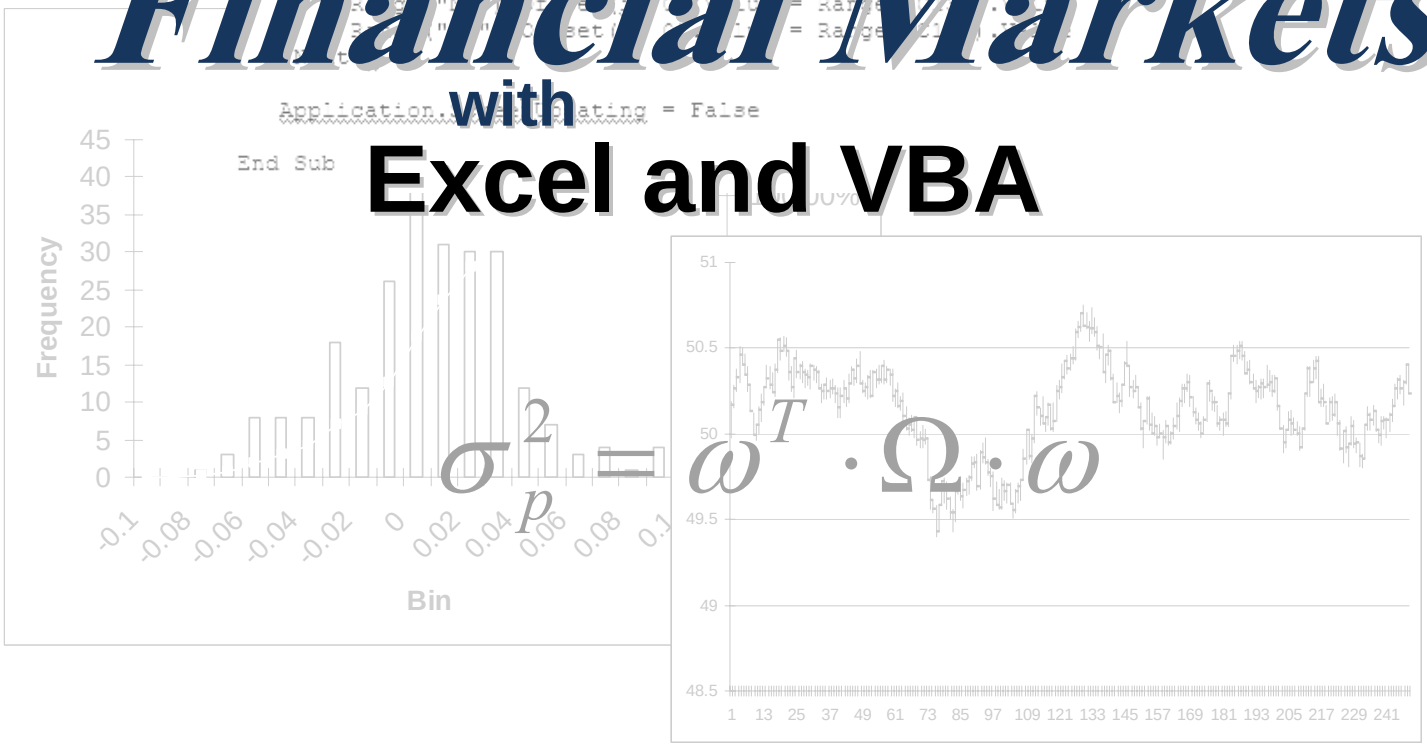
$$C = S \cdot N(d_1) - X \cdot e^{-rt} \cdot N(d_2)$$

$$\mu_{2|1} = \mu_2 + \rho \frac{\sigma_2}{\sigma_1}(z_1 - \mu_1)$$

Modeling *Financial Markets*

with Excel and VBA



$$\sigma_p^2 = \omega^T \cdot \Omega \cdot \omega$$

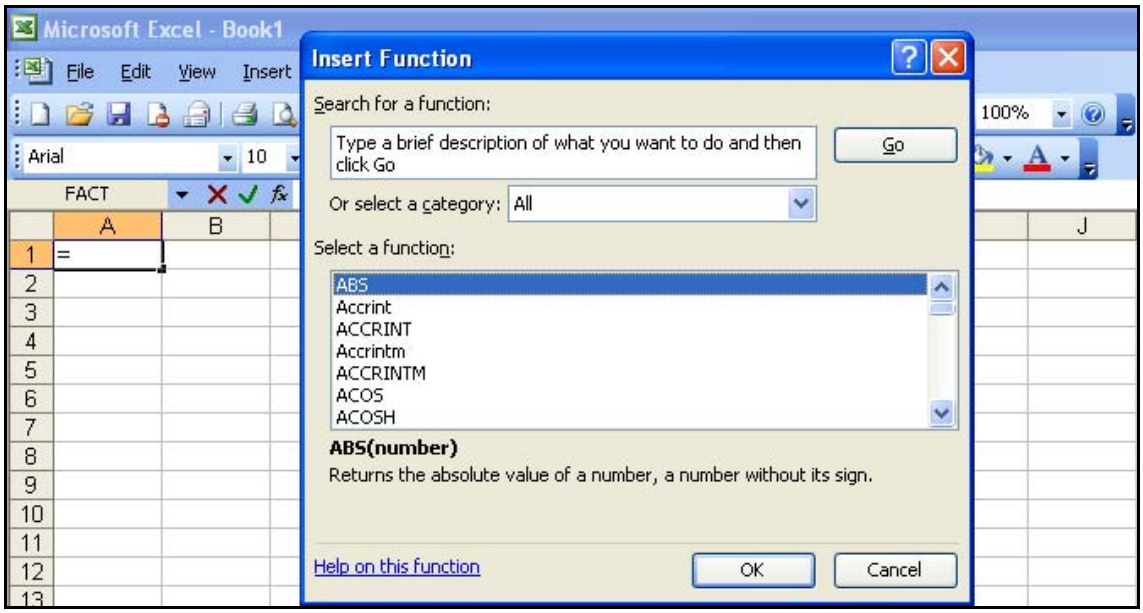
Ben Van Vliet

May 9 , 2011

I. GETTING STARTED WITH EXCEL

Excel has a wealth of tools and functionality to facilitate financial modeling. Too much, in fact, to cover in this text. You should become comfortable with investigating the Excel visual development environment on your own. Indeed, this is the only way to really learn Excel. Nevertheless, this study guide should help is pointing you toward those capabilities of Excel that are widely used in the financial industry.

The primary functionality of Excel is its library of hundreds of built-in functions. To view a list of these functions, click on the f_x icon.



Highlighting a function name will show a brief description, the parameter list and provide a link to the associated help file. You should take some time to familiarize yourself with Excel's help files. They provide a wealth of information, including function formulas, to speed development.

A. Things You Should Be Familiar With

An Excel **spreadsheet application** contains a **workbook**, which is a collection of **worksheets**. There are many, many built-in functions in Excel. Some of them are related to financial topics, for example, IRR(), DURATION(), NPV(), PMT(), and ACCINT().

BE CAREFUL!

You should be careful when using any Excel function as the formula used for calculation may be different from what you expect. ***Never assume that a function calculates something the way you think it does.*** *Always verify the formula using the help files.*

Spreadsheet errors are pervasive*. As Dr. Ray Panko points out, "consistent with research on human error rates in other cognitive tasks, laboratory studies and field examinations of real-world spreadsheets have confirmed that developers make uncorrected errors in 2%-5% of all formulas… Consequently, nearly all large spreadsheets are wrong and in fact have multiple errors… Spreadsheet error rates are very similar to those in traditional programming."*[1] *Be sure to test your spreadsheet calculations. Testing should consume* ***25%-40%*** *of development time.*

Here is a brief list of built-in **Excel functions you should be familiar with**:

AVERAGE	**COVAR**	**EXP**	**LOG**	**NORMSDIST**	**SUM**
CORREL	**DATE**	**IF**	**MAX/MIN**	**RAND**	**TRANSPOSE**
COUNT	**DAYS360**	**INTERCEPT**	**MINVERSE**	**SLOPE**	**VAR**
COUPDAYSBS	**DURATION**	**LINEST**	**MMULT**	**STDEV**	**VLOOKUP**

You should look at the descriptions and **help files** of these functions as well as the many others. You will not be able to memorize all the Excel functions, so the key is to know what kinds of functions are available in Excel and find them quickly.

To use a function, simply type = into cell, then the function name, then the **parameters**, or **input arguments**. For example:

```
=EXP( 5 )
```

The **return value** of the function will appear in the cell.

Some functions require an array of data as a parameter. This is accomplished using **ranges**. Usually we set these parameters by pointing and clicking on a cell, rather than by typing the code. For example, given values in cells A1 through A5, the sum of these values can be calculated in cell B1 as:

[1] Panko, Raymond R. 2006. "Recommended Practices for Spreadsheet Testing." *EuSpRIG 2006 Conference Proceedings.* p. 73 ff.

```
=SUM( A1:A5 )
```

Some functions accept more than one parameter, which are separated by commas. If a cell used as an input argument is in another worksheet, the cell reference is preceded by the sheet name. For example:

```
=IF( Sheet1!A1 > 0, 1, 0 )
```

Matrix functions often return arrays. To make this work, highlight the entire range, type in the function, and press **Control-Shift-Enter**. For example, given a matrix of values in cells A1 to B2, we can put the transposed matrix into cells C1 to D2 by highlighting C1 to D2, entering the formula, and then pressing Control-Shift-Enter:

```
=TRANSPOSE( A1:B2 )
```

Other matrix functions include MMULT, MINVERSE, and MDETERM.

B. Working with Cells

So far we have looked at **absolute cell referencing**. An absolute reference uses the column letter and row number. **Relative cell referencing** shows the number of rows and columns up and to the left of the current cell. For example, *if R1C1 Reference Style is turned on*, this formula refers to the cell one row up and one column to the left:

```
=R[-1]C[-1]
```

To turn on the R1C1 Reference Style, go to the Office Button | Excel Options | Formulas and click it on. In general, relative referencing is confusing, and absolute referencing is preferred.

Often times we need to copy formulas over some range. This is easily done, by left-clicking on the square in lower right-hand corner of the highlighted cell and dragging downward. In this example, cell B1 contains the formula:

```
=SUM( $A$1:A1 )
```

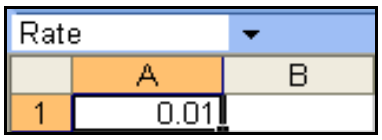
After copying this formula down, cell B8 contains:

```
=SUM( $A$1:A8 )
```

	A	B	C
1	1	1	
2	2	3	
3	3	6	
4	4	10	
5	5	15	
6	6	21	
7	7	28	
8	8	36	
9	9		
10	10		

Notice that the first cell in range reference is **locked** using the dollar signs, $A$1. This means that as you copy formulas to adjacent cells, neither the column nor the row in the first reference will change. Clicking on **F4** iterates through the four possible combinations of fixed cell references. For example: A1, no lock. $A1, only column locked. A$1, only row locked. $A$1, both column and row locked.

Sometimes we like to use **named ranges**. Rather than using cell references then, we can use the range name as a parameter instead. For example, we can name cell A1 as Rate in the name box.



Then, in cell B2 we can use this value as a parameter thusly:

```
=EXP( Rate )
```

Of course, you can always do your own math in Excel using operators to implement a formula. For example, the simple present value equation can be implemented as follows:

$$\Pr esentValue = \frac{CashFlow}{(1+Rate)^{Ttime}}$$

```
=100 / ( 1.08 ) ^ .25
```

C. Working with Data

Often times we use Excel to store data. This is convenient especially in light of Excel's look-up functionality, which searches for a value in a data table based upon a condition. Here is a simple example borrowed from Investopedia[2]:

	A	B	C	D	E	F
1	Data Table					
2	U.S. Treasury	Rate		Bond	Benchmark	Benchmark Yield
3	2 Year	3.04		XYZ Corp.	30 Year	4.63
4	5 Year	3.86		ABC Inc.	2 Year	3.04
5	10 Year	4.14		PDQ & Co.	5 Year	3.86
6	30 Year	4.63		MNO, Ltd.	10 Year	4.14

In this example, cell F3 contains the following formula:

```
=VLOOKUP( E2,$A$3:$B$6, 2, False )
```

In VLOOKUP function call (V stands for vertical, there is also HLOOKUP for horizontal), E2 is the look-up condition. A3:B6 is the table range. 2 indicates to compare the look-up condition to column one in the data table and return the corresponding value from column two. True as the last parameter compares on an exact or approximate match is returned. False returns only an exact match. The values in the data table must be in ascending order, or it may not work right.

[2] See "Microsoft Excel Features For The Financially Literate," by Barry Nielsen, CFA.

D. Working with Dates and Times

If we enter a date into Excel, say 01/09/2011, we can format its appearance in several ways by right-clicking and selecting Format Cells… However, Excel itself keeps track of the data as an integer value. We can use Excel's built-in date functions to perform date calculations. For example, to find the amount of time between two dates:

	A
1	1/9/2011
2	7/18/2011
3	190
4	.5205

Cell A3 contains the number of days between the two dates using either:

```
=A2 – A1
=DATEDIF( A1, A2, "d" )
```

The formula in cell A4 is:

```
=YEARFRAC( A1, A2, 3 )
```

For information on the parameters for the YEARFRAC function, see the Excel help files.

E. Other Functionality

You should also familiarize yourself with other capabilities of the Excel visual development environment. Most often, there is more than one way to accomplish any particular task. And, many times there are wizards and visual cues to walk you through development. For example, some of the following are also available by highlighting an Excel range and **right-clicking**.

What	How	Description
Saving Files	File Menu or Toolbar	Opening and saving Excel files
Copy / Paste	Edit Menu or Toolbar	Editing cells and worksheets
Cell Formatting	Format Menu or Toolbar	Changing cell appearance, also setting decimal style
Adding Toolbars	View | Toolbars	Using tools available in other toolbars
Excel Options	Tools | Options	Changing default Excel settings
Charts	Chart Wizard Icon	Creating charts in Excel using the Chart Wizard
Sorting Data	Data | Sort or Icon	Sorting data ascending or descending
Auditing	Tools | Formula Auditing	Trace cells that are inputs into a formula, or other cells that depend on the current cell
VBA Editor	Tools | Macro | Visual Basic Editor	Launch the VBA development environment

We will look at more Excel functionalities over the course of this study guide.

F. Getting Price Data from Yahoo! Finance

Historical stock price data is available for free on Yahoo! To get data:

- Go to Yahoo! Finance.
- Type in the symbol IBM.
- Click on Historical Prices.
- Select a date range, say Jan 1, 2010 to Dec 31, 2010. Check Daily data. Click Get Prices.
- At the bottom, click Download to Spreadsheet and save to IBM.csv.

This .csv file will open automatically in Excel.

G. LAB 1: Calculating Volatility and Covariance

Get one year of price data for Microsoft (MSFT) and Intel (INTC).

- Calculate the daily returns for each stock using the continuous compounding formula.
- Calculate the average daily return for each.
- Calculate the total return for each stock over the five years.
- Calculate the daily variances and standard deviations of returns.
- Calculate the annualized volatility of each stock.
- Calculate the covariance and correlation of returns between the two stocks.
- Create a line chart showing the prices of each stock.

How do you know your answers are right?

II. Advanced Excel Functionality

A. Goal Seek

Goal Seek enables what-if analysis. What-if analysis is a process of changing the inputs into a formula or model to see how they change the outcome of the formula or model. If you know the outcome of a formula, but not the input value that will generate that outcome, you can use Excel's Goal Seek. Goal Seek can find a specific outcome of a formula by changing the value of a cell that is used as an input into that formula.

	A	B	C
1	**Discount Rate**		0.41041
2	**Growth Rate**		0.00
3	**Year**	**Cash Flow**	**Present Value**
4	0	-1000	-1000.00
5	1	500	354.51
6	2	500	251.35
7	3	500	178.21
8	4	500	126.35
9	5	500	89.59
10			
11	**Net Present Value**		0.00

In each of the Present Value cells C4:C9, the formulas are copied down from C4 as:

```
=B4 / ( 1 + $C$2 ) ^ A4
```

The total Net Present Value is the sum of the individual present values in cells C4:C9. Using Tools | Goal Seek, set cell C11 to a value of 0 by changing cell C1.

Remember, Goal Seek is a fairly simple tool. It is used only for a single output and a single input. We will learn more powerful techniques later on.

B. Data Tables

If we want to look at how changing the value of an input affects the value output over a range of possible inputs, we can use Excel Data Tables. That is, we can try out different inputs without having to retype the formula over and over.

Continuing the prior example, we can use a **one-variable data table**, in cells E7:E13, to contain a range of possible discount rates. The question is how do various discount rates change the net present value? Cell F6 contains the formula =C11.

	E	F
5	**Discount Rate**	**NPV**
6		0.00
7	**0.00**	1500.00
8	**0.10**	895.39
9	**0.20**	495.31
10	**0.30**	217.78
11	**0.40**	17.58
12	**0.50**	-131.69
13	**0.60**	-246.14

Highlight the range outlined, click on Data | Table. Then, leave Row Input Cell blank and set Column Input Cell to $C$1.

A two-variable data table uses two input cells. What if our model had a both a growth rate and a discount rate? As both these variables change, the net present value changes.

	E	F	G	H	I
5	**Discount Rate**	**Growth Rate**			
6	0.00	**.00**	**0.10**	**0.20**	**0.30**
7	**0.00**	1500.00	2052.55	2720.8	3521.55
8	**0.10**	895.39	1272.73	1725.255	2263.59
9	**0.20**	495.31	763.86	1083.333	1460.72
10	**0.30**	217.78	415.61	649.1154	923.08
11	**0.40**	17.58	167.58	343.3391	548.19
12	**0.50**	-131.69	-15.10	120.5333	277.64
13	**0.60**	-246.14	-153.59	-46.6309	76.51

This time, cell E6 contains the formula =C11.

C. Pivot Tables

The Excel pivot table is a great way to report data. It sorts and sums the original data for analysis and presentation.

D. Histograms

A **histogram** shows the distribution of data. The height of each bar in a histogram is determined by the frequency of occurrences in the particular bin. The total area of the histogram is the number of data points. Dividing each bar by the total area will show the relative frequency with the total area equal to one. Thus, the histogram estimates the **probability density function**, $f(x)$. From there, **the cumulative density function**, $F(x)$, can be estimated through summation of the relative frequencies.

The widths of the bins can be calculated as:

$$bin\ size = \frac{\max(x) - \min(x)}{n}$$

Usually a value of n between 15 and 20 works out best.

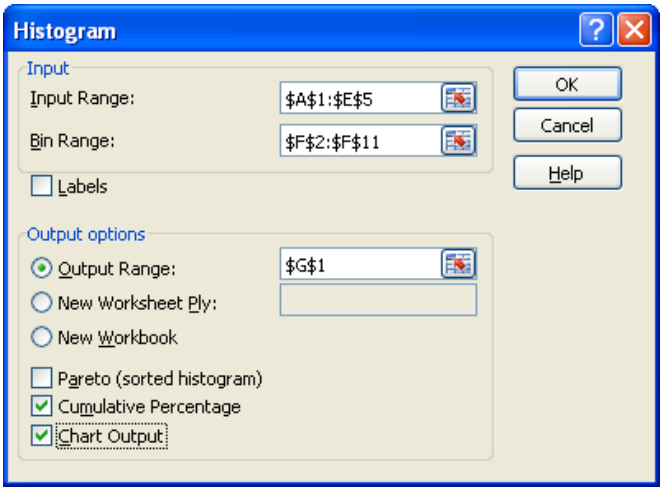
Consider the following random data:

	A	B	C	D	E
1	6	7	5	9	8
2	8	8	3	0	3
3	1	7	6	5	3
4	6	2	3	4	3
5	6	8	8	8	1

And the following bins:

	F
1	Bins
2	0
3	1
4	2
5	3
6	4
7	5
8	6
9	7
10	8
11	9

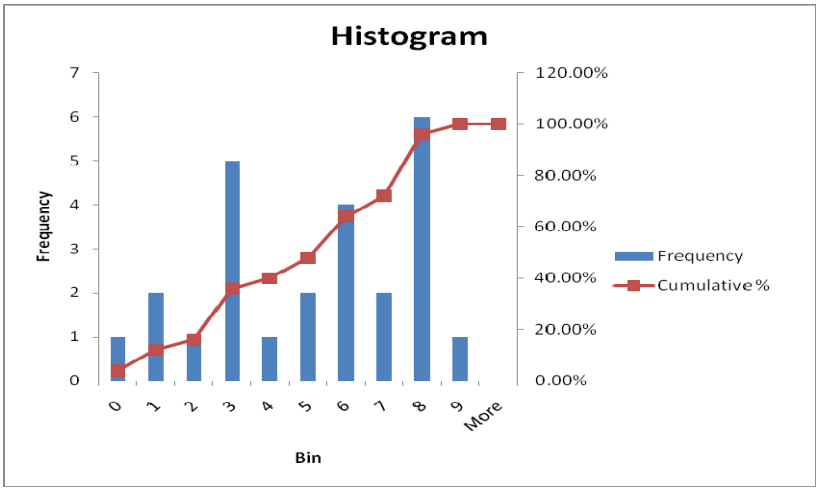
Click in Data | Data Analysis | Histogram. Populate the Histogram window as shown:



The output shows the frequency and cumulative percentage distributions.

	G	H	I
1	*Bin*	*Frequency*	*Cumulative %*
2	0	1	4.00%
3	1	2	12.00%
4	2	1	16.00%
5	3	5	36.00%
6	4	1	40.00%
7	5	2	48.00%
8	6	4	64.00%
9	7	2	72.00%
10	8	6	96.00%
11	9	1	100.00%
12	More	0	100.00%

Notice in the chart output that the cumulative percentage ($F(x)$) has a range of 0 to 1.



E. Calculating Portfolio Volatility

We calculate the one period continuous return on a stock as the natural log of the price relative:

$$r_i = \ln\left(\frac{S_i}{S_{i-1}}\right)$$

Where S_i is the closing price of the current period, and S_{i-1} is the closing price of the prior period. The average (or expected) return on a stock is calculated as:

$$\bar{r} = E(r) = \frac{\sum_{i=1}^{n} r_i}{n}$$

Where n is the number of periods. The sample variance of returns (=VAR() in Excel) is calculated as:

$$\sigma^2 = \frac{\sum_{i=1}^{n} (r_i - \bar{r})^2}{n-1}$$

The population variance (=VARP() in Excel) is the same, only the denominator is simply n rather than $n - 1$. The sample standard deviation of return is simply the square root of the variance.

Now, if we have a **portfolio** of m stocks, where the proportion (or percentage weight) on each stock is $\omega_1 \ldots \omega_m$, then the one period return on the portfolio is calculated as:

$$r_{p,i} = \sum_{j=1}^{m} r_j \cdot \omega_j$$

Where:

$$\sum_{j=1}^{m} \omega_j = 1$$

Notice that the average (or expected) return of the portfolio over n periods is equal to the average returns of the constituent stocks times their respective weights:

$$\bar{r}_p = E(r_p) = \frac{\sum_{i=1}^{n} r_{p,i}}{n} = \sum_{j=1}^{m} \bar{r}_j \cdot \omega_j$$

We might naively think that the portfolio variance behaves similarly. But, this is not the case. To calculate the portfolio variance we must account for the covariances between each pair of stocks. The covariance of returns on two stocks j and k is given by:

$$\sigma_{j,k} = COV(r_j, r_k) = \frac{\sum_{i=1}^{n} (r_{j,i} - \bar{r}_j)(r_{k,i} - \bar{r}_k)}{n}$$

The calculation of portfolio variance is given by:

$$\sigma_p^2 = \sum_{j=1}^{m} \sigma_j^2 \cdot \omega_j^2 + 2 \cdot \sum_{j=1}^{m} \sum_{k=j+1}^{m} \omega_j \cdot \omega_k \cdot \sigma_{j,k}$$

In matrix notation, these calculations are greatly simplified. If we let the average returns on the *m* stocks be:

$$\overline{R} = \begin{bmatrix} \bar{r}_1 \\ \bar{r}_2 \\ \vdots \\ \bar{r}_m \end{bmatrix} = \begin{bmatrix} E(r_1) \\ E(r_2) \\ \vdots \\ E(r_m) \end{bmatrix}$$

And the vector of weights is:

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_m \end{bmatrix}$$

Then the expected return on the portfolio for period *i* is:

$$\bar{r}_p = E(r_p) = \overline{R}^T \cdot \omega$$

Where *T* denotes the transpose, in this case of the expected return matrix. The portfolio variance calculation can be shortened to:

$$\sigma_p^2 = \omega^T \cdot \Omega \cdot \omega$$

Where Ω is the covariance matrix:

$$\Omega = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,m} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m,1} & \sigma_{m,2} & \cdots & \sigma_{m,m} \end{bmatrix}$$

Likewise, the correlation matrix *P* is:

$$\mathrm{P} = \begin{bmatrix} 1 & \rho_{1,2} & \cdots & \rho_{1,m} \\ \rho_{2,1} & 1 & \cdots & \rho_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{m,1} & \rho_{m,2} & \cdots & 1 \end{bmatrix}$$

F. LAB 2: Calculating Portfolio Metrics

Get five years of price data for IBM (IBM) and Intel (INTC).

- Calculate the portfolio volatility using matrix functions in Excel.

II. VISUAL BASIC FOR APPLICATIONS

Visual Basic for Applications (VBA) is a programming language that runs behind Excel. The **Microsoft Developer Network (MSDN)** website has a wealth of documentation—both how to's and references—on Excel/VBA at:

- http://msdn.microsoft.com/en-us/library/bb979621(v=office.12).aspx

Specifically, the Excel 2007 Developer Reference link will lead you to **How Do I… in Excel 2007** and the **Excel Object Model Reference** which maps the Excel Object Model. If you encounter object references in this text that you are unfamiliar with, you should first refer to the MSDN reference for information.

Here is a brief list of built-in **Excel objects you should be familiar with**:

APPLICATION	FONT	SELECTION[3]
CHART	LISTOBJECT	WORKBOOKS
DIALOG	PARAMETER	WORKSHEETS
ERROR	RANGE	CELLS[4]

You should look at the descriptions of these objects as well as others. You will not be able to memorize all the Excel objects, so the key is to know what kinds of objects are available in VBA and find and learn about them quickly. As you will see, these objects consist of three parts:

1. **Properties**, which are attributes of an object.
2. **Methods**, which are functionalities of an object.
3. **Events**, which are actions that are initiated within the scope of an object, but handled by methods outside of an object.

Another good site for VBA code and explanation is Anthony's VBA Page at:

- http://www.anthony-vba.kefra.com/

As with all things Excel, the best way to learn is to jump in and start doing. To add VBA code to your spreadsheet, open the Visual Basic Editor (VBE) environment by clicking on Developer | Visual Basic. (If the Developer tab is not available, click on the Office button in the upper right-hand corner, then click on Excel Options, and turn on Show Developer tab in the Ribbon.) In the VBE, add a module. From the menu bar, click Insert | Module. A blank code window will appear. We use the VBE create and modify procedures—functions and sub-routines. Every Excel file can contain VBA code.

---

[3] The Selection object may not appear in the MSDN reference list. But, it is an important object. The Selection object "represents the active selection, which is a highlighted block of text or other elements in the document that a user or a script can carry out some action on." For more information from within MSDN, you should search on *Selection Object*.

[4] The Cells object may not appear in the MSDN reference list. This kind of problem is a common occurrence in technology documentation. It's all part of what makes technology so hard. For more information from within MSDN, you should search on *Cells Object*.

A **procedure** is a block of code enclosed in Sub and End Sub statements or in Function and End Function statements. The difference between a **sub-routine** and a **function** is that a sub-routine has no return value. Because of this, we use the two in different ways and for different reasons. Both sub-routines and functions may have input parameters. First, let's create a function:

```
Function Add(a, b)
    c = a + b
    Add = c
End Function
```

In Excel, you can call this function in the same fashion as calling any of Excel's built-in functions. This code works, but does not employ good programming practices. We are better off writing it as:

```
Option Explicit

Function Add(a As Double, b As Double) As Double
    Dim c As Double
    c = a + b
    Add = c
End Function
```

Here, **option explicit** forces us to declare our variables before we define them. Variables, such as a, b, and c, are declared with a type. In this case the type is double, which is a floating point number.

Variables are physical memory locations inside the computer. VBA has several variable types to hold different kinds of data. The most commonly used are: **Boolean** (true/false), **Char** (character), **Date** (Date and Time), **Double** (floating point number), **Integer**, **Object** (any type), **String** (series of characters), **Variant** (any type).

Notice that the function code defines the number and type of parameters the function expects when called, as well as the return type. The **return value** is set in VBA by setting the name of the function equal to some value.

We can also create a function that accepts a Range as an input parameter:

```
Function Sum_Range( A As Range ) As Double
    Dim total As Double
    Dim r, c As Integer

    For r = 1 To A.Rows.Count
        For c = 1 To A.Columns.Count
            total = total + A(r, c)
        Next c
    Next r
    Sum_Range = total
```

```
End Function
```

A **Range** represents a cell, a row, a column, or a rectangular selection of contiguous cells. For more information on the Range object, see the Excel Object Model Reference on the MSDN website. Given date in cells A1 to A5, we can call this function in Excel as:

```
=Sum_Range( A1:A5 )
```

Now, let's write a sub-routine.

```
Sub Sum_Values()

    Dim a, b As Double

    a = Range("A1").Value
    b = Range("A2").Value
    Range("B1").Value = a + b

End Sub
```

This sub-routine is a **macro**. An Excel macro contains instructions to be executed. We often use macros to eliminate the need to repeat steps of common performed tasks. We can cause this sub-routine to run using a button. To add a button to your spreadsheet, click open View | Toolbars | Forms. From the Forms toolbar, left-click on the button icon and paint an area on your spreadsheet. When the Assign Macro window shows up, select Sum_Values and click OK. When you click on the button, the sum of A1 and A2 should appear in B1.

A. Variable Scope

As soon as a variable goes out of **scope** it can no longer be accessed and it loses its value. Variables can be given different scopes based upon how we declare them.

Procedure-level scope variables are declared using Dim or Const inside a procedure.

```
Public Sub MyProcedure()
      Dim a As Integer
      a = 7
End Sub
```

When the procedure is done running, the variable or constant goes out of scoped and is destroyed. **Module-level scope** variables are declared above the procedure definitions stay within scope after the procedure is done running.

```
Dim a As Integer

Public Sub MyProcedure()
      a = 7
End Sub
```

All variables with this level of scope are available to all procedures that are within the module. **Project-Level** or **Workbook-Level** variables are declared at the top of any standard public module and are available to all procedures in all modules.

B. Conditional Statements

Often we need to test for equality or inequality. We do this with an If...Then statement. The general syntax is this: if the condition to validate is true, then execute some code.

```
Public Sub MyProcedure()

   Dim a as Integer = 2
   Dim b as Integer = 3

   If a < b Then
       MsgBox( “True” )
   End If

End Sub
```

If need to add lines of code in the case where the expression evaluates to false, we use If...Then...Else. This syntax is:

```
If a > b Then
     MsgBox( “True” )
Else
     MsgBox( “False” )
End if
```

If multiple evaluation conditions exist, we can use a Select...Case statement. You can think of a Select…Case statement as a series of if statements. The syntax is:

```
Select Case a
     Case Is < 0
        …
     Case 1 To 5
        …
     Case Is > 5
        …
     Case Else
        …
End Select
```

D. Loops

For repeated execution of a block of code, we can use one of several repetition or iteration structures. The general syntax is of a Do While loop is:

```
Do While a < 10
     a = a + 1
Loop
```

This line of code inside the Do While loop will executed repetitively until the condition evaluates to false. The syntax of the Do…Loop While is:

```
Do
     a = a + 1
Loop While a < 10
```

The most commonly used repetition structure is the For loop. The For loop syntax is:

```
For a = 1 to 10 Step 1
     …
Next a
```

In this case, the Step 1 is optional because the For loop will by default increment a by 1 each time through the loop. However, any other incrementation would require the **Step** clause.

```
For a = 50 to 0 Step -2
     …
Next a
```

It's often convenient to use range offsets in conjunction with sub-routines in order to fill a range with values. For example:

```
Sub Fill_Range()
    Dim i As Integer
    For i = 0 To 10
        Range("A1").Offset(i, 0).Value = i
    Next i
End Sub
```

In this example, the Offset adds *i* rows and 0 columns to the absolute cell reference A1.

F. Using Excel Functions in VBA

We can call Excel functions from VBA code:

```
Function MyAverage(data As Range) As Double

    Dim avg As Double
    avg = Application.Average(data)
    MyAverage = avg

End Function
```

Note, however, that using Excel functions in VBA has performance implications. Excel functions run slower in VBA than equivalent VBA functions! So, it's better to write your own functions in VBA and call those, rather than using the Excel function.

G. Recording Macros

Here is an example showing how to record a VBA macro to calculate the average of five numbers.

Suppose you want to put the numbers 1 through 20 in cells A1 through A20, sum up those numbers and put the total in cell A21. Then you want to make cell A21 bold, centered and with a yellow background. If this was a task you needed to do repetitively, you could record a macro to perform these steps.

To record this macro, click Developer | Record Macro. For now, leave the default name to Macro1 and click OK. Excel will now record every move you make in the form of VBA code. Follow the tasks defined above until you are finished, then click Stop Recording button on the macro toolbar. The VBA recorder should have written something like this:

```
Sub Macro1()
'
' Macro1 Macro
' Macro recorded 1/6/2011 by Ben
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "1"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "2"
    Range("A1:A2").Select
    Selection.AutoFill Destination:=Range("A1:A20"), Type:=xlFillDefault
    Range("A1:A20").Select
    Range("A21").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-20]C:R[-1]C)"

    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .Weight = xlMedium
        .ColorIndex = xlAutomatic
    End With
    With Selection.Interior
        .ColorIndex = 6
        .Pattern = xlSolid
    End With
    Selection.Font.Bold = True
    With Selection
        .HorizontalAlignment = xlCenter
    End With
End Sub
```

If you associate a button with this sub-routine you can run it again and again.

Recording macros is one of the best ways to learn how to program in VBA. If you don't know the code to accomplish some task, try recording it and see what comes out! Often, the code recorder writes fairly messy code. If you walk through what it writes and clean it up, you'll learn a lot about VBA!

H. LAB 3: Programming VBA

Get one year of price data for AXP.

- Write a VBA function that calculates the average return.
- Create a VBA macro that makes a chart of the prices.

I. Arrays

An array is a set of contiguous memory locations all of the same data type. Each element in an array can be referred to by its index. The **ReDim** statement resizes an array that has previously been declared.

```
Option Explicit
Option Base 1

Function CovarMatrix(data As Range) As Variant

    Dim r As Integer
    Dim c As Integer
    Dim n As Integer

    Dim rows_count As Integer
    Dim cols_count As Integer

    rows_count = data.rows.Count
    cols_count = data.Columns.Count

    Dim avgs As Variant
    avgs = Averages(data)

    Dim matrix() As Double
    ReDim matrix(cols_count, cols_count)

    For c = 1 To cols_count
        For n = 1 To cols_count
            For r = 1 To rows_count

                matrix(c, n) = matrix(c, n) + (data(r, c) - avgs(c)) *
                                              (data(r, n) - avgs(n))

            Next r
            matrix(c, n) = matrix(c, n) / rows_count
        Next n
    Next c

    For r = 2 To cols_count
        For c = 1 To r - 1
            matrix(r, c) = matrix(c, r)
        Next c
    Next r

    CovarMatrix = matrix

End Function

Function Averages(data As Range) As Variant

    Dim r As Integer
    Dim c As Integer
```

```
    Dim avgs() As Double
    ReDim avgs(data.Columns.Count)

    For c = 1 To data.Columns.Count
        For r = 1 To data.rows.Count
            avgs(c) = avgs(c) + data(r, c)
        Next r
        avgs(c) = avgs(c) / data.rows.Count
    Next c
    Averages = avgs

End Function
```

Given the following data, the CovarMatrix function will accept the range A1:D19 as the input parameter and require Ctrl+Shift+Enter to return the matrix.

	A	B	C	D
1	**INTC**	**IBM**	**MSFT**	**WMT**
2	0.00000	-0.01215	0.00422	-0.01117
3	-0.00346	-0.00649	-0.02838	-0.01432
4	-0.00996	-0.00346	0.00679	0.01821
5	0.01145	0.00994	-0.03408	-0.01264
6	0.00543	-0.01049	0.02915	0.01993
7	-0.01637	0.00792	-0.00340	0.03147
8	0.01687	-0.00211	-0.01216	-0.01416
9	0.02478	0.01584	0.01402	-0.00111
10	-0.03218	-0.00401	-0.01151	-0.01438
11	0.01182	0.01771	-0.02279	0.01081
12	0.00147	-0.02945	-0.00353	-0.00022
13	-0.01131	-0.00959	-0.00322	0.00201
14	-0.04605	-0.02755	-0.03210	0.03341
15	0.02050	0.00494	-0.00100	0.01948
16	-0.01585	-0.00291	0.04049	-0.02533
17	0.01179	0.00461	-0.00932	-0.01688
18	-0.01592	-0.02062	-0.00681	0.01274
19	-0.02622	-0.01110	-0.00358	0.00609
20	0.01057	0.01848	-0.01248	0.00325

J. Calculating Beta

The **capital asset pricing model (CAPM)** states that the expected return on a stock is a linear function of the market return less the risk free rate. The CAPM calculates a theoretical required rate of return of a stock r_s given a return on a market index r_i, as per:

$$E(r_s) = r_f + \beta_s (E(r_i) - r_f)$$

$E(r_i) - r_f$ is called the **market risk premium (RP_i)** , thus the risk premium for a stock $E(r_s) - r_f$ is equal to the market risk premium times β.

From the CAPM, the **Beta** (β_s) of a stock (or a portfolio of stocks) is a number describing the relationship of its returns relative those of a market index. A stock's Beta will be zero if its returns are completely un-related to the returns of the index. A Beta of one means that the stock's returns will tend to be like the index's returns. A negative Beta means that the stock's returns generally move opposite to the index's returns.

Beta can be calculated for by using either regression of the stock's returns against the index returns (=SLOPE()) or as:

$$\beta_s = \frac{COV(r_s, r_i)}{\sigma_i^2}$$

Beta is also a measure of the sensitivity of the stock's returns to index returns. That is, Beta represents the stocks systematic risk, risk that cannot be diversified away. Thus, Beta is also the market hedge ratio.

Consider a portfolio of stocks valued at $100 million. This portfolio has a beta of 1.08 relative to the S&P 500. If the S&P 500 futures contract is currently at 1300.00, how do we fully hedge this portfolio?

Since we are long the stocks, we will need to sell futures in the appropriate amount. That way, if the market goes down, the loss on our stocks will be offset by a gain in the futures. Since each E-mini S&P 500 futures contract has a value of $50 times value of the S&P 500 index, we would sell:

$$\frac{Portfolio\ Value}{Index\ Value \times Contract\ Size} \times Beta = \frac{100{,}000{,}000}{1300 \times 50} \times 1.08 = 1662\ \ futures\ contracts$$

Arbitrage pricing theory (APT) states that the expected return of a stock is a linear function of multiple risk factors—macro-economic, fundamental or indices. Thus, where the CAPM that has only one Beta, APT has multiple betas. Each risk factor has a beta indicating the sensitivity of the stock to that risk factor.

$$E(r_s) = r_f + b_1(RP_1) + b_2(RP_2) + \cdots + b_n(RP_n)$$

K. LAB 4: Beta

- Calculate the Beta of IBM using regression and the Beta formula.
- Given a portfolio of the following positions:

Stock	Shares	Price
IBM	2000	164.82
INTC	12000	21.69
WMT	5000	56.07
XOM	5000	83.93

What is the optimal hedge ratio?

L. Financial Data

Thus far the only financial data we have considered is **price data.** But there are other types of financial data too.

Price Data

Price data consists of the bid and ask prices and quantities, and trade prices and quantities for securities and derivatives.

Valuation Data

Valuation data is different from price data. For some financial instruments—bonds, swaps, and all OTC derivatives—no price data exists, or if it does, it is a highly guarded secret. For these, valuation data is all there is. That is, the price exists only in theory, and, furthermore, is not a firm bid or offer that any market maker is obliged to honor.

Fundamental Data

Fundamental data consists of everything that is disclosed in 10-Q quarterly and 10-K annual reports, including key business items, such as earnings, sales, inventories, and rents.

Calculated Data

Given fundamental data, calculated data includes ROE, price to book, beta, forecasted dividends, free cash flow, etc.

Economic Data

Economic data, such as CPI and GDP, are key indicators often used in financial analysis and trading.

M. Correlation and Ranking

Pearson's correlation is obtained by dividing the covariance of two random variables by the product of their standard deviations:

$$\rho_{i,j} = \frac{\sigma_{i,j}}{\sigma_i \sigma_j}$$

The Pearson correlation is +1 in the case of a perfectly linear correlation, −1 in the case of a perfectly negative correlation. All other values are between −1 and +1. A zero correlation the two random variables are uncorrelated.

Often, we **rank** fundamental or calculated data. Given the following raw earnings per share data in column A, the ranks are fond using Excel's RANK() formula.

	A	B
1	0.25	4
2	0.36	5
3	-0.22	2
4	-0.06	3
5	1.52	6
6	-0.29	1

Here, the formula in cell B1 is copied down to B6 as:

```
=RANK( A1, $A$1:$A$6, 1 )
```

If multiple data points are the same (i.e. there are **ties**), we generally find the average of those ranks. So, if ranked data points 4, 5 and 6 are the same, then the average is (4 + 5 + 6) / 3 = 5, so all three data points get a rank of 5.

Spearman's rank correlation is a non-parametric measure of statistical dependence between two random variables. It assesses how well the relationship between two variables can be described using a monotonic function. If there are no repeated data values, a perfect Spearman correlation of +1 or −1 occurs when each of the variables is a perfect monotone function of the other.

A simple procedure is normally used to calculate Spearman's correlation. The n raw scores are converted to ranks x_i, y_i, and the differences $d_i = x_i - y_i$ between the ranks of each observation on the two variables are calculated. If there are no tied ranks, then ρ is given by:

$$\rho = 1 - \frac{6 \cdot \sum d_i^2}{n \cdot (n^2 - 1)}$$

Given the following EPS data on two stocks, ABC and XYZ:

	A	B	C	D	E	F
1	**ABC**	**XYZ**	**Rank ABC**	**Rank XYZ**	**d**	**d^2**
2	0.25	0.45	1	3	-2	4
3	1.32	0.36	4	2	2	4

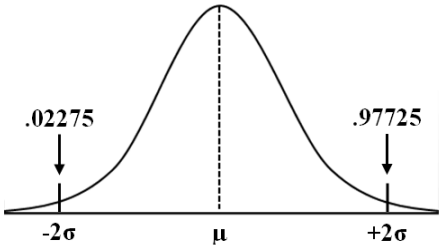
4	1.06	-0.5	2	1	1	1
5	1.21	0.65	3	4	-1	1
6					**Sum:**	10

The Spearman's rank correlation is:

$$\rho = 1 - \frac{6 \cdot 10}{4 \cdot (16 - 1)} = 0$$

N. Distribution Fitting Example

Suppose that, given raw fundamental data (e.g. earnings per share, price-to-book ratio, etc.), we wish to fit the data to a normal distribution, between plus and minus 2 standard deviations. The probabilities associated with this range are 2.275% and 97.725%:



These probabilities can be found easily in Excel using the NORMSDIST() function. In the following table, given raw data in column A, we can convert it to the new normalized score in column C.

	A	B	C
1	**Raw Data**	**Cumulative Probability**	**New Z-Score**
2	-.50	**.02275**	**-2**
3	-.25	.36364	-.34874
4	-.22	.40455	-.24159
5	-.18	.45909	-.10272
6	0	.70454	.53749
7	.10	.84089	.99813
8	.20	**.99725**	**2**

Given data x_1 through x_n where $i = 1...n$, the cumulative probabilities in column B are found as:

$$F(x_i) = P(x \le x_i) = F(x_1) + \frac{x_i - x_{i-1}}{x_n - x_1} \cdot (F(x_n) - F(x_1))$$

The Excel formulae for generating the data in this table are:

```
CELL B2:       = NORMSDIST( -2 )
CELL B8:       = NORMSDIST( 2 )
CELL B3 – B7:  = ( A3 - $A$2 ) / ( $A$8 - $A$2 ) * ( $B$8 - $B$2 ) + $B$2
CELL C2 – C8:  = NORMSINV( B2 )
```

In this next table, given the *ranks* of the raw data in column A, we can convert it to the new normalized score in column C.

	A	B	C
1	**Raw Rank**	**Cumulative Probability**	**New Z-Score**

2	1	**.022750**	**-2**
3	2	.181833	-.90840
4	3	.340917	-.40996
5	4	.500000	0
6	5	.659083	.40996
7	6	.818167	.90840
8	7	**.997250**	**2**

The Excel formulae for generating the data are the same as in Table 1. The difference between simple ranking and distribution fitting is that using ranks is like fitting to a uniform distribution.

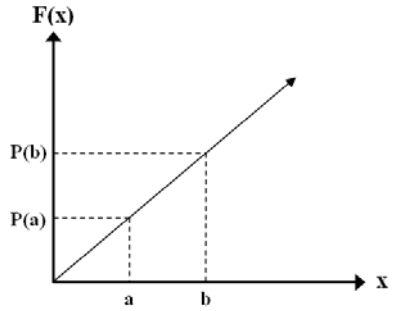


Figure 1: Simple Ranking Fits to a Uniform Distribution

As can been seen from Figure 1, two ranks in the neighborhood of *P(a)* will map the appropriate distance apart, as will two points in the neighborhood of *P(b)*, because of the constant slope of *F(x)* in a uniform distribution.

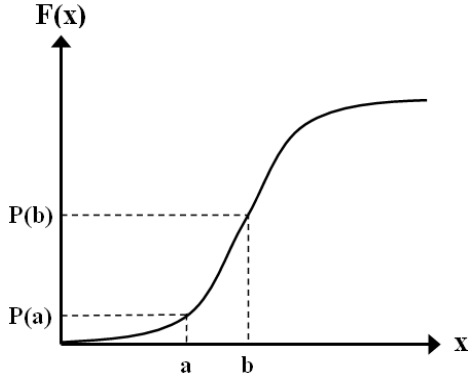


Figure 2: Fitting Ranked Data to a Normal Distribution

Fitting the ranks to a normal distribution is different. As can be seen in Figure 2, two points in the neighborhood of *P(a)*—such as data points with ranks 1 and 2—will map further away than will two points in the neighborhood of *P(b)*—such as data with ranks 455 and 456—because the slope of *F(x)* not constant, and is steeper at *b* than *a*.

So, distribution fitting takes differences in the ranks of observations (or in some cases the observations themselves), and imposes a distributional prior as to how much importance gaps in neighboring observations should have. The distributional method determines the importance of outliers.

A variation on the ranking theme is to scale the ranks by the *difference* between data points, so that points with larger differences between them have a correspondingly large gap between their ranks. This is typically done by placing the differences into bins. The steps are as follows:

Step 1: Sort the data from low to high.
Step 2: Find the *differences* between points.
Step 3: Put the differences into bins 1…*m* according to their size. That is, small differences go into bin 1, and the largest differences go into bin *m*.
Step 4: To assign ranks to the data, give the smallest data point a rank of 1, and then add the *bin value* to each successive rank, so that each value gets assigned a rank that differs from the previous rank by the bin value. Thus, there will be gaps in the numbering.
Step 5: Finally, proceed with distribution fitting as before.

Here is a numerical example using 3 difference bins to illustrate this technique.

	A	B	C	D
1	**Raw Data**	**Difference**	**Difference Bin**	**Raw Rank**
2	-1			1
3	-.5	.5	2	3
4	-.3	.2	1	5
5	-.2	.1	1	6
6	0	.2	1	7
7	.1	.1	1	8
8	.5	.4	2	10
9	2	1.5	3	13

In this table, given the sorted raw data in column A, we can convert it to the new raw ranks in column D. These raw ranks can be used as inputs into the previous example to generate a normalized score.

O. Scaling and Ranking in Practice

Financial data is often categorized for the purpose of generating factor indicators. For example, valuation factors are often scaled by the industry sector. Volatility factors are sometimes scaled by capitalization group. For cross-sectional analysis, these groups are often defined by fundamental data. For time-series analysis, these groups may be time periods, months, days, or intra-day periods.

We only scale a given factor by category if we believe the differences between the groups are systemic and meaningless. It also may be the case that the factor is otherwise too volatile or unstable to generate meaningful forecasts. For example, relative value is more stable than absolute.

If the distribution of data in each group, sector, or category is different, then scaling by group may not help much, unless you use a ranking method. Table 4 contains some sample data that should illustrate the value of ranking by groups.

	A	B	C	D	E	F	G	H	I
1	Group	Raw Data							
2	A	22	25	26	28	30	35	36	39
3	B	5	6	6	7	7	7	8	9

	A	B	C	D	E	F	G	H	I
4	Group	Raw Ranks							
5	A	1	2	3	4	5	6	7	8
6	B	1	2	2	5	5	5	7	8

In the first table here, the data for group A clearly indicates a different distribution. By ranking the data by group, we can compare apples to apples from a normalized, z-score perspective.

Some caveats with respect to scaling, ranking and z-scoring should be noted.

- Z-scoring will not prevent most of the bad values from being in group 1. This does not fit most people's intuitive definition of group neutral.
- Z-scoring a factor by cross-section has the meaning of ranking the particular stock's value on that factor relative to other stocks at that point in time. This is typical for benchmark-aware strategies.
- Z-scoring a factor through time, stock by stock, creates a factor that points to unusually high or low values relative to each stocks own history. This is typical for non-portfolio based strategies.

P. Double Normalizing

It is also possible to double normalize. That is, we normalize one way, then the other. In this case the order of normalization—cross-sectional first, or time-series first—is important to the final meaning of the factor.

Example 1

For example, in the case of performing time-series normalization first, then cross-sectional normalization, consider the data for IBM:

IBM		
Month	**Factor Data**	**Z-Score**
March	1.02	-.23
April	2.21	.96
May	1.00	-.25
June	3.52	2.27

After the time-series normalization, the factor data and z-score for June clearly appear to be unusually high. However, after a cross-sectional normalization, as can be seen next, most other stocks seem to also be high in June.

Stocks	
Symbol	**June Z-Scores**
IBM	2.27
LUV	1.95
INTC	2.35
WMT	2.02

So, 2.27 is nothing special in terms of upward movement.

Example 2

In the alternative case, where we perform the cross-sectional normalization first, then time-series normalization, consider the data:

Stocks		
Symbol	**June Factor Data**	**Z-Score**
IBM	3.52	1.52
LUV	.60	-.92
INTC	2.99	1.08
WMT	1.25	-.38

After the cross-sectional normalization, IBM looks particularly good. After the time-series normalization, as can be seen in next, IBM's June Z-Score is high relative to its own history in that cross-sectional score.

IBM	
Month	**Factor Z-Score**
March	1.13
April	.98
May	1.01
June	1.52

Relative to its *usual* Z-score it looks good, but not quite as good as it looked in after the cross-sectional normalization because IBM appears to score consistently high in this factor.

Q. LAB 5: Scaling and Ranking

- Given the following fundamental data and two stocks:

	A	B
1	ABC	XYZ
2	12.50	0.12
3	9.82	0.25
4	11.77	0.17
5	15.43	0.05
6	19.03	0.31

Calculate the Spearman's rank correlation between the two.

- Given the following data:

	A	B	C
1	Month	Stock	Raw Data
2	Jan	IBM	2.52
3		WMT	1.19
4		MSFT	.45
5		XOM	5.36
6	Feb	IBM	2.10
7		WMT	1.11
8		MSFT	.45
9		XOM	5.13
10	Mar	IBM	2.48
11		WMT	1.36
12		MSFT	.47
13		XOM	4.59
14	Apr	IBM	2.52
15		WMT	1.43
16		MSFT	.49
17		XOM	4.23

Performing a double normalization—time-series first, then cross-sectional.

III. INTRODUCTION TO SIMULATION

A model is a representation of reality. Traditionally, models are mathematical equations, which are attempts at analytical or closed form, solutions to problems of representation.

"All models are wrong. Some models are useful." -George Box

These equations enable estimation or prediction of the future behavior of the system from a set of input parameters, or initial conditions. However, many problems are too complex for closed form equations.

Simulation methods are used when it is unfeasible or impossible to develop a closed form model. Simulation as a field of study is a set of algorithms that depend upon the iterative generation of random numbers to build a distribution of probable outcomes. Because of their reliance on iteration, sometimes millions of them, simulation is accomplished only through the use of computer programs.

Simulation is especially useful when applied to problems with a large number of input distributions, or when considerable uncertainty exists about the value of inputs. Simulation is a widely-used method in financial risk analysis, and is especially successful when compared with closed-form models which produce single-point estimates or human intuition. Where simulation has been applied in finance, it is usually referred to as Monte Carlo simulation.

Monte Carlo methods in finance are often used to calculate the value of companies, to evaluate investments in projects, to evaluate financial derivatives, or to understand portfolio sensitivities to uncertain, external processes such as market risk, interest rate risk, and credit risk. Monte Carlo methods are used to value and analyze complex portfolios by simulating the various sources of market uncertainty that may affect the values of instruments in the portfolio.

Monte Carlo methods used in these cases allow the construction of probabilistic models, by enhancing the treatment of risk or uncertainty inherent in the inputs. When various combinations of each uncertain input are chosen, the results are a distribution of thousands, maybe millions, of what-if scenarios. In this way Monte Carlo simulation considers random sampling of probability distribution functions as model inputs to produce probable outcomes.

Central to the concept of simulation is the generation of random numbers. A random number generator is a computer algorithm designed to generate a sequence of numbers that lack any apparent pattern. A series of numbers is said to be (sufficiently) random if it is statistically indistinguishable from random, even if the series was created by a deterministic algorithm, such as a computer program. The first tests for randomness were published by Kendall and Smith in the *Journal of the Royal Statistical Society* in 1938.

These frequency tests are built on the Pearson's chi-squared test, in order to test the hypothesis that experimental data corresponded with its theoretical probabilities. Kendall and Smith's null hypotheses were that each outcome had an equal probability and then from that other patterns in random data would also be likely to occur according to derived probabilities.

For example, a serial test compares the outcome that one random number is followed by another with the hypothetical probabilities which predict independence. A runs test compares how often sequences of numbers occur, say five 1s in a row. A gap test compares the distances between occurrences of an outcome. If a data sequence is able to pass all of these tests, then it is said to be random.

As generation of random numbers became of more interest, more sophisticated tests have been developed. Some tests plot random numbers on a graph, where hidden patterns can be visible. Some of these new tests are: the monobit test which is a frequency test; the Wald–Wolfowitz test; the information entropy test; the autocorrelation test; the K-S test; and, Maurer's universal statistical test.

A. Uniform Distribution

Parameters *a* and *b*, the lower and upper bounds.

Probability density:

$$f(x) = \frac{1}{b-a}$$

Cumulative distribution function F(x):

$$F(x) = \frac{x-a}{b-a}$$

Expected value of x:

$$E(x) = \frac{a+b}{2}$$

Variance of x:

$$V(x) = \frac{(b-a)^2}{12}$$

The Linear Congruential Generator (LCG) will generate uniformly distributed integers over the interval 0 to m - 1:

$$u_i = (cu_{i-1} + d) \mod k$$

The generator is defined by the recurrence relation, where u_i is the sequence of pseudorandom values, and 0 < m, the modulus, 0 < c < k, the multiplier, and 0 < d < m, the increment. u_0 is called the seed value.

VBA:

```
Public Function LCG( c As Double, d As Double, k As Double, _
                     u0 As Double ) As Double
    LCG = (c * u0 + d) Mod k
End Function
```

	A	B	C	D
1	c	6578	=LCG(B1,B2,B3,B4)	=C1/($B$3-1)
2	d	1159	=LCG($B$1,$B$2,$B$3,C1)	=C2/($B$3-1)
3	k	7825	‘’	‘’
4	u0	5684	‘’	‘’

Excel:

```
=MOD( c * u0 + d, k )
```

The real problem is to generate uniformly distributed random numbers over the interval 0 to 1, what we call the *standard uniform* distribution, where the parameters $a = 0$ and $b = 1$. A standard uniform random number, u_s, can be accomplished by dividing the LCG random integer by k – 1 as in Table 1. However, Excel and VBA already have functions that return standard uniform random numbers:

Excel:

```
=RAND()
```

VBA:

```
Public Function UniformRand() As Double
      UniformRand = Rnd()
End Function
```

Generating Uniformly Distributed Random Numbers:

VBA:

```
Sub Generate()
    Dim i as Integer
    For i = 0 To Range("A1").Value
        Range("A2").Offset(i).Value = Rnd()
    Next i
End Sub
```

In any case, the state of the art in uniform random number generation is the **Mersenne Twister** algorithm. Most statistical packages, including MatLab, use this algorithm for simulation.

Turning a standard uniform random number, u_s, into a uniformly distributed random number, u, over the interval a to b.

Excel:

```
= a + RAND() * ( b - a )
```

VBA:

```
Public Function Uniform( a As Double, b As Double ) As Double
```

```
    Uniform = a + Rnd() * ( b – a )
End Function
```

Generating Uniformly Distributed Random Integers:

Turning a standard uniform random number, u_s, into a uniformly distributed random *integer* of the interval *a* to *b*:

Excel:

```
= FLOOR( a + RAND() * ( b – a + 1 ), 1 )
```

VBA:

```
Public Function Uniform( a As Double, b As Double ) As Double
    Uniform = Int( a + Rnd() * (b - a + 1) )
End Function
```

III. CONTINUOUS DISTRIBUTIONS

A. Inverse Transform Method

The inverse transform method generates random numbers from any probability distribution given its cumulative distribution function (cdf). Assuming the distribution is continuous, and that its probability density is actually integratable, the inverse transform method is generally computationally efficient.

The inverse transform methods states that if *f(x)* is a continuous function with cumulative distribution function *F(x)*, then *F(x)* has a uniform distribution over the interval *a* to *b*. The inverse transform is just the inverse of the cdf evaluated at *u*:

$$x = F^{-1}(u)$$

The inverse transform method works as follows:

1. Generate a random number from the standard uniform distribution, u_s.
2. Compute the value *x* such that $F(x) = u$. That is, solve for x so that $F^{-1}(u) = x$.
3. *x* is random number drawn from the distribution *f*.

C. Exponential Distribution

Parameter β, the scale parameter. The exponential distribution arises when describing the inter-arrival times in a (discrete) Poisson process.

Probability density:

$$f(x) = \frac{1}{\beta} e^{-x/\beta}$$

Derivation of the cumulative distribution function F(x):

$$F(x) = \int_0^x f(x)dx$$

$$F(x) = \int_0^x \frac{1}{\beta} e^{-x/\beta} dx$$

$$F(x) = -\int_0^x -\frac{1}{\beta} e^{-x/\beta} dx$$

$$F(x) = -e^{-x/\beta}\Big|_0^x$$

$$F(x) = -e^{-x/\beta} + e^{-0/\beta}$$

$$F(x) = 1 - e^{-x/\beta}$$

Expected value of x:

$$E(x) = \beta$$

Variance of x:

$$V(x) = \beta^2$$

To generate a random number from an exponential distribution:

$$u_s = F(x)$$

So that:

$$x = F^{-1}(u_s)$$

Solve for x:

$$u_s = 1 - e^{-x/\beta}$$

$$u_s - 1 = -e^{-x/\beta}$$

$$\ln(1 - u_s) = -x/\beta$$

$$x = -\beta \ln(1 - u_s)$$

Notice that if u_s is a uniformly distributed random number between 0 and 1, then $1 - u_s$ is also a uniformly distributed random number between 0 and 1. Thus,

$$x = -\beta \ln(u_s)$$

is equivalent to the prior solution.

EXCEL:

```
= -$A$4 * LN( 1 - RAND() )
```

VBA:

```
Function Random_Exp( beta As Double ) As Double
    Random_Exp = -beta * Log(1 - Rnd())
End Function
```

E. Triangular Distribution

Parameters a, b, and m, the lower and upper bounds and the mode or most likely value, so that a ≤ m ≤ b.

Probability density:

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(b-a)(m-a)} & if\ a \le x \le m \\ \dfrac{2(b-x)}{(b-a)(b-m)} & if\ m \le x \le b \end{cases}$$

Cumulative distribution function F(x):

$$F(x) = \begin{cases} \dfrac{(x-a)^2}{(b-a)(m-a)} & if\ a \le x \le m \\ 1 - \dfrac{(b-x)^2}{(b-a)(b-m)} & if\ m \le x \le b \end{cases}$$

Expected value of x:

$$E(x) = \frac{a+b+m}{3}$$

Variance of x:

$$V(x) = \frac{a^2 + b^2 + m^2 - ab - am - bm}{18}$$

To generate a random number from a triangular distribution:

$$u = F(x)$$

So that:

$$x = F^{-1}(u)$$

Solve for x_s is standard triangular, where $a = 0$, $b = 1$, and where:

$$m_s = \frac{m-a}{b-a}$$

And, therefore:

$$x_s = \begin{cases} \sqrt{m_s u_s} & if\ u_s \le m_s \\ 1 - \sqrt{(1-m_s)(1-u_s)} & if\ u_s > m_s \end{cases}$$

So that x is triangular(a, b, m):

$$x = a + x_s(b-a)$$

EXCEL:

VBA:

```
Function STriangular( m As Double ) As Double

    Dim us As Double
    us = Rnd()

    If us < m Then
         …
    Else
         …
    End If

End Function


Function Triangular( a As Double, b As Double, m As Double ) As Double

    Dim ms As Double
    ms = (m - a) / (b - a)

    Triangular = a + STriangular( ms ) * (b - a)

End Function
```

F. Normal Distribution

Parameters μ and σ.

Probability density:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

Cumulative distribution function F(x):

$$F(x) = \text{approximation?}$$

The cdf of the *standard* normal distribution, where μ = 0 and σ = 1, is approximated in Excel in the NormsDist() function.

EXCEL:

```
=NORMSDIST( z )
```

VBA:

```
Function SNormCDF( z As Double ) As Double

    Dim a As Double, b As Double, c As Double, d As Double
    Dim e As Double, f As Double, x As Double, y As Double, z As Double

    a = 2.506628
    b = 0.3193815
    c = -0.3565638
    d = 1.7814779
    e = -1.821256
    f = 1.3302744

    If z > 0 Or z = 0 Then
         x = 1
    Else
         x = -1
    End If

    y = 1 / (1 + 0.2316419 * x * z)

    SNormCDF = 0.5 + x * (0.5 - (Exp(-z * z / 2) / a) * _
               (y * (b + y * (c + y * (d + y * (e + y * f))))))
End Function
```

Expected value of x:

$$E(x) = \mu$$

Variance of x:

$$V(x) = \sigma^2$$

To generate a random number from a normal distribution:

$$u = F(z)$$

So that:

$$z = F^{-1}(u)$$

Solve for x:

$$z = \text{approximation?}$$

Generating Random Numbers from the Standard Normal Distribution:

To generate a z_s, a random number drawn from the *standard* normal distribution, $\mu = 0$ and $\sigma = 1$.

EXCEL:

```
= NORMSINV( RAND() )
```

VBA:

There are three ways to generate standard normal random numbers. Here is the first way:

```
Function Random_SNorm1() As Double
    Dim u1 As Double
    Dim u2 As Double
    u1 = Rnd()
    u2 = Rnd()
    Random_SNorm1 = Sqr(-2 * Log(u1)) * Cos(2 * 3.1415927 * u2)
End Function
```

Here is the second way using an approximation to the Normal Inverse CDF:

```
Function SNorm_InverseCDF( p As Double ) As Double
    ‘ Dims are left out for brevity.
    a1 = -39.6968303
    a2 = 220.9460984
    a3 = -275.9285104
    a4 = 138.3577519
    a5 = -30.6647981
    a6 = 2.5066283
```

```
    b1 = -54.4760988
    b2 = 161.5858369
    b3 = -155.6989799
    b4 = 66.8013119
    b5 = -13.2806816

    c1 = -0.0077849
    c2 = -0.3223965
    c3 = -2.4007583
    c4 = -2.5497325
    c5 = 4.3746641
    c6 = 2.938164

    d1 = 0.0077847
    d2 = 0.3224671
    d3 = 2.4451341
    d4 = 3.7544087

    p_low = 0.02425
    p_high = 1 - p_low

    q = 0#
    r = 0#

    Select Case p

        Case Is < p_low
            q = Sqr(-2 * Log(p))
            SNorm_InverseCDF = (((((c1 * q + c2) * q + c3) * q + c4) _
                               * q + c5) * q + c6) / ((((d1 * q + d2) _
                               * q + d3) * q + d4) * q + 1)
        Case Is < p_high
            q = p - 0.5
            r = q * q
            SNorm_InverseCDF = (((((a1 * r + a2) * r + a3) * r + a4) _
                               *  r + a5) * r + a6) * q / (((((b1 * r _
                               + b2) * r + b3) * r + b4) * r + b5) * _
                               r + 1)
        Case Is < 1
            q = Sqr(-2 * Log(1 - p))
            SNorm_InverseCDF = -(((((c1 * q + c2) * q + c3) * q + c4) _
                               * q + c5) * q + c6) / ((((d1 * q + d2))_
                               * q + d3) * q + d4) * q + 1)
        End Select
End Function
```

```
Function Random_SNorm2() As Double
     Random_SNorm2 = SNorm_InverseCDF( Rnd() )
End Function
```

Here is the third way which works because of the central limit theorem. However, the previous two ways should be preferred.

```
Function Random_SNorm3() As Double
     Random_SNorm3 = Rnd + Rnd + Rnd + Rnd + Rnd + Rnd + Rnd + Rnd + _
                     Rnd + Rnd + Rnd + Rnd - 6
End Function
```

Generating Random Numbers from a Normal Distribution:

To generate a random number drawn from a normal distribution, with parameters μ and σ:

$$z = \mu + z_s \sigma$$

VBA:

```
Function Random_Norm( mu As Double, sigma As Double ) As Double
     Random_Norm = mu + Random_SNorm3() * sigma
End Function
```

G. Lognormal Distribution

Parameters μ_y and σ_y.

Probability density:

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} e^{\frac{-(\ln(x-\mu))^2}{2\sigma^2}}$$

Cumulative distribution function F(x):

$$F(x) = ?$$

Expected value of x:

$$E(x) = e^{\mu+\sigma^2/2}$$

Variance of x:

$$V(x) = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)$$

To generate a random number from a lognormal distribution:

EXCEL:

```
= EXP( NORMSINV( RAND() ) )
```

VBA:

```
Function Random_LogN() As Double
     Random_LogN = exp( Random_SNorm3() )
End Function
```

H. Generalized Inverse Transform Method

Probability density:

$$E.g.: \quad f(x) = .003x^2 \quad 0 \le x \le 10$$

Cumulative distribution function F(x):

$$F(x) = \int_0^x .003x^2 dx = .001x^3$$

Notice that this is a probability density because the total area under the curve from 0 to 10 is 1. That is:

$$F(x) = \int_0^{10} .003x^2 dx = 1$$

For the inverse transform method, we must solve for F^{-1}. We set $u = F(x)$ and solve for u.

$$u = F(x) = .001x^3$$

$$x = F^{-1}(u) = (1000 \cdot u)^{\frac{1}{3}}$$

To generate a random number from this probability density, if $u_s = .701$, then:

$$x = (1000 \cdot .701)^{\frac{1}{3}} = 8.88$$

Should we, on the other hand, wish to truncate this distribution and, say, generate a random number only between 2 and 5, then we must scale u_s over the range $F(2)$ to $F(5)$ thusly:

$$u = F(a) + u_s(F(b) - F(a))$$

$$u = F(2) + u_s(F(5) - F(2))$$

$$u = .008 + u_s(.125 - .008)$$

Again, if $u_s = .701$, then u = .090. The random number x drawn from $f(x)$ over the range $2 < x < 5$ is:

$$x = (1000 \cdot .090)^{1/3} = 4.48$$

Thus, we can generalize the inverse transform method as:

$$x = F^{-1}(u) = F^{-1}(F(a) + u_s(F(b) - F(a)))$$

IV. DISCRETE DISTRIBUTIONS

A. Bernoulli Trials

Parameter p.

Probability density:

$$p(X = x) = \begin{cases} 1 - p & for\ x = 0 \\ p & for\ x = 1 \end{cases}$$

Cumulative distribution function F(x):

$$F(x) = 1 - p$$

Expected value of x:

$$E(x) = p$$

Variance of x:

$$V(x) = p(1 - p)$$

To generate a random number from a Bernoulli distribution:

EXCEL:

```
= IF( RAND() < p, 1, 0 )
```

VBA:

```
Function Random_Bernoulli( p As Double ) As Double
    Dim u as Double
    u = Rnd()
    If u <= p Then
        Random_Bernoulli = 0
    Else
        Random_Bernoulli = 1
    End It
End Function
```

B. Binomial Distribution

Parameters p and n.

Number of successes in n independent trials, each with probability of success p.

Probability:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Where:

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}$$

Cumulative distribution function F(x):

$$F(x) = \sum_{i=0}^{\lfloor x \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$$

Expected value of x:

$$E(x) = np$$

Variance of x:

$$V(x) = np(1-p)$$

To generate a random number from a Binomial distribution?

EXCEL and VBA:

Series of Bernoulli Trials

Normal Approximation of the Binomial:

If n is large, then an approximation to B(n, p) is the normal distribution N($\mu = np$, $\sigma = np(1-p)$). The approximation gets better as n increases. So, to decide if n is large enough to use the normal, both np and $n(1-p)$ must be greater than 5.

C. Trinomial Distribution

Parameters p, q and n.

The obvious way to extend the Bernoulli trials concept is to add ties. Thus, each *de Moivre* trial yields success (x = 1), failure (x = -1), or a tie (x = 0). The probability of success is p, of failure q, and of a tie is 1 – p – q.

Probability:

$$P(X = x, Y = y) = \frac{n!}{x!\,y!(n-x-y)!}\,p^x q^y (1-p-q)^{n-x-y}$$

Expected value of x and y:

$$E(x) = np \quad \mathrm{E}(y) = nq$$

Variance of x:

$$V(x) = np(1-p) \quad V(y) = nq(1-q)$$

To generate a random number from a de Moivre trial:

$$x = \begin{cases} 1 & \text{if } \mathrm{u_s} \le \mathrm{p} \\ 0 & \text{if } \mathrm{p} \le \mathrm{u_s} \le \mathrm{p+q} \\ -1 & \text{if } \mathrm{p+q} \le \mathrm{u_s} \le 1 \end{cases}$$

EXCEL and VBA:

Series of de Moivre Trials

D. Poisson Distribution

Parameter λ.

Number of events that occur in an interval of time when events are occurring at a constant rate, say exponentially.

Probability:

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \quad for\ x \in \{0,1,...\}$$

Cumulative Distribution Function:

$$F(x) = e^{-\lambda}\sum_{i=0}^{\lfloor x \rfloor}\frac{\lambda^{i}}{i!} \quad if\ x \geq 0$$

Expected value of x and y:

$$E(x) = \lambda$$

Variance of x:

$$V(x) = \lambda$$

To generate a random number from a Poisson distribution:

Step 1: Let $a = e^{-\lambda}$, b = 1, and i = 0.
Step 2: Generate u_s and let $b = b \cdot u_s$. If b < a, then return x = 1. Otherwise, continue to Step 3.
Step 3: Set i = i + 1. Go back to Step 2.

VBA:

E. Empirical Distributions

A cumulative distribution function gives the probability than a random variable X is less that a given value x. So,

$$F(X) = P(X \leq x)$$

An empirical distribution uses actual data rather than a theoretical distribution function. In the table below, $n = 1000$ observations are made with $i = 4$ outcomes, $x_i = \{ 100, 200, 1000, 5000 \}$. The outcomes are sorted from low to high. Then calculated is the probability of each outcome, where the empirical probability, $P(x_i)$, is the ratio of the count of each outcome, n_x, to the total number of trials.

i	x_i	n_x	$P(x_i)$	$F(x_i)$	If $u_s = .71$,
1	100	500	.50	.50	
2	200	100	.10	.60	
3	1000	250	.25	.85	x = 1000
4	5000	150	.15	1.0	
		Σ = 1000	**Σ = 1.0**		

VBA:

```
Function Empirical() As Double

    Dim us As Double
    us = Rnd()

    Dim x As Double

    Select Case us
        Case Is < 0.5
            x = 100
        Case 0.5 To 0.6
            x = 200
        Case 0.6 To 0.85
            x = 1000
        Case 0.85 To 1
            x = 5000
    End Select

    Empirical = x

End Function
```

F. Linear Interpolation

Linear interpolation is a method of curve fitting using linear polynomials. If the coordinates of two points are known, the linear interpolant is the straight line between these points. For a point in the interval, the coordinate values, x and y, along the straight line are given by:

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - x_0}$$

Solving for *y* we get:

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

V. GENERATING CORRELATED RANDOM NUMBERS

What does correlated mean? Correlation is the tendency for two series of random numbers to diverge in tandem, above or below, from their respective means.

$$\rho_{x_i,x_2} = \frac{\sigma_{x_1,x_2}}{\sigma_{x_i}\sigma_{x_2}} = \frac{\sum_{i=1}^{n}((x_{1,i} - \mu_{x_1})(x_{2,i} - \mu_{x_2}))}{\sigma_{x_1}\sigma_{x_2}}$$

A. Bivariate Normal

Parameters μ_1, μ_2, σ_1, σ_2, and ρ.

Expected value of z_2 given z_1 is:

$$\mu_{2|1} = \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(z_1 - \mu_1)$$

Variance of z_2 given z_1 is:

$$\sigma_{2|1}^2 = \sigma_2^2(1 - \rho^2)$$

To generate correlated random numbers, z_1 and z_2, from two normal distributions:

$$z_1 = \mu_1 + z_{s(1)}\sigma_1$$

Then:

$$z_2 = \mu_{2|1} + z_{s(2)}\sigma_{2|1}$$

B. Multivariate Normal

Random multivariate normal numbers (i.e. correlated random numbers for normal distributions) can be generated by multiplying a vector of standard normal random numbers, Z_s, by the Cholesky decomposition, L, of the correlation matrix, C, as follows:

$$Z_{s*} = LZ_s$$

The Cholesky decomposition is in the lower left triangle and main diagonal of a square matrix. The elements in the upper right triangle are 0.

VBA:

```
Function Cholesky( mat As Range ) As Double()
     Dim A As Variant, L() As Double, S As Double
     Dim n As Double, m As Double
     A = mat
     n = mat.Rows.Count
     m = mat.Columns.Count
     If n <> m Then
         Cholesky = "?"
         Exit Function
     End If

     ReDim L(1 To n, 1 To n)
     For j = 1 To n
         S = 0
         For K = 1 To j - 1
             S = S + L(j, K) ^ 2
         Next K

         L(j, j) = A(j, j) – S

         If L(j, j) <= 0 Then Exit For

         L(j, j) = Sqr(L(j, j))

         For i = j + 1 To n
             S = 0
             For K = 1 To j - 1
                 S = S + L(i, K) * L(j, K)
             Next K
             L(i, j) = (A(i, j) - S) / L(j, j)
         Next i
     Next j
     Cholesky = L
End Function
```

Given the following correlation matrix:

$$C = \begin{bmatrix} 1 & -.503 & .183 \\ -.503 & 1 & -.803 \\ .183 & -.803 & 1 \end{bmatrix}$$

The Cholesky decomposition matrix of C is:

$$L = \begin{bmatrix} 1.000 & 0 & 0 \\ -.503 & .864 & 0 \\ .183 & -.823 & .538 \end{bmatrix}$$

Given a vector of standard normal random numbers (i.e. *z*'s):

$$Z_s = \begin{bmatrix} .517 \\ 1.516 \\ -.151 \end{bmatrix}$$

The vector of correlated standard normal random variables, Z_{s*}, is:

$$Z_{s*} = LZ_s = \begin{bmatrix} .517 \\ 1.050 \\ -1.234 \end{bmatrix}$$

To convert the individual z_{s*}'s in Z_{s*} to z_i's for each $z_i \sim N(\mu_i, \sigma_i)$:

$$z_i = \mu_i + z_{i*}\sigma_i$$

VI. MODELING REAL TIME FINANCIAL DATA

A process is a set of steps that converts inputs into outputs. There are many classifications of processes: continuous and discrete, stable and non-stable, stationary and dynamic, convergent and divergent, cyclical and non-cyclical, linear and non-linear, and deterministic and stochastic.

A stochastic process introduces randomness, to represent uncertainty around a central tendency of outcomes. This uncertainty is represented by a probability distribution. Given an initial state, there are many (maybe infinite!) sequences or paths a stochastic process could potentially generate. In a discrete time case, a stochastic process is what we call a time series. A stochastic process is said to be stationary if its probability distribution does not change over time. That is, if the probability distribution is normal, for example, then the mean and variance do not change over time. Sometimes, we may manipulate a set of financial data in an attempt to force stationarity.

A time series is a sequence of data where each sample or measurement occurs at a consistent time interval. Now, financial data is not generated by a process; it is generated by the interplay of buyers and sellers, supply and demand. But, we do attempt to model financial data with mathematical processes to aid in valuation and forecasting.

There are many such models. Some of the widely known models are: autoregressive (AR) models, moving average (MA) models, autoregressive moving average (ARMA), auto-regressive integrated moving average (ARIMA) models, and non-linear models such as autoregressive conditional heteroskedasticity (ARCH) and the GARCH family of models.

A diffusion process is a continuous-time Markov process that produces continuous random sequences, called sample paths. A stochastic differential equation (SDE) is a differential equation that includes at least one stochastic process, so that SDEs incorporate random movement, sometimes called noise, around a central tendency.

A martingale is a stochastic process where the expected value of a sample x_t, (i.e. at time t, given all prior samples up to x_{t-1}), is equal to x_{t-1}. A discrete-time martingale satisfies:

$$E(x_{t+1} \mid x_1,...,x_t) = x_t$$

A random-walk process is sometimes modeled as a Markov chain, where the past, present, and future states of a random variable are all independent. Thus, in a Markov process:

$$P(X_{x+1} = x \mid X_1 = x_1, X_2 = x_2,..., X_n = x_n) = P(X_{n+1} = x \mid X_n = x_n)$$

A stochastic process is a Markov process if the conditional probability distribution of future states depends solely upon the present state and is, therefore, independent of all prior states.

A continuous-time stochastic process is a Levy process if it starts at 0, contains jumps (which will be described later) and has stationary and independent increments. By stationary, we mean that the probability distribution of any segment of the random sequence, $x_T - x_t$ depends only upon the length of the time interval $T - t$. So, increments with equally long intervals must be identically distributed. A Wiener process is a continuous-time stochastic process where $x_T - x_t$ is normally distributed with $\mu = 0$ and $\sigma^2 = T - t$.

Brownian motion is another continuous-time stochastic process useful for modeling the prices of stocks and other financial instruments. In Brownian motion, the expected value of the next price is equal to the last price. Geometric Brownian motion is a continuous-time stochastic process where the logarithm of the random variable follows Brownian motion. It is also called a Wiener process. A stochastic process S_t is to be geometric Brownian motion if it satisfies the following stochastic differential equation:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

where W_t is a Wiener process and μ is the drift rate and σ the volatility. For an initial value S_0 the equation we can find a random value at some time t in the future:

$$S_t = S_0 e^{(\mu - \sigma^2/2)t + \sigma W_t}$$

Where S_t is a log-normally distributed with expected value:

$$\mathrm{E}(S_t) = S_0 e^{\mu t}$$

and variance:

$$\sigma_{S_t}^2 = S_0^2 e^{2\mu t}(e^{\sigma^2 t} - 1)$$

This conclusion can be verified using Itō's lemma, where the continuous rate of return $r = ln(S_t/S_0)$ is normally distributed. A Wiener process W_t has independent, normally distributed changes such that:

$$W_t = W_t - W_0 \sim N(0, t) \qquad (1)$$

That is, the expected value and variance of a Wiener process is:

$$E(W_t) = 0 \text{ and } V(W_t) = t \qquad (2)$$

This variance is important because is shows that the standard deviation is the square root of t, and so it is that stock volatility scales with the square root of time.

The following proof shows the connection to between μdt from geometric Brownian motion and the normally distributed, continuously compounding drift term (μ - ½ · σ^2) · t. Given the assumption that stock price follows geometric Brownian motion, with Wiener process W:

$$S_t - S_0 = dS_t = S_t(\mu dt + \sigma dW_t) \qquad (3)$$

This says that the change in the price of the stock is equal to the price of the stock times a mean drift rate times the change in time plus the standard deviation times some random variable.

The reason we use geometric Brownian motion to model stock price paths is because it encapsulates two widely observed phenomena in financial markets:

1. The long term trends of markets, represented by the mean term.

2. The white noise or random price movements or volatility around the trend, represented by the standard deviation term, which scales with the square root of time.

A. Financial Market Data

Exchanges provide for continuous quoting of bids and offers in shares or contracts listed on the exchange. From an automated trading perspective, a full quote consists of seven things: the symbol, the quantity on the highest bid price, the highest bid price, the lowest ask price, the quantity on the lowest ask price, the last price traded, and the quantity of the last trade (or the sum of the quantities of the consecutive trades on that price).

Symbol	Bid Qty	Bid Price	Ask Price	Ask Qty	Last Price	Last Qty
ES	195	125025	125050	456	125025	33

Quote 1

TICK

The term tick has different meanings in finance. The tick size is the minimum price increment—the minimum bid-ask spread—that is the difference between the highest bid and the lowest offer. In the example above the inside market (i.e. the highest bid and lowest ask) is 1250.25 to 1250.50, where the tick size is .25. The value of the minimum tick increment is not, in fact 25 cents; the contract size is much larger. In this sense, if a trader makes a tick profit on a trade, he has made .25 of a point. For the S&P 500 E-mini contract the value of a tick is $12.50. The whole-point value is often referred to as the handle. In the above example, 1250 is the handle. If the contract increases to 1254, we say the price is up four handles. In the case of this contract, a tick is a quarter handle.

A tick—as in tick data or uptick or downtick—refers to trade that has occurred on an exchange as been broadcasted to market particpants. In Quote 1 above, a tick would be reflected by a change in the Last Price and Last Qty fields. A days worth of tick data would contain all the information about all the trades that occurred that day. A tick in this sense contains at least four things:

- Ticker symbol
- Price
- Volume or quantity
- Time

Now, some of these data elements may be represented by a value, by a change from the previous value, or by a predefined increment. For example, given data representing a trade as follows:

Ticker Symbol	Price	Quantity	Time
ES	125025	33	13:43:12.100

Trade 1

Then, to save bandwidth, the *next* tick may show the trade data in an abbreviated format:

Ticker Symbol	Price	Quantity	Time
ES	-1	10	12

Trade 2

The new price of the new trade is 125000, because -1 means subtract 1 increment (e.g. .25) from the previous price. The time of the trade is 13:43:12.112, the time of the previous trade plus 12 milliseconds.

B. Modeling Tick Data

The goal is to simulate the arrival of tick data. Clearly, there are 3 stochastic processes at work: the price, the quantity, and the arrival time interval. To do this, we will use the exponential distribution to simulate the inter-arrival times of new ticks. We will use a trinomial distribution to simulate price movements—up, down, or sideways. And, we will use an empirical distribution to simulate quantities.

VBA:

```
Function deMoivre( p As Double, q As Double ) As Double

    Dim us As Double
    us = Rnd()

    Dim v As Double

    Select Case us

        Case Is < p
            v = 1
        Case p To p + q
            v = -1
        Case p + q To 1.0
            v = 0
    End Select

    deMoivre = v

End Function
```

```
Function Exponential( beta As Double ) As Double

    Exponential = Int(-beta * Log(1 - Rnd()) * 1000 + 1)

End Function
```

```
Function Empirical() As Double

    Dim us As Double
    us = Rnd()

    Dim v As Double

    Select Case us
        Case Is < 0.4
            v = 100
        Case 0.4 To 0.5
            v = 200
```

```
        Case 0.5 To 0.6
            v = 300
        Case 0.6 To 0.7
            v = 400
        Case 0.7 To 0.8
            v = 500
        Case 0.8 To 0.9
            v = 1000
        Case 0.9 To 0.925
            v = 1500
        Case 0.925 To 0.95
            v = 2000
        Case 0.95 To 0.975
            v = 5000
        Case 0.975 To 1
            v = 10000
    End Select

    Empirical = v

End Function
```

The initial time is 0, the initial price 50.00, and the initial quantity is 0. The Excel formula to generate millisecond arrival intervals is:

```
=Exponential( 0.5 )
```

The Excel formula to generate price movements is:

```
=B3 + deMoivre( 0.333, 0.333 ) * 0.01
```

The Excel formula to generate random quantity:

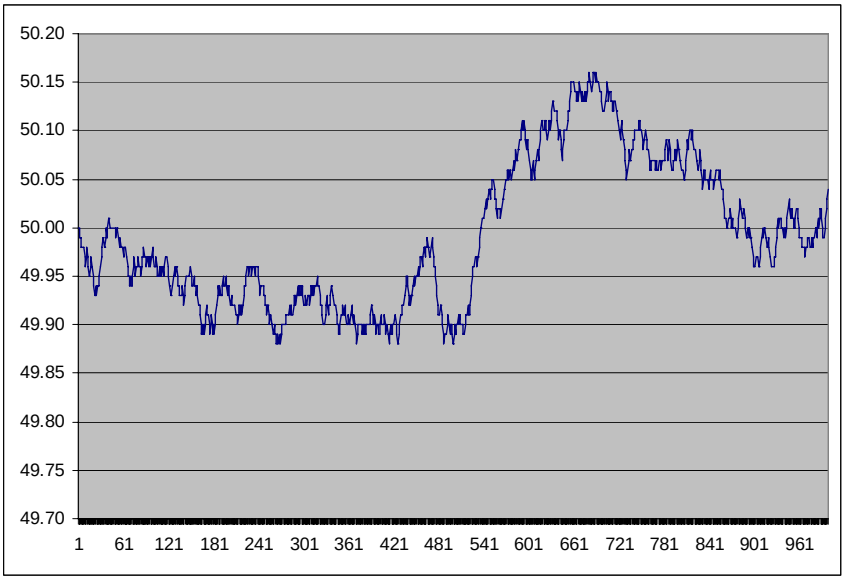
```
=Empirical()
```

These formulae generated the following random tick data:

Times	Price	Qty
0	50.00	0
667	50.00	300
1018	49.99	100
38	49.99	1000
84	49.98	100
651	49.98	2000
1599	49.98	100

47	49.98	300
100	49.97	100
861	49.96	100
287	49.97	100
68	49.98	500
95	49.97	100
503	49.96	200

The tick data produces the following chart:



Random Tick Data

Now, what we would like to do is convert the random tick data (with random inter-arrival times) into a time series of bars (with constant time intervals), where a bar contains four data elements over the time interval:

- Open Price
- High Price
- Low Price
- Closing Price

Using the tick data, there are a random number of ticks that occur over each one minute interval. (The number of ticks per interval follows a Poisson distribution.) By summing the inter-arrival times until the minute is over, i.e. Σ Times > 60000, we can find the Excel range of the ticks that occurred in that minute. The bar data is then:

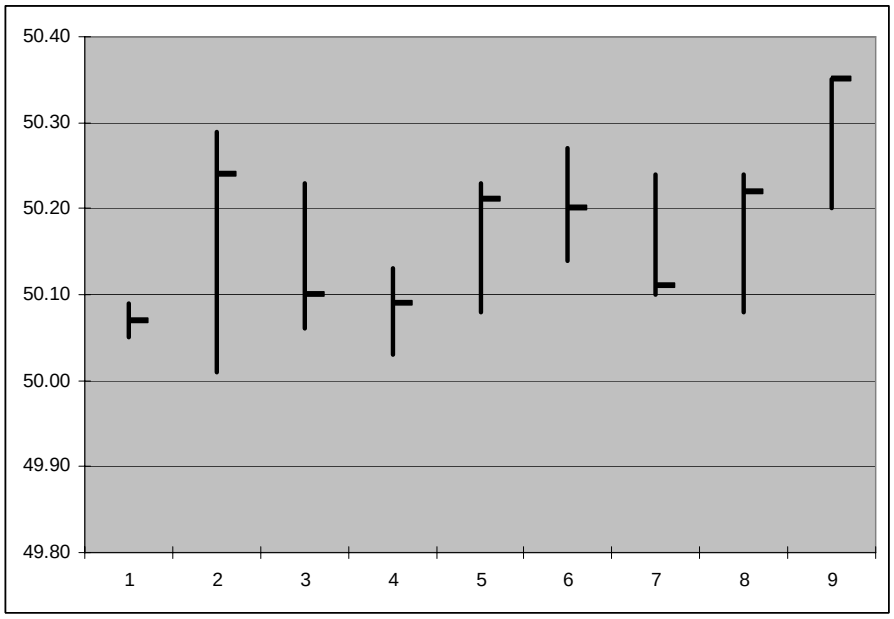
- Open Price = *first price in the range*
- High Price = MAX(*range*)
- Low Price = MIN(*range*)
- Closing Price = *last price of the range*

The tick data generated the following bars:

Open	High	Low	Close

50.00	50.09	50.05	50.07
50.06	50.29	50.01	50.24
50.23	50.23	50.06	50.10
50.11	50.13	50.03	50.09
50.08	50.23	50.08	50.21
50.21	50.27	50.14	50.20
50.20	50.24	50.10	50.11
50.10	50.24	50.08	50.22
50.21	50.35	50.20	50.35

The bar data produces the following chart:



Bar Data

This data can be used to find the continuous returns for each minute interval as per:

$$r_i = \ln(P_i / P_{i-1})$$

Where P_i equals the closing price for minute i.

Close	Returns
50.07	
50.24	0.0034
50.10	-0.0028
50.09	-0.0002
50.21	0.0024
50.20	-0.0002
50.11	-0.0018
50.22	0.0022
50.35	0.0026

These rates of return, *r*, are approximately normally distributed:

$$r \sim N(\mu, \sigma^2)$$

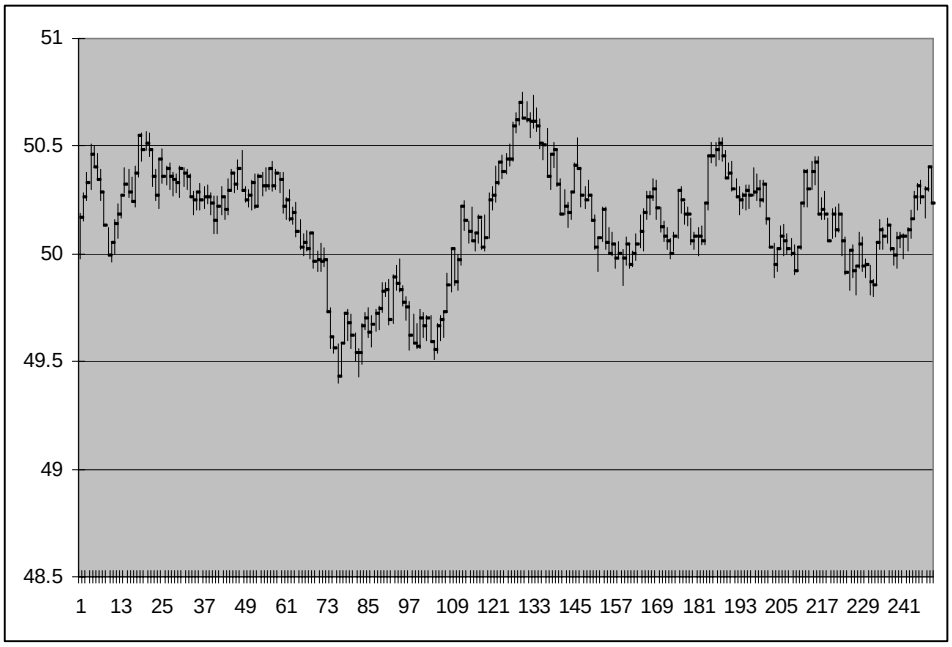
To demonstrate the approximation of returns to normality, we need to generate a lot more than 10 returns. The following code will generate around 250 bars.

```
Option Explicit
Option Base 1

Public Type Tick
    Time As Double
    Price As Double
    Quantity As Double
End Type

Sub Run_Simulation()

    Application.ScreenUpdating = False

    Randomize

    Dim ticks(30000) As Tick
    Dim p As Double
    p = 50

    Dim i As Integer
    Dim j As Integer

''''''''''''''''''''''''''
'' Create random ticks ''
''''''''''''''''''''''''''

    For i = 1 To 30000
         ticks(i).Time = Exponential(0.5)
         ticks(i).Price = p + deMoivre(0.333, 0.333) * 0.01
         ticks(i).Quantity = Empirical()
         p = ticks(i).Price
    Next i

'''''''''''''''''''''''''''
'' Create bars from ticks ''
'''''''''''''''''''''''''''

    Range("A1").value = "Open"
    Range("B1").value = "High"
    Range("C1").value = "Low"
    Range("D1").value = "Close"

    Dim count As Double
    Dim cursor As Double
    Dim time_sum As Double
```

```
    cursor = 1

    For j = 1 To 30000

        time_sum = time_sum + ticks(j).Time

        If (time_sum > 60000) Then
            Range("A2").Offset(count).value = ticks(cursor).Price
            Range("B2").Offset(count).value = Max(ticks, cursor, j - 1)
            Range("C2").Offset(count).value = Min(ticks, cursor, j - 1)
            Range("D2").Offset(count).value = ticks(j - 1).Price

            cursor = j
            count = count + 1
            time_sum = time_sum - 60000
        End If

    Next j

    Application.ScreenUpdating = True

End Sub
```

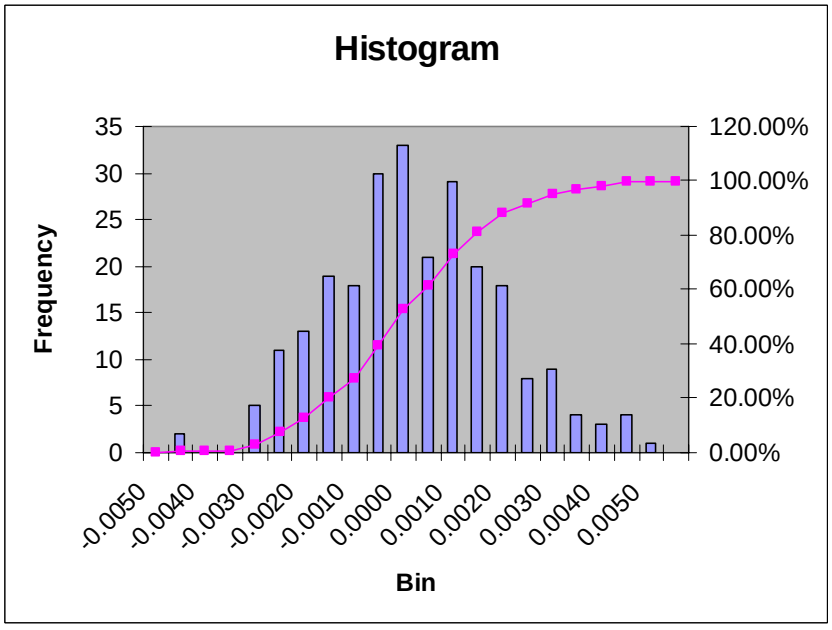
```
Function Max(ts() As Tick, start As Double, finish As Double) As Double

    Dim i As Integer
    Dim value As Double

    For i = start To finish
        If ts(i).Price > value Then
            value = ts(i).Price
        End If
    Next i

    Max = value

End Function

Function Min(ts() As Tick, start As Double, finish As Double) As Double

    Dim i As Integer
    Dim value As Double
    value = 9999999

    For i = start To finish
        If ts(i).Price < value Then
            value = ts(i).Price
        End If
    Next i

    Min = value

End Function
```

The code above created bars with the following chart:



Randomly generated bar data

The histogram of the log returns of the one minute time series appear to be approximately normally distributed:

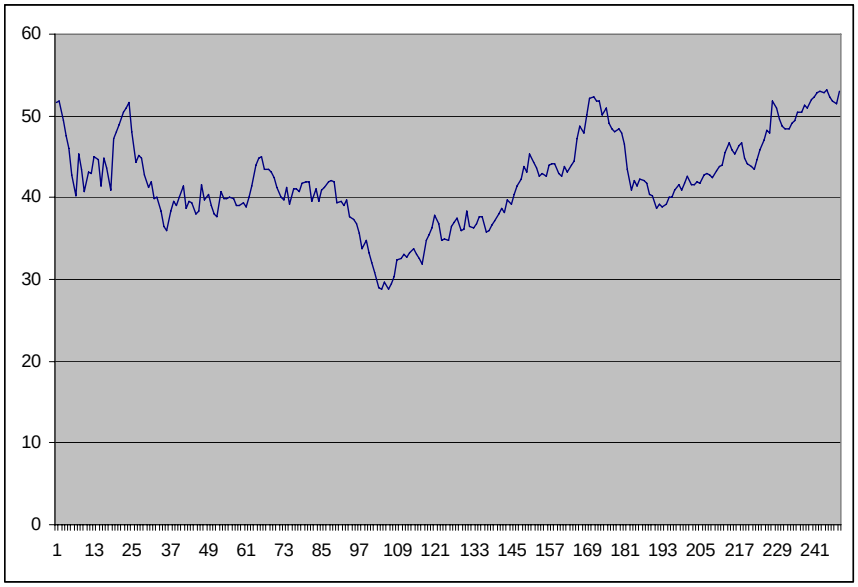


Distribution of log returns

The mean return was very close to 0 at .000005, and the standard deviation was .0017. Given more bar data, we would see the normal distribution even more clearly.

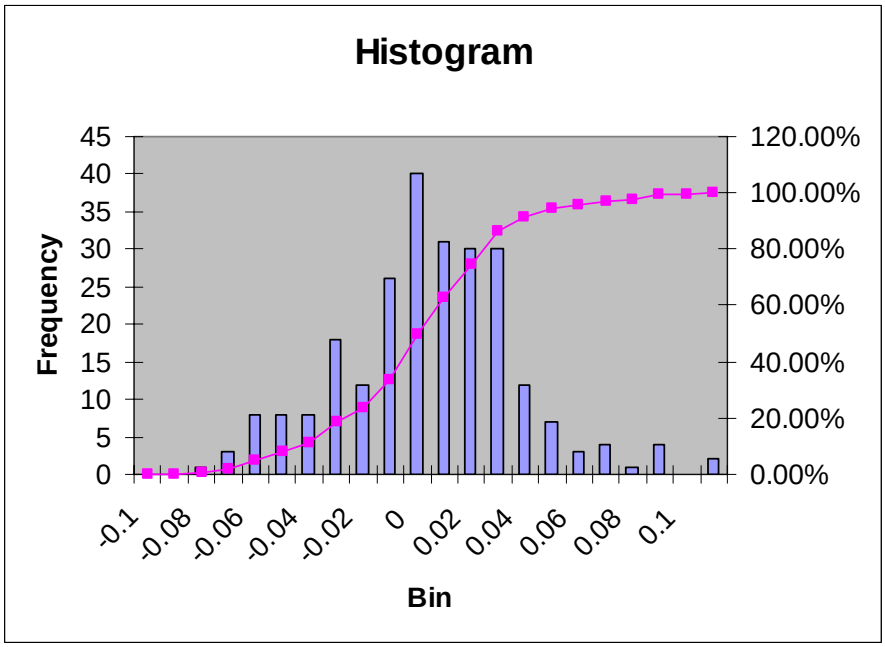
C. Modeling Time Series Data

Here is an actual chart using real daily closing price data of Boeing stock (symbol BA) from 10-02-2008 to 10-02-2009.



Price path of BA stock

The histogram for the daily log returns for BA over the year also appears to be approximately normally distributed:



Distribution of BA log returns

The mean log return is .00011 with a standard deviation .0335.

The goal now is to simulate price data without first creating ticks. To simplify, we will just generate closing prices. We can do this easily using the normal distribution.

$$S_{t+1} = S_t e^{\mu t + z_s \sigma \sqrt{t}}$$

If $t = 1$, then we can generate a one period ahead random price as per:

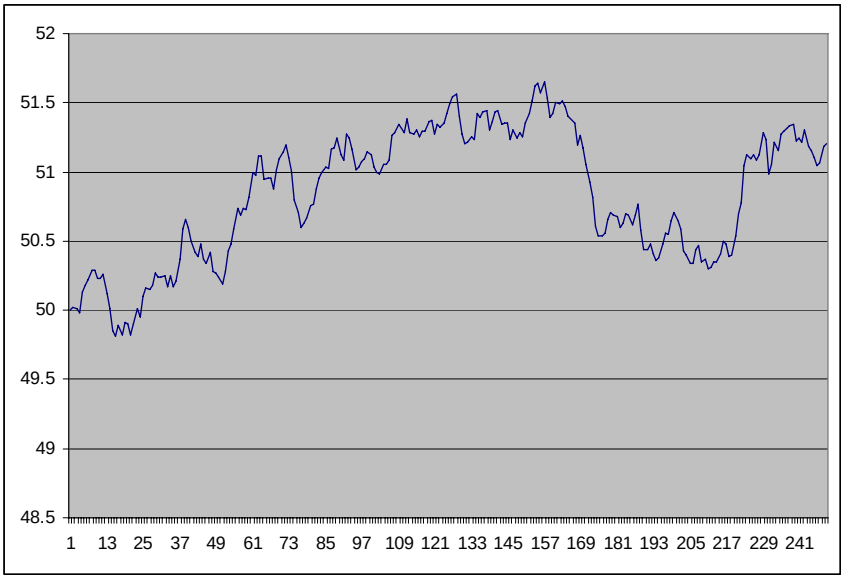
$$S_{t+1} = S_t e^{\mu + z_s \sigma}$$

Cell D2 contains the following Excel formula:

EXCEL: = D1 * EXP($B$1 + NORMSINV(RAND()) * $B$2)

	A	B	C	D
1	Mean	0.000005		50
2	StDev	0.0017		50.01666
3				49.94792
4				49.94042
5				49.90814

The chart output for this appears as follows:



Random price path

D. LAB 6: Augmenting the Simple Price Path Simulation

- **Odd lots** happen. How could we change our model to include occasional odd lot trades?
- The rate of trading activity is not constant. How could we change our model to incorporate the fact that trading volume is greater at the open and at the close of the trading day?

1. Fat Tails

The assumption in the previous model is that the standard deviation remains constant over the price path, which is almost certainly not representative of the real world. Stocks typically exhibit periods of low volatility and periods of high volatility, usually for exogenous or fundamental reasons—market shocks, earnings reports, etc. Fat tails are a property of financial data distributions relative to normal distributions which have thin tails. In an attempt incorporate extreme events with greater probability than a normal distribution would imply, mixture models that mix distributions together have been developed. As a simple example, the code below implements a jump process, where 1% of the time the width of the distribution generating the random returns increases from one standard deviation, or volatility, to another.

Now, for simplicity, let's set $\mu \approx 0$, so that:

$$S_t = S_{t-1} e^{z_s \sigma}$$

VBA:

```
Function Mixture_of_Normals( price As Double, vol1 As Double, _
                             vol2 As Double ) As Double
    Dim us As Double
    us = Rnd()

    If us < 0.99 Then
        Mixture_of_Normals = Random_Price(price, vol1)
    Else
        Mixture_of_Normals = Random_Price(price, vol2)
    End If

End Function


Function Random_Price( price As Double, vol As Double ) As Double

    Random_Price = price * Exp(mu + Application.NormSInv(Rnd()) * vol)

End Function
```

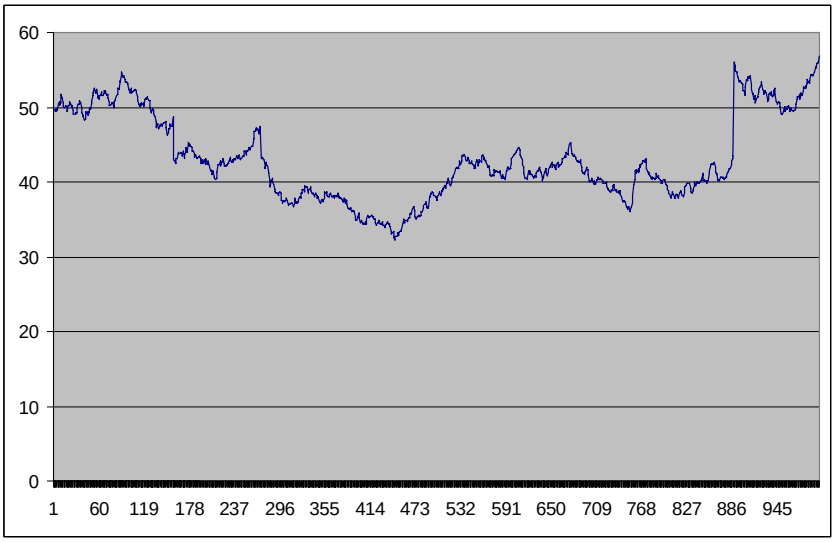
	A	B	C	D
1	Mean	0		50
2	StDev 1	0.01		49.50734
3	StDev 2	0.10		49.43878
4				49.7491
5				49.68668

In cell D2 and copied down is the following formula:

EXCEL:

```
= Mixture_Of_Normals( D1, $B$2, $B$3 )
```

This code produced the following sample price path:



Random price path with jumps

A histogram of the log returns will show a normal-like distribution but with fatter tails.

2. Stochastic Volatility Models

Stochastic volatility models treat volatility as a random process, governed by state variables, and may include a serial correlation and a tendency to revert to a long-run mean value. Stochastic volatility models are one way to resolve the shortcoming of many financial models that assume constant volatility over some time horizon, which is certainly a contradiction of widely observed phenomena. Stochastic volatility models are used to value and manage risk associated with derivative instruments.

The **Heston model**, proposed by Dr. Steven Heston (currently of the University of Maryland), represents the evolution of volatility by assuming that it is not constant, but is rather itself a random variable. The basic model assumes that the price S_t follows:

$$dS_t = \mu \cdot S_t \cdot dt + \sqrt{\nu_t} \cdot S_t \cdot dW_t^S$$

where ν_t, the instantaneous variance, is a Cox-Ingersol-Ross (CIR) process:

$$d\nu_t = \kappa \cdot (\theta - \nu_t) \cdot dt + \xi \cdot \sqrt{\nu_t} \cdot dW_t^\nu$$

Where dW_t^S and dW_t^ν are Wiener processes, or random walks, with correlation ρ. The parametersare:

- μ is the rate of return of the asset.

- θ is the **long-run variance.** That is, as t tends to infinity, the expected value of ν_t tends to θ.
- κ is the rate at which ν_t reverts to θ.
- ξ is the **volatility of volatility.** That is, the variance of ν_t.

a. ARCH(1)

An autoregressive conditional heteroscedasticity (ARCH) model considers the variance of the current period return, (i.e. the error term) to be a function of the prior period squared errors. We use it finance, to account for volatility clustering, i.e. periods of high volatility tending to be followed by periods of low volatility. The ARCH(1) equation is:

$$\hat{\sigma}_{t+1}^2 = \gamma + \alpha r_t^2$$

b. GARCH(1,1)

An generalized autoregressive conditional heteroscedasticity (GARCH) model considers the variance of the current period return, (i.e. the error term) to be a function of the prior period squared errors and the prior period estimated of variance. We use it finance, to account for volatility clustering, i.e. periods of high volatility tending to be followed by periods of low volatility. The GARCH(1,1) equation is:

$$\hat{\sigma}_{t+1}^2 = \gamma + \alpha r_t^2 + \beta \hat{\sigma}_t^2$$

Here is an Excel implementation of the GARCH(1,1) formula:

	A	B	C	D	E	F
1	**gamma**	0.0001	**Price**	**Return**	**GARCH**	**Vol**
2	**Alpha**	0.4	50			
3	**Beta**	0.6	51	0.0198	0.01	**0.1**
4			53.44	0.0467	0.0062	**0.0788**
5			53.97	0.0098	0.0044	**0.0669**
6			50.57	-0.0651	0.0028	**0.0531**

Where, cells C2 and C32 contain initial prices, cell D3 (and copied down) contains the formula:

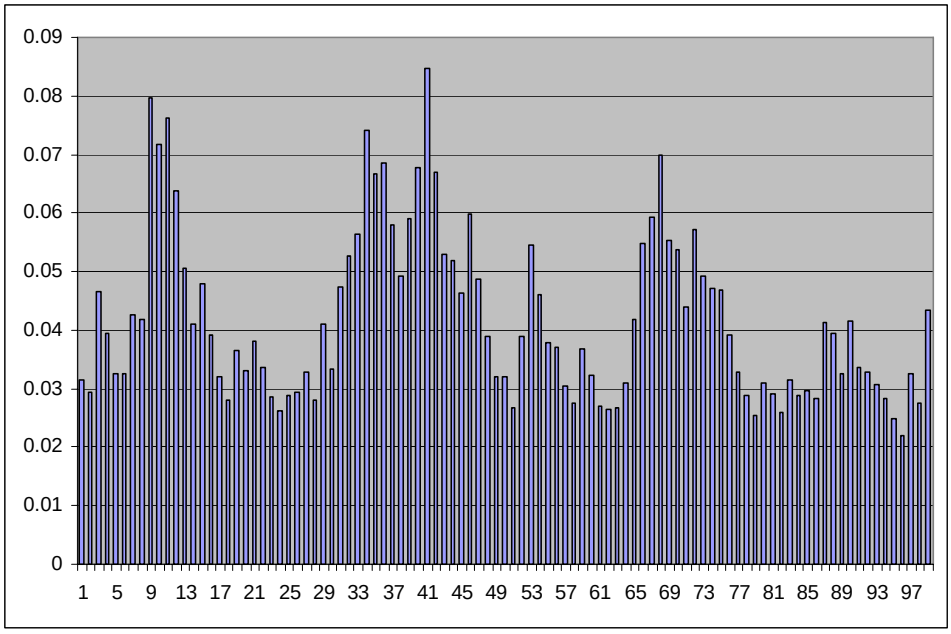
```
= LN( C3 / C2 )
```

Cell E3 is an initial value for the GARCH forecast and column F is the square root of column E. The GARCH equation in E4 and copied down is:

EXCEL:

```
= $B$1 + $B$2 * D3 ^ 2 + $B$3 * E3
```

The following chart, of the data in column F, shows the kind of volatility clustering commonly observed in markets.



Volatility Clustering

b. Estimating Volatility

Often when analyzing financial data, we often estimate volatility over a period of time in the past. This is easily done if we have a time series of price data, where we use log returns to calculate the standard deviation of returns. How though do we estimate volatility given only one day of data? We estimate one day standard deviation using **close-to-close** data as follows:

$$\sigma_{CC} = \sqrt{\left[\ln\left(\frac{C_i}{C_{i-1}}\right)\right]^2}$$

However, this method certainly does not capture all of the information of intra-day volatility. A stock could close at 50 one day, gap open to 53 the following day, trade down to 44 and close back at 50. In this case, using this close-to-close calculation would not be a very good indicator of volatility since "0" is not a good description of what happened.

To better account for one-period volatility, other, more efficient methods have been proposed which use intra-period highs and lows to estimate volatility. These methods are often grouped under the term **extreme value estimators**. Since several models that we use in financial markets are based on the assumption of continuous time, it is more intuitive to examine the entire time period rather than simply the ends. The

most well-known of the extreme value estimators have been proposed by **Parkinson** (1980), and **Garman and Klass** (1980). The Parkinson's equation uses the intra-period high and low thusly:

$$\sigma_P = .601\sqrt{\left(\ln\frac{H_i}{L_i}\right)^2}$$

The Garman Klass estimator, which the intra-period high and low as well as the open and close data, has the form:

$$\sigma_{GK} = \sqrt{\left[\frac{1}{2}\left(\ln\frac{H_i}{L_i}\right)^2 - (2\ln(2)-1)\left(\ln\frac{C_i}{O_i}\right)^2\right]}$$

Notice that these equations represent and estimate of the one period historical volatility of the underlying. You may notice, however, that neither of these models take into account gaps, either up or down, from the previous days close. Volatility that happens over night will in neither of these models be accounted for. For this and other reasons there are dozens of variations of these two extreme value estimators currently in use.

VII. MODELING OPTIONS

A call option on a stock grants the owner of that option the right to buy the underlying stock at the strike price at expiration (in the case of a European option), or at any time up to expiration (in the case of an American option). The writer of a call option is then obligated to sell the stock at the strike price when the owner of the call option exercises his or her right.

A put option on a stock grants the owner of that option the right to sell the underlying stock at the strike price at expiration, or at any time up to expiration. The writer of a put option is then obligated to buy the stock at the strike price when the owner of the put option exercises his or her right.

Before we begin to discuss option pricing, lets quickly review an important relationship —put-call parity. Put-call parity states that the value of a call at a given strike implies a certain value for the corresponding put. If there is divergence from parity, then an arbitrage opportunity would exist and trading could take risk-free profits until the divergence from parity is eliminated.

Put-call parity states that the price of the stock, the price of a call at a given strike and expiration, and the price of a put with that same strike and expiration, are all interrelated and simultaneously solved according to:

$$C + Xe^{-rt} = P + S_0$$

Where, C is the call price, P the put price, X the strike, S the stock price, r the interest rate and t the time till expiration.

A. The Simulation Way

Recall that the t time ahead price can be simulated in the following way:

$$S_t = S_0 e^{\mu t + z_s \sigma \sqrt{t}}$$

Also, the call option payoff at time t can be expressed as:

$$C = \max(S_t - X, 0)$$

Where X is the strike price of the option. Using VBA we generate many simulations of the stock price to generate many call option payoffs.

VBA:

```
Function Sim_Eur_Call(S As Double, X As Double, r As Double, _
                      t As Double, sigma As Double) As Double

   Dim sum_payoffs As Double
   Dim i As Integer

   For i = 1 To 1000
       ST = S * Exp(Application.NormSInv(Rnd) * sigma * Sqr(t))
       sum_payoffs = sum_payoffs + Max(ST - X, 0#)
   Next i

   Sim_Eur_Call = Exp(-r * t) * (sum_payoffs / 1000)

End Function
```

```
Function Max(a As Double, b As Double) As Double
    If a >= b Then
        Max = a
    Else
        Max = b
    End If
End Function
```

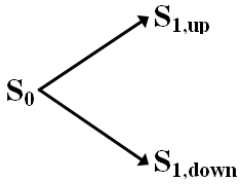
B. The Cox-Ross-Rubenstein (CRR) or Binomial Way

Recall that the binomial distribution considers two outcomes—success or failure. Lets assume that with stocks, success means the stock goes up, and failure means the stock goes down. But how much does it go up or down?

We can calculate the magnitude of an up moves and down moves using the annualized volatility of the stock as:

$$Up = e^{\sigma\sqrt{\Delta t}} \qquad \text{and} \qquad Down = e^{-\sigma\sqrt{\Delta t}}$$

Given a Bernoulli trial, the stock price can move up by a factor of *Up* or down by a factor of *Down*.



$$S_{1,Up} = S_0 \cdot Up \qquad \text{and} \qquad S_{1,Down} = S_0 \cdot Down$$

Now, the present value before the Bernoulli trial (i.e. $t = 0$) must be the sum of the two discounted prices (i.e. up and down) after the Bernoulli trial, so that:

$$S_0 = q\_up \cdot S_{1,Up} + q\_down \cdot S_{1,Down}$$

If r is 1 plus the interest rate, then solving the system of linear equations yields discount factors of:

$$q\_up = \frac{r - Down}{r \cdot (Up - Down)} \qquad \text{and} \qquad q\_down = \frac{Up - r}{r(Up - Down)}$$

After each trial, the test is:

$$C = q\_up \cdot Max(S \cdot Up - X, 0) + q\_down \cdot Max(S \cdot Down - X, 0)$$

For a European call on a non-dividend paying stock, given n number of trials, the call price follows the binomial distribution:

$$C = \sum_{i=1}^{n} \binom{n}{i} \cdot q\_up^{\,i} \cdot q\_down^{\,n-i} \cdot Max(S \cdot Up^{\,i} \cdot Down^{\,n-i} - X, 0)$$

Where the number of combinations is:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

In the following simple code example, the change in time is 1 day. So, if time t is set to .75, then there will be 273 Bernoulli trials (i.e. 273 / 365 = .75).

VBA:

```
Function Bin_Eur_Call(S As Double, X As Double, r As Double, _
                      t As Double, sigma As Double) As Double

    Dim Periods As Double
    Dim rf As Double
    Dim Up As Double
    Dim Down As Double
    Dim q_up As Double
    Dim q_down As Double

    Periods = Int(t * 365)

    rf = Exp(r / 365)

    Up = Exp(sigma * Sqr(1 / 365))
    Down = Exp(-sigma * Sqr(1 / 365))

    q_up = (rf - Down) / (rf * (Up - Down))
    q_down = 1 / rf - q_up

    Bin_Eur_Call = 0

    For i = 0 To Periods
        Bin_Eur_Call = Bin_Eur_Call + Application.Combin(Periods, i) _
                       * q_up ^ i * q_down ^ (Periods - i) _
                       * Application.Max(S * Up ^ i * Down ^ _
                       (Periods - i) - X, 0)
    Next i

End Function
```

C. The Black-Scholes Way

The simulation and binomial methods for pricing options both approach the normal distribution. The Black Scholes formulae for calls *C* and puts *P* assumes a normal distribution of returns.

$$C = S \cdot N(d_1) - X \cdot e^{-rt} \cdot N(d_2)$$
$$P = X \cdot e^{-rt} \cdot N(-d_2) - S \cdot N(-d_1)$$

Where: $d_1 = \dfrac{\ln(S/X) + (r + \sigma^2/2) \cdot t}{\sigma \cdot \sqrt{t}}$ and $d_2 = d_1 - \sigma \cdot \sqrt{t}$

VBA:

```
Public Function BS_Eur_Call(S As Double, X As Double, r As Double, _
                            t As Double, sigma As Double) As Double

    Dim d1 As Double
    Dim d2 As Double

    d1 = (Log(S / X) + (r + sigma ^ 2 / 2) * t) / (sigma * Sqr(t))
    d2 = d1 - sigma * Sqr(t)

    BS_Eur_Call = S * Application.NormSDist(d1) - X * Exp(-r * t) * _
                  Application.NormSDist(d2)

End Function
```

D. Option Greeks

The **Greeks—delta, gamma, theta, vega** and **rho—**represent the sensitivities of the price of an option to changes in the input parameters.

Delta measures the rate of change of the option price given a change in the price of the underlying asset. It is the first derivative with respect to *S*. For a European call on a non-dividend paying stock, delta is:

$$Delta = N(d_1)$$

Gamma measures the rate of change in the delta given a change in the price of the underlying asset. It is the second derivative with respect to *S*. For a European call (or put) on a non-dividend paying stock, gamma is:

$$Gamma = \frac{N'(d_1)}{S \cdot \sigma \cdot t}$$

Theta measures the rate of change of the option price given a change in the time to expiration. It is the first derivative with respect to time. For a European call on a non-dividend paying stock, theta is:

$$Theta = \frac{S \cdot N'(d_1) \cdot \sigma}{2\sqrt{t}} - r \cdot X \cdot e^{-rt} \cdot N(d_2)$$

Vega measures the rate of change of the option price given a change in volatility of the underlying asset. It is the first derivative with respect to sigma. For a European call (or put) on a non-dividend paying stock, vega is:

$$Vega = S \cdot \sqrt{t} \cdot N'(d_1)$$

Rho measures the rate of change of the option price given a change in interest rates. It is the first derivative with respect to the rate. For a European call on a non-dividend paying stock, rho is:

$$Rho = X \cdot t \cdot e^{-rt} \cdot N(d_2)$$

Where the standard normal pdf is:

$$N'(x) = \frac{1}{\sqrt{2\pi}} \cdot e^{-x^2/2}$$

Sensitivities in finance are also often found using **perturbation.**

E. Implied Volatility

Often, the price of an option in the market place is observed, but the volatility that the price implies is unknown. To find the volatility level that generates a given price we can use an iterative bisection method.

VBA:

```
Function Implied_Vol(S As Double, X As Double, r As Double, _
                     t As Double, price As Double) As Double

    Dim High As Double
    Dim Low As Double
    Dim test_price As Double
    Dim test_vol As Double

    High = 1
    Low = 0

    Do While (High - Low) > 0.00001

        test_vol = (High + Low) / 2
        test_price = BS_Eur_Call(S, X, r, t, test_vol)

        If (test_price > price) Then
            High = test_vol
        Else
            Low = test_vol
        End If
    Loop
    Implied_Vol = test_vol
End Function
```

F. American Options

```
Function Bin_Amer_Call(S As Double, X As Double, r As Double, _
                            t As Double, sigma As Double) As Double
    Dim Periods As Double
    Dim rf As Double
    Dim Up As Double
    Dim Down As Double
    Dim q_up As Double
    Dim q_down As Double

    Periods = Int(t * 365)

    rf = Exp(r / 365)

    Up = Exp(sigma * Sqr(1 / 365))
    Down = Exp(-sigma * Sqr(1 / 365))

    q_up = (rf - Down) / (rf * (Up - Down))
    q_down = 1 / rf - q_up

    Dim OptionReturnEnd() As Double
    Dim OptionReturnMiddle() As Double
    ReDim OptionReturnEnd(Periods + 1)

    For i = 0 To Periods
          OptionReturnEnd(i) = Max(S * Up ^ i * Down _
                              ^ (Periods - i) - X, 0)
    Next i

    For i = Periods - 1 To 0 Step -1
        ReDim OptionReturnMiddle(i)
        For j = 0 To i
           OptionReturnMiddle(j) = Max(S * Up ^ j * Down ^ (i - j) _
                                  - X, q_down * OptionReturnEnd(j) _
                                  + q_up * OptionReturnEnd(j + 1))
        Next j
        ReDim OptionReturnEnd(i)
        For j = 0 To i
            OptionReturnEnd(j) = OptionReturnMiddle(j)
        Next j
    Next i
    Bin_Amer_Call = OptionReturnMiddle(0)
End Function
```

VIII. OPTIMIZATION

Optimization is a field of study that focuses on methods for minimizing or maximizing the value of objective functions. Finding the optimal solution may not be difficult. One could guess every possible solution and pick the one with the most favorable output. But, finding the solution in a computationally efficient way is very difficult, and very interesting to management scientists. Thus, optimization focuses on systematic ways of choosing input values that lead to the optimal solution in the minimum number of steps.

Maximize the objective function: $f(x)$

Subject to the constraints:

Linear programming (**LP**) problems involve the optimization of a linear objective function, subject to linear equality and inequality constraints. Linear programs can be expressed in the form:

$$\textbf{Maximize:} \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n \qquad \text{or,} \qquad \textbf{Maximize:} \quad c^Tx$$

$$\textbf{Subject to:} \quad \begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\le b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\le b_1 \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\le b_1 \end{aligned} \qquad \textbf{Subject to:} \quad Ax \le b$$

where **x** is the vector of input variables. **c** is a vector of known coefficients. **b** is the right hand side vector. **A** is the matrix of constraint coefficients. The objective function is $\mathbf{c}^T\mathbf{x}$. The **simplex** algorithm finds numerical solutions to linear programming problems.

Nonlinear programming (**NLP**) is the process of solving maximization and minimization problems where some of the constraints or the objective function are nonlinear. Non-linear optimization problems can be solved by using gradient ascent (or descent) to find points where the gradient of the objective function is zero. Excel Solver uses the **Generalized Reduced Gradient** (GRG) algorithm for nonlinear problems.

The hardest optimization problems to solve are discontinuous and/or non-smooth problems, where multiple feasible regions may exist each with their own locally optimal solution. Excel's Premium Solver uses an **evolutionary genetic algorithm** to solve for the global optimum.

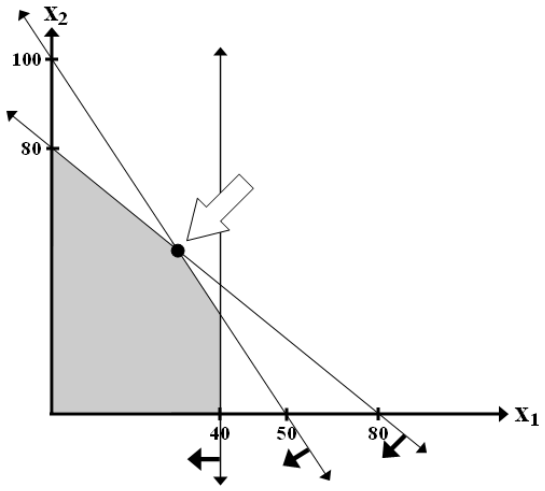
Problems with optimization: no solutions, infinite solutions, one solution. Is it global or just local?

A. Linear Optimization

If we represent the following linear program on a graph:

Maximize:	$3x_1 + 2x_2$	
Subject to:	$2x_1 + x_2$	≤ 100
	$x_1 + x_2$	≤ 80
	x_1	≤ 40
	$x_1, x_2 \geq 0$	

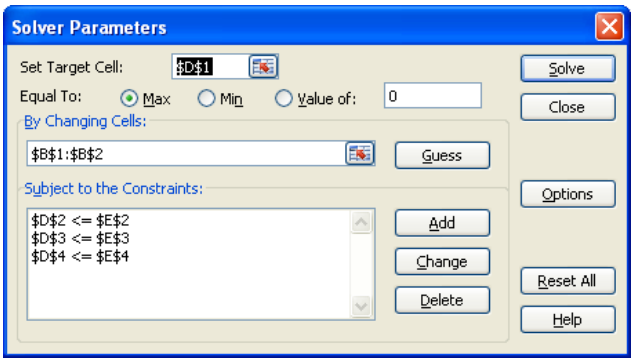
We see that the set of **feasible solutions** is shown in the shaded area.



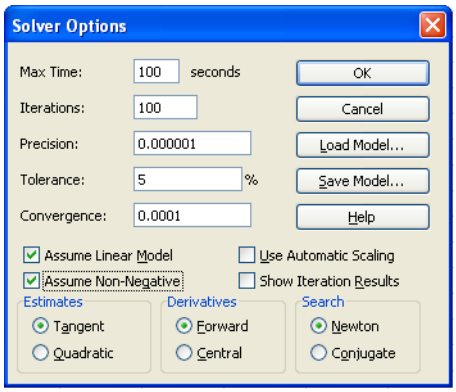
In Excel:

	A	B	C	D	E
1	x_1	20	**Maximize:**	180	
2	x_2	60	**Subject to:**	100	100
3				80	80
4				20	40

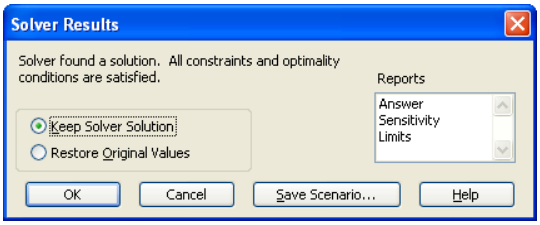
Open the Solver window via Data | Solver. Set the problem parameters in Solver:



In the Solver window, click on Options. In the Options window click on Assume Linear Model and Assume Non-Negative as shown:



Click OK. Then in the main Solver window, click OK.



Notice on the graph above, that the objective function is maximized (at 180) and all constraints are satisfied when $x_1 = 20$ and $x_2 = 60$. At the optimal solution, the first two constraints are **binding**. The optimal point always occurs at the intersection of constraints, or a corner point.

B. LAB 7: Nonlinear Optimization

- **Use Solver to find the optimal values for:**

Maximize:	$x_1^2 + x_2$	
Subject to:	$x_1 + x_2^2$	≥ 3
	$x_1^2 + x_2^2$	≤ 4

C. Efficient Portfolios

An **efficient portfolio** is the set of stock positions that generates the lowest variance of returns given a particular level of expected return. Or, it is the set of positions that generates the highest return given a particular level of risk. That is:

Minimize:	$\sigma^2 = \omega^T \Omega \omega$	or,	**Maximize:**	$E(r) = \omega r$
Subject to:	$E(r) \geq \omega r$		**Subject to:**	$\sigma^2 \geq \omega^T \Omega \omega$

The **efficient frontier** is the set of all efficient portfolios.

Minimize: $\dfrac{E(r) - c}{\sigma_p}$

Subject to: $\sum \omega_i = 1$

$\omega_i \geq 0$

For example, given the following covariance matrix:

	A	B	C	D	E
1		Covariance Matrix			
2		IBM	WMT	XOM	T
3	IBM	0.00012	0.00004	0.00008	0.00006
4	WMT	0.00004	0.00008	0.00005	0.00004
5	XOM	0.00008	0.00005	0.00013	0.00007
6	T	0.00006	0.00004	0.00007	0.00009

And the following expected returns:

	F	G
1		Expected
2		Return
3	IBM	0.00048
4	WMT	0.00007
5	XOM	0.00033
6	T	0.00036

The following are also included:

	B	C	D
8	c	0.001	
9			
10	Weights		
11	IBM	1	
12	WMT	0	
13	XOM	0	
14	T	0	
15	Sum	1	

16			
17	**Portfolio Mean**		0.000484
18	**Portfolio Sigma**		0.011167
19	**Theta**		-0.04619

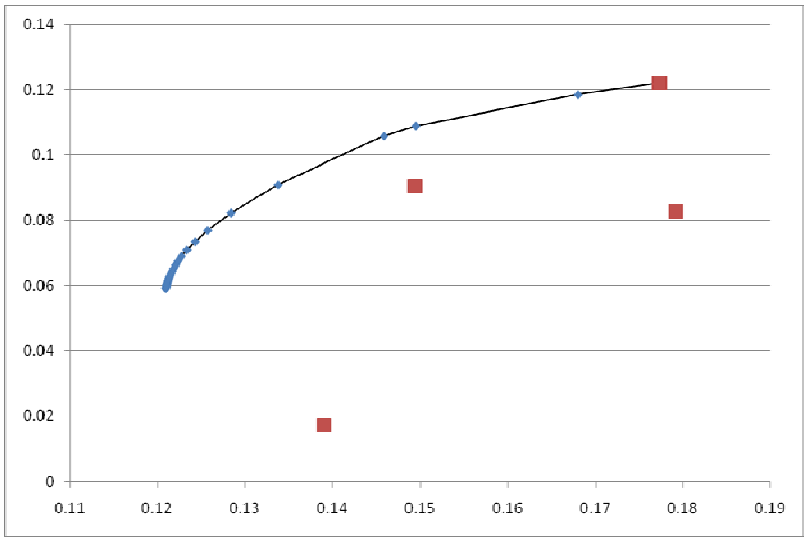
The Excel code for the Portfolio Mean, Portfolio Sigma and Theta is:

```
D17:  =MMULT(TRANSPOSE(C11:C14),G3:G6)
D18:  =SQRT(MMULT(TRANSPOSE(C11:C14),MMULT(B3:E6,C11:C14)))
D19:  =(D17-C8)/D18
```

The VBA code to solve the optimization problem iteratively is:

```
Sub Solve()

    Application.ScreenUpdating = False

    'Range("I2:O36").ClearContents

    For i = 1 To 35
        Range("C8").Value = (-0.006 + i * 0.0002)

'''''''''''''''''''' Solver Code ''''''''''
  SolverReset
  SolverOk SetCell:="$D$19", MaxMinVal:=1, ValueOf:="0", ByChange:="$C$11:$C$14"

  SolverAdd CellRef:="$C$15", Relation:=2, FormulaText:="1"
  SolverAdd CellRef:="$C$11:$C$14", Relation:=3, FormulaText:="0"

  SolverSolve UserFinish:="True"
''''''''''''''''''''''''''''''''''''''''''''''

        Range("I1").Offset(i, 0).Value = Range("C8").Value
        Range("J1").Offset(i, 0).Value = Range("D17").Value * 252
        Range("K1").Offset(i, 0).Value = Range("D18").Value * Sqr(252)
        Range("L1").Offset(i, 0).Value = Range("C11").Value
        Range("M1").Offset(i, 0).Value = Range("C12").Value
        Range("N1").Offset(i, 0).Value = Range("C13").Value
        Range("O1").Offset(i, 0).Value = Range("C14").Value
    Next i

   Application.ScreenUpdating = False

End Sub
```

The VBA macro generates the following chart:


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0.11
0.12
0.13
0.14
0.15
0.16
0.17
0.18
0.19

D. Capital Budgeting

A firm has $60M budget to fund six projects under consideration. The projects have the following expected cash flows:

	A	B	C	D	E	F	G
1	Rate	.05					
2	Year	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6
3	1	(5.00)	(5.00)	(15.00)	(1.00)	(40.00)	(2.00)
4	2	(7.00)	(10.00)	10.00	(2.00)	20.00	(3.00)
5	3	(10.00)	6.00	7.00	(8.00)	15.00	(8.00)
6	4	10.00	6.00	2.00	5.00	10.00	8.00
7	5	10.00	5.00	1.00	5.00	10.00	5.00
8	6	5.00	3.00	-	3.00	5.00	3.00
9	7	2.00	-	-	2.00	-	1.00
10	NPV	1.47	2.44	3.26	2.01	12.80	1.91

The investments are by percentage:

12	Investment	0%	100%	0%	100%	30%	100%

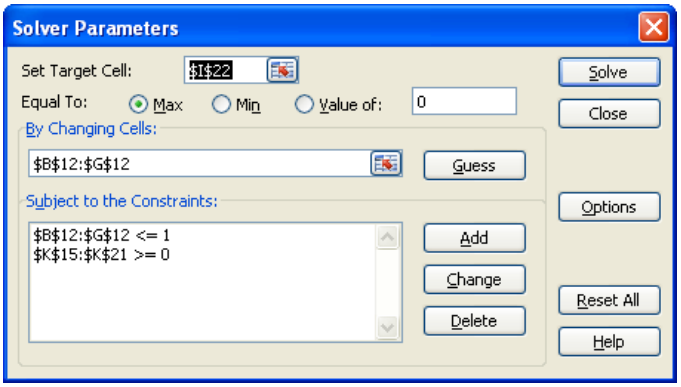
The weighted cash flows are cash flows times the weights:

14	Year	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6
15	1	-	(5.00)	-	(1.00)	(12.00)	(2.00)
16	2	-	(10.00)	-	(2.00)	6.00	(3.00)
17	3	-	6.00	-	(8.00)	4.50	(8.00)
18	4	-	6.00	-	5.00	3.00	8.00
19	5	-	5.00	-	5.00	3.00	5.00
20	6	-	3.00	-	3.00	1.50	3.00
21	7	-	-	-	2.00	-	1.00
22	NPV	0.00	2.44	0.00	2.01	3.84	1.91

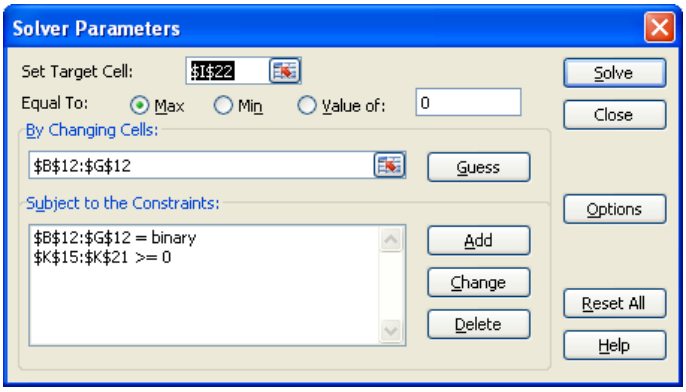
The total investment of $60M occurs over three years, and the constraint is that the firm cannot have a negative surplus in any year.

14	Total CF	Budget	Surplus
15	(20.00)	20.00	-
16	(9.00)	20.00	11.00
17	(5.50)	20.00	14.50
18	22.00	-	22.00
19	18.00	-	18.00
20	10.50	-	10.50
21	3.00	-	3.00
22	10.21	Total NPV	

The Solver window looks like this:



Under the constraint that projects can only be either fully funded or not at all (i.e. no partial fundings can occur), the weights can be binary as per the Solver window:



Under the binary weights, the optimal outcome is:

12	**Investment**	**0%**	**0%**	**100%**	**100%**	**0%**	**100%**

And,

14	**Total CF**	**Budget**	**Surplus**
15	(18.00)	20.00	2.00
16	5.00	20.00	25.00
17	(9.00)	20.00	11.00
18	15.00	-	15.00
19	11.00	-	11.00
20	6.00	-	6.00
21	3.00	-	3.00
22	**7.19**	**Total NPV**	

APPENDIX I: MATRIX MATH PRIMER

An *m x n* matrix, that is with dimensions *m* by *n*, is a array of *m* rows and *n* columns of numbers, called elements.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ a_{3,1} & a_{3,2} & \dots & a_{3,n} \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}$$

If $m = n$ then we call the matrix a square matrix. A = B, if and only if they each have the same number of rows and the same number of columns and every element of A is equal to every element of B; that is $a_{ij} = b_{ij}$ for all i,j. A matrix is said to equal 0, if and only if every element is equal to zero. That is, if A = 0, then $a_{ij} = 0$.

$$A = 0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

A matrix that consists of one row and several columns is called a row matrix or a row vector.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \end{bmatrix}$$

A matrix that consists of one column and several rows is called a column matrix or a column vector.

$$\begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \dots \\ a_{m,1} \end{bmatrix}$$

If $m = n$ then I, the identity matrix, has ones along the main diagonal.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A. Matrix Transpose

To transpose a matrix, simply take the columns and turn them into rows. For example,

$$\begin{bmatrix} 4 & 3 \\ 5 & 7 \end{bmatrix}^T = \begin{bmatrix} 4 & 5 \\ 3 & 7 \end{bmatrix}$$

EXCEL:

```
=TRANSPOSE( A1:D4 )
```

B. Addition and Subtraction

To add two matrices or to subtract one matrix from another, the two must have the same dimensions and:

$$\lfloor a_{i,j} \rfloor \pm \lfloor b_{i,j} \rfloor = \lfloor a_{i,j} \pm b_{i,j} \rfloor$$

For example,

$$\begin{bmatrix} 4 & 3 \\ 5 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 5 \\ 6 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 8 \\ 11 & 9 \end{bmatrix}$$

EXCEL:

```
=A1:B2 + C1:D2
```

C. Scalar Multiplication

To multiply a matrix by a scalar, that is a one by one matrix, multiply each element in the matrix by the scalar.

$$c\lfloor a_{i,j} \rfloor = \lfloor c \cdot a_{i,j} \rfloor$$

For example,

$$3 \cdot \begin{bmatrix} 1 & 6 \\ 4 & 7 \end{bmatrix} = \begin{bmatrix} 3 & 18 \\ 12 & 21 \end{bmatrix}$$

EXCEL:

```
=3 * A1:B2
```

D. Matrix Multiplication

In order to multiply two matrices together, say A times B, the number of columns in A must equal the number of rows in B. The solution matrix will have dimensions equal to the number of rows in A and the number of columns in B. Then, take the i^{th} row of A and multiply by the j^{th} column of B. For example,

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 7 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 8 \\ 2 & 6 \end{bmatrix}$$

$$AB = \begin{bmatrix} 2 & 3 \\ 1 & 7 \end{bmatrix} \begin{bmatrix} 4 & 8 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} 2 \cdot 4 + 3 \cdot 2 & 2 \cdot 8 + 3 \cdot 6 \\ 1 \cdot 4 + 7 \cdot 2 & 1 \cdot 8 + 7 \cdot 6 \end{bmatrix} = \begin{bmatrix} 14 & 34 \\ 18 & 50 \end{bmatrix}$$

Note that matrix multiplication is non commutative. So, BA ≠ AB.

EXCEL:

```
=MMULT( A1:B2, C1:D2 )
```

E. Matrix Inversion

Consider a square matrix, A. If there exists a square matrix ,B, such that AB = BA = I, then B is called the inverse of A. That is, $B = A^{-1}$. Also then, A is said to be invertible or nonsingular. Matrix A is nonsingular if and only if $D(A) \neq 0$. Consider a system of linear equations such that AX = B. If A is nonsingular, then $A^{-1}AX = IA^{-1}B$. Then, $X = A^{-1}B$ is the solution. To find A^{-1}, where:

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 7 \end{bmatrix}$$

To find A^{-1}, augment A to get [A | I].

$$[A \mid I] = \left[\begin{array}{cc|cc} 2 & 3 & 1 & 0 \\ 4 & 7 & 0 & 1 \end{array}\right]$$

Then, apply row reduction to find [I | A^{-1}].

$$A^{-1} = \begin{bmatrix} 3.5 & -1.5 \\ -2 & 1 \end{bmatrix}$$

Given that AX = B, the system can be solved by computing $A^{-1}B$.

EXCEL:

```
=MINVERSE( A1:B2 )
```

APPENDIX II: CALCULUS PRIMER

A. Differentiation

If y = f(x), then the instantaneous rate of change of y with respect to x, which is the slope of the tangent line at x, is called the derivative of f or y with respect to x and is given by :

$$m_{\tan} = y' = \frac{\partial y}{\partial x} = f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Rather than using the definition of the derivative, however, there are rules that give us a mechanical process for differentiation. Rules of differentiation:

Rule 1: If c is a constant, then $\frac{d}{dx}(c) = 0$

Rule 2: If n is any real number, then $\frac{d}{dx}(x^n) = nx^{x-1}$

Rule 3: If c is a constant, then $\frac{d}{dx}(cf(x)) = cf'(x)$

Rule 4: If f and g are differentiable, then $\frac{d}{dx}(f(x) \pm g(x)) = f'(x) \pm g'(x)$

Rule 5: If f and g are differentiable, then $\frac{d}{dx}(f(x) \cdot g(x)) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$

Rule 6: If f and g are differentiable, then $\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x) \cdot g(x) - f(x) \cdot g'(x)}{(g(x))^2}$

Rule 7: If y =f(u) and u=g(x), then $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$

Rule 8: If f is differentiable, then $\frac{d}{dx}(f(x)^n) = nf(x)^{n-1} \cdot f'(x)$

Rule 9: If c is a constant, then $\frac{d}{dx}(\ln(c)) = \frac{1}{c}$

Rule 10: If f is differentiable, then $\frac{d}{dx}(\ln(f(x)) = \frac{f'(x)}{f(x)}$

Rule 11: If f is differentiable, then $\frac{d}{dx}(e^{f(x)}) = e^{f(x)} \cdot f'(x)$

Rule 12: If c is a constant and f is differentiable, then $\frac{d}{dx}(c^{f(x)}) = c^{f(x)} \cdot \ln(c) \cdot f'(x)$

Rule 13: If f and g are differentiable, then $\frac{d}{dx}(f \circ g) = f'(g(x)) \cdot g'(x)$.

Note then that if y = f(x), then $dy = f'(x) \cdot dx$.

A partial derivative of a function of several variables is its derivative with respect to one of those variables with the others held constant. Let z = f (x,y) .To find ∂z/∂x, treat f as a function of x i.e. hold y constant and then differentiate. To find ∂z/∂y treat f as a function of y i.e. hold x constant and then differentiate.

B. Taylor's Theorem

Taylor's theorem gives an approximation of a differentiable function near a given point by polynomials, whose coefficients depend only on the derivatives of the function at that point.

$$f(x_1) \approx f(x_0) + \frac{1}{1!}f'(x_0)(x_1 - x_0) + \frac{1}{2!}f''(x_0)(x_1 - x_0)^2 + \ldots + \frac{1}{n!}f^n(x_0)(x_1 - x_0)^n$$

C. Integration

If y = f(x) such that F'(x) = f(x), then F is the antiderivative of f. An antiderivative of f is a function whose derivative is f. To integrate means to find:

$$\int f(x)dx = F(x) + c, \text{ if and only if } F'(x) = f(x)$$

When we find an antiderivative, we are finding the family of antiderivatives where c is a constant.

Rule 1: If k is a constant, then $\int kdx = kx + c$

Rule 2: $\int x^n dx = \frac{1}{n+1}x^{n+1} + c$

Rule 3: $\int x^{-1}dx = \ln(x) + c$

Rule 4: $\int e^{f(x)} \cdot f'(x)\, dx = e^{f(x)} + c$

Rule 5: If k is a constant, then $\int kf(x)dx = k\int f(x)dx$

Rule 6: $\int (f(x) \pm g(x))dx = \int f(x)dx \pm \int g(x)dx$

Rule 7: $\int k^{f(x)}dx = \frac{1}{\ln(k)}k^{f(x)} + c$

Rule 8: $\int \frac{1}{f(x)}dx = \ln(f(x)) + c$

Integration by Parts: $\int f(x) \cdot g'(x) = f(x) \cdot g(x) - \int f'(x) \cdot g(x)$

D. Fundamental Theorem

The fundamental theorem of calculus allows one to compute the definite integral of a function by using any one of its infinitely many antiderivatives.

$$\int_a^b f(x)dx = F(b) - F(a)$$

For functions that are not integratable, we use the trapezoidal rule to estimate the area under the curve.