Finite Difference Methods

Kannan Singaravelu, CQF

1 Finite Difference Methods

Finite Difference Methods (FDM) are one of the popular numerical methods used in computational finance. FDM are discretizations used for solving differential equations by approximating them with difference equations. It is one of the simplest and the oldest methods to solve differential equations. These techniques were applied to numerical applications as early as 1950s.

FDM are similar in approach to the (binomial) tress. However, instead of discretizing asset prices and the passage of time in a tree structure, it discretizes in a grid - with time and price steps - by calculating the value at every possible grid points.

Explicit, Implicit and Crank-Nicolson are the three popular approaches of FDM. The explicit methods are simple to implement, but it does not always converge and largely depends on the size of the time and asset step. Explicit methods are unstable when compared to other two methods. Finite Difference approach is peferred for low dimensional problem, usually upto 4 dimensions.

2 Differentiation Using The Grid

The Binomial method contains the diffusion - the volatility - in the tree structure whereas in FDM, the *'tree'* is fixed and we change the parameters to reflect the changing diffusion. We will now define the grid by specifying the time step δt and asset step δs and discretize S and t as

$$S = i\delta s$$

and time to maturity as

$$t = T - k\delta t$$

where $0 \leq i \leq I$ *and* $0 \leq k \leq K$

Here i and k are respective steps in the grid and we can write the value of the option at each grid points as

$$V_i^k = V(i\delta S, T - k\delta t)$$

3 Approximating Greeks

The greeks terms, the Black–Scholes equation can be written as

$$\Theta + \frac{1}{2}\sigma^2 S^2 \Gamma + rS\Delta - rV = 0$$

Assume that we know the option value at each grid points, we can extract the derivatives of the option using Taylor series expansion.

Approximating Θ

We know that the first derivative of option as,

$$\frac{\partial V}{\partial t} = \lim_{h \to 0} \frac{V(S, t+h) - V(S,t)}{h}$$

We can then approximate the time derivative from our grid using

$$\frac{\partial V}{\partial t}(S,t) \approx \frac{V_i^k - V_i^{k+1}}{\delta t}$$

Approximating Δ

From the lecture, we know that the central difference has much lower error when compared to forward and backward differences. Accordingly, we can approximate the first derivative of option with respect to the underlying as

$$\frac{\partial V}{\partial S}(S,t) \approx \frac{V_{i+1}^k - V_{i-1}^k}{2\delta S}$$

Approximating Γ

The gamma of the option is the second derivative of option with respective to the underlying and approximating it we have,

$$\frac{\partial V^2}{\partial S^2}(S,t) \approx \frac{V_{i+1}^k - 2V_i^k + V_{i-1}^k}{\delta S^2}$$

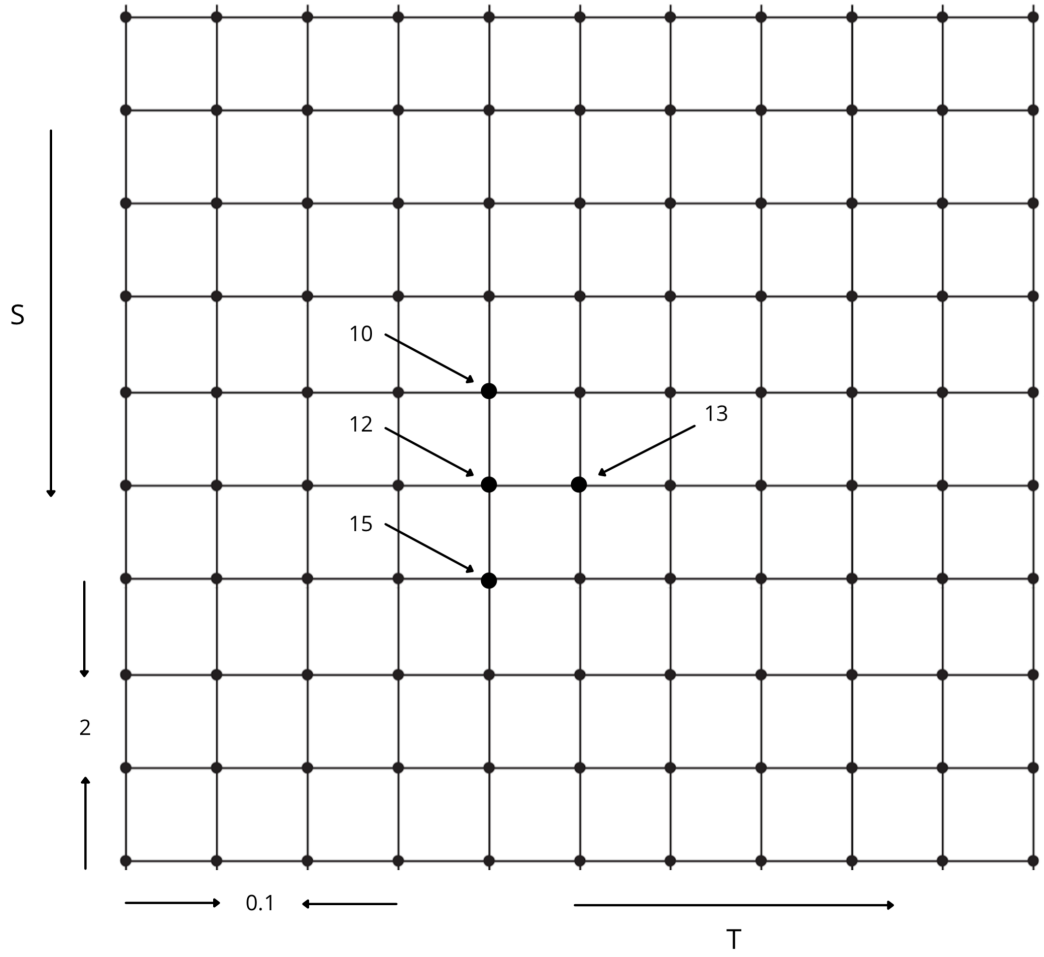
Import Required Libraries

[ ]:
```
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Importing libraries
import pandas as pd
from numpy import *

# Import cufflinks
import cufflinks as cf
```

```
cf.set_config_file(offline=True, dimensions=((1000,600)))

# Set max row and columns to 300 and 100
pd.set_option('display.max_rows', 300)
pd.set_option('display.max_columns', 100)
```

Example

Suppose that we know the value of the option on the below grid points, we can then easily evaluate the greeks as follows



From the grid, we can estimate the

$$\Theta = \frac{12 - 13}{0.1} = -10$$

$$\Delta = \frac{15 - 10}{2x2} = 1.25$$

$$\Gamma = \frac{15 - 2x12 + 10}{2x2} = 0.25$$

Black Scholes Equation is a relationship between the option value and greeks. If we know the option value at the expiration, we can step back to get the values prior to it such that $V_k^i = V_{k-1}^i - \Theta * dt$. This approach is called Explicit Finite Difference Method because the relationship between the option values at time step k is a simple function of the option values at time step $k-1$.

4 Option Pricing Techniques

As with other option pricing techniques Explicit Finite Difference methods are used to price options using what is essentially a three step process.

Step 1: Generate the grid by specifying grid points. **Step 2:** Specify the final or initial conditions. **Step 3:** Use boundary conditions to calculate option values and step back down the grid to fill it.

European Option

To price an option, we generate a finite grid of a specified asset and time steps for a given maturity. Next, we specify the initial and boundary conditions to calculate payoff when S and T equals zero. We then step back to fill the grid with newer values derived from the earlier values.

Specify Parameters

[ ]:
```
# Specify the parameters for FDM
T    = 1                                # time to maturity in years
E    = 100                              # strike price
r    = .05                              # riskfree rate
vol  = .20                              # volatility
Flag = 1                                # Flag = 1 for call, -1 for puts
EType = 1                               # EType = 1 for american, 0 for european
NAS  = 20                               # number of asset steps

ds   = 2* E / NAS                       # asset step size
dt   = (0.9/vol**2/NAS**2)              # time step size, for stability

NTS  = int(T / dt) + 1                  # number of time steps
dt   = T / NTS                          # time step size [Expiration as int # of␣
 ↪time steps away]
```

4.1 Generate Grid

Build the grid with the above input parameters

[ ]:
```
# Create asset steps i*ds
s = arange(0, (NAS+1)*ds,ds)
s
```

[ ]:
```
# Create time steps k*dt
t = T-arange(NTS*dt,-dt,-dt)
t
```

[ ]:
```
# Verify the steps size
s.shape, t.shape
```

[ ]:
```
# Initialize the grid with zeros
grid = zeros((len(s),len(t)))

# Subsume the grid points into a dataframe
# with asset price as index and time steps as columns
grid = pd.DataFrame(grid, index=s, columns=around(t,3))
grid
```

4.2 Set up Payoff

Specify payoffs at expiration for calls

$$V_i^0 = max(i\delta s - E, 0)$$

[ ]:
```
# Set Payoff at Expiration
if Flag == 1:
    grid.iloc[:,0] = maximum(s - E, 0)
else:
    grid.iloc[:,0] = maximum(E - s, 0)
```

[ ]:
```
# Verify the grid
grid
```

[ ]:
```
# Store payoff for early exercise
p = []
for j in arange(0, NAS+1):
    p.append(grid.iloc[j,0])
```

4.3 Fill the Grid

Specify boundary condition at S = 0

$$V_0^k = (1 - r\delta t)V_0^{k-1}$$

Specify boundary condition at S = ∞

$$V_i^k = 2V_{i-1}^k - V_{i-2}^k$$

[ ]:
```
# k is counter
for k in range(1, len(t)):
    for i in range(1,len(s)-1):
        delta = (grid.iloc[i+1,k-1] - grid.iloc[i-1,k-1]) / (2*ds)
        gamma = (grid.iloc[i+1,k-1]-2*grid.iloc[i,k-1]+grid.iloc[i-1,k-1]) /␣
↪(ds**2)
```

```
        theta = (-0.5* vol**2 * s[i]**2 * gamma) - (r*s[i]*delta) + (r*grid.
↪iloc[i,k-1])

        grid.iloc[i,k] = grid.iloc[i,k-1] - (theta*dt)

    # Set boundary condition at S = 0
    grid.iloc[0,k] = grid.iloc[0,k-1] * (1-r*dt) # ds = rsdt + sigma*sdx, s= 0,␣
↪ds = 0

    # Set boundary condition at S = infinity  # gamma = 0, so you can linearly␣
↪extract
    grid.iloc[len(s)-1,k] = 2*grid.iloc[len(s)-2,k] - grid.iloc[len(s)-3,k]

    if EType==1:
       for i in range(0,len(s)):
        grid.iloc[i,k] = maximum(grid.iloc[i,k], p[i])

# Round grid values to 2 decimals
grid = around(grid,3)
```

[ ]:
```
# Output the option values
# grid.iloc[0:15,:]
grid
```

[ ]:
```
# Print out stock, payoff and option value
data = {
    "Stock": s,
    "Payoff": p,
    "Option": grid.iloc[:,-1]
}

option_value_2D = pd.DataFrame(data)
option_value_2D                          # print(option_value_2D.
↪to_string(index=False))
```

[ ]:
```
# Plot option value and payoff
option_value_2D[['Payoff', 'Option']].iplot(title='Payoff & Option Value')
```

5 User Defined Function

Let's subsume above grid calculation into a function for ease of use. All we have to do is to combine the above code blocks into a single function.

[ ]:
```
def fdm_option(strike, sigma, rate, ttm, nas, is_call=True, american=False):

    # Specify flag as 1 for calls and -1 for puts
    # Specify american=True for early exercise and False for european
```

```
    ds = 2*strike/nas                    # asset step size
    dt = 0.9/sigma**2/nas**2             # for stability

    nts = int(ttm / dt) + 1              # time step size, alternatively use␣
↪fixed size 10 on stability issue
    dt = ttm/nts                         # time step

    s = arange(0,(nas+1)*ds,ds)
    t = ttm-arange(nts*dt,-dt,-dt)

    # Initialize the grid with zeros
    grid = zeros((len(s),len(t)))
    grid = pd.DataFrame(grid, index=s, columns=around(t,2))

    # Set boundary condition at Expiration
    flag = 1 if is_call else -1
    grid.iloc[:,0] = abs(maximum(flag * (s - strike), 0))

    # Store payoff for early exercise
    p = []
    for j in arange(0, nas+1):
        p.append(grid.iloc[j,0])

    for k in range(1, len(t)):
        for i in range(1,len(s)-1):
            delta = (grid.iloc[i+1,k-1] - grid.iloc[i-1,k-1]) / (2*ds)
            gamma = (grid.iloc[i+1,k-1]-2*grid.iloc[i,k-1]+grid.iloc[i-1,k-1]) /␣
↪(ds**2)
            theta = (-0.5* sigma**2 * s[i]**2 * gamma) - (rate*s[i]*delta) +␣
↪(rate*grid.iloc[i,k-1])

            grid.iloc[i,k] = grid.iloc[i,k-1] - dt*theta

        # Set boundary condition at S = 0
        grid.iloc[0,k] = grid.iloc[0,k-1] * (1-rate*dt)

        # Set boundary condition at S = infinity
        grid.iloc[len(s)-1,k] = abs(2*(grid.iloc[len(s)-2,k]) - grid.
↪iloc[len(s)-3,k])

        # Check for early exercise
        if american==True:
            for i in range(0,len(s)):
                grid.iloc[i,k] = maximum(grid.iloc[i,k], p[i])

    # round grid values to 4 decimals
```

```
    return around(grid,2)
```

[ ]:
```
# Call the function to price options
fdm_grid = fdm_option(100,0.2,0.05,1,20)
fdm_grid
```

5.1 Option Values

Let's now get the value of European & American call and put option for K=100, Volatility=20%, Rate=5%, T=1 and NAS=60.

[ ]:
```
# Output the option values
call = fdm_option(100,0.2,0.05,1,60,is_call=True, american=True).loc[100,1]
put = fdm_option(100,0.2,0.05,1,60,is_call=False, american=True).loc[100,1]

# Print the values
print(f"Call Option Value is {call:0.4f}")
print(f"Put Option Value is {put:0.4f}")
```

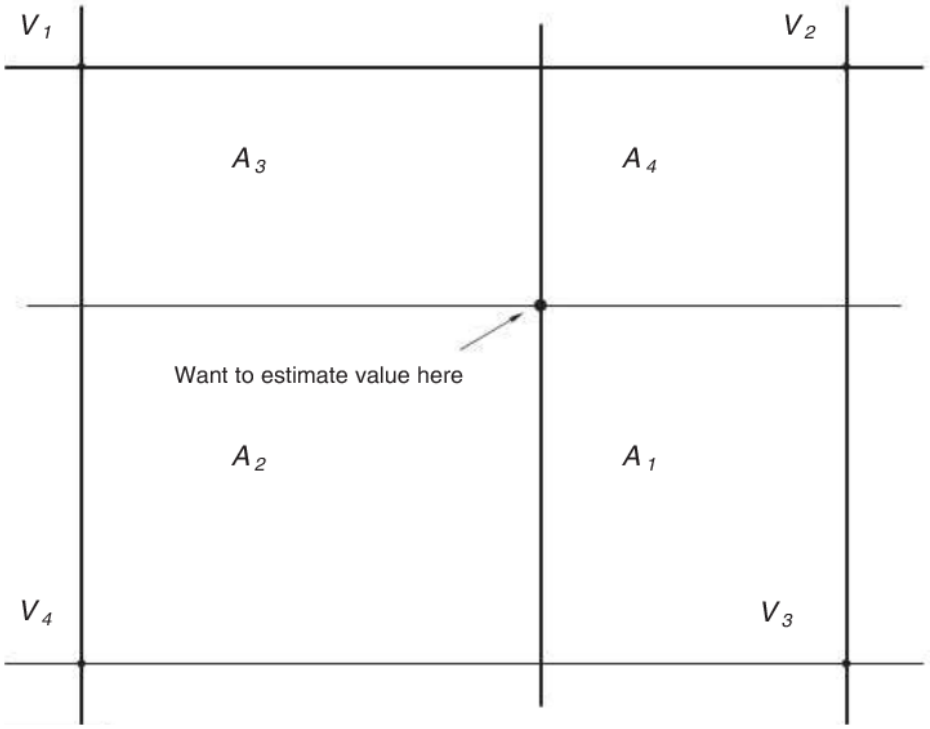
5.2 Visualize the payoff

[ ]:
```
# Plot Option Payoff
# Axis titles are not rendered correctly on the graph. This is a bug in cufflinks
fig = fdm_grid.iplot(kind = 'surface', title='Option values by Explicit FDM',␣
↪xTitle='Spot', yTitle='Maturiy', zTitle='Option Value', asFigure=True)
fig.show()
```

[ ]:
```
# Save the figure - you might have to install kaleido for earlier version of␣
↪plotly
# !pip install -U kaleido

# Save as portable network graphics
fig.write_image("images/fdm_option.png")
```

6 Bilinear Interpolation

We have generated the grid and filled it with the possible option values. However, if we have to estimate option value or its derivatives on the mesh points, how can we estimate the value at points in between? The simplest way is to do a two-dimensional interpolation method called **Bilinear Interpolation**.



The option value can be estimated using the values from the nearest neighbouring values. Assume $V_1, V_2, V_3 and V_4$ are the option values from the nearest neighbour and $A_1, A_2, A_3 and A_4$ are the areas of the rectanges made by the four corners and the interior points, we can approximate the option value at the interior points as

$$\frac{\sum_{i=1}^{4} A_i V_i}{\sum_{i=1}^{4} A_i}$$

[ ]:
```
def bilinear_interpolation(asset_price, ttm, df):

    # Find relevant rows and columns
    col1 = df.columns[df.columns < ttm][-1]
    col2 = df.columns[df.columns >= ttm][0]
    row1 = df.index[df.index < asset_price,][-1]
    row2 = df.index[df.index >= asset_price,][0]

    # Define points and areas
    V = [df.loc[row1, col1], df.loc[row1, col2],
           df.loc[row2, col2], df.loc[row2, col1]]

    A = [(row2 - asset_price) * (col2 - ttm),
           (row2 - asset_price) * (ttm - col1),
           (asset_price - row1) * (ttm - col1),
           (asset_price - row1) * (col2 - ttm)]
```

```
    # Interpolate values
    return sum(array(V)*array(A))/sum(array(A))
```

[ ]:
```
# Option value, approximated
bilinear_interpolation(105,0.25,fdm_grid)
```

[ ]:
```
# Verify rows and columns - Examples
# col1 = grid.columns[grid.columns < 0.3][-1]
# col2 = grid.columns[grid.columns >= 0.3][0]
# row1 = grid.index[grid.index < 110][-1]
# row2 = grid.index[grid.index >= 110][0]

# Nearest neighbours grid points
# [row1,col1], [row1, col2], [row2, col2], [row2, col1]

# Get option values
# V = [grid.loc[row1,col1], grid.loc[row1,col2], grid.loc[row2, col2], grid.
 ↪loc[row2, col1]]

# Areas of the rectangle made by four corners and interior points
# A = [(row2-105) * (col2-0.3),
#      (row2-105) * (0.3-col1),
#      (105-row1) * (0.3-col1),
#      (105-row1) * (col2-0.3)]

# Option value in mesh points, approximated
# sum(array(V)*array(A))/sum(array(A))
```

7 Convergence Analysis

Let's now compare option pricing for various asset steps (NAS) with black scholes price.

[ ]:
```
# Iterate over asset steps (NAS)
nas_list = [10,20,30,40,50,60]
fdmoption = []
for i in nas_list:
    fdmoption.append(fdm_option(100,0.2,0.05,1,i).loc[100,1])
fdmoption
```

[ ]:
```
# Call black scholes class
from blackscholes import BS

# Instantiate black scholes object
option = BS(100,100,0.05,1,0.20)
bsoption = round(option.callPrice,2)
bsoption = bsoption.repeat(len(nas_list))
```

```
# Range of option price
bsoption
```

[ ]:
```
# Subsume into dataframe
df = pd.DataFrame(list(zip(bsoption,fdmoption)), columns=['BS', 'FDM'],␣
 ↪index=nas_list)
df['dev'] = df['FDM'] - df['BS']
df['% dev'] = round(df['dev'] / df['BS'] * 100.,2)

# Output
print("BS - FDM Convergence over NAS")
df
```

8 References

- Python Resources
- Paul Wilmott (2007), Paul Wilmott introduces Quantitative Finance

Python Labs by Kannan Singaravelu.