5.1 Binomial Tree For Option Pricing

The two most popular models for using binomial trees to price options are

- **Cox et al. (1979) (CRR for short)** whose extra degree of freedom is to set
  
  \[ ud = 1 \]

  thus
  
  \[ u = e^{\sigma \sqrt{\Delta t}}, \quad d = e^{-\sigma \sqrt{\Delta t}}, \quad q = \frac{e^{r \Delta t} - d}{u - d} \]

- **Rendleman and Bartter (1979)** who choose:

  \[ q = \frac{1}{2} \]

  and so

  \[ u = e^{(r - \frac{1}{2} \sigma^2) \Delta t + \sigma \sqrt{\Delta t}}, \quad d = e^{(r - \frac{1}{2} \sigma^2) \Delta t - \sigma \sqrt{\Delta t}}. \]

We wish to generate a stock price tree, so denote the value of the underlying asset after timestep \( i \) and upstate \( j \) by \( S_{ij} \) and we have that:

\[ S_{ij} = S_0 u^i d^{j-i} \]

First download and start with this template code:
First declare and initialise the Black Scholes parameters for your chosen problem. Here we are going to value a Black Scholes vanilla European call option with, \( S_0 = 100, X = 100, T = 1, r = 0.06 \) and \( \sigma = 0.2 \), so declare variables for each of these. Next add in an integer to store the number of steps in the tree and call it \( n \). Finally add in some local variable to describe the tree, so we have the timestep length \( dt \), \( u \), \( d \) and \( q \). Your code should look like:

```cpp
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

// Template code for the binomial tree
* Instructions to build the code are here
* http://www.maths.manchester.ac.uk/~pjohnson/Tutorials/node1.html
*
int main() {
    // declare and initialise Black Scholes parameters
    // declare and initialise tree parameters (steps in tree)
    // declare and initialise local variables (u,d,p)
    // create storage for the stock price tree and option price tree
    // setup and initialise the stock price tree
    // setup and initialise the final conditions on the option price tree
    // loop through time levels, setting the option price at each node in the tree
    // output the estimated option price
}
```

Then calculate the values for \( dt \), \( u \), \( d \) and \( q \), using the appropriate formula, you should check the value of these against those found in my lecture notes (click here, slide 19).

Next we need to create some storage for the values of the stock at each node in the tree. Declare a `vector of vectors` `stockTree`, into which we will place our stock price nodes.
// create storage for the stock price tree and option price tree
vector<vector<double>> stockTree(n+1,vector<double>(n+1));

where this has been intialised to an $n + 1 \times n + 1$ 2D array, and $n$ is the number of nodes in the tree.

Now use a for loop and the function pow to input the value of the stock at each node in the tree, where $S_{i,j} \rightarrow \text{stockTree}[i][j]$.

Now print out the vector stockTree[i][j] to the screen. Your code might look something like this

```cpp
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

/* Template code for the binomial tree
 * Instructions to build the code are here
 * http://www.maths.manchester.ac.uk/~pjohson/Tutorials/node1.html */

int main()
{
    // declare and initialise Black Scholes parameters
    double S0 = 100., X = 100., T = 1., r = 0.06, sigma = 0.2;
    int n = 3;
    // declare and initialise local variables (u,d,q)
    double dt, u, d, q;
    dt = T / n;
    u = exp(sigma * sqrt(dt));
    d = exp(-sigma * sqrt(dt));
    q = (exp(r * dt) - d) / (u - d);
```

```
// create storage for the stock price tree and option price tree
vector<vector<double>> stockTree(n+1,vector<double>(n+1));
// setup and initialise the stock price tree
for(int i=0;i<=n;i++)
{
    for(int j=0;j<=i;j++)
    {
        stockTree[i][j]=S0*pow(u,j)*pow(d,i-j);
        cout << i << " " << j << " " << stockTree[i][j] << endl;
    }
    cout << endl;
}
/** OUTPUT */
0 0 100
1 0 89.0947
1 1 112.24
2 0 79.3787
2 1 100
2 2 125.978
3 0 70.7222
3 1 89.0947
3 2 112.24
3 3 141.398
*/

Compare the values with those from the example in the lectures.

5.2 The Option Value Tree

Let us generate a simple example so that we can compare results at every stage to something that we can work out on paper. This is an important idea in debugging, to solve the problem and do all of your bug checking on a small scale before attempting the full problem.

First declare a vector of vectors which shall hold the values of the option

```
// create storage for the stock price tree and option price tree
vector<vector<double>> valueTree(n+1,vector<double>(n+1));
```

and set it to the same size as stockTree. Here we use the same relation

\[ V_{i,j} \rightarrow \text{valueTree}[i][j]. \]

Now fill in the final values of the tree, given that we have already first generated the stock tree:

\[ V_{n,j} = \text{payoff}(S_{n,j}), \]
where payoff is the appropriate function for the type of option we are solving for. For a
European call option you should get

```cpp
for (int j = 0; j <= n; j++)
{
    valueTree[n][j] = max(stockTree[n][j] - X, 0.);
    cout << n << " " << j << " " << valueTree[n][j] << endl;
}
```

Now we need to loop **backwards** through the tree to generate the value at each node
using the equation:

\[ V_{ij} = e^{-r\Delta t}(qV_{i+1,j+1} + (1 - q)V_{i+1,j}) \]

You will need to do the following.

- Write *for* loops to move backwards through the vector array calculating the value of
  the option at each node.
- Print out the tree to screen (with n=3) and compare to the simple example (click [here](#), slide 19) to check your code is working.
- If your values don’t match - try to work out why!!
- Now print out the value at (0,0) increasing the number of steps in the tree. Do the
  results look feasible? Compare them against the exact values from the formula.

Finally your code should look like this

```cpp
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

/*@ Template code for the binomial tree
 * Instructions to build the code are here
 * http://www.maths.manchester.ac.uk/~pjohnson/Tutorials/node1.html
 */
int main()
{
    // declare and initialise Black Scholes parameters
    double S0 = 100., X = 100., T = 1., r = 0.06, sigma = 0.2;
    // declare and initialise tree parameters (steps in tree)
    int n = 3;
    // declare and initialise local variables (u,d,q)
    double dt, u, d, q;
    dt = T / n;
    u = exp(sigma * sqrt(dt));
    d = exp(-sigma * sqrt(dt));
    q = (exp(r * dt) - d) / (u - d);
    // create storage for the stock price tree and option price tree
```
vector<vector<double>> stockTree(n+1,vector<double>%(n+1));
// setup and initialise the stock price tree
for(int i=0;i<=n;i++)
{
    for(int j=0;j<=i;j++)
    {
        stockTree[i][j]=S0*pow(u,j)*pow(d,i-j);
    }
}
vector<vector<double>> valueTree(n+1,vector<double>%(n+1));
for(int j=0;j<=n;j++)
{
    valueTree[n][j]=max(stockTree[n][j]-X,0.);
}
for(int i=n-1;i>=0;i--)
{
    for(int j=0;j<=i;j++)
    {
        valueTree[i][j] = exp(-r*dt)*( q*valueTree[i+1][j+1] + (1-q)*valueTree[i+1][j] );
    }
}
cout<<"V(S=100,t=0) = ",valueTree[0][0]<<endl;
/** OUTPUT
V(S=100,t=0) = 11.552
*/

Finally, you could try to improve your code with the following:

- Create a function returning the value of the binomial tree for a set of given parameters.
- Write a code storing two time-levels, and compare (at every stage if needed) with the previous code.
- Is it possible to store just one time-level? Try to write a code for this.
- Do you notice any difference (time taken for computation) between the codes with different storage requirements?

5.3 Evaluating Convergence Properties

Taking the code from last time, we can finish this off by taking the main algorithm out into a function and allowing the payoff function to be set independently. A representative code might look like the following:

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
```
```cpp
#include <vector>
#include <algorithm>
using namespace std;

/* Template code for the binomial tree 
* Instructions to build the code are here 
* http://www.maths.manchester.ac.uk/~pjohnson/Tutorials/node1.html */

// return the payoff of the function you want to evaluate
double payoff(double S, double X)
{
    return max(S-X, 0.);
}

// return the value of the binomial tree
double binomialTree(double S0, double X, double T, double r, double sigma, int n)
{
    // declare and initialise local variables (u,d,q)
    double dt, u, d, q;
    dt = T/n;
    u = exp(sigma * sqrt(dt));
    d = exp(-sigma * sqrt(dt));
    q = (exp(r * dt) - d) / (u - d);
    // create storage for the stock price tree and option price tree
    vector<vector<double>> stockTree(n+1, vector<double>(n+1));
    // setup and initialise the stock price tree
    for (int i=0; i<=n; i++)
    {
        for (int j=0; j<=i; j++)
        {
            stockTree[i][j]=S0*pow(u,j)*pow(d,i-j);
        }
    }
    vector<vector<double>> valueTree(n+1, vector<double>(n+1));
    for (int j=0; j<=n; j++)
    {
        valueTree[n][j]=payoff(stockTree[n][j],X);
    }
    for (int i=n-1; i>=0; i--)
    {
        for (int j=0; j<=i; j++)
        {
            valueTree[i][j] = exp(-r * dt) * (q * valueTree[i+1][j+1] + (1-q) * valueTree[i+1][j]);
        }
    }
    return valueTree[0][0];
}

int main()
```

Now we wish to investigate what happens when $N$ is increasing. Write the following loop in your code to output some results to file, you will need to adjust the output filename depending on your preference for where the file should be saved.

```c++
// open output stream
ofstream output("binomial-convergence.csv");
// check it is open
if(output.is_open())
{
    cout << "File opened successfully" << endl;
    // output various n and V_n to file
    for(int n=3;n<=1000;n++)
    {
        output << n<<" , " << binomialTree(S0,X,T,r,sigma,n) << endl;
    }
    cout << "File write complete" << endl;
}
else
{
    cout << "File could not be opened for some reason.\n";
}
/** OUTPUT
File opened successfully
File write complete */
```

Run the code for $S_0 = 100$ and $X = 100$ and then import the data into a plotting package and visualise the results. You should see the following pattern.
The pattern can be explained by realising that the nodes at terminal time will either be placed exactly on the strike price, or above/below depending on whether the number of steps is odd or even. We will see quite a different pattern if you choose $S_0 \neq X$, so for example set $S_0 = 97.3467$ in your code and rerun the results. After putting this in a plotting package you should see something like this.

To produce this graph I have set $n = 25$ as the minimum grid size to show up the pattern better. This time we see humps as well as an odd-even effect. The humps are again caused by the positioning of the grid nodes relative to the strike price. Although the convergence of the tree is $O(1/n)$, the humps make it very difficult to achieve high accuracy. There are several papers that try to address this, including Leisen and Reimer (1996) and Heston and Zhou (2000), whilst Joshi (2007) gives a good comparison of different methods when
American options are being priced.

References


