Binomial Tree

Kannan Singaravelu, CQF

1 Binomial Tree

The binomial tree method was pioneered by Cox, Ross, and Robinstein in 1979. The binomial model allows the stock to move up or down a specific amount over the next time step. Given the initial stock price `S`, we allow it to either go up or down by a factor `u` and `v` resulting in the value uS and vS after the next time step. Extending this random walk after two time steps, the asset will either be at u^2S, if there are two up moves or at v^2S if there are two down moves or at uvS if an up was followed by a down move or vice versa. We can extend this to any number of time step (or branches) depending on how complex you want the model to be or until expiration.

This structure where nodes represent the values taken by the asset is called the **binomial tree**.

1.1 Import Libraries

We'll import the required libraries that we'll use in this example.

[ ]:
```
# Import math functions from NumPy
from numpy import *

# Import plotting functions from helper
from helper import plot_asset_path, plot_probability, plot_binomial_tree
```

1.2 Price Path

The probability of reaching a particular node in the binomial tree depends on the numbers of distinct paths to that node and the probabilities of the up and down moves. The following figures shows the number of paths to each node and the probability of reaching to that node.

[ ]:
```
# Plot asset price path
plot_asset_path()
```

1.3 Path Probability

```
[ ]: # Plot node probability
     plot_probability()
```

1.4 Risk Neutral Probability

Risk-neutral measure is a probability measure such that each share price today is the discounted expectations of the share price. From Paul's lecture, we know the formula for u, v, p' and V are as follows,

\$ u = 1+ $\{\sigma\sqrt{\delta t}\}$\$

\$ v = 1- $\{\sigma\sqrt{\delta t}\}$\$

The underlying instrument will move up or down by a specific factor u or v per step of the tree where $u \geq 1$ and $0 < v \leq 1$.

\$ p ' = $1\frac{}{2+\frac{r\sqrt{\delta t}}{2\sigma}\$}$

where, p' the risk-neutral probability.

\$ V = $1\frac{}{1+r\delta t(p'V@\{+\}+(1-p')V@\{-\})\$}$

where, V is the option value which is present value of some expectation : sum probabilities multiplied by events.

1.5 Building Binomial Tree

Next, we will build a binomial tree using the risk neutral probability. Building a tree is a multi step process which involves.

Step 1: Draw a n-step tree

Step 2: At the end of n-step, estimate terminal prices

Step 3: Calculate the option value at each node based on the terminal price, exercise price and type

Step 4: Discount it back one step, that is, from n to n-1, according to the risk neutral probability

Step 5: Repeat the previous step until we find the final value at step 0

1.5.1 Binomial Pricing Model

Let's now define a binomial option pricing function

```
[ ]: # Create a user defined function
     def binomial_option(spot: float, strike: float, rate: float, sigma: float, time:␣
      ↪float, steps: int, output: int=0) -> ndarray:

         """
         binomial_option(spot, strike, rate, sigma, time, steps, output=0)
         Function for building binomial option tree for european call option payoff
```

```
    Params
    ------
    spot         int or float      - spot price
    strike       int or float      - strike price
    rate         float             - interest rate
    sigma        float             - volatility
    time         int or float      - expiration time
    steps        int               - number of trees
    output       int               - [0: price, 1: payoff, 2: option value, 3:␣
↪option delta]

    Returns
    -------
    out:         ndarray
    An array object of price, payoff, option value and delta specified by the␣
↪output parameter

    """

    # params
    ts = time/steps
    u = 1+sigma*sqrt(ts)
    v = 1- sigma*sqrt(ts)
    p = 0.5+rate*sqrt(ts)/(2*sigma)
    df = 1/(1+rate*ts)

    # initialize arrays
    px = zeros((steps+1, steps+1))
    cp = zeros((steps+1, steps+1))
    V = zeros((steps+1, steps+1))
    d = zeros((steps+1, steps+1))

    # binomial loop

    # forward loop
    for j in range(steps+1):
        for i in range(j+1):
            px[i,j] = spot*power(v,i)*power(u,j-i)
            cp[i,j] = maximum(px[i,j]-strike, 0)

    # reverse loop
    for j in range(steps+1, 0, -1):
        for i in range(j):
            if (j==steps+1):
                V[i,j-1] = cp[i,j-1]
                d[i,j-1] = 0
            else:
                V[i,j-1] = df*(p*V[i,j]+(1-p)*V[i+1,j])
                d[i,j-1] = (V[i,j]-V[i+1,j])/(px[i,j]-px[i+1,j])

    results = around(px,2), around(cp,2), around(V,2), around(d,4)

    return results[output]
```

[ ]:
```
# Asset price
px = binomial_option(100,100,0.05,0.2,1,4,0)
px
```

[ ]:
```
# Intrinsic value of call options
cp = binomial_option(100,100,0.05,0.2,1,4,1)
cp
```

[ ]:
```
# Option price
opx = binomial_option(100,100,0.05,0.2,1,4,2)
opx
```

[ ]:
```
# Option delta
d = binomial_option(100,100,0.05,0.2,1,4,3)
d
```

[ ]:
```
# Binomial Option Price
print(f"European Call Option Price using Binomial Tree Method: {opx[0,0]:.2f}")
```

1.5.2 Visualize the Binomial Tree

[ ]:
```
# Plot a 4-Step Binomial Tree
plot_binomial_tree(px[0,0], px, opx, d)
```

2 References

- Numpy Documentation
- Paul Wilmott (2007), Paul Wilmott introduces Quantitative Finance

Python Labs by Kannan Singaravelu.