()
    exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
    t=(exp-today).days


    option = FDImplicitEu(S0=3000, K=row['Strike'], r=0.06, T=t,␣
↪sigma=row["Implied Volatility"], M=3, N=3, is_call=True)
    return option.price()
```

[38]:
```
def getpricesIMPut(row):

    today = datetime.today()
    exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
```

```
    t=(exp-today).days


    option = FDImplicitEu(S0=3000, K=row['Strike'], r=0.06, T=t,␣
↪sigma=row["Implied Volatility"], M=3, N=3, is_call=False)
    return option.price()
```

2.3.1 Applying Implicit Finite Difference on AMZN Call

[39]:
```
IMCall=AMZN_ATM_calls.apply(getpricesIMCall, axis=1)
IMCall=pd.DataFrame(IMCall,columns=["Implicit-Finite-Price"])
IMCall
```

[39]:

	Implicit-Finite-Price
0	1229.121317
1	395.008569
2	393.734795
3	1210.816928
4	392.461022
5	1201.664733
6	389.913475
7	1183.360343
8	388.639701
9	1174.208148
10	387.365928
11	386.092154
12	1146.751564
13	79.757459
14	0.087503
15	382.270834
16	1137.599369
17	0.830767
18	379.080416
19	374.901466
20	1103.088307
21	372.811991
22	2.293166
23	368.633041
24	1084.541086
25	1078.358679
26	76.552670
27	364.454091
28	362.364616
29	7.121800
30	1059.811458
31	358.185666
32	74.797114

```
33           356.096191
34           354.006716
35           351.917241
36            73.919336
37           349.827766
38          1035.081830
39             0.162550
40            73.041558
```

2.3.2 Applying Implicit Finite Difference on AMZN Puts

[40]:
```
IMPut=AMZN_ATM_puts.apply(getpricesIMPut, axis=1)
IMPut=pd.DataFrame(IMPut,columns=["Implicit-Finite-Price"])
IMPut
```

[40]:

	Implicit-Finite-Price
0	843.810805
1	861.933454
2	875.525441
3	-114.755226
4	-120.699966
5	880.056103
6	-107.996249
7	925.362727
8	970.669350
9	-5.299536
10	-101.237272
11	-94.478295
12	1061.282597
13	-87.719318
14	-119.938736
15	-80.960341
16	-119.558121
17	-74.201364
18	-67.442387
19	-119.177506
20	-60.683410
21	-53.924434
22	-47.165457
23	-118.796891
24	1333.122337
25	-40.406480
26	-33.647503
27	1423.735584
28	-118.416276
29	-26.888526
30	1514.348830

```
31          -118.035661
32           -20.129549
33           -26.080519
34            -6.611595
35             0.147382
36          1695.575324
37           -20.545174
38           -19.037119
```

2.4 Applying Crank-Nicolson Finite Difference on AMZN

[41]:
```python
def getpricesCNCall(row):

    today = datetime.today()
    exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
    t=(exp-today).days


    option = FDCnEu(S0=3000, K=row['Strike'], r=0.06, T=t, sigma=row["Implied␣
↪Volatility"], M=3, N=3, is_call=True)
    return option.price()
```

[42]:
```python
def getpricesCNPut(row):

    today = datetime.today()
    exp=datetime.strptime(row["Expiration Date"],"%Y-%m-%d")
    t=(exp-today).days


    option = FDCnEu(S0=3000, K=row['Strike'], r=0.06, T=t, sigma=row["Implied␣
↪Volatility"], M=3, N=3, is_call=False)
    return option.price()
```

2.4.1 Applying Crank-Nicolson Finite Difference on AMZN Call

[43]:
```python
CNCall=AMZN_ATM_calls.apply(getpricesCNCall, axis=1)
CNCall=pd.DataFrame(CNCall,columns=["Crack-Nicolsan-Price"])
CNCall
```

[43]:

	Crack-Nicolsan-Price
0	1229.006953
1	282.008275
2	281.964577
3	1210.719243
4	281.920879
5	1201.575388

6	281.833483
7	1183.287679
8	281.789785
9	1174.143824
10	281.746087
11	281.702389
12	1146.712259
13	-54.491789
14	-4232.540023
15	281.571295
16	1137.568405
17	-2621.784399
18	280.300688
19	277.210678
20	1103.067884
21	275.665672
22	-1401.704150
23	272.575661
24	1084.521006
25	1078.338713
26	-51.066634
27	269.485651
28	267.940645
29	-895.268484
30	1059.791836
31	264.850634
32	-49.895541
33	263.305629
34	261.760624
35	260.215618
36	-49.309994
37	258.670613
38	1035.062666
39	-3428.575279
40	-48.724447

2.4.2 Applying Crank-Nicolson Finite Difference on AMZN Put

[44]:
```
CNPut=AMZN_ATM_puts.apply(getpricesCNPut, axis=1)
CNPut=pd.DataFrame(CNPut,columns=["Crack-Nicolsan-Price"])
CNPut
```

[44]:

	Crack-Nicolsan-Price
0	828.360379
1	846.648089
2	860.363871
3	-272.001425

```
4             -59.393248
5             864.935798
6            -271.782935
7             910.655073
8             956.374347
9             604.409366
10           -271.564445
11           -271.345955
12           1047.812895
13           -271.127464
14            -69.746433
15           -270.908974
16            -74.923026
17           -270.690484
18           -270.471994
19            -80.099618
20           -270.253504
21           -270.035013
22           -269.816523
23            -85.276211
24           1322.128539
25           -269.598033
26           -269.379543
27           1413.567087
28            -90.452803
29           -269.161053
30           1505.005635
31            -95.629396
32           -268.942563
33            544.990755
34           -268.505582
35           -268.287092
36           1687.882731
37            432.897849
38            374.171732
```

2.5 Comparision of Call Prices

[45]:
```
Call_Pricing = pd.concat([EXCall,IMCall,CNCall],axis=1)
Call_Pricing
```

[45]:

	Explicit Finite-Price	Implicit-Finite-Price	Crack-Nicolsan-Price
0	1.315389e+03	1229.121317	1229.006953
1	1.373144e+03	395.008569	282.008275
2	1.369803e+03	393.734795	281.964577
3	1.302724e+03	1210.816928	1210.719243
4	1.366462e+03	392.461022	281.920879

5	1.296391e+03	1201.664733	1201.575388
6	1.359781e+03	389.913475	281.833483
7	1.283726e+03	1183.360343	1183.287679
8	1.356440e+03	388.639701	281.789785
9	1.277393e+03	1174.208148	1174.143824
10	1.353099e+03	387.365928	281.746087
11	1.349758e+03	386.092154	281.702389
12	1.258395e+03	1146.751564	1146.712259
13	2.047491e+02	79.757459	-54.491789
14	1.048320e+07	0.087503	-4232.540023
15	1.339735e+03	382.270834	281.571295
16	1.252063e+03	1137.599369	1137.568405
17	1.888368e+06	0.830767	-2621.784399
18	1.332351e+03	379.080416	280.300688
19	1.323885e+03	374.901466	277.210678
20	1.219929e+03	1103.088307	1103.067884
21	1.319652e+03	372.811991	275.665672
22	1.628382e+06	2.293166	-1401.704150
23	1.311187e+03	368.633041	272.575661
24	1.200559e+03	1084.541086	1084.521006
25	1.194102e+03	1078.358679	1078.338713
26	1.995635e+02	76.552670	-51.066634
27	1.302721e+03	364.454091	269.485651
28	1.298488e+03	362.364616	267.940645
29	4.184483e+05	7.121800	-895.268484
30	1.174732e+03	1059.811458	1059.791836
31	1.290023e+03	358.185666	264.850634
32	1.973614e+02	74.797114	-49.895541
33	1.285790e+03	356.096191	263.305629
34	1.281557e+03	354.006716	261.760624
35	1.277324e+03	351.917241	260.215618
36	1.962604e+02	73.919336	-49.309994
37	1.273091e+03	349.827766	258.670613
38	1.148905e+03	1035.081830	1035.062666
39	3.325392e+07	0.162550	-3428.575279
40	1.951594e+02	73.041558	-48.724447

2.6 Comparision of Put Prices

[46]:
```
Put_Pricing = pd.concat([EXPut,IMPut,CNPut],axis=1)
Put_Pricing
```

[46]:

	Explicit Finite-Price	Implicit-Finite-Price	Crack-Nicolsan-Price
0	1.655705e+02	843.810805	828.360379
1	1.774905e+02	861.933454	846.648089
2	1.864306e+02	875.525441	860.363871
3	-1.538743e+02	-114.755226	-272.001425

4	-3.208825e+01	-120.699966	-59.393248
5	1.894106e+02	880.056103	864.935798
6	-1.526579e+02	-107.996249	-271.782935
7	2.192107e+02	925.362727	910.655073
8	2.490109e+02	970.669350	956.374347
9	1.082575e+07	-5.299536	604.409366
10	-1.514416e+02	-101.237272	-271.564445
11	-1.502253e+02	-94.478295	-271.345955
12	3.086111e+02	1061.282597	1047.812895
13	-1.490089e+02	-87.719318	-271.127464
14	-3.533180e+01	-119.938736	-69.746433
15	-1.477926e+02	-80.960341	-270.908974
16	-3.695358e+01	-119.558121	-74.923026
17	-1.465763e+02	-74.201364	-270.690484
18	-1.453599e+02	-67.442387	-270.471994
19	-3.857535e+01	-119.177506	-80.099618
20	-1.441436e+02	-60.683410	-270.253504
21	-1.429273e+02	-53.924434	-270.035013
22	-1.417109e+02	-47.165457	-269.816523
23	-4.019713e+01	-118.796891	-85.276211
24	4.874119e+02	1333.122337	1322.128539
25	-1.404946e+02	-40.406480	-269.598033
26	-1.392783e+02	-33.647503	-269.379543
27	5.470122e+02	1423.735584	1413.567087
28	-4.181891e+01	-118.416276	-90.452803
29	-1.380619e+02	-26.888526	-269.161053
30	6.066124e+02	1514.348830	1505.005635
31	-4.344068e+01	-118.035661	-95.629396
32	-1.368456e+02	-20.129549	-268.942563
33	2.680885e+07	-26.080519	544.990755
34	-1.344129e+02	-6.611595	-268.505582
35	-1.331966e+02	0.147382	-268.287092
36	7.258130e+02	1695.575324	1687.882731
37	2.824844e+07	-20.545174	432.897849
38	1.113678e+08	-19.037119	374.171732

- We see that prices of both Calls and Puts are similar with using three different method.
- There are some difference in the prices possibly due to raw data inaccuaracy

```
[ ]:
```

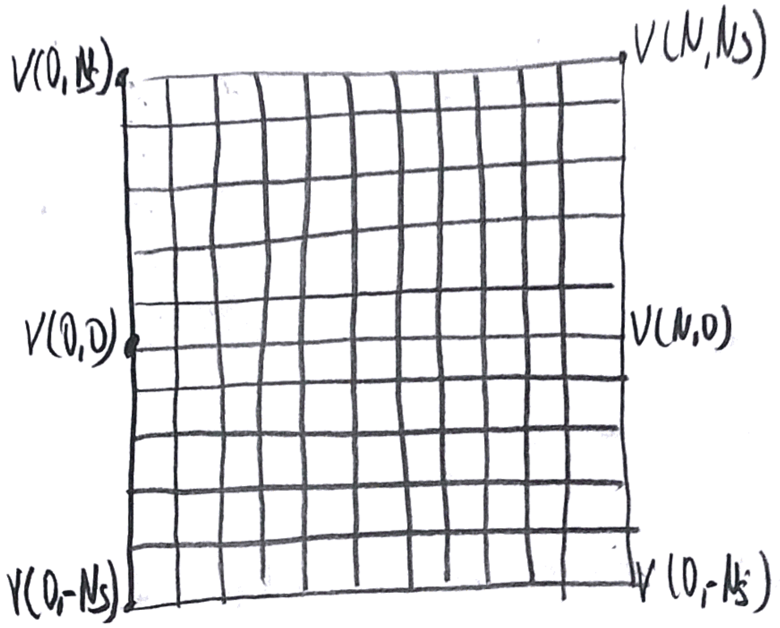
Problem 3

1)

→ First Split the time by N where $\Delta t = T/N$

i is the time step ~~Δt~~, $2\Delta t$, ... $(i-1)\Delta t$, $i\Delta t$

Dividing stock price by $2N+1$, $J\Delta x$



$$\frac{\partial V}{\partial t} + 2\cos(S)\cdot\frac{\partial V}{\partial S} + 0.2 S^{1/2}\frac{\partial^2 V}{\partial S^2} - rV = 0$$

$$\frac{\partial V}{\partial t} = \frac{V_{i+1,j} - V_{i,j}}{\partial t}, \quad \frac{\partial V}{\partial S} = \frac{V_{i+1,j+1} - V_{i+1,j-1}}{2\Delta x}$$

$$\frac{\partial^2 V}{\partial S^2} = \frac{V_{i+1,j+1} - 2V_{i+1,j} + V_{i+1,j-1}}{(\Delta x)^2}$$

$$\frac{V_{i+1,j} - V_{i,j}}{dt} + 2\cos(j\Delta x)\cdot\frac{V_{i+1,j+1} - V_{i+1,j-1}}{2dx} + 0.2(j\Delta x)^{1/2}\frac{V_{i+1,j+1} - 2V_{i+1,j} + V_{i+1,j-1}}{(\Delta x)^2} - rV_{i+1,j} = 0$$

$$V_{i,j} = V_{i+1,j} - \frac{\Delta t\, 2\cos(j\Delta x)}{\cancel{2}\Delta x}\left(V_{i+1,j+1} - V_{i+1,j-1}\right) - \frac{\Delta t}{(\Delta x j)^2}\, 0.2(j\Delta x)^{1/2}\left(V_{i+1,j+1} - 2V_{i+1,j} + V_{i+1,j-1}\right)$$

$$- r\cdot V_{i+1,j}$$

$$V_{i,j} = V_{i+1,j+1} - \frac{\Delta t\cos(j\Delta x)}{\Delta x j} - \Delta t(0.2(j\Delta x)^{-1/2} + V_{i+1,j}\left(0.4\,\Delta t\,(j\Delta x)^{+1/2} + t r\right) + V +$$

$$V_{i-1,j-1}\left(\frac{\Delta t.\cos(j\Delta x)}{4 j x}\right) - \Delta t\ 0.2\,(j\Delta x)^{-1/2}$$

$$V_{i,j} = V_{i+1,j-1}\Delta t\left(\frac{\cos(\Delta x_j)}{j\Delta x} - 0.2(j\Delta x)^{-1/2}\right) - V_{i+1,j+1}(\Delta t)\left(\frac{\cos(\Delta x_j)}{j\Delta x} + 0.2(j\Delta x)^{-1/2}\right)$$

$$+ V_{i+1,j}\left(1 - r + 0.4dt(\Delta x_j)^{-1/2}\right)$$

2)

Yes we need to define boundary conditions if the stock price is large and where $i = N - \Delta t$

$V_{i,N_s} - V_{i,N_s-1} = \Delta x$ or $V_{i,N_s} = V_{i,N_s-1} + \Delta x$

if the stock price and where $i = N - \Delta t$

$V_{i,-N_s} - V_{i,-N_s+1} = 0$ $V_{i,-N_s} = V_{i,-N_s+1}$

Backward shifting enable us to find $V_{i,-N_s}$ V_{i,N_s} by dividing the line which is surrounded by boundaries