Axel Kind
Assistant Professor
Corporate Finance Division
Department of Economics and
Business Administration (WWZ)
University of Basel
Spring 2011

Quantitative Security Analysis, Course-Nr. 21996

Solution Exercise Set 3: Binomial-Tree Option Pricing

1 Convergence of the Binomial Option Price

1. Implement a function *BinomialTree.m* that can price put and call options of either for European or American type using the binomial-tree method in the spirit of Cox, Ross and Rubinstein (1979). Distinguish between option types {Call or Put} and exercise types {European or American} by using the `switch`-command.

```
function value = BinomialTree(putCall,exerciseType,S0,K,...
    sigma,r,div,T,n)
%
% This function calculates the American or European option value by a
% binomial tree model.
%
% value = BinomialTree(putCall,exerciseType,S0,K,sigma,r,div,T,n)
%
% Input:        putCall:        indicates the option type
%                               putCall = 'Call' for a call option
%                               putCall = 'Put' for a put
%                               option
%               exerciseType:   European ('E') or American ('A')
%                               exercise type
%               S:              Stock price
%               K:              Strike price
%               sigma:          Volatility
%               r:              riskfree rate
%               div:            Dividend yield
%               T:              Time to maturity
%               n:              Number of steps in the tree
%
% OUTPUT:       value:          option value
%
% EXAMPLE: value =...
%          BinomialTree('Put','A',50,50,0.4,0.1,0.05,0.5,500)

% Parameters for binomial tree
dt = T/n;
u  = exp(sigma*sqrt(dt));
d  = 1/u;
p  = (exp((r-div)*dt)-d)/(u-d);
q  = 1-p;
disc = exp(-r*dt);
% Initialize matrices
stockM = zeros(n+1,n+1);
optionM = zeros(n+1,n+1);
% Create stock tree
stockM(1,1) = S0;
for j = 2:n+1
    for i = 1:j-1
        stockM(i,j) = stockM(i,j-1)*u;
    end
    stockM(i+1,j) = stockM(i,j-1)*d;
end
% Set z parameter to calculate the option payoff depending on the
% selected option type (Call, Put)
```

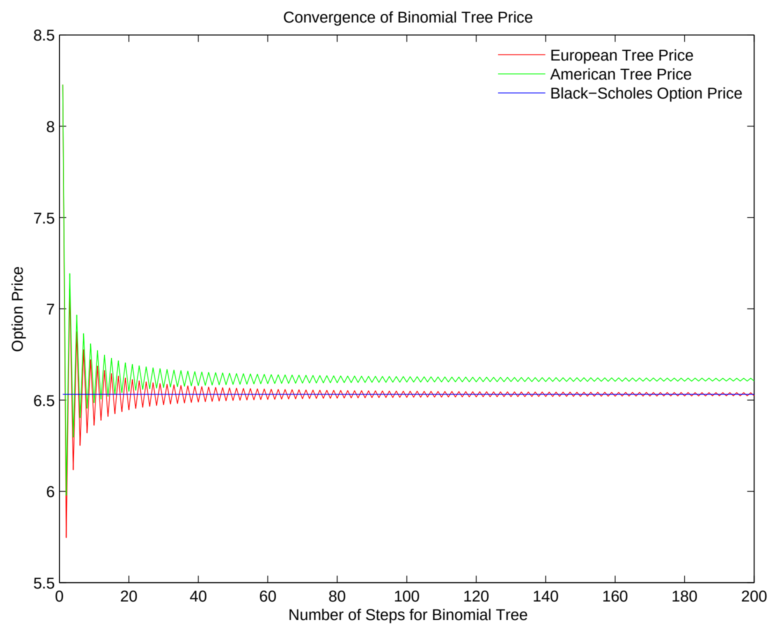
```
switch putCall
    case 'Call'
        z = 1;
    case 'Put'
        z = -1;
    otherwise
        error('Check option type!');
end
% Insert terminal values
optionM(:,end) = max(z*(stockM(:,end)-K),0);
% Value call by working backward from time n-1 to time 0
switch exerciseType
    % European option
    case 'E'
        for j=n:-1:1;
            for i=j:-1:1;
                optionM(i,j) = disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1));
            end
        end;
    % American option
    case 'A'
        for j=n:-1:1;
            for i=j:-1:1;
                optionM(i,j) = max(z*(stockM(i,j)-K),disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1)));
            end
        end
    otherwise
        error('Check exercise type!');
end
value = optionM(1,1);
```

2. Use the code from Exercise 2.1 to plot the convergence of the put option (`exerciseType = 'E'`) specified in Table 1 to the Black-Scholes price by steadily increasing the number of steps in the binomial tree. Plot in the same figure the convergence of the American counterpart (`exerciseType = 'A'`) to this option.

Parameter	MATLAB
Option type	`putCall = 'Put';`
Exercise type A	`exerciseType = 'A';`
Exercise type B	`exerciseType = 'E';`
Stock price	`stockPrice = 50;`
Strike price	`strikePrice = 50;`
Volatility	`sigma = 0.5;`
Riskfree interest rate	`r = 0.05;`
Dividend yield	`div = 0.02;`
Time-to-maturity	`T = 2;`
Steps in binomial tree	`stepsV = 1:200;`

Table 1: Parameter for plotting convergence

```
% m-file to plot the convergence of the binomial put option price to
% the Black-Scholes price and the convergence of the american put option
% price.

close all
clear all

% Assign parameter values
putCall = 'Put';
exerciseTypeA = 'E';
exerciseTypeB = 'A';
stockPrice = 50;
strikePrice = 50;
sigma = 0.5;
r = 0.05;
div = 0.02;
T = 0.5;
stepsV = 1:100;
lengthSteps = length(stepsV);

% Inititialize value vector
valueA = nan(lengthSteps,1);
valueB = nan(lengthSteps,1);

% Loop over the increasing number of steps
counter = 0;
for iSteps = stepsV
    counter = counter + 1;
    % Calculate European put option price
    valueA(counter) = ...
        BinomialTree(putCall,exerciseTypeA,stockPrice,strikePrice,...
        sigma,r,div,T,iSteps);
    % Calculate American put option price
    valueB(counter) = ...
        BinomialTree(putCall,exerciseTypeB,stockPrice,strikePrice,...
        sigma,r,div,T,iSteps);
end
% Compute the Black-Scholes option price
bsValue = BlackScholesPriceGreeks(putCall,stockPrice,...
    strikePrice,sigma,r,T,div);

% Plot the binomial tree price
figure;
plot(stepsV,valueA,'r');
hold on
plot(stepsV,valueB,'g');
% Plot the Black-Scholes option price
plot([stepsV(1) stepsV(end)],[bsValue bsValue]);
% Add x-axis and y-axis description
xlabel('Number of Steps for Binomial Tree');
ylabel('Option Price');
% Add legend to plot
legend('European Tree Price','American Tree Price','Black-Scholes Option Price');
legend('boxoff');
% Add title
title('Convergence of Binomial Tree Price');
% Save plot
saveas(gcf,'Convergence.eps','psc2');
```



2 Exercise Boundary

Modify the function *BinomialTree.m* to explicitly consider the exercise boundary for an American call or put option. Plot the exercise boundary for the option in Table 1 with 500 steps as shown in Figure 1. Consider only values where the exercise boundary is not zero. Save the function under *ExerciseBoundary.m*.

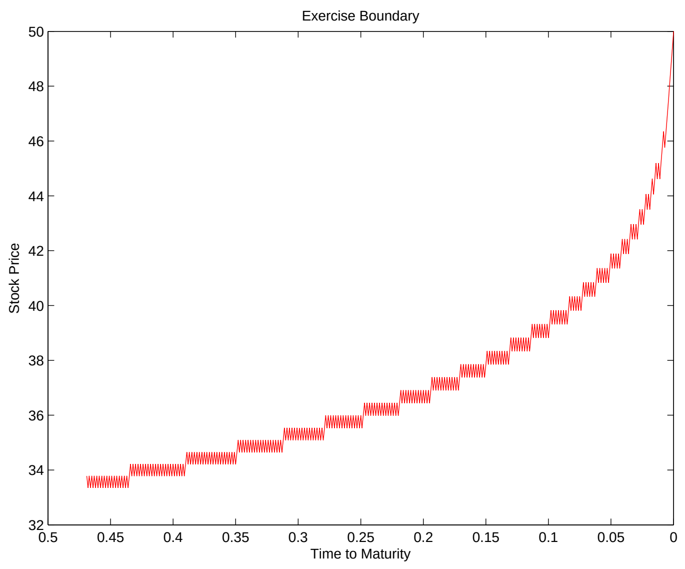
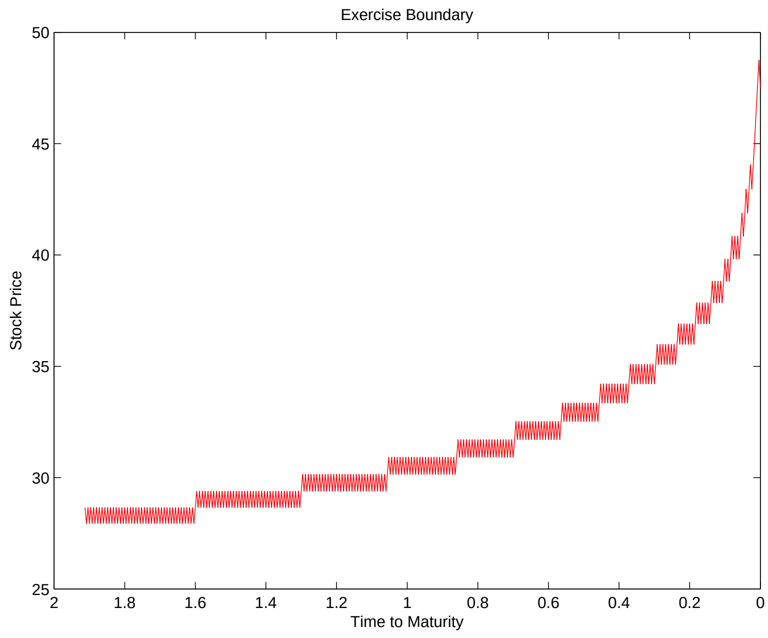


Figure 1: Exercise Boundary

```
function value = ExerciseBoundary(putCall,S0,K,sigma,r,div,T,n)
%
% This function calculates the American ption value by a
% binomal tree model and plots the exercise boundary.
%
% value = ExerciseBoundary(putCall,S0,K,sigma,r,div,T,n)
%
% Input:        putCall:        indicates the option type
%                               putCall = 'Call' for a call option
%                               putCall = 'Put' for a put
%                               option
%               S:              stock price
%               K:              strike price
%               sigma:          Volatility
%               r:              riskfree rate
%               div:            dividend yield
%               T:              time to maturity
%               n:              number of steps in the tree
%
% OUTPUT:       value:          Product value
%               plot:           a plot depicting the exercise boundary of
%                               the option over time
%
% EXAMPLE: value =...
%          ExerciseBoundary('Put',50,50,0.4,0.1,0.05,2,500)

close all

% Parameters for binomial tree
dt = T/n;
u  = exp(sigma*sqrt(dt));
d  = 1/u;
p  = (exp((r-div)*dt)-d)/(u-d);
q  = 1-p;
disc = exp(-r*dt);

% Initialize matrices
stockM = zeros(n+1,n+1);
optionM = zeros(n+1,n+1);
exBoundM = zeros(n+1,n+1);

% Create stock tree
stockM(1,1) = S0;
for j = 2:n+1
    for i = 1:j-1
        stockM(i,j) = stockM(i,j-1)*u;
    end
    stockM(i+1,j) = stockM(i,j-1)*d;
end

% Set z parameter to calculate the option payoff depending on the
% selected option type (Call, Put)
switch putCall
    case 'Call'
        z = 1;
        % Exercise boundary at last time step
        exBoundM(:,n+1) = stockM(:,n+1)>K;
    case 'Put'
        z = -1;
        % Exercise boundary at last time step
        exBoundM(:,n+1) = stockM(:,n+1)<K;
    otherwise
        error('Check option type!');
end
```

```
% Insert terminal values
optionM(:,end) = max(z*(stockM(:,end)-K),0);

% Value call or put by working backward from time n-1 to time 0
for j=n:-1:1;
    for i=j:-1:1;
        % Calculate intrinsic value of call option
        innerValue = z*(stockM(i,j)-K);
        % Calculate option value at particular node
        optionM(i,j) = max(innerValue,disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1)));
        % Check whether early exercise is taken or not
        if innerValue > disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1))
            exBoundM(i,j) = 1;
        end
    end
end

exBoundM = max(exBoundM.*stockM);
value = optionM(1,1);
% Create a time vector
exBoundTimeV  = dt*(n:-1:0);
% Use only values where the exercise boundary is not zero
exBoundTimeV  = exBoundTimeV(exBoundM > 0);
exBoundM = exBoundM(exBoundM > 0);
% plot exercise boundary
figure;
plot(exBoundTimeV,exBoundM,'r');
% Change x-axis direction that time to maturity is on the left of the plot
set(gca,'XDir','reverse')
xlabel('Time to Maturity');
ylabel('Stock Price');
title('Exercise Boundary');
saveas(gcf,'ExerciseBoundary.eps','psc2');
```



3 Greeks in Binomial Trees

Complete the function *BinomialTree.m* such that it is able to extract the Greeks from the tree by using the following formulas and save it as *BinomialTreeGreeks.m*:

(i) Delta $= \Delta \approx \frac{C_u - C_d}{S_0 \cdot u - S_0 \cdot d}$

(ii) Gamma $= \Gamma \approx \frac{\Delta_u - \Delta_d}{(S_0 \cdot uu - S_0 \cdot dd)/2}$

(iii) Vega[1] $= V \approx \frac{C(\sigma + 0.01) - C(\sigma - 0.01)}{2 \cdot 0.01}$

(iv) Theta $= \Theta \approx \frac{C_{ud} - C}{2 \cdot (T/n)}$

(v) rho $= \rho \approx \frac{C(r + 0.01) - C(r - 0.01)}{2 \cdot 0.01}$

Calculate the greeks using the parameters from Table 1 with 500 steps for the American put. To compute the option's vega and rho already inside the function *BinomialTreeGreeks.m*, apply `feval` for re-evaluation of the option price at different volatility and risk-free rate levels.

[1]Although Vega is not a Greek letter, it is commonly subsumed as such.

```
function [value,delta,gamma,theta,vega,rho] = BinomialTreeGreeks(putCall,...
    exerciseType,S0,K,sigma,r,div,T,n)
%
% This function calculates the American or European option value by a
% binomial tree model.
%
% [value,delta,gamma,theta,vega,rho] = ...
%       BinomialTreeGreeks(putCall,exerciseType,S0,K,sigma,r,div,T,n)
%
% Input:        putCall:        indicates the option type
%                               putCall = 'Call' for a call option
%                               putCall = 'Put' for a put
%                               option
%               exerciseType:   European ('E') or American ('A')
%                               exercise type
%               S:              Stock price
%               K:              Strike price
%               sigma:          Volatility
%               r:              riskfree rate
%               div:            Dividend yield
%               T:              Time to maturity
%               n:              Number of steps in the tree
%
% OUTPUT:       value:          option value
%               delta:          dV/dS
%               gamma:          dV^2/d^2S
%               theta:          dV/dT
%               vega:           dV/dsigma
%               rho:            dV/dr
%
% EXAMPLE: [value,delta,gamma,theta,vega,rho] = ...
%          BinomialTreeGreeks('Put','A',50,50,0.4,0.05,0.02,2,500)

% Parameters for binomial tree
dt = T/n;
u  = exp(sigma*sqrt(dt));
d  = 1/u;
p  = (exp((r-div)*dt)-d)/(u-d);
q  = 1-p;
disc = exp(-r*dt);

% Initialize matrices
stockM = zeros(n+1,n+1);
optionM = zeros(n+1,n+1);

% Create stock tree
stockM (1,1) = S0;
for j = 2:n+1
    for i = 1:j-1
        stockM(i,j) = stockM(i,j-1)*u;
    end
    stockM(i+1,j) = stockM(i,j-1)*d;
end

% Set z parameter to calculate the option payoff depending on the
% selected option type (Call, Put)
switch putCall
    case 'Call'
        z = 1;
    case 'Put'
        z = -1;
    otherwise
        error('Check option type!');
end
% Insert terminal values
optionM(:,end) = max(z*(stockM(:,end)-K),0);
```

```
% Value call by working backward from time n-1 to time 0
switch exerciseType
    % European option
    case 'E'
        for j=n:-1:1;
            for i=j:-1:1;
                optionM(i,j) = disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1));
            end
        end;
    % American exercise type
    case 'A'
        for j=n:-1:1;
            for i=j:-1:1;
                optionM(i,j) = max(z*(stockM(i,j)-K),disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1)));
            end
        end
    otherwise
        error('Check exercise type!');
end
value = optionM(1,1);
% Calculate greeks
if nargout > 1
    delta = (optionM(1,2)-optionM(2,2))/(stockM(1,2)-stockM(2,2));
end
if nargout > 2
    deltaUp = (optionM(1,3)-optionM(2,3))/(stockM(1,3)-stockM(2,3));
    deltaDown = (optionM(2,3)-optionM(3,3))/(stockM(2,3)-stockM(3,3));
    gamma = (deltaUp-deltaDown)/((stockM(1,3)-stockM(3,3))/2);
end
if nargout > 3
    theta = (optionM(2,3)-optionM(1,1))/(2*dt);
end
% Calculate vega and rho with re-evaluation
if nargout > 4
    vegaUp = feval(@BinomialTreeGreeks,putCall,exerciseType,S0,K,...
        sigma+0.01,r,div,T,n);
    vegaDown = feval(@BinomialTreeGreeks,putCall,exerciseType,S0,K,...
        sigma-0.01,r,div,T,n);
    vega = (vegaUp - vegaDown)/(2*0.01);
end
if nargout > 5
    rhoUp = feval(@BinomialTreeGreeks,putCall,exerciseType,S0,K,sigma,...
        r+0.01,div,T,n);
    rhoDown = feval(@BinomialTreeGreeks,putCall,exerciseType,S0,K,sigma,...
        r-0.01,div,T,n);
    rho = (rhoUp - rhoDown)/(2*0.01);
end
```

```
value =

    9.5081
delta =

   -0.3626
gamma =

    0.0145
theta =

   -1.8742
vega =

   25.5789
rho =

  -38.1940
```

4 Implied Volatility

On April 5, 2010, the share of Google Inc. (GOOG) traded at 571.19\$ (S). At the same time, American call-options written on Google were priced as shown in Table 2 (in \$). The maturities of the options are shown in parenthesis.

	Maturity				
Strike	0.04 years	0.13 years	0.20 years	0.45 years	0.80 years
660	0.45	1.86	3.60	10.50	20.90
640	1	3.30	5.70	14.50	27.90
620	2.15	5.93	8.90	20.20	34.86
600	5.30	10.75	14.30	26.81	42.80
580	11.5	17.30	23.00	36.80	48.73
560	21.30	28.10	32.80	46.50	60.40
540	36.60	43.10	45.90	59.50	73.00

Table 2: Google call options

In addition, you know that Google never pays dividends (`div` = 0). The risk-free rate, given by the one-year Treasury bill rate on the 5th of April, 2010, is `r` = 0.44%.

1. Program a routine that calculates the implied volatility based on binomial trees according to Cox, Ross and Rubinstein (1979). Use the (American) option on Google with maturity 0.20 years and the strike of 580 to show the convergence to the Black-Scholes implied volatility with 1 to 100 steps. Solve the task by creating three files:

   (i) A function *BinomialTreeCall.m* that prices an American call option in a binomial tree framework.

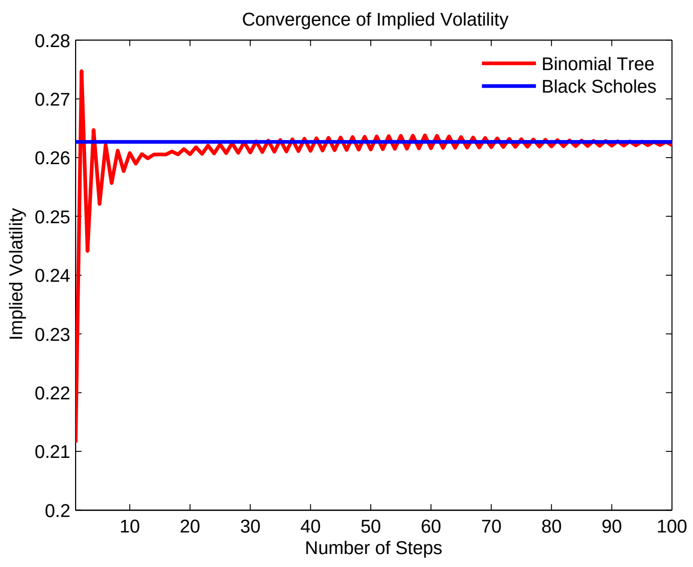
   (ii) A function *BinomialCallDiff.m* that computes the difference of the function *BinomialTreeCall.m* and the given option price and a function *BlackScholesDiff.m* that in turn computes the difference between the function *BlackScholesPriceGreeks.m* and the given option price.

   (iii) A script-file that calls the function *BinomialTreeCall.m* for the implied volatility and shows the convergence to the Black-Scholes implied volatility.

   As you know, the American Call will not be exercised prior to expiration, hence the implied volatility computed from a binomial tree should converge to the European Black-Scholes implied volatility by increasing the number of steps.

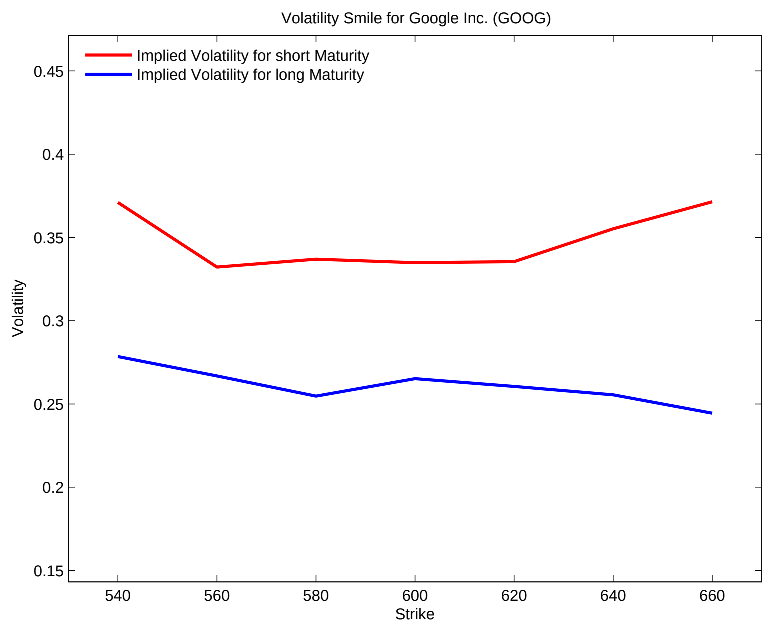
```
function value = BinomialTreeCall(S0,K,sigma,r,T,n)
%
% Parameters for binomial tree
dt = T/n;
u  = exp(sigma*sqrt(dt));
d  = 1/u;
p  = (exp(r*dt)-d)/(u-d);
q  = 1-p;
disc = exp(-r*dt);

% Initialize matrices
stockM = zeros(n+1,n+1);
optionM = zeros(n+1,n+1);

% Create stock tree
stockM (1,1) = S0;
for jStep = 2:n+1
    for iState = 1:jStep-1
        stockM(iState,jStep) = stockM(iState,jStep-1)*u;
    end
    stockM(iState+1,jStep) = stockM(iState,jStep-1)*d;
end

% Create a matrix of option values; insert terminal values
optionM(:,end) = max(stockM(:,end)-K,0);
% values call by working backward from time n-1 to time 0
% American exercise type
for jStep=n:-1:1;
   for iState=jStep:-1:1;
       optionM(iState,jStep) = max(stockM(iState,jStep)-K,...
       disc*(p*optionM(iState,jStep+1)+q*optionM(iState+1,jStep+1)));
   end
end;
value = optionM(1,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function diff = BinomialCallDiff(S0,K,sigma,r,T,n,price)

diff = BinomialTreeCall(S0,K,sigma,r,T,n) - price;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function value = BlackScholesDiff(putCall,S0,K,sigma,r,T,div,price)

value = BlackScholesPriceGreeks(putCall,S0,K,sigma,r,T,div) - price;
```

```
% script-file that shows the convergence of the binomial tree implied
% volatility to the Black-Scholes implied volatility

clear all
close all

% Definition of input parameters:
S0        = 571.19;
K         = 580;
r         = 0.0044;
price     = 23.00;
T         = 0.2;
div       = 0;
putCall   = 'Call';
stepsV    = 1:100;
lenStepsV = length(stepsV);

options = optimset('fzero');
% TolX, 1e-7: tolerance level
options = optimset(options, 'TolX', 1e-7);
% Initialize vector for implied volatilites
implVolBinomial = nan(lenStepsV,1);
% Loop for the number of steps
counter = 0;
for n = stepsV
    counter = counter + 1;
    % Calculate impl. volatility in binomial tree
    implVolBinomial(counter) = fzero(@(sigma)...
        BinomialCallDiff(S0,K,sigma,r,T,n,price), [0.001 1.5], options);
end

% Calculate Black-Scholes impl. volatility
implVolBS  = fzero(@(sigma)...
    BlackScholesDiff(putCall,S0,K,sigma,r,T,div,price), [0.001 1.5], options);

figure;
plot(stepsV,implVolBinomial,'r-','LineWidth',2);
hold on
plot([stepsV(1) stepsV(end)],[implVolBS implVolBS],'b-','LineWidth',2);
title('Convergence of Implied Volatility');
xlabel('Number of Steps');
ylabel('Implied Volatility');
axis([stepsV(1) stepsV(end) 0.2 0.28]);
l = legend('Binomial Tree','Black Scholes');
legend(l,'boxoff')
saveas(gcf, 'ImpliedVolaConvergence.eps','psc2');
```



2. Create two figures using the above data. The call option prices are provided in the Excel-file *GoogleExSet3.xls* and can be imported to MATLAB using the command `xlsread('GoogleExSet3')`.
   (i) Plot the implied volatility depending on the option's strike for the shortest and the longest maturity. What do you observe?
   (ii) Depict the implied volatility surface obtained from the Google options. The volatility surface is a 3-dimensional plot of the implied volatility (z-axis) depending on the option's moneyness (x-axis) (here defined as $ln(S/K)$) and the corresponding maturity (y-axis).

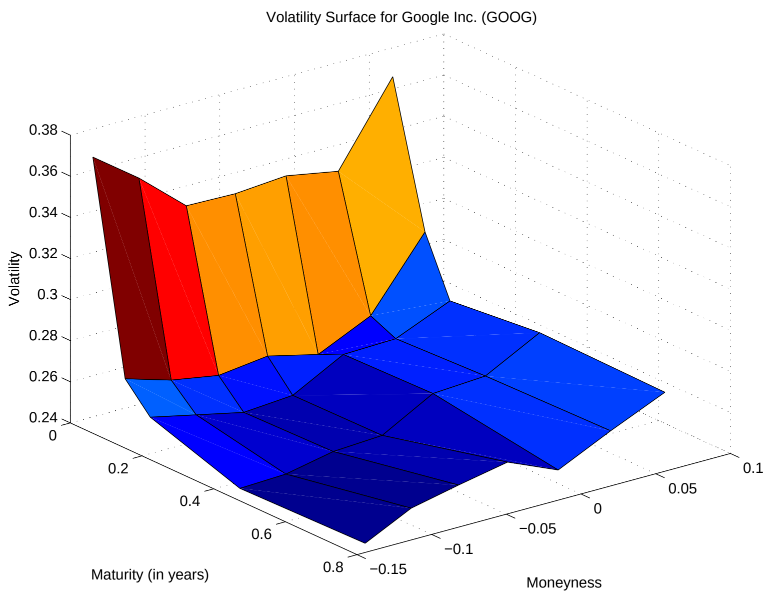
```
% m-file that calculates implied volatilities and plots volatility
% smiles and the volatility surface, respectively.

close all
clear all

optionData = xlsread('GoogleExSet3','A3:F10');

% Input parameter
S0      = 571.19;
div     = 0;
r       = 0.0044;
putCall = 'Call';

KV      = optionData(2:end,1);
TV      = optionData(1,2:end);
priceM  = optionData(2:end,2:end);
nStrike = length(KV);
mMat    = size(TV, 2);


% Initialize matrix for the implied-volatility values
implVolM = nan(nStrike,mMat);

% Loop for different maturities
for jMat = 1:mMat
    T = TV(1,jMat);
    % Loop for the different strike prices
    for iStrike = 1:nStrike
        price   = priceM(iStrike,jMat);
        K       = KV(iStrike);
        try
            % Calculation of the implied volatility by referring to the
            % earlier computed file saved under ImpliedVolatility.m
            implVolM(iStrike, jMat) = ImpliedVolatility(putCall,S0,K,r,T,div,price);
            % Add nan to the value matrix if the function was not able to
            % solve for the implied volatility.
        catch ME
            implVolM(iStrike,jMat) = nan;
        end
    end
end

% Compute vector for moneyness
moneynessV = log(S0./KV);

% Plot volatility smile
figure;
plot(KV, implVolM(:,1),'r-','LineWidth',2);
hold on
plot(KV, implVolM(:,mMat),'b-','LineWidth',2);
hold off
title('Volatility Smile for Google Inc. (GOOG)');
xlabel('Strike');
ylabel('Volatility');
% Set the range of axis: [x_min x_max y_min y_max]
axis([min(KV)-10 max(KV)+10 min(min(implVolM))-0.1 max(max(implVolM))+0.1]);
% Add a legend
legend('Implied Volatility for short Maturity',...
    'Implied Volatility for long Maturity','location','NorthWest');
% Remove the box around the legend
legend('boxoff')
% Save the plot
saveas(gcf, 'VolatilitySmile.eps','psc2');
```



```
function implVol = ImpliedVolatility(putCall,S0,K,r,T,div,price)

options = optimset('fzero');
% TolX, 1e-7: tolerance level
options = optimset(options, 'TolX', 1e-7);
% func: function handle
func = @(sigma) BlackScholesDiff(putCall,S0,K,sigma,r,T,div,price);
% [0.001 1.5] interval for volatility
implVol = fzero(func, [0.001 1.5], options);
```

The phenomenon called *volatility smile* implies that options on the same underlying have different implied volatilities depending on the moneyness. More precisely deep in- and out-of-the-money options have higher implied volatilities than at-the-money options. This is due to the fact of the normality assumption of stock returns and the log-normal distribution of stock prices of the pricing model. An analysis of option data shows that since the crash in October 1987, investors allocate a higher probability to events taking place in the tails of the distribution (e.g. Derman and Kani (1994)). Therefore, options with low and high strike prices are priced more expensively than implied by the log-normal distribution. For longer maturities, a *volatility smirk* (reverse skew) is observed. Only in-the-money calls and out-of-the-money puts are priced more expensively than implied by the model. Explanations for the positive impact on these option prices are the high demand for in-the-money calls that are an almost perfect equivalent to holding the stock but with higher leverage as out-of-the-money puts, where investors are willing to pay a premium to have a downside protection.

```
% Plot volatility surface
figure;
surf(moneynessV,TV,implVolM');
title('Volatility Surface for Google Inc. (GOOG)');
xlabel('Moneyness');
ylabel('Maturity (in years)');
zlabel('Volatility');
% Set the y-axis to evolve backwards
set(gca,'YDir','reverse');
% Save the plot
saveas(gcf, 'VolatilitySurface.eps','psc2');
```



5 Known Dividend Yield in Binomial Trees

In the script, three ways to handle dividends in a binomial tree are considered. These are: (*i*) continuous dividends, (*ii*) discrete dividend yields, and (*iii*) discrete cash dividends. In the following exercise your task is to adjust the geometry of the stock price evolution to handle the case of *discrete dividend yields*.

Create a new function *BinomialTreeKnownYield.m* that adjusts the stock-price tree such that the function prices European as well as American options with a known discrete dividend yield. A known dividend yield means that the dividend is a constant percentage of the stock price and, hence the binomial tree is still recombining. Whereas for European options you only have to know the number of ex-dividend dates, for American options you need to take account of the exact exercise dates. Therefore, pass the dividend dates as a vector to an input variable `divDatesV`. For example, `divDatesV = [0.5 1 1.5]` means that dividends are paid out at dates 0.5, 1, and 1.5. Calculate the put option value given in Table 3.

Parameter	MATLAB
Option type	`putCall = 'P';`
Exercise type	`exerciseType = 'A';`
Stock price	`stockPrice = 50;`
Strike price	`strikePrice = 45;`
Volatility	`sigma = 0.4;`
Riskfree interest rate	`r = 0.05;`
Dividend dates	`divDatesV = [0.5 1 1.5];`
Dividend yield	`divV = [0.01 0.01 0.03];`
Time-to-maturity	`T = 2;`
Steps in binomial tree	`n = 6;`

Table 3: Parameters for put option with known dividend yield

```
function value = BinomialTreeKnownYield(putCall,exerciseType,S0,K,...
    sigma,r,divV,divDatesV,T,n)
%
% This function calculates the American or European option value with the
% CRR binomal tree model by considering a payout of a known dividend yield
% at particular steps in the tree.
%
% value = BinomialTreeKnownYield(putCall,exerciseType,S0,K,sigma,r,divV,...
%         divDatesV,T,n)
%
% Input:        putCall:        indicates the option type
%                               putCall = 'Call' for a call option
%                               putCall = 'Put' for a put
%                               option
%               exerciseType:   European ('E') or American ('A')
%                               exercise type
%               S:              Stock price
%               K:              Strike price
%               sigma:          Volatility
%               r:              riskfree rate
%               divV:           Vector of dividend yield
%               divDatesV:      vector of dividend dates
%               T:              Time to maturity
%               n:              Number of steps in the tree
%
% OUTPUT:       value:          option value
%
% EXAMPLE: value =...
%          BinomialTreeKnownYield('Put','A',50,45,0.4,0.05,[0.01 0.01 0.03]...
%          ,[0.5 1 1.5],2,6)

if ~isempty(divDatesV)
    if divDatesV(end) > T
        error('Last dividend is paid after expiration.');
    end
end

% Parameters for binomial tree
dt = T/n;
u  = exp(sigma*sqrt(dt));
d  = 1/u;
p  = (exp(r*dt)-d)/(u-d);
q  = 1-p;
disc = exp(-r*dt);
% number of dividend dates
nDates = length(divDatesV);
% Initialize matrices
stockM = zeros(n+1,n+1);
optionM = zeros(n+1,n+1);

% Set z parameter to calculate the option payoff depending on the
% selected option type (Call, Put)
switch putCall
    case 'Call'
        z = 1;
    case 'Put'
        z = -1;
    otherwise
        error('Check option type!');
end
```

```
% Create stock tree
switch exerciseType
    case 'E'
        % in the European case the current stock price is replaced by
        % S0*prod(ones(1,length(divV))-divV)
        stockM(1,1) = S0*prod(ones(1,length(divV))-divV);
        for j = 2:n+1
            for i = 1:j-1
                stockM(i,j) = stockM(i,j-1)*u;
            end
            stockM(i+1,j) = stockM(i,j-1)*d;
        end
    case 'A'
        % Build stock tree with known dividend yield
        stockM (1,1) = S0;
        iDate = 1;
        for j = 2:n+1
            if iDate <= nDates
                    % Adjust the stock value for dividend payment
                if (j-1)*dt <= divDatesV(iDate) && j*dt > divDatesV(iDate);
                    % If the period includes an ex-dividend date set
                    % divYield equal to divV.
                    divYield = divV(iDate);
                    iDate = iDate + 1;
                else
                    % If the period includes no ex-dividend date set
                    % divYield equal to zero.
                    divYield = 0;
                end
            else
                divYield = 0;
            end
            for i = 1:j-1
                stockM(i,j) = stockM(i,j-1)*(1-divYield)*u;
            end
            stockM(i+1,j) = stockM(i,j-1)*(1-divYield)*d;
        end
    otherwise
        error('Check exercise type!');
end

% Insert terminal values
optionM(:,end) = max(z*(stockM(:,end)-K),0);

% values call by working backward from time n-1 to time 0
switch exerciseType
    % European option
    case 'E'
        for j=n:-1:1;
            for i=j:-1:1;
                optionM(i,j) = disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1));
            end
        end;
    % American option
    case 'A'
        for j=n:-1:1;
            for i=j:-1:1;
                optionM(i,j) = max(z*(stockM(i,j)-K),disc*(p*optionM(i,j+1)+q*optionM(i+1,j+1)));
            end
        end
    otherwise
        error('Check exercise type!');
end
value = optionM(1,1);



BinomialTreeKnownYield('Put','A',50,45,0.4,0.05,[0.01 0.01 0.03],[0.5 1 1.5],2,6)
ans =
    7.0113
```

6 Rate of Convergence

As you know, the option value calculated by a binomial tree converges to the true option price by increasing the number of periods in the tree. A sequence of lattice converges with order $\beta > 0$ if there is a constant $c > 0$ such that $|\hat{p}_n - p| \leq c \cdot n^{-\beta}$, where $\hat{p}_n$ denotes the estimated model value, p is the true value and n denotes the number of steps. This can be written in *big-O notation* as $|\hat{p}_n - p| = O(n^{-\beta})$. To achieve a certain precision level, the constant c and the order β is of importance.
By setting $|\hat{p}_n - p|$ equal to ϵ and taking the log of $\epsilon \leq c \cdot n^{-\beta}$ we obtain

$$ln(e) = ln(c) - \beta ln(n), \tag{1}$$

i.e. the bounding function becomes a straight line with slope $-\beta$ and shift c. The higher the β the faster is the convergence of the CRR binomial tree to the true value. Show in this exercise that the order of convergence, β, is one for an European call option. To achieve this, you need (i) to plot the error, ϵ_n, against the number of steps, n, and (ii) to calculate the β in a OLS regression of $ln(\epsilon)$ on $ln(c)$ and $ln(n)$. The parameter values to be used are given in Table 4.

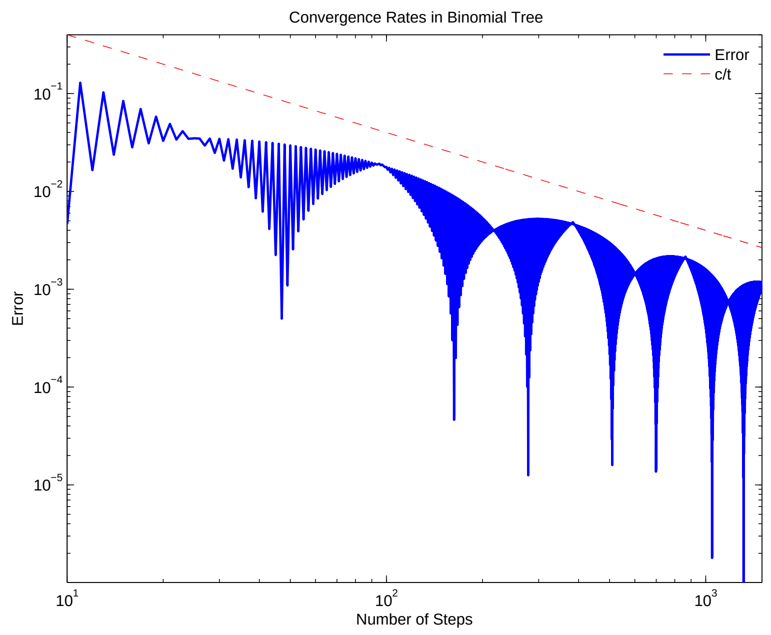
Parameter	MATLAB
Option type	`putCall = ’P’;`
Exercise type	`exerciseType = ’E’;`
Stock price	`stockPrice = 50;`
Strike price	`strikePrice = 48;`
Volatility	`sigma = 0.4;`
Riskfree interest rate	`r = 0.05;`
Dividend yield	`div = 0.00;`
Time-to-maturity	`T = 1;`
Steps in binomial tree	`stepsV = 10:1:1500;`
Constant	`c = 4`

Table 4: Parameters for rate of convergence

```
% m-file to estimate the order of the error convergence
clear all
clc
close all
% Parameter values and vector initialization
putCall = ’Put’;
exerciseType = ’E’;
S = 50;
K = 48;
sigma = 0.4;
r = 0.05;
div = 0;
T = 1;
stepsV = 10:1:1500;
lengthSteps = length(stepsV);
```

```
valueBinTree = nan(lengthSteps,1);
err = nan(lengthSteps,1);
% Value of constant
c = 4;
% Create vector constant for the OLS regression
const = ones(lengthSteps,1)*c;
% Calculate Black-Scholes option value
trueValueBS = BlackScholesPriceGreeks(putCall,S,K,sigma,r,T,div);
% Loop over the increasing number of steps
counter = 0;
for iSteps = stepsV
    counter = counter + 1;
    % Calculate binomial tree option value
    valueBinTree(counter) = ...
        BinomialTree(putCall,exerciseType,S,K,sigma,r,div,T,iSteps);
    % Calculate the error
    err(counter) = trueValueBS - valueBinTree(counter);
end

% Create plot
figure;
loglog(stepsV,abs(err),'linewidth',1.5);
hold on
loglog(stepsV,c./stepsV,'r--');
hold off
legend('Error','c/t');
legend('boxoff')
xlabel('Number of Steps');
ylabel('Error');
title('Convergence Rates in Binomial Tree');
axis tight
saveas(gca,'ConvergenceRateSteps.eps','psc2');
% x-variables of OLS-regression
x = [log(const) log(stepsV')];
% y-variable of OLS-regression
y = log(abs(err));
% Calculate beta of OLS regression
beta = (x'*x)\(x'*y);
% Print result for order of convergence to MATLAB command window
fprintf('The Beta equals %1.4f.\n',beta(2));
```

```
The Beta equals -1.0031.
```


Convergence Rates in Binomial Tree
Error
c/t
10^-1
10^-2
10^-3
10^-4
10^-5
Error
10^1
10^2
10^3
Number of Steps