

IFA/QFN VBA Tutorial

Notes prepared by Keith Wong

Chapter 6: Simulation

6-1: Numerical Approximation

Suppose we have $y = g(x)$, where g is differentiable. We want to find x for a given y . If g^{-1} is difficult to formulate, we can use some numerical methods to approximate x . One method is called the Newton's method, which is used to find a root of a function. The recursive relation is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

We start from choosing a guess value of x_0 . Then using the recursion several times we can get a pretty good approximation of x .

In our problem, to find x from $y = g(x)$, we simply find the root of $g(x) - y = 0$ by Newton's method. The equation becomes:

$$x_{n+1} = x_n - \frac{g(x_n) - y}{g'(x_n)}.$$

In the Newton's method, we can specify the number of significant figures to determine when the iteration should stop. During the iteration, the computer compares the existing value and the new calculated value. If they are equal under the said number of significant figures, the iteration should stop.

The problem of using the Newton's method is that we have to approximate the derivative of g for each iteration. We can approximate the derivative by $\frac{f(x_n + dx) - f(x_n)}{dx}$. Here are two assisting functions to calculate the order of magnitude of a number and the derivative of a function at a point.

```
Function magnitude(ByVal x As Double) As Integer
    'the order of magnitude of a number
    x = Abs(x)
    If x = 0 Then
        magnitude = 0
    Else
        magnitude = Int(Log(x) / Log(10#))
    End If
End Function
```

```
Function derivative(ByVal f As String, ByVal x As Double, Optional param) As Double
    'Approximate the derivative of f at point x
    Dim dx As Double
    If x = 0 Then dx = 2# ^ -30 Else dx = x / (2# ^ 30)

    If IsArray(param) Then
        derivative = (Application.Run(f, x + dx, param) - Application.Run(f, x, param)) / dx
    Else
        derivative = (Application.Run(f, x + dx) - Application.Run(f, x)) / dx
    End If
End Function
```

Note: The optional parameter `param` in `derivative` function is an array which, if provided, will be passed to the function `f`.

Here is an implementation of the Newton's method.

```

Function invFunc(ByVal y As Double, ByVal f As String, Optional ByVal sigFig As Integer = 6, _
    Optional ByVal initGuessValue As Double = 1, Optional param) As Double
    'Newton's method to approximate x from 0 = f(x) - y
    If sigFig <= 0 Then
        sigFig = 6
    ElseIf sigFig > 15 Then
        sigFig = 15
    End If

    Dim oldX As Double, newX As Double, df As Double
    oldX = initGuessValue
    Do
        If IsArray(param) Then
            df = Application.Run("derivative", f, oldX, param)
        Else
            df = Application.Run("derivative", f, oldX)
        End If

        If df = 0 Then Exit Do

        If IsArray(param) Then
            newX = oldX - (Application.Run(f, oldX, param) - y) / df
        Else
            newX = oldX - (Application.Run(f, oldX) - y) / df
        End If

        If magnitude(newX) < -100 Then
            invFunc = 0
            Exit Function
        End If

        Dim compX As Double
        compX = Round(newX * 10# ^ (sigFig - magnitude(newX)))

        If compX = Round(oldX * 10# ^ (sigFig - magnitude(newX))) Then
            invFunc = compX / 10# ^ (sigFig - magnitude(newX))
            Exit Function
        End If

        oldX = newX
    Loop
    invFunc = Round(newX * 10# ^ (sigFig - magnitude(newX))) / _
        10# ^ (sigFig - magnitude(newX))
End Function

```

An improved method (based on code complexity and speed) is the Secant method. This method approximates the derivative of f by $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$ instead. So the recursive relation is:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \cdot f(x_n),$$

or

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{g(x_n) - g(x_{n-1})} \cdot [g(x_n) - y].$$

We can see that we need two initial guess values. Here is an implementation of the Secant method.

```
Function invFuncS(ByVal y As Double, ByVal f As String, Optional ByVal sigFig As Integer = 6, _
    Optional ByVal initGuessValue1 As Double = 0.01, Optional ByVal initGuessValue2 As Double _
    = 1, Optional param) As Double
'Secant method to approximate x from  $\theta = f(x) - y$ 
If sigFig <= 0 Then
    sigFig = 6
ElseIf sigFig > 15 Then
    sigFig = 15
End If

Dim oldX As Double, olderX As Double, newX As Double
olderX = initGuessValue1
oldX = initGuessValue2

Do
    If IsArray(param) Then
        newX = oldX - (oldX - olderX) / (Application.Run(f, oldX, param) - Application.Run _
            (f, olderX, param)) * (Application.Run(f, oldX, param) - y)
    Else
        newX = oldX - (oldX - olderX) / (Application.Run(f, oldX) - Application.Run(f, olderX)) _
            * (Application.Run(f, oldX) - y)
    End If

    If magnitude(newX) < -100 Then
        invFuncS = 0
        Exit Function
    End If

    Dim compX As Double
    compX = Round(newX * 10# ^ (sigFig - magnitude(newX)))

    If compX = Round(oldX * 10# ^ (sigFig - magnitude(newX))) Then
        invFuncS = compX / 10# ^ (sigFig - magnitude(newX))
        Exit Function
    End If
    olderX = oldX
    oldX = newX
Loop
End Function
```

Another method is to use a very hidden sheet (5-7-5) with GoalSeek (5-8-44) or Solver functions to do the approximation. This involves creating Excel formulas for your problem, putting them in some cells of the very hidden sheet, and applying approximation functions. For solver usage with VBA, visit <http://peltiertech.com/Excel/SolverVBA.html>. You will probably record a macro to generate codes for solver.

Example of using the Newton's method and Secant method functions.

```
Function ch6_1_func(x)
    func = x ^ 2 + 2 * x + 1
End Function

Sub ch6_1()
    MsgBox invFunc(0, "ch6_1_func", 12) & vbCrLf & invFuncS(0, "ch6_1_func", 12)
End Sub
```

You can see that the answers are not accurate if the derivative of the f at the point of answer is zero.

Let's try! Corner

Create a function to calculate the implied volatility of a European Call under the Black-Scholes model. The implied volatility is the volatility (σ) calculated from a given call price (See Exercise 2 question 5). You will probably create another function to facilitate the input to `invFunc` or `invFuncS`.

```
Function Black_Scholes_Call_Implied_Vol(ByVal S0 As Currency, ByVal T As Double,  
ByVal K As Currency, ByVal r As Double, ByVal delta As Double, ByVal price As Double)  
As Double
```

6-2: Simulating Continuous Random Variables

We can use the inverse transformation method to obtain a realized value of a random variable X .

First we generate a $U(0, 1)$ random number. ($U(0,1)$ stands for a uniform distribution between 0 and 1) Treating the number as the probability for $X < x$, we can obtain x by

$$X = F^{-1}[U(0,1)],$$

where F^{-1} is the inverse of the cumulative distribution function of X .

Most of the distribution functions (including their inverse) can be found in the `WorksheetFunction` class.

To generate a $U(0, 1)$ random number, we simply do the following:

```
'generate a U(0,1) random variable  
Dim U As Double  
Randomize  
U = Rnd
```

Then we can use the inverse transformation method to get a simulated value of the desired random variable. For instance, we want 10 simulated values of a standard normal random variable. We simply do this:

```
Dim U As Double, strArray(9) As String, i As Integer
Randomize

'simulate 10 standard normal random variable
For i = 0 To 9
    'generate a U(0,1) random variable
    U = Rnd
    'calculate the simulated value of a normal random variable
    strArray(i) = Round(WorksheetFunction.NormSInv(U), 6)
Next i

MsgBox Join(strArray, vbCrLf)
```

One note is that if you want to simulate a large number of values, your computer will hang for a long time. To prevent this, insert a `DoEvents` statement somewhere in your loop. This statement will free some computing resources to other programs in your system. Also, disable automatic calculation and screen updating also helps. For example:

```
Application.Calculation = xlCalculationManual
Application.ScreenUpdating = False
For i = 0 To 9999
    'generate a U(0,1) random variable
    U = Rnd
    'calculate the simulated value of a normal random variable
    strArray(i) = Round(WorksheetFunction.NormSInv(U), 6)
    DoEvents
Next i
Application.Calculation = xlCalculationAutomatic
Application.ScreenUpdating = True
```

Another note: Instead of generating the random numbers directly to the cells, consider creating a two-dimensional array and put the random numbers to the array. After all the numbers are simulated, assign the array to the range of cells. This may be faster.

Let's try! Corner

Add a sheet to an existing workbook. Simulate 10,000 exponential random variables of mean 1. Output them to the sheet.

Let's try! Corner

Given

$$F(x) = \begin{cases} 0.5x & \text{for } 0 \leq x < 1 \\ 0.5 + 0.25x & \text{for } 1 \leq x < 2 \end{cases}$$

Simulate 5,000 values of this distribution to a sheet.

6-3: Simulating Discrete Random Variables

Since the functions are discrete, we cannot use the methods described in 6-1 to find the result of the inverse transformation. We shall try to formulate another algorithm to determine the simulated value.

For example, to simulate a Bernoulli random variable with $p = 0.4$, we have $F(0) = 0.6$ and $F(1) = 1$. If a $U(0, 1)$ random number is 0.2, we will assign the simulated value to be 0. If the $U(0, 1)$ value is 0.8, the simulated result will be 1. Generally, for $0 \leq U < 0.6$, the simulated value is 0, and for $0.6 \leq U < 1$, the simulated value is 1.

Therefore, an algorithm to determine the simulated value is:

```
x = 0
Do Until F(x-1) ≤ U < F(x)
  x = x + 1
Loop
return x
```

An implementation:

```
Function discDistInv(ByVal u As Double, ByVal pmf As String, Optional param) As Long
  'determine the inverse of a discrete distribution
  Dim x As Long, oldFx As Double, newFx As Double
  x = 0
  oldFx = 0
  Do
    If IsArray(param) Then
      newFx = oldFx + Application.Run(pmf, x, param)
    Else
      newFx = oldFx + Application.Run(pmf, x)
    End If

    If oldFx <= u And u < newFx Then
      discDistInv = x
      Exit Function
    End If

    x = x + 1
    oldFx = newFx
  Loop
End Function
```

Let's try! Corner

Simulate 10,000 values of a B(4, 0.6) random variable (B(4, 0.6) stands for a binomial distribution with $n = 4$ and $p = 0.6$).

6-4: Simulate a Compound Random Variable

A compound random variable in the form $S = \sum_{i=1}^N X_i$, where N is a frequency random variable and each X_i is a severity random variable. To simulate this kind of variable, consider simulating N first. Given a simulated N , we can simulate X_i .

Let's try! Corner

We are trying to simulate a stop-loss insurance.

Simulate $S = \sum_{i=1}^N X_i$, where N follows a Poisson distribution with mean 8 and each X_i follows a Pareto

distribution with $\alpha = 2$ and $\theta = 5,000$. (The pdf and cdf are in the Exam C table.) For each simulated value, calculate $\min(S, 10,000)$ (the amount which the insurer pays) and $\max(S - 10,000, 0)$ (the amount which the reinsurer pays).

6-5: Monte Carlo Simulation

To simulate expected values of a random variable, we can use the law of large numbers on a group of simulated values.

1. Simulate r independent and identically distributed random variables Y_1, Y_2, \dots, Y_r .
2. Calculate the corresponding $g(Y_1), g(Y_2), \dots, g(Y_r)$.
3. $\bar{g}(r) = \frac{g(Y_1) + g(Y_2) + \dots + g(Y_r)}{r}$ gives an estimate of $E[g(Y)]$.

To simulate	Use the following for $g(y)$
Mean	Y
k th moment	y^k
cdf $F_Y(t)$	$I(y \leq t)$ [This means if $y \leq t$ we have 1, otherwise 0]
call option	$(y - K)_+$ [This means $\max(y - K, 0)$]

For example, to simulate the mean of a random variable X , we simulate X 10000 times to get $X_1, X_2, \dots, X_{10000}$. Then the mean is estimated by (the sum of the simulated values / 10000). To simulate the variance, we

use $\frac{1}{r-1} \sum_{k=1}^r (Y_k - \bar{Y})^2$. To simulate $F_Y(t)$, first calculate $I(Y_i \leq t)$ and take the mean of these values.

Let's try! Corner

Simulate the mean and variance of a Pareto distribution with $\alpha = 3$ and $\theta = 500$ using 50,000 samples. Compare your result to the theoretical values. You will find formulas in the Exam C table.

Let's try! Corner

Insurance for a city's snow removal costs covers four winter months.

1. There is a deductible of 10,000 per month.
2. The insurer assumes that the city's monthly costs are independent and normally distributed with mean 15,000 and standard deviation 2,000.

Simulate the insurer's yearly claim cost using 10,000 runs of simulation for each month.

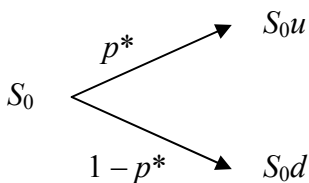
6-6: Simulating Stock Prices

6-6-1: Stock Prices under the Binominal Tree

Consider a binomial model for a stock price which pays dividends at rate δ :

- At time 0, the price is S_0
- At time h , its price S_h can be equal to S_0u or S_0d ($u > 1$; $d < 1$)
- The continuously compounded risk-free interest rate is r

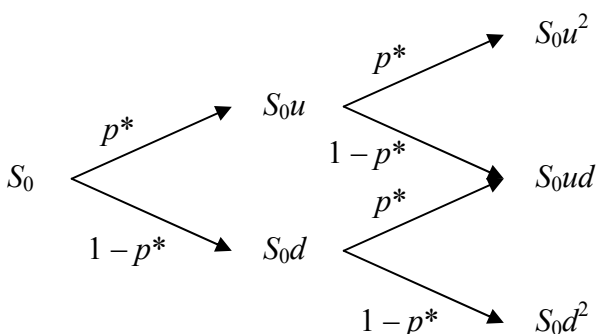
We can then formulate the following one-step binomial tree:



p^* is the probability of going up for a period h under the risk-neutral measure. We can calculate p^* by the formula

$$p^* = \frac{e^{(r-\delta)h} - d}{u - d}.$$

Extending to a two-step tree is easy:



To simulate stock price under a binomial model, we just simulate the path of a stock price.

Here is an implementation of simulating a risk-neutral stock price.

```
Function simBinomialStock(ByVal S0 As Currency, ByVal u As Double, ByVal d As Double, _
    ByVal r As Double, Optional ByVal delta As Double = 0, Optional ByVal step As Integer = 1, _
    Optional ByVal h As Double = 1) As Currency
    'S0 is the initial stock price
    'u is the ratio of the new stock price to the initial stock price if going up
    'd is the ratio of the new stock price to the initial stock price if going down
    'r is the continuous interest rate
    'delta is the continuous dividend rate
    'step is the number of steps of the binomial tree
    'h is the time period of each step
    Dim param(1), nSuccess As Integer

    'Get the simulated binomial variable
    param(0) = step                    'n = step
    param(1) = (Exp((r - delta) * h) - d) / (u - d) 'p = p*
    nSuccess = discDistInv(Rnd, "simBinomialStock_func", param)

    'Calculate the stock price
    simBinomialStock = S0 * u ^ nSuccess * d ^ (step - nSuccess)
End Function

Private Function simBinomialStock_func(ByVal x As Integer, param As Variant) As Double
    'return a binomial pmf
    simBinomialStock_func = WorksheetFunction.BinomDist(x, param(0), param(1), False)
End Function
```

Let's try! Corner

Simulate 5000 risk-neutral stock prices after 1 year and calculate the mean of these prices using a 100-step binomial tree ($h = 0.01$), with $S_0 = 100$, $u = 1.002$, $d = 0.998$, $r = 0.05$, $\delta = 0.02$. Do this 10 times and compare the 10 numbers you get.

If the stock follows a Geometric Brownian Motion (GBM), instead of u and d , we are provided with a volatility (σ). We can approximate the stock price by an n -step binomial tree. There are several methods to link σ to u and d . One method is to use the following formulas:

$$u = \exp[(r - \delta)h + \sigma\sqrt{h}]$$

$$d = \exp[(r - \delta)h - \sigma\sqrt{h}]$$

Therefore we can see that it is very easy to implement this kind of simulation.

Let's try! Corner

Simulate 5000 risk-neutral stock prices after 1 year and calculate the mean of these prices using a 100-step binomial tree ($h = 0.01$), with $S_0 = 100$, $\sigma = 0.2$, $r = 0.05$, $\delta = 0.02$. Do this 10 times and compare the 10 numbers you get.

6-6-2: Stock Prices following Geometric Brownian Motion

If a stock price follows GBM, it will have the following stochastic differential equation (SDE) under the risk-neutral measure:

$$dS(t) = rdt + \sigma dZ(t).$$

Thus the formula for the stock price:

$$S(t) = S(0) \exp[(r - \delta - 0.5\sigma^2)t + \sigma Z(t)],$$

where $Z(t)$ follows a normal distribution with mean 0 and variance t .

Thus to simulate a risk-neutral stock price, first simulate a $N(0, T)$ random variable. Next plug the simulated $N(0, T)$ into the stock price formula.

```
Function simGBMStock(ByVal S0 As Currency, ByVal sigma As Double, _
    ByVal r As Double, Optional ByVal delta As Double = 0, _
    Optional ByVal t As Double = 1) As Currency
'S0 is the initial stock price
'sigma is the volatility of the stock price
'r is the continuous interest rate
'delta is the continuous dividend rate
't is the elapsed time
simGBMStock = S0 * Exp((r - delta - 0.5 * sigma ^ 2) * t + _
    sigma * WorksheetFunction.NormInv(Rnd, 0, Sqr(t)))
End Function
```

Let's try! Corner

Simulate 5000 risk-neutral stock prices after 1 year and calculate the mean of these prices using a GBM simulation method, with $S_0 = 100$, $\sigma = 0.2$, $r = 0.05$, $\delta = 0.02$. Do this 10 times and compare the 10 numbers you get.

6-7: Simulating European Options

The key to using simulation to price any financial derivatives is to use the risk-neutral pricing method. If there is a derivative whose payoff is paid at maturity T , the price of the derivative is given by

$$V = e^{-rT} E^* [\text{payoff at } T],$$

where E^* is the expected value under the risk-neutral measure.

To estimate $E^*[\text{payoff at } T]$, we use the sample mean of a group of simulated values of payoff at T under risk-neutral measure.

For example, to price a European Call, we can simulate 10,000 risk-neutral stock prices. For each price S_i , calculate $(S_i - K)_+$, take the mean of the 10,000 values, and finally multiply the mean with e^{-rT} .

Let's try! Corner

Consider a non-dividend-paying stock with $S(0) = \$41$ and $\sigma = 30\%$.

- 1 By construct a binomial tree with 100 time steps, simulate the price of a \$40-strike European call on the stock maturing after 1 year. Use $r = 8\%$ and 10,000 simulations.
- 2 Redo (1) with a GBM model.
- 3 Compare the result of (1) and (2) with the theoretical price.

6-8: Simulating Path-Dependent Options

A path-dependent European option has a terminal payoff that does not only depend on the terminal stock price $S(T)$, but also on some immediate values of the stock before expiration.

One example is the Asian option. For example, for an arithmetic average-strike European call with 1 year maturity and observation period of 4 months, the terminal payoff is

$$\left[\frac{S(1/3) + S(2/3) + S(1)}{3} - K \right]_+.$$

To simulate a single terminal payoff, we need to simulate $S(1/3)$, $S(2/3)$ and $S(1)$. In the Black-Scholes model, this can be achieved by

$$S(t_2) = S(t_1) \exp[(r - \delta - 0.5\sigma^2)(t_2 - t_1) + \sigma(Z(t_2) - Z(t_1))]$$

and that $Z(t_2) - Z(t_1) \sim N(0, t_2 - t_1)$ and is independent of all events happened on or before t_1 .

To simulate $S(1/3)$, $S(2/3)$ and $S(1)$, we can first simulate $S(1/3)$ with $S(0)$ and a $N(0, 1/3)$ simulated value. Then we can proceed to simulate $S(2/3)$ with $S(1/3)$ and another $N(0, 1/3)$ simulated value. Finally $S(1)$ can be simulated with similar methods.

Let's try! Corner

Simulate the price of an arithmetic average-strike European call with strike price 40, 1 year maturity and observation period of 4 months, using the GBM model on a non-dividend-paying stock with $S(0) = \$41$ and $\sigma = 30\%$. Run your simulation 10,000 times to get the average.

6-9: Simulating Basket Options

A basket option is an option written on a portfolio of assets. Let S_1, S_2, \dots, S_n be n stocks in a portfolio, and let w_1, w_2, \dots, w_n be their weights, the price of the portfolio at time t is given by:

$$V(t) = \sum_{i=1}^n S_i(t)w_i .$$

We have the payoff for a K -strike call option on the portfolio:

$$\text{payoff} = [V(T) - K]_+$$

To price this kind of options, we need to simulate the risk-neutral price of the portfolio. However, normally S_1, S_2, \dots, S_n will be correlated in some way, with a correlation matrix

$$\Sigma = \begin{pmatrix} 1 & \rho_{12} & \dots & \rho_{1n} \\ \rho_{21} & 1 & \dots & \rho_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{n1} & \rho_{n2} & \dots & 1 \end{pmatrix} .$$

To simulate n correlated stocks, we use the following algorithm:

- 1 Simulate n uncorrelated standard normal random variables, ε_k 's.
- 2 Decompose the correlation matrix to $\mathbf{A}\mathbf{A}^T$, where \mathbf{A} is a lower triangular matrix. (This is known as the Cholesky decomposition on the correlation matrix.)
- 3 Calculate the n correlated standard normal random variables using the formula

$$Z_k = \sum_{i=1}^k a_{ki} \varepsilon_i, \text{ where } a_{ki} \text{ is the } k\text{th row, } i\text{th column entry of the matrix } \mathbf{A}.$$

- 4 Obtain the correlated $N(0, t)$ random variables by

$$Z_k(t) = \sqrt{t} \cdot Z_k.$$

- 5 Calculate the stock prices respectively.

Here is an implementation:

```
Function simCorrGBMStock(S0() As Currency, delta() As Double, sigma() As Double, _
    corr() As Double, ByVal r As Double, Optional ByVal T As Double = 1) As Currency()
    'S0 is a vector of initial stock prices
    'delta is a vector of dividend rates
    'sigma is a vector of volatilities
    'corr is the correlation matrix
    'r is the risk-free rate
    'T is the time elapsed
    Dim ST() As Currency, epsilon() As Double, A() As Double, Z() As Double, n As Integer

    n = UBound(S0) - LBound(S0) + 1
    ReDim ST(n - 1)
    ReDim epsilon(n - 1)
    ReDim A(n - 1, n - 1)
    ReDim Z(n - 1)

    'Simulate n uncorrelated standard normal
    Dim K As Integer, i As Integer
    For K = 0 To n - 1
        epsilon(K) = WorksheetFunction.NormSInv(Rnd)
    Next K

    'Calculate the Cholesky decomposition
    A = Chol(corr)

    'Calculate the n correlated N(0,t)
    For K = 0 To n - 1
        For i = 0 To K
            Z(K) = A(K, i) * epsilon(i) * Sqr(T)
        Next i
    Next K

    'Calculate the stock prices
    For K = 0 To n - 1
        ST(K) = S0(K) * Exp((r - delta(K) - 0.5 * sigma(K) ^ 2) * T + _
            delta(K) * Z(K))
    Next K

    simCorrGBMStock = ST
End Function
```

The Chol function is used to do the Cholesky decomposition:

```
Function Chol(cov)
' Cholesky decomposition
' Cov = L x L matrix of covariances
Dim SumSq As Double, SumPr As Double, h As Integer
Dim i As Integer, j As Integer, A() As Double, L As Integer
L = UBound(cov) - LBound(cov) + 1
ReDim A(L - 1, L - 1)

For i = 0 To L - 1
    SumSq = 0
    For h = 0 To i - 1
        SumSq = SumSq + A(i, h) * A(i, h)
    Next h
    A(i, i) = Sqr(cov(i, i) - SumSq)
    For j = i To L - 1
        SumPr = 0
        For h = 0 To i - 1
            SumPr = SumPr + A(i, h) * A(j, h)
        Next h
        A(j, i) = (cov(i, j) - SumPr) / A(i, i)
    Next j
Next i
Chol = A
End Function
```

Let's try! Corner

Write a function to value the price of a European Call basket options using 5,000 simulations. Test your function with $r = 0.1$, $K = 50$, $T = 1$ and four stocks of equal weight:

Stock 1: $S_0 = 70$, $\delta = 0.02$, $\sigma = 0.25$

Stock 2: $S_0 = 40$, $\delta = 0$, $\sigma = 0.3$, $\rho_{12} = 0.1$

Stock 3: $S_0 = 50$, $\delta = 0.02$, $\sigma = 0.35$, $\rho_{13} = -0.1$, $\rho_{23} = 0.3$

Stock 4: $S_0 = 45$, $\delta = 0$, $\sigma = 0.3$, $\rho_{14} = -0.3$, $\rho_{24} = 0.2$, $\rho_{34} = 0.6$

6-10: Simulating American Options

There are several methods to simulate an American Option. One method is the Longstaff-Schwartz Least-Square approach.

The concern in valuating an American option is that at every time point, we are choosing whether or not we exercise the option or hold the option. In the Longstaff-Schwartz approach,

- the value of exercising is the payoff that one can get from early exercise;
- the value of continuing (that is, not exercising the option) is obtained from least-square regression.

It is quite difficult to formulate the VBA codes for this approach. You can refer to the corresponding section in the lecture notes of FIN4250 and the sample code for an explanation and implementation of this method.

Exercise 6

1. Write a function to calculate the implied volatility of a European Put option.
2. Simulate 1,000 lognormal random variables with $\mu = 0.15$ and $\sigma = 0.2$.
3. Simulate 1,000 compound random variables, with the frequency random variable following a negative binomial distribution with $r = 5$ and $p = 0.4$, and each of the severity random variable following a Gamma distribution with $\alpha = 2$ and $\theta (= \beta) = 500$.
4. Simulate the price and standard deviation of a European Put with $S_0 = 50$, $r = 0.08$, $\delta = 0.05$, $\sigma = 0.3$, $K = 55$, $T = 0.75$, using 10,000 simulations.
5. Repeat question 4 if the put is an arithmetic average-strike European one with an observation period of 3 months.
6. Redo the Let's Try! Corner of section 6-9 to calculate the price of a European put.