

Matlab Course 2015-2016

Sachapon Tungsong

Queen Mary University of London

s.tungsong@qmul.ac.uk

July 18, 2016

OVERVIEW OF LECTURE 4

- Monte Carlo simulation - the basics
 - ① Tossing a coin - Bernoulli trials
 - ② Rolling a die - relevant distributions
- Generating a Brownian Motion
- Binomial option pricing
- Black-Scholes option pricing
- Pricing a European option
- Pricing an American option

MONTE CARLO BASICS

- Monte Carlo simulation does not rely on calculus to get an answer.
- Monte Carlo works for any random process.
- However, you never quite know what the mean should be exactly. You only know the approximate mean.
- We need a large sample size for a more accurate simulation result.

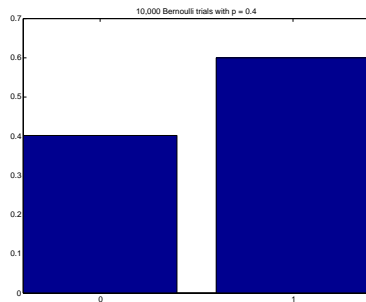
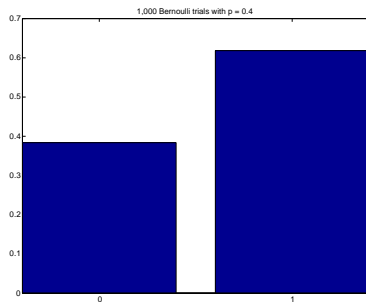
BERNOULLI TRIAL

- Generate 1,000 trials, in each of which you toss a weighted coin with the probability of landing head = 0.4.

```
1      % Bernoulli trial with a probability of heads being 0.4
2
3      N = 1000; % Number of trials
4      Y = rand(N,1); %Generate 1,000 random numbers in the interval [0,1]
5      s = (Y < 0.4);
6      p = sum(s);
7      q = sum(1-s);
8
9      [p,q] %Print out the results
10     bar([0,1], [p,q]/N) %Plot a bar graph
11     title('1,000 Bernoulli trials with p = 0.4')
```

- Now try generating 10,000 trials and compare the values of p and q. Which experiment gives value of p closer to the true value (0.4)?

BERNOULLI TRIAL



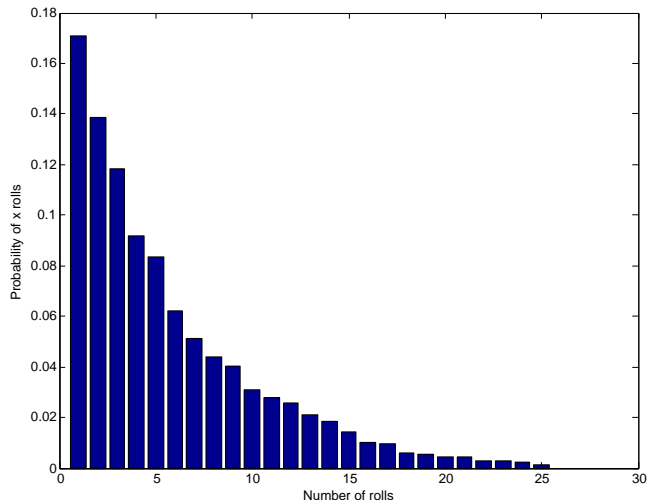
ROLLING A DIE

- Write simulation codes to find out the following:
 1. How many times do you have to roll a die to get a '6'?
 2. How many times do you have to roll a die until you get four occurrences of '3's?
 3. What is the sum of rolling three 6-sided dice?

GEOMETRIC DISTRIBUTION - SAMPLE CODE

```
1  N = 10000; %Sample size, i.e., number of experiments
2  M = 100; %Number of throws in each experiment
3  p = 1/6; %Probability of getting 6 rolling a 6-sided die
4  x = zeros(N,1); %Number of times you have to roll a die to get a 6.
5  for i = 1:N
6      n = 0; %Reset the total throws before getting a 6 for each experiment
7      for j = 1:M
8          if (rand < p)
9              n = n + 1;
10             if (n==1)
11                 x(i) = j;
12             end
13         end
14     end
15 end
16
17 sum(x==1)%Number of times you roll the die once to get a 6
18 sum(x==2)%Number of times you roll the die twice to get a 6
19
20 %Plotting, truncating graph at x =25
21 for i = 1:25
22     f(i) = sum(x==i);
23 end
24
25 bar (f/N)
26 xlabel('Number of rolls');
27 ylabel('Probability of x rolls');
```

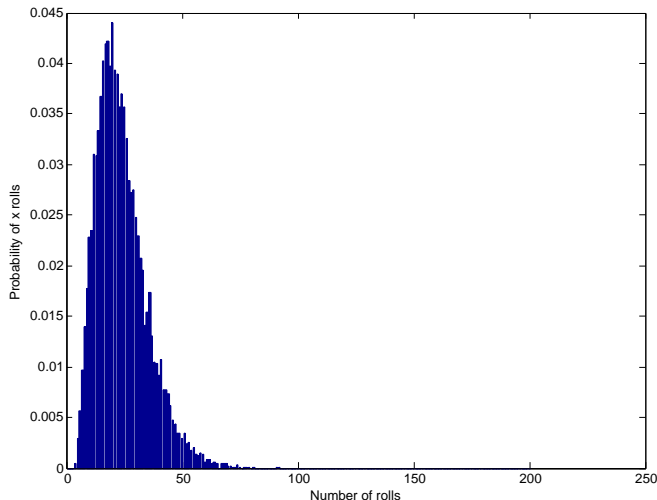
GEOMETRIC DISTRIBUTION



NEGATIVE BINOMIAL DISTRIBUTION - CODE

```
1      %The number of rolls until you get four '3's.
2
3      N = 10000; %Sample size, i.e., number of experiments
4      M = 200; %Number of throws in each experiment
5      p = 1/6; %Probability of getting 3 from rolling a 6-sided die
6      x = zeros(N,1);
7
8      for i = 1:N
9          n = 0; %Reset the total throws before getting a 3 for each experiment
10         x(i) = M; %Initialize the no. of rolls b4 getting a 3 to max no. of throws
11         for j = 1:M
12             if (rand < p)
13                 n = n + 1;
14                 if (n==4)
15                     x(i) = j;
16                 end
17             end
18         end
19     end
20
21
22     %Plotting
23     f = zeros(M,1);
24     for i = 1:M
25         f(i) = sum(x==i);
26     end
27
28     bar (f/N)
29     xlabel('Number of rolls');
30     ylabel('Probability of x rolls');
```

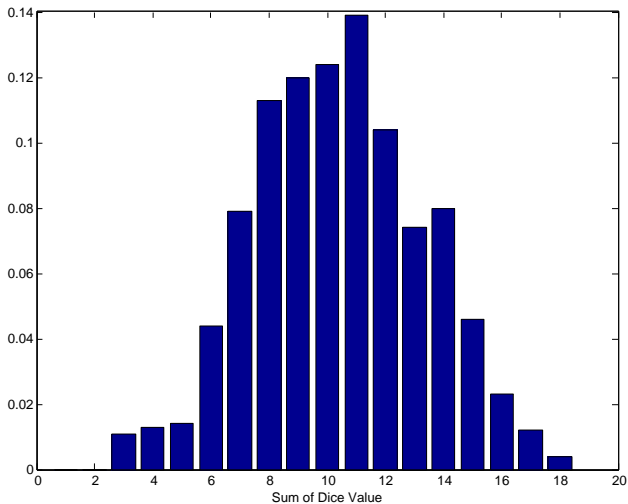
NEGATIVE BINOMIAL DISTRIBUTION



SUM OF ROLLING 3 DICE - CODE

```
2      %Sum of rolling three 6-sided dice
3
4      N = 1000; %Sample size
5      m = 3; %number of dice
6      s = 6; %Total number of sides of a die
7      x = zeros(N,1);
8
9      dice1 = floor(rand(N,1)*s+1);
10     dice2 = floor(rand(N,1)*s+1);
11     dice3 = floor(rand(N,1)*s+1);
12
13     x = dice1+dice2+dice3;
14
15     %Plotting the PDF
16     for i = 1:(m*s);
17         f(i) = sum(x==i);
18     end
19
20     bar(f/N)
21     xlabel('Sum of Dice Value');
22     ylabel('Probability');
```

SUM OF ROLLING 3 DICE



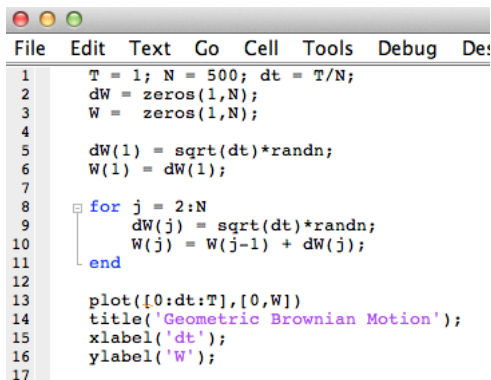
BROWNIAN MOTION

- Brownian motion, is described mathematically by the Wiener process W , a continuous-time stochastic process. It is a random walk, which is used to characterize a stock price path. Generating a discrete Brownian motion is equivalent to generating a random function $W : [0, \infty) \rightarrow R$ using the equation

$$W(kh) = \sqrt{h}(Z_1 + Z_2 + \dots + Z_k) \quad (1)$$

for $k = 1, 2, \dots$, $h > 0$ is a positive time step, and Z_1, \dots, Z_k are independent $N(0, 1)$ random variables.

GENERATING A BROWNIAN MOTION

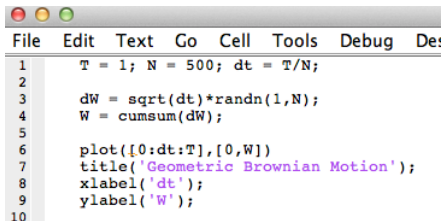


A screenshot of a MATLAB script editor window. The window has a title bar with three colored buttons (red, yellow, green) and a menu bar with options: File, Edit, Text, Go, Cell, Tools, Debug, and Desktop. The script is as follows:

```
1 T = 1; N = 500; dt = T/N;
2 dW = zeros(1,N);
3 W = zeros(1,N);
4
5 dW(1) = sqrt(dt)*randn;
6 W(1) = dW(1);
7
8 for j = 2:N
9     dW(j) = sqrt(dt)*randn;
10    W(j) = W(j-1) + dW(j);
11 end
12
13 plot([0:dt:T],[0,W])
14 title('Geometric Brownian Motion');
15 xlabel('dt');
16 ylabel('W');
17
```

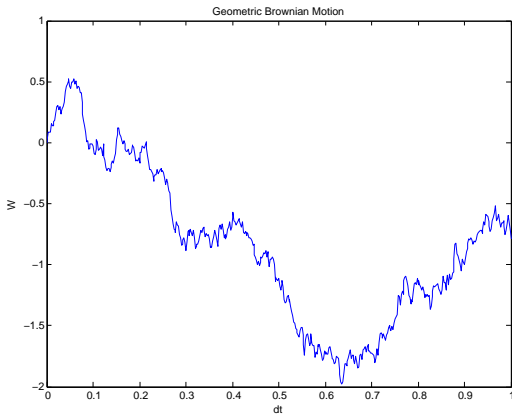
GENERATING A BROWNIAN MOTION

- We can improve our code by making it more concise as follows. The key functions we need to familiarize ourselves with is `randn` and `cumsum`. Because MATLAB is an interpreted language, for loops are inefficient. The function `cumsum` calculates the cumulative sum performed by the for loop in the previous code.



```
File Edit Text Go Cell Tools Debug De:  
1      T = 1; N = 500; dt = T/N;  
2  
3      dW = sqrt(dt)*randn(1,N);  
4      W = cumsum(dW);  
5  
6      plot([0:dt:T],[0,W])  
7      title('Geometric Brownian Motion');  
8      xlabel('dt');  
9      ylabel('W');  
10
```

A BROWNIAN MOTION PLOT



RAND and RANDN

```
x = rand
```

Returns a scalar whose value is drawn from the standard uniform distribution on the open interval $(0,1)$.

```
x = rand(m,n) or rand([m,n])
```

Returns returns an m-by-n matrix.

```
x = randn
```

Returns a scalar whose value is drawn from the standard normal distribution $N(0,1)$.

```
x = randn(m,n) or randn([m,n])
```

Returns returns an m-by-n matrix.

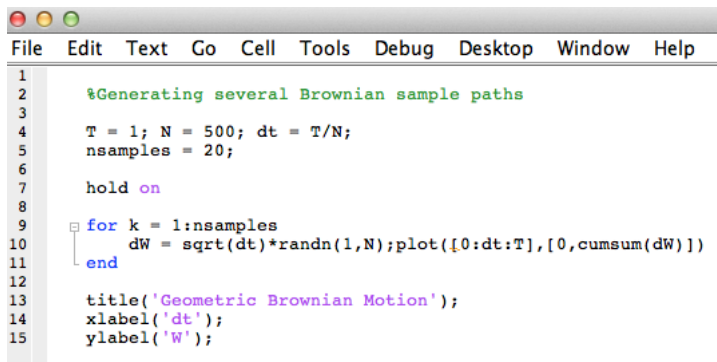
```
B = cumsum(A)
```

Returns the cumulative sum along different dimensions of an array. If A is a vector, `cumsum(A)` returns a vector containing the cumulative sum of the elements of A . If A is a matrix, `cumsum(A)` returns a matrix the same size as A containing the cumulative sums for each column of A .

```
B = cumsum(A,dim)
```

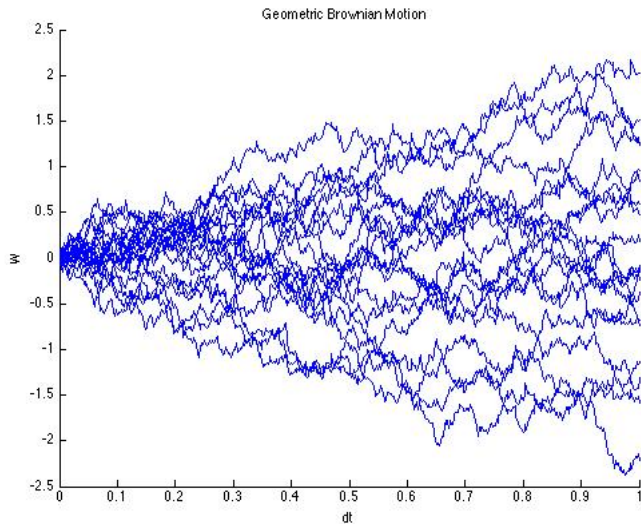
Returns the cumulative sum of the elements along the dimension of A specified by scalar `dim`. For example, `cumsum(A,1)` works along the first dimension (the columns).

GENERATING 20 BROWNIAN MOTIONS



```
1
2 %Generating several Brownian sample paths
3
4 T = 1; N = 500; dt = T/N;
5 nsamples = 20;
6
7 hold on
8
9 for k = 1:nsamples
10     dW = sqrt(dt)*randn(1,N); plot([0:dt:T],[0,cumsum(dW)])
11 end
12
13 title('Geometric Brownian Motion');
14 xlabel('dt');
15 ylabel('W');
```

20 BROWNIAN MOTION PATHS



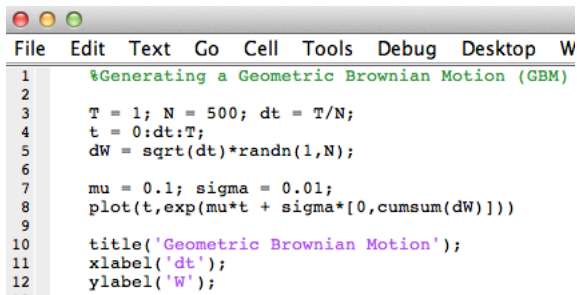
GEOMETRIC BROWNIAN MOTION

- Because a Brownian motion can be both positive and negative while a share price can never be negative, some modifications need to be made. Modern replacement of BM is to take the exponential. The result is called the Geometric Brownian Motion (GBM). Mathematically,

$$S(t) = S(0) \exp(\mu t + \sigma W(t)) \quad (2)$$

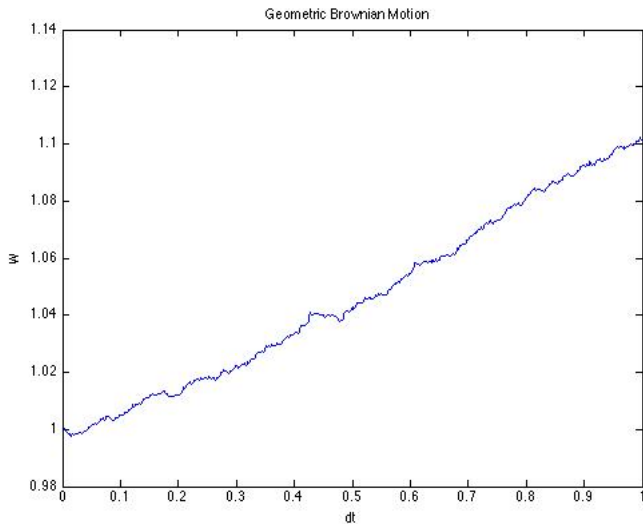
for $t > 0$, where $\mu \in \mathbb{R}$ is called the *drift* and σ is the *volatility*.

GENERATING A GBM

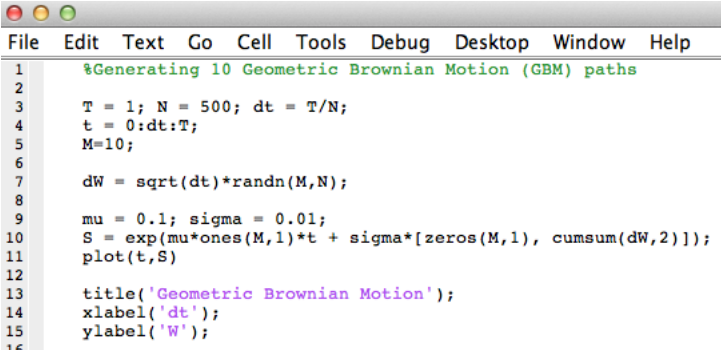


```
1 %Generating a Geometric Brownian Motion (GBM)
2
3 T = 1; N = 500; dt = T/N;
4 t = 0:dt:T;
5 dW = sqrt(dt)*randn(1,N);
6
7 mu = 0.1; sigma = 0.01;
8 plot(t,exp(mu*t + sigma*[0,cumsum(dW)]))
9
10 title('Geometric Brownian Motion');
11 xlabel('dt');
12 ylabel('W');
```

A GBM PLOT

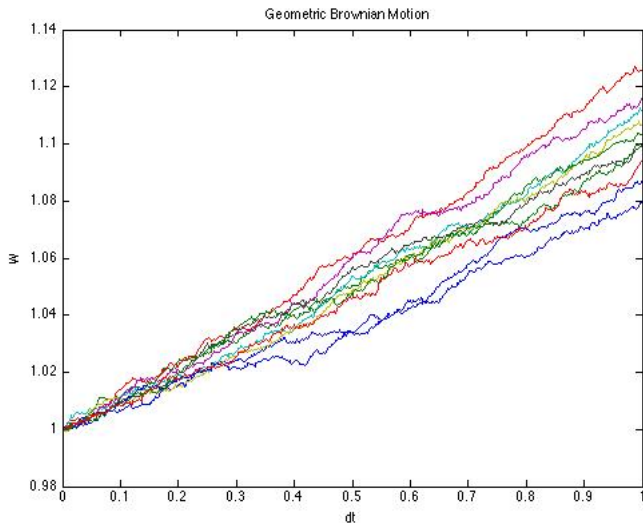


GENERATING 10 GBM PATHS



```
1 %Generating 10 Geometric Brownian Motion (GBM) paths
2
3 T = 1; N = 500; dt = T/N;
4 t = 0:dt:T;
5 M=10;
6
7 dW = sqrt(dt)*randn(M,N);
8
9 mu = 0.1; sigma = 0.01;
10 S = exp(mu*ones(M,1)*t + sigma*[zeros(M,1), cumsum(dW,2)]);
11 plot(t,S)
12
13 title('Geometric Brownian Motion');
14 xlabel('dt');
15 ylabel('W');
```

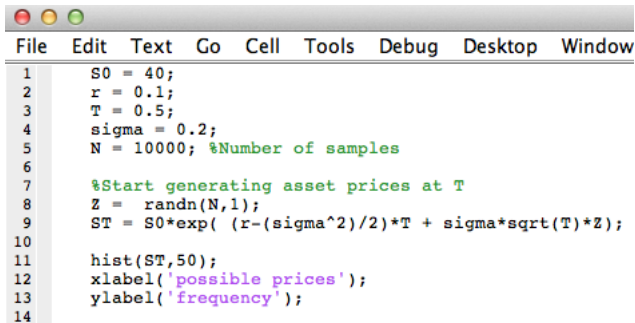

PLOT OF 10 GBM PATHS



GENERATING FUTURE ASSET PRICES

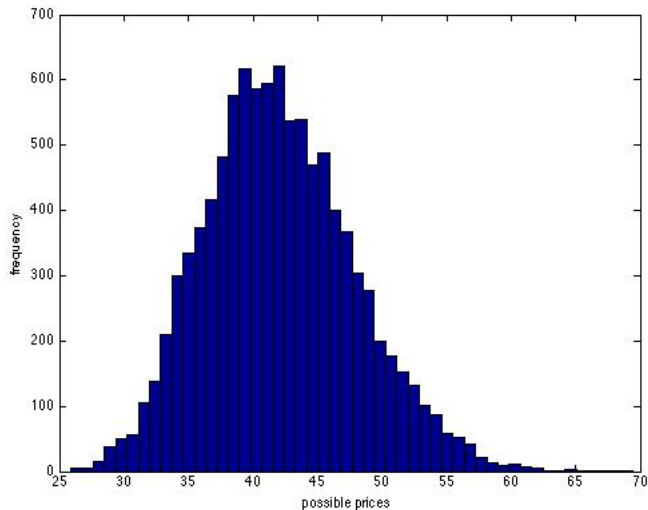
- Sometimes we may be interested ONLY in the final values at expiry $S(T)$. As such, there is no need to generate the sample paths. Using the mathematical fact that $W(T) \sim N(0, T)$ to generate share prices $S(T)$ at expiration time T

GENERATING FUTURE ASSET PRICES



```
File Edit Text Go Cell Tools Debug Desktop Window
1 S0 = 40;
2 r = 0.1;
3 T = 0.5;
4 sigma = 0.2;
5 N = 10000; %Number of samples
6
7 %Start generating asset prices at T
8 Z = randn(N,1);
9 ST = S0*exp( (r-(sigma^2)/2)*T + sigma*sqrt(T)*Z);
10
11 hist(ST,50);
12 xlabel('possible prices');
13 ylabel('frequency');
14
```

GENERATING FUTURE ASSET PRICES



OPTION PRICING

- European options
 - ① Binomial tree
 - ② Black-Scholes-Merton
- American options
 - ① Binomial tree

BINOMIAL PRICING

```
[AssetPrice,OptionValue] =  
binprice(S,K,r,t,h,Vol,Type,DivRate,Div,ExDiv);
```

Calculates the value (price) of a call or put option using binomial pricing method.

Arguments:

S: Underlying asset price. A scalar.

K: Option's exercise price. A scalar.

r: Risk-free interest rate. Decimal value.

t: Option's time until maturity in years. A scalar.

Increment: Time increment. A scalar.

BINOMIAL PRICING

Vol: Underlying asset's volatility. A scalar.

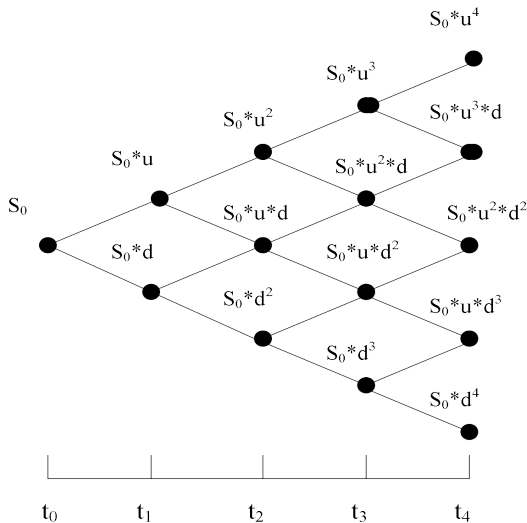
Type: Specifies whether the option is a call (Type = 1) or a put (Type = 0). A scalar.

DivRate: (Optional) The dividend rate, as a decimal fraction. A scalar. Default = 0.

Div: (Optional) The dividend payment at an ex-dividend date, ExDiv. A row vector. Default = 0.

ExDiv: (Optional) Ex-dividend date, specified in number of periods. A row vector. Default = 0. Set DivRate = 0 if you enter values for Div and ExDiv.

BINOMIAL TREE



BINOMIAL PRICING EXAMPLES

- Price a European call on a stock whose price is currently USD 100 and volatility is 20 percent per annum. The strike price of the option is USD 105. Option expires in 15 months. Risk-free interest rate is currently 4.5 percent per annum. Using 1-month time increment.
- Price a European call with the same features as the first option using 1-day time increment.
- Price a European call with the same features as the first option with dividend rate 2 percent.
- Price a European put with the same features as the first option.
- Compare the values of the above options.

BINOMIAL PRICING EXAMPLES

```
2      %Binomial option pricing
3 -    S = 100;
4 -    K = 105;
5 -    r = 0.045;
6 -    t = 15/12;
7 -    h = 1/12;
8 -    Vol = 0.2;
9 -    Type = 1; %Call option
10
11     [AssetPrice,OptionValue] = binprice(S,K,r,t,h,Vol,Type);
```

```
2      %Binomial option pricing
3 -    S = 100;
4 -    K = 105;
5 -    r = 0.045;
6 -    t = 15/12;
7 -    h = 1/12;
8 -    Vol = 0.2;
9 -    Type = 1; %Call option
10 -    DivRate = 0.02;
11
12     [AssetPrice,OptionValue] = binprice(S,K,r,t,h,Vol,Type,DivRate);
```

BLACK SCHOLES PRICING

```
[Call,Put] = blsprice(S,K,r,t,Vol,DivRate);
```

Calculates the value (price) of a call and put option using Black-Scholes pricing method. Any input argument may be a scalar, vector, or matrix. When a value is a scalar, that value is used to price all the options. If more than one input is a vector or matrix, the dimensions of all non-scalar inputs must be identical.

r , t , Vol , and $DivRate$ must be expressed in consistent units of time.

BLACK SCHOLES PRICING

Arguments:

S: Underlying asset's current price.

K: Option's exercise price.

r: Annualized, continuously compounded risk-free rate.

t: Option's time until maturity in years.

Vol: Annualized volatility of the asset, i.e., annualized standard deviation of the continuously compounded asset return.

DivRate: (Optional) Annualized, continuously compounded yield of the underlying asset over the life of the option, expressed as a decimal number. (Default = 0). For example, for options written on stock indices, DivRate could represent the dividend yield. For currency options, DivRate could be the foreign risk-free interest rate.

BLACK-SCHOLES PRICING EXAMPLES

- Price European put and call stock options that expire in three months with an exercise price of USD 95. Assume that the underlying stock pays no dividend, trades at USD 100, and has a volatility of 50 percent per annum. The risk-free rate is 10 percent per annum.
- Price European call and put options expiring in 4 months on a currency that is worth USD 0.8 and has a volatility of 12 percent per year. Strike is USD 0.7. Provided that the domestic and foreign risk-free interest rates are 6 percent and 8 percent, respectively.

BLACK-SCHOLES PRICING EXAMPLES

```
2      %Pricing stock options using Black-Scholes
3      S = 100;
4      K = 95;
5      r = 0.1;
6      t = 0.25;
7      Vol = 0.5;
8
9      [call,put] = blsprice(S,K,r,t,Vol);
```

```
1
2      %Pricing FX options using Black-Scholes
3      S = 0.8;
4      K = 0.7;
5      r = 0.06;
6      t = 4/12;
7      Vol = 0.12;
8      DivRate = 0.08;
9
10     [call,put] = blsprice(S,K,r,t,Vol,DivRate);
```

PRICING AN AMERICAN OPTION

- Higham (2002) published his implementation of Binomial pricing in MATLAB for both European and American options. Using the following relationships:

$$A = \frac{\exp(-r dt) + \exp((r + \sigma^2) dt)}{2} \quad (3)$$

$$u = A + \sqrt{A^2 - 1} \quad (4)$$

$$d = \frac{1}{u} \quad (5)$$

$$p = \frac{\exp(r dt) - d}{u - d} \quad (6)$$

AMERICAN OPTION PRICING EXAMPLES

- Price an American put option on a stock whose current price is USD 45, with a volatility of 20 percent per year. The option strike price is USD 45, maturity is 3 years. The current risk-free interest rate is 5 percent.
- Price an American call option on a stock whose current price is USD 100, with a volatility of 30 percent per year. The option strike price is USD 101, maturity is 2 years. The current risk-free interest rate is 5 percent.

AMERICAN PUT OPTION

```
2      %%% Pricing American put option using Binomial method %%%
3      %%% Option specifications %%%
4      S = 45;
5      K = 45;
6      r = 0.05;
7      sigma = 0.2;
8      T = 3;
9
10     %%% Binomial parameters %%%
11     N = 512; %Number of interval
12     dt = T/N; %Size of time step
13     A = 0.5*(exp(-r*dt) + exp((r + sigma^2)*dt));
14     up = A + sqrt(A^2 - 1);
15     down = 1/up;
16     p = (exp(r*dt) - down)/(up-down);
17
18     %%% Branching the tree %%%
19     BinTree = S*up.^((N:-1:0)') .* down.^((0:N)');
20
```

AMERICAN PUT OPTION

```
20
21     %%% Compute option value at maturity %%%
22     BinTree = max(K-BinTree,0);
23
24     %%% Compute expectations %%%
25     Expect = p*eye(N+1,N+1) + (1-p)*diag(ones(N,1),1);
26
27     for i = N:-1:1 %Going back one step at a time
28         DiscountedVal = Expect(1:i,1:i+1)*BinTree*exp(-r*dt);
29         %Discounting value one step back
30         shareprices = S*up.^((i-1:-1:0)').*down.^((0:i-1)');
31         %Share prices at time step
32         BinTree = max(DiscountedVal, K-shareprices);
33         %Comparing discounted vs intrinsic value for exercise decision
34     end
35
36     Option_price = BinTree;
37     display('The option price is'),display(Option_price)
38
```