1. Simulating Binomial Trees are known to take up a lot of memory. For a typical Binomial Tree simulating a typical option calculation requiring 100 steps one needs to create a stock price matrix consisting of S[101][101] and Option matrix of C[101][101] (according to Segupta solutions). The program provides to simulate the pricing of European/American call/put options. The S[N+1][N+1] need not be created and C[n][i] can be obtained as follows

**C[i]: = max(0,** Sun-idi **- X) Call Option**

#pragma once

#include "targetver.h"

#include <fstream>

#include <stdio.h>

#include <tchar.h>

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

const int maxSize = 11; //global variable

double termVal(char iType, double strkPr, double termPr);

void printTree(double tree[maxSize], int numSteps, double timeStep);

int main()

{

//cout.open("bino.txt");

double S = 60.; // price of stock or underlying security

double K = 60.; // strike price

double r = 0.04; // interest rate

double T = 0.60; // remaining life

double sig = 0.20; // volatility

double stk[maxSize], stk0[maxSize];

double amer[maxSize], euro[maxSize];

char optType;

string optName;

int i, j, n;

double dt, u, d, p, emrdt, exVal;

n = maxSize - 1; // number of steps; must be < maxSize

optType = 'p'; // c for call p for put

optName = "call";

if(optType =='p') optName = "put";

dt = T / n; // step size in years

u = exp(sig \* sqrt(dt)); // up movement multiplier

d = 1/u; // down movement multiplier

emrdt = exp(-r \* dt); // discount factor per step

// print inputs

cout << "Option Type = " << optName << endl;

cout << "So = " << S << endl;

cout << "K = " << K << endl;

cout << "r(annual interest rate) = " << r << endl;

cout << "T(Time to expiration - years) = " << T << endl;

cout << "sig = " << sig << endl;

cout << "maxSize = " << maxSize << endl;

cout << "u = " << u << " d = " << d << endl;

cout << "dt(Step size in years) = " << dt << endl;

// p is risk neutral probability of up movement

p = (exp(r \* dt) - d) / (u - d);

stk[0] = S\*pow(d,n); // initial stock price at time 0

// generate stock price tree

for(int i = 1; i <= n; i++){

stk[i] = stk[i - 1] \* u \* u;

}

for(j=0; j<=n; j++){

amer[j] = termVal(optType, K, stk[j]);

euro[j] = termVal(optType, K, stk[j]);

exVal = termVal(optType, K, stk[j]);//S\*pow(u,2\*i-j));

if (exVal > amer[j]) amer[j] = exVal;

}

// generate option value tree

for(j = n; j >= 0; j--) // counts steps backward

{

for(i = 0; i < j; i++)

{

euro[i] = ((1-p) \* euro[i] + p

\* euro [i+1]) \* emrdt;

amer[i] = ((1-p) \* amer[i] + p

\* amer [i+1]);

}

}

cout << endl << "Stock price tree" << endl << endl;

printTree(stk, n, dt);

cout << endl;

cout << "Value of the American " << optName << " option: ";

cout << amer[0];

cout << endl << endl;

cout << endl << "American " << optName << " value tree";

cout << endl << endl;

printTree(amer, n, dt);

cout << endl;

cout << "Value of the European " << optName << " option: ";

cout << euro[0];

cout << endl << endl;

cout << endl << "European " << optName << " value tree";

cout << endl << endl;

printTree(euro, n, dt);

cout << endl;

system("PAUSE");

return 0;

}

double termVal(char iType, double strkPr, double termPr)

{

/\*

functions parameters:

iType: position type, c, p, or u (underlying)

strkPr: strike price of option or purchase price of u

termPr: terminal price of u

\*/

double posVal = 0.;

switch (iType)

{

case 'u':

posVal = termPr - strkPr;

break;

case 'c':

if(termPr > strkPr) posVal = termPr - strkPr;

break;

case 'p':

if(termPr < strkPr) posVal = strkPr - termPr;

break;

}

return posVal;

}

void printTree(double tree[maxSize], int numSteps, double timeStep)

{

int i, j, k;

cout << "Step";

cout.setf (ios::fixed);

for(j = 0; j <= numSteps; j++)

{

cout.width(6);

cout << j;

}

cout << endl;

cout << "Time";

cout.precision(2);

cout.setf (ios::fixed);

cout.setf(ios::showpoint);

for(j = 0; j <= numSteps; j++)

{

cout.width(6);

cout << j \* timeStep;

}

cout << endl;

for(i = 0; i <= numSteps; i++)

{

cout << endl;

for (k = 1; k <= i+1; k++) {cout << " ";}

cout.width(4);

cout << tree[i];

}

return;

}

