



```

/*****
Copyright ©2009, Kuberre Systems, Inc. All rights reserved. For more information, address Kuberre Systems, Inc,
805 Turnpike Street, North Andover, MA 01845.
*****/

```

```

#include "stdio.h"
#include "KBSHansaMatrix.h"
#include <math.h>

```

```
using namespace std;
```

```
short TrinomialAmerican(boolean CallPut, double AssetP, double Strike,
                        double RiskFree, double Div, double Time, double Vol, long nSteps)

```

```

{
    /*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Computes the Boyle (1986) Trinomial Tree for American Call/Put Option Values based
    % on the following inputs:
    % CallPut      =   Call = 1, Put = 0
    % AssetP      =   Underlying Asset Price
    % Strike      =   Strike Price of Option
    % RiskFree    =   Risk Free rate of interest
    % Div         =   Dividend Yield of Underlying
    % Time        =   Time to Maturity
    % Vol         =   Volatility of the Underlying
    % nSteps      =   Number of Time Steps for Trinomial Tree to take
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */

```

```

    double dt = Time / nSteps;          // Allocates the time steps
    double cc = RiskFree - Div;         // Specifies the cost of carry (r - D)
    HansaMatrix* b;
    HansaMatrix* strike;

```

```

    if(CallPut){
        b = new HansaMatrix(1,1,1);
    }
    else {
        b = new HansaMatrix(-1,1,1);
    }

```

```

    strike = new HansaMatrix(Strike,1,1);
    HansaMatrix& RR = expo(RiskFree * dt);
    // The magnitude of an up movement

```



Hansa



```
HansaMatrix& Up = expo(Vol * sqrt(2 * dt));
// The magnitude of a down movement

HansaMatrix& Down = clone(Up)^(-1);
// Specifies the probability of up, down and mid moves for trinomial tree
HansaMatrix& P_up = ((expo(cc * dt / 2) - expo(-Vol * sqrt(dt / 2))).
    dotDiv((expo(Vol * sqrt(dt / 2)) - expo(-Vol * sqrt(dt / 2)))) ^ 2;
HansaMatrix& P_down = ((expo(Vol * sqrt(dt / 2)) - expo(cc * dt / 2)).
    dotDiv((expo(Vol * sqrt(dt / 2)) - expo(-Vol * sqrt(dt / 2)))) ^ 2;
HansaMatrix& one = ones(1,1);
HansaMatrix& P_mid = one - P_up - P_down;

HansaMatrix& Df = expo(-RiskFree * dt);

// Sets up the asset movements on the trinomial tree
HansaMatrix& Value = zeros(1,(2*nSteps+1));
Value.selectRow(1);
for(int i = 0; i<=(2*nSteps); i++)
{
    int state = i + 1;
    HansaMatrix& U = clone(Up);
    HansaMatrix& D = clone(Down);

    HansaMatrix& movement = (b * (AssetP * (U ^ max(i - nSteps, 0)) *
        (D ^ max(nSteps * 2 - nSteps - i, 0)) - strike));
    movement.selectRow(1);
    Value(state) = max(0, movement[1]);

    U.unload();
    D.unload();
}

// Works backwards to determine the price of the option
for(int TT = (nSteps - 1); TT>=0; TT--)
{
    for(int i = 0; i<=(TT * 2); i++)
    {
        int state = i + 1;
        HansaMatrix& U = clone(Up);
        HansaMatrix& D = clone(Down);
        HansaMatrix& Pu = clone(P_up);
```



Hansa



```
HansaMatrix& Pm = clone(P_mid);
HansaMatrix& Pd = clone(P_down);

HansaMatrix& price1 = ((Pu * Value[state + 2] + Pm *
                      Value[state + 1] + Pd * Value[state]) * Df);
HansaMatrix& price2 = (b * (AssetP * (U ^ max(i - TT, 0)) *
                      (D ^ max(TT * 2 - TT - i, 0)) - strike));

price1.selectRow(1);
price2.selectRow(1);
Value(state) = max(price1[1], price2[1]);

U.unload();
D.unload();
Pu.unload();
Pm.unload();
Pd.unload();
    }
}
double Trinomial = Value[1];
return SUCCESS;
}
```