Integrating Python with other Tools

QF 205 L12

Learning Objectives

- One of Python's very attractive feature is its ability to integrate with other technologies
- While it certainly is not the perfect technology, the ability to integrate with other technologies make it an invaluable tool
- In this lecture, we illustrate Python's integrability with Excel.
- Connecting with Excel, we may regard Excel as a user interface with Python as a compute-core

Contents

- Brief Look at Excel for Computation
- Monte Carlo Pricing of Black-Scholes Option on Excel
- Excel Connectivity with PyWin32
- Monte Carlo Pricing of Black-Scholes Option on Excel with Python Compute-Core
- Preparing Excel for Bloomberg Options/Futures Data

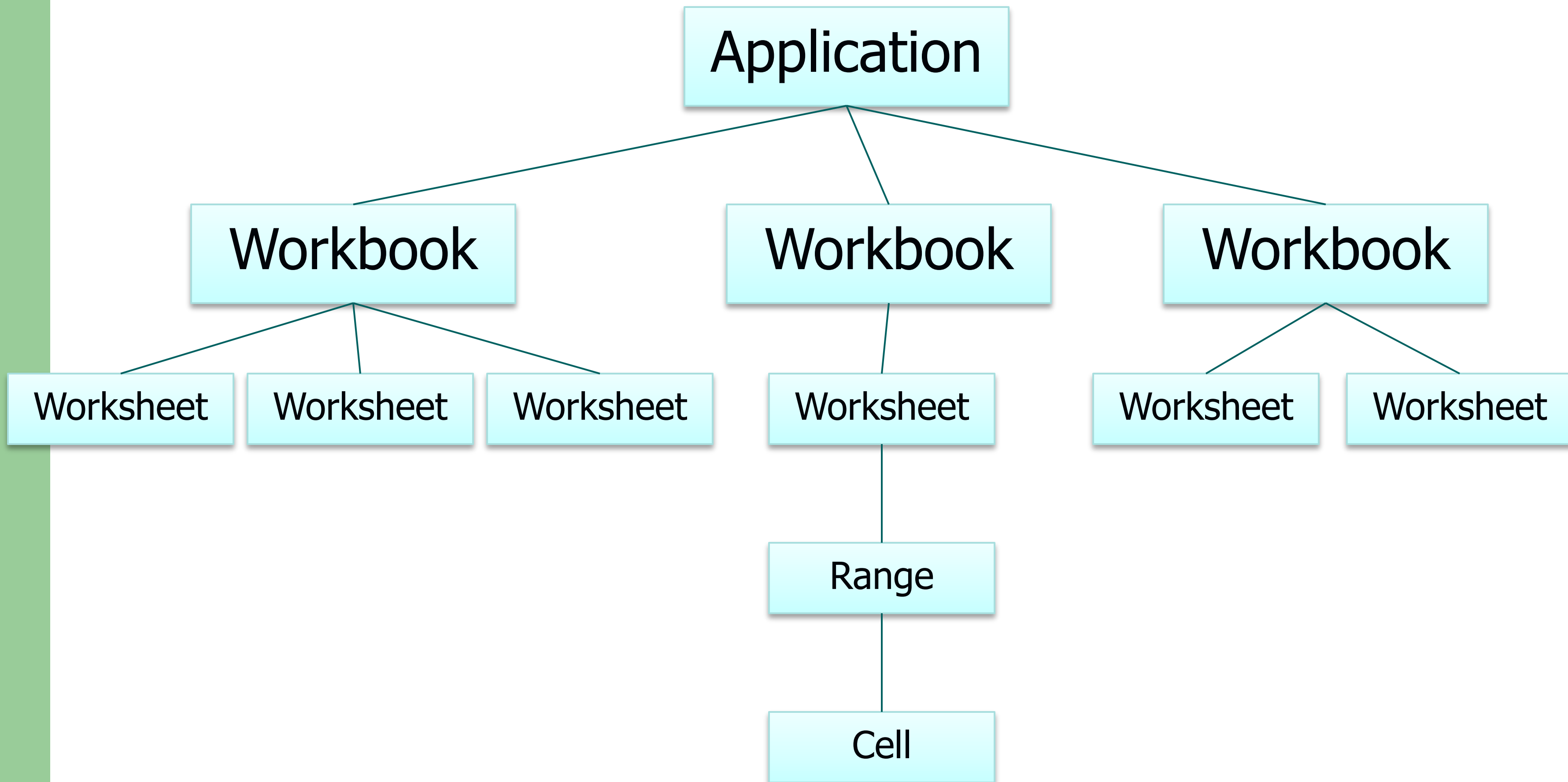
Brief Look at Excel for Computation

Computation on Excel

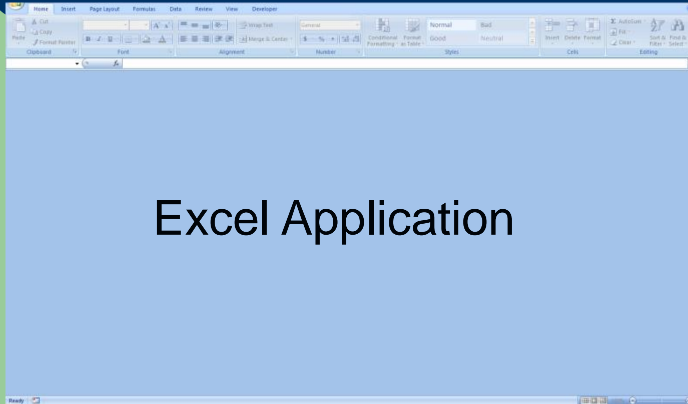
- Data is entered into cells

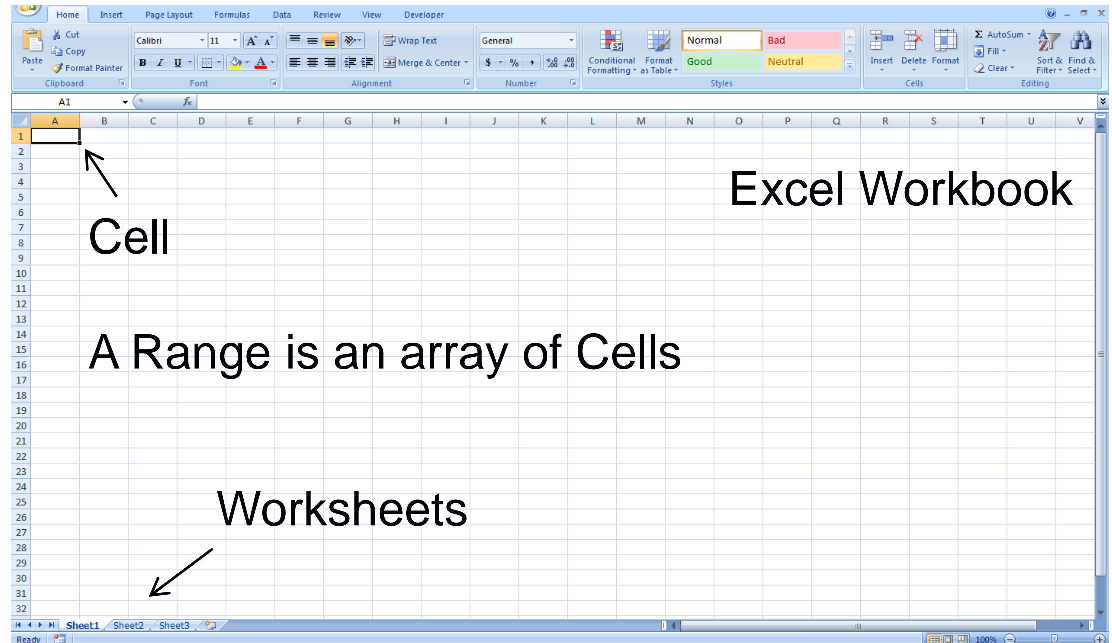
- Computation is done in one of two ways:
  - Worksheet formulas
  - Control of Excel Objects using VBA

Structure of Excel – Fundamental Objects



Structure of Excel – Fundamental Objects





Worksheet Formulas

	A	B	C	D	E	F	G	H	I	J
1	0.370849									
2	0.458217									
3	0.075678									
4	0.48763									
5	0.54637									
6	0.571308									
7	0.496747									
8	0.919701									
9	0.767984									
10	0.812269									
11	0.723925									
12	0.935963									
13	0.3381									
14	0.388087									
15	0.508631									
16	0.40647									
17	0.643239									
18	0.37584									
19	0.487638									
20	0.268013									
21	10.58266									
22										
23										
24										

A21 =SUM(A1:A20)

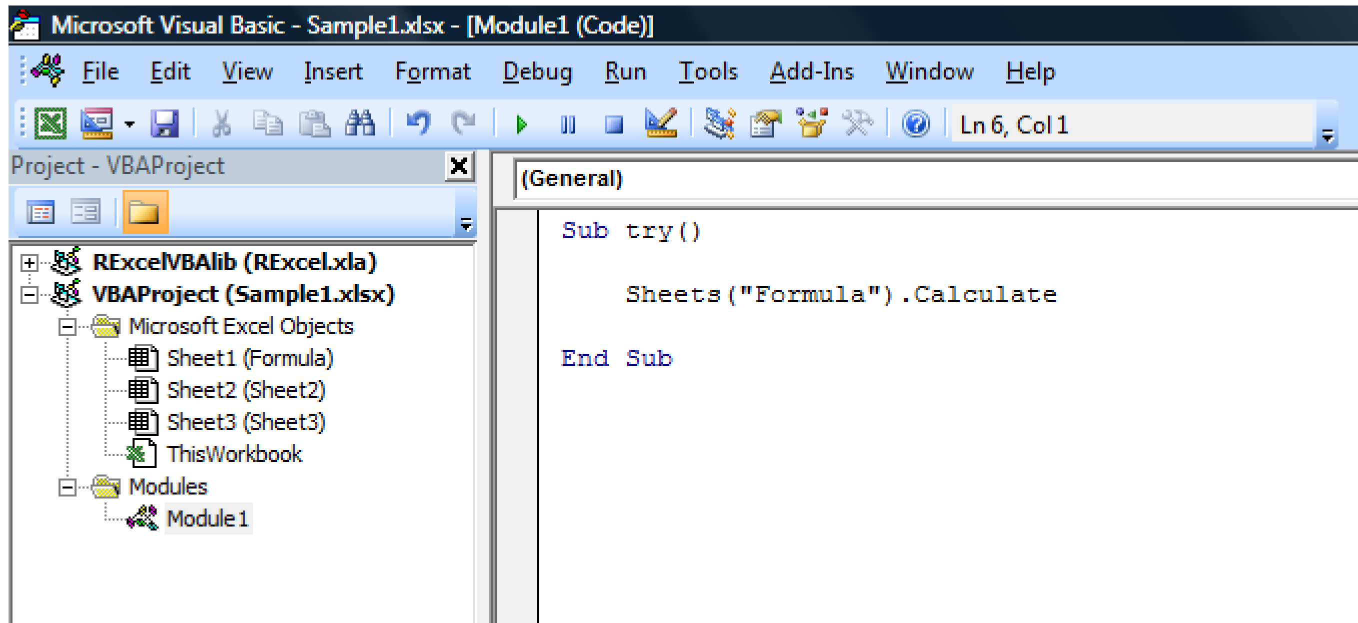
=RAND()

Double-click on a cell and click away
Notice the numbers changing

That is an example of RECALCULATION

Excel recalculates its sheets contingent upon certain events to maintain an updated status

Recalculate with VBA



1) Alt-F11 to start VBE
2) Right click the Project Explorer and select Insert Module to create new module
3) Key in the above VBA code
4) Click F5 to run the code

Cell Data Types

- The value contained in a cell can be one of the following types:

Number (1)
Text (2)
Logical Value (4)
Error (16)
Array (64)

	A	B	C
1		1	
2	12/1/2010	1	
3	abc	2	
4	TRUE	4	
5	#NAME?	16	
6			
7			

- The number in the parenthesis is the value returned by the function Type(*value*)

- Notice that the empty cell and the date are considered numbers

Format

- A cell that contains a number that is not empty may have a format – it is the format that determines whether it is displayed as a date or not

C2 =CELL("format",A2)

	A	B	C	D	E
1		1			
2	12/1/2010	1	D1		
3	abc	2			
4	TRUE	4			
5	#NAME?	16			
6					

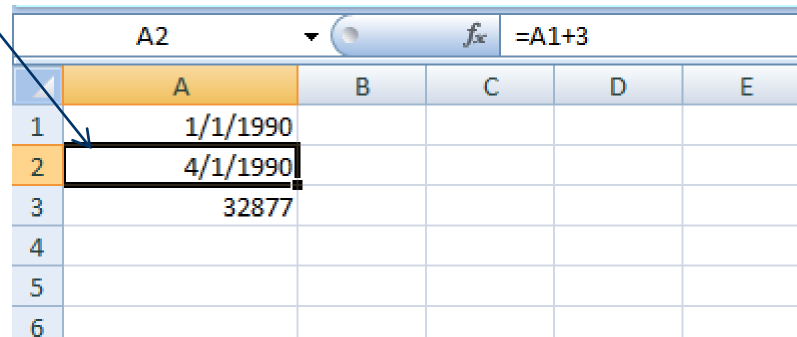
If the Excel format is	The CELL function returns
General	"G"
0	"F0"
#,##0	",0"
0.00	"F2"
#,##0.00	",2"
$#,##0_);($#,##0)	"C0"
$#,##0_);[Red]($#,##0)	"C0-"
$#,##0.00_);($#,##0.00)	"C2"
$#,##0.00_);[Red]($#,##0.00)	"C2-"
0%	"P0"
0.00%	"P2"
0.00E+00	"S2"
# ?/? or # ??/??	"G"
m/d/yy or m/d/yy h:mm or mm/dd/yy	"D4"
d-mmm-yy or dd-mmm-yy	"D1"
d-mmm or dd-mmm	"D2"
mmm-yy	"D3"
mm/dd	"D5"
h:mm AM/PM	"D7"
h:mm:ss AM/PM	"D6"
h:mm	"D9"
h:mm:ss	"D8"

Source: http://office.microsoft.com/en-us/excel/HP100623921033.aspx

Ain’t it confusing

Each cell has many properties (attributes).

The value that A2 contains is 32877 (pasted as value into A3)
It has a formula “=A1+3”
It has a format that renders the value as a date



Extracting the information

	A	B	C	D	E	F
1	1/1/1990					
2	4/1/1990	32877	4/1/1990	=A1+3		
3	32877					
4						
5						

=CELL("contents",A2)

=TEXT(A2,"d/m/yyyy")

=formula_string(A2)

```
Function formula_string(celref As Range)
    formula_string = celref.Formula
End Function
```

VBE

Microsoft Visual Basic - Sample1.xlsm - [Module2 (Code)]

File Edit View Insert Format Debug Run Tools Add-Ins Window Help

Type a question for help

Project - VBAProject

- RExcelVBAlib (RExcel.xla)
- VBAProject (Sample1.xlsm)
  - Microsoft Excel Objects
    - Sheet1 (Formula)
    - Sheet2 (Cell_Format)
    - Sheet3 (Confusing)
    - ThisWorkbook
  - Modules
    - Module1
    - Module2

(General) | formula_string

```
Function formula_string(celref As Range)
    formula_string = celref.Formula
End Function
```

Top level Excel objects that are “alive”

VBA codes

Immediate

```
4/1/1990
=A1+3
```

VBA talks to Excel objects

Microsoft Visual Basic - Sample1.xlsm - [Module3 (Code)]

File Edit View Insert Format Debug Run Tools Add-Ins Window Help

Project - VBAProject

- RExcelVBAlib (RExcel.xla)
- VBAProject (Sample1.xlsm)
  - Microsoft Excel Objects
    - Sheet1 (Formula)
    - Sheet2 (Cell_Format)
    - Sheet3 (Confusing)
    - ThisWorkbook
  - Modules
    - Module1
    - Module2
    - Module3

(General) test

```
Sub test()

    Debug.Print Application.Name
    Debug.Print ThisWorkbook.Name
    Debug.Print Sheets("Formula").Range("A1:C3").Address

End Sub
```

Methods and fields/attributes in the Excel Object Model

Immediate

```
Microsoft Excel
Sample1.xlsm
$A$1:$C$3
```

Excel Object Model

Excel Object Model Overview - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://msdn.microsoft.com/en-us/library/wss56bz7(VS.80).aspx

Most Visited Getting Started Latest Headlines SourceForge.net: Files Yahoo! Finance: GOO...

Excel Object Model Overview

Home Library Learn Downloads Support Community Sign in | United States - English | Preferences

MSDN Library
Development Tools and Languages
Visual Studio 2005
Visual Studio
Visual Studio Tools for Office
Automating Applications Using the O...
Excel Object Model Overview

Community Content

Add code samples and tips to enhance this topic.

More...

Visual Studio Tools for the Microsoft Office System

Excel Object Model Overview

msdn

To develop solutions that use Microsoft Office Excel, you can interact with the objects provided by the Excel object model. This topic introduces the most important classes:

- Microsoft.Office.Interop.Excel.Application
- Microsoft.Office.Interop.Excel.Workbook
- Microsoft.Office.Interop.Excel.Worksheet
- Microsoft.Office.Interop.Excel.Range

For the most part, the object model directly emulates the user interface. For example, the **Application** object represents the entire application, and each **Workbook** object contains a collection of `Worksheet` objects. From there, the major abstraction representing cells is the **Range** object, which allows you to work with individual cells or groups of cells.

Microsoft Visual Studio 2005 Tools for the Microsoft Office System (VSTO 2005) extends many of these native objects into host items and host controls that can be used in document-level customizations. These controls have additional functionality, including data binding capabilities and events. For example, a native Excel **Microsoft.Office.Interop.Excel.Range** object is extended into a Microsoft.Office.Tools.Excel.NamedRange control, which can be bound to data and exposes events. For more information about host items and host controls, see Host Items and Host Controls Overview.

For complete information about the Excel 2003 object model, see the VBA documentation that is installed with Excel or see "Welcome to the Microsoft Office Excel 2003 VBA Language Reference" (http://go.microsoft.com/fwlink/?linkid=27951). For complete information about the Excel 2007 object model, see the VBA documentation that is installed with Excel or see the 2007 Microsoft Office system developer content on the MSDN Web site (http://go.microsoft.com/fwlink/?LinkId=72870).

Accessing Objects in an Excel Project

When you create a new application-level project for Excel by using Microsoft Visual Studio 2005 Tools for the 2007 Microsoft Office System (VSTO 2005 SE), Visual Studio automatically creates a ThisAddIn.vb or ThisAddIn.cs code file. You can access the Application object by using `Me.Application` or `this.Application`.

When you create a new document-level project for Excel by using VSTO 2005, you have the option of creating a new Excel Application or Excel Template project. VSTO 2005 automatically creates the following code files in your new Excel project for both document and template projects.

Visual Basic	C#
ThisWorkbook.vb	ThisWorkbook.cs

Done

Monte Carlo Pricing of Black-Scholes Option on Excel

Recap: Black-Scholes Theory

Assumption 1: Stock price follows the Geometric Brownian Motion

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$$

Assumption 2: The market satisfied the Principle of No-Arbitrage
(i.e. there is no free lunch)

Conclusion:

Call Option Price

$$S_0 N(d_1) - K e^{-rT} N(d_2)$$

Put Option Price

$$-S_0 N(-d_1) + K e^{-rT} N(-d_2)$$

$$d_1 = \frac{\log(S_0 / K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\log(S_0 / K) + (r - \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$d_1 = d_2 + \sigma\sqrt{T}$$

The Risk Neutral Pricing Framework

- In this framework, which is equivalent to Black-Scholes Theory as was stated, the stock price satisfies the Geometric Brownian Motion with drift equal to the risk-free rate:

$$\frac{dS_t}{S_t} = r\,dt + \sigma\,d\tilde{W}_t$$

- And the call option price is given by:

$$C_0 = \tilde{E}\left[e^{-rT}\max(S_T - K, 0)\right]$$

Properties of Brownian Motion

Continuous

Given $t1 < t2 < t3 \cdots$
$W_{t1}, W_{t2} - W_{t1}, W_{t3} - W_{t2}, \ldots$
are independent

$$W_t$$

$W_t - W_s \sim N(0, t-s)$

$W_0 = 0$
$0 < t < T$

Fundamental Building Block of Stochastic Calculus

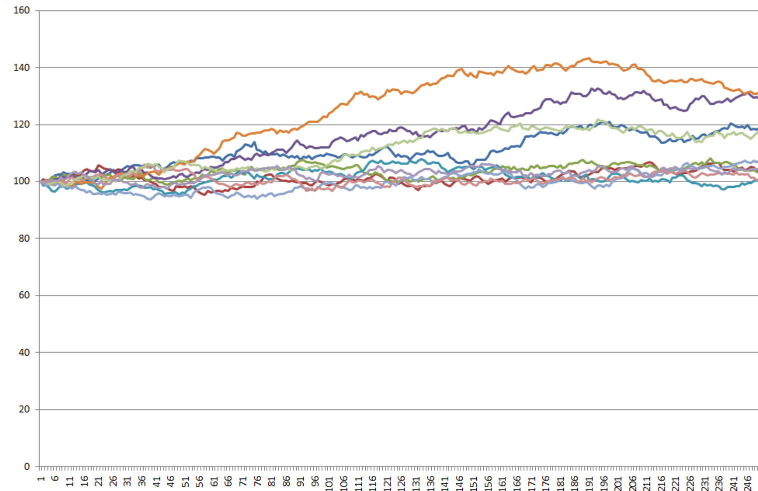
Monte Carlo Option Pricing

- Step 1: Discretize the GBM

$$\frac{S_{(i+1)\delta t} - S_{i\delta t}}{S_{i\delta t}} = r\delta t + \sigma(W_{(i+1)\delta t} - W_{i\delta t}) = r\delta t + \sigma\sqrt{\delta t}Z$$

where $\delta t = \frac{T}{N}, Z \sim N(0,1)$

- Step 2: Simulate M price paths on Excel spreadsheet



Monte Carlo Option Pricing

- Step 3: Estimate the (call) option price from the formula

$$C_0 \approx e^{-rT} \times \frac{\max(S^1(T)-K,0)+\max(S^2(T)-K,0)+\cdots+\max(S^M(T)-K,0)}{M}$$

Implementation on Excel

- Let's plan!
- 1 sheet for parameters: Input Sheet
- 1 sheet for price paths (numerics): Output-Paths Sheet
- 1 chart for price paths (graphs): Paths Chart Sheet
- 1 sheet for option price: Output-Price Sheet

Input

	A	B	C	D	E	F	G	H
1	S0		100					
2	r		0.1					
3	sigma		0.1					
4	T		1					
5	n = T/dt		250					
6	m, Number of paths		10					
7	Strike		100					

GENERATE

Paths Chart | Input | Output-Paths | Output-Price

Ready

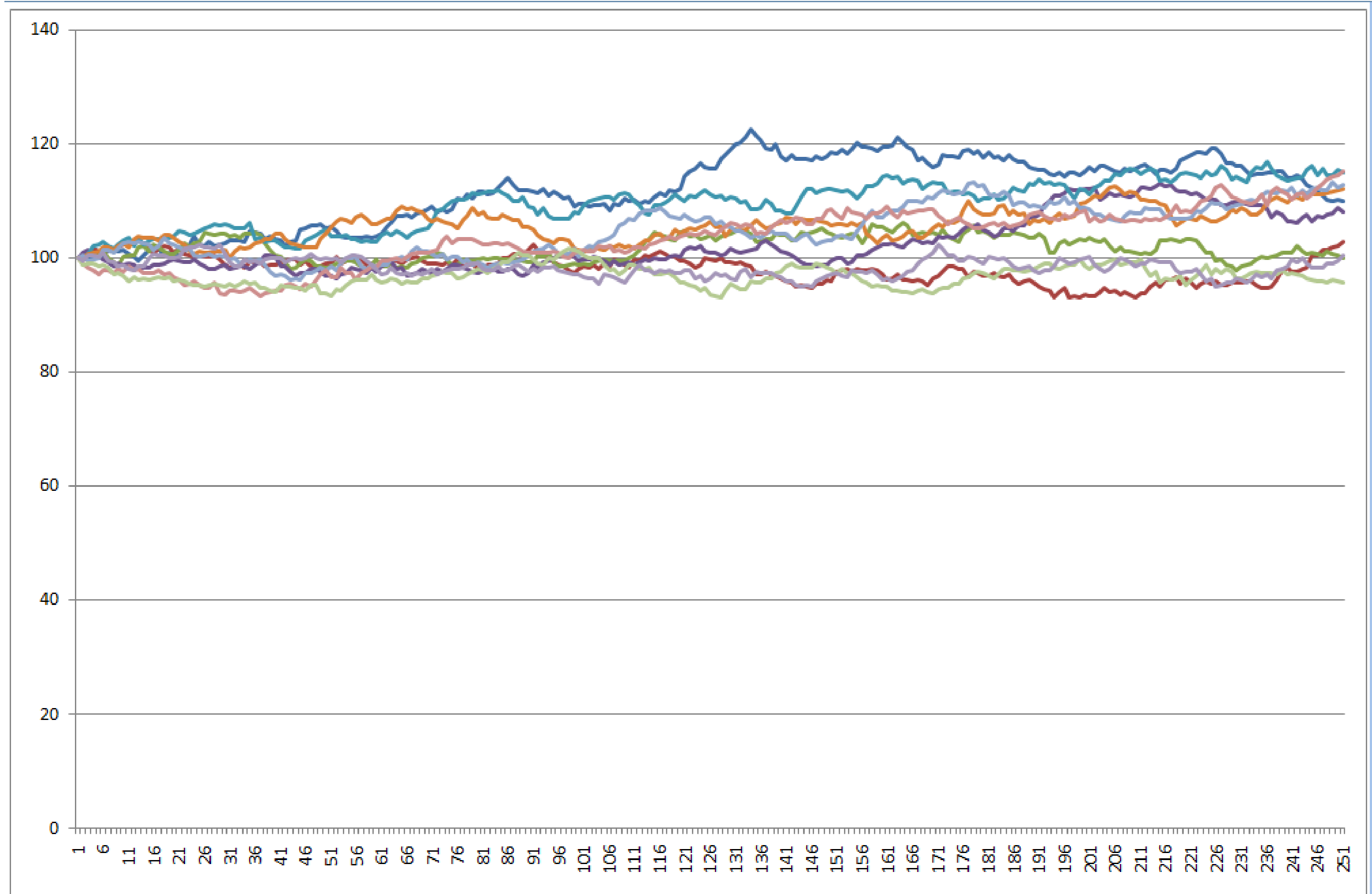
Output-Paths

	A	B	C	D	E	F	G	H	I	J	K
1	100	100	100	100	100	100	100	100	100	100	
2	100.0466	99.69623	100.0446	100.8468	100.0497	100.0283	100.4603	99.01095	98.64312	99.38205	
3	99.68973	100.2996	100.7078	101.3445	100.6377	100.6348	99.35802	98.2413	99.05048	100.515	
4	100.1568	101.6311	100.8734	100.5022	102.0503	100.9784	99.71318	97.75659	98.96965	100.4091	
5	100.076	101.2169	101.0112	100.2401	102.3401	101.0359	99.89633	97.04492	98.63351	100.2161	
6	100.4483	101.0626	101.3659	99.78883	102.8051	101.9627	101.3339	97.90846	98.74527	100.8061	
7	101.3919	101.3125	100.0005	99.16854	102.1078	101.9511	101.3176	97.69862	98.25508	99.43346	
8	101.0835	101.45	99.22104	98.54744	101.5254	101.434	100.8486	97.31678	97.0837	99.33079	
9	101.1337	101.7825	98.47378	98.98082	102.4336	102.0688	101.5663	98.02354	97.32589	98.60004	
10	101.3035	102.3337	100.2817	98.85781	102.9744	102.4513	102.6657	97.61055	96.94039	98.68055	
11	100.6988	102.4105	100.3406	98.98424	103.5836	101.9506	102.7676	98.52035	95.98951	98.12405	
12	100.2408	102.9907	100.5078	98.77528	102.1268	103.3188	102.5636	97.87043	96.66578	97.74438	
13	99.47893	102.9646	101.9242	98.0842	103.4341	103.7946	102.7918	98.06553	96.22915	98.63269	
14	100.468	102.4568	102.1819	98.18935	102.4289	103.5403	102.6622	97.20987	96.28984	98.49814	
15	101.3604	102.0398	101.0816	98.18766	103.1092	103.3956	102.3479	97.21048	96.15689	99.97551	
16	101.855	100.7475	102.2551	98.71066	103.3287	103.4223	101.3801	97.22337	96.45322	100.5627	
17	101.582	101.6714	101.7471	98.73008	103.1668	103.0282	102.0352	97.71063	96.61477	100.5187	
18	100.4204	102.0526	101.1487	99.04243	103.2381	104.1031	103.4331	97.18264	96.32895	100.4577	
19	100.7767	101.8475	100.6472	99.35633	103.5869	103.9444	102.5042	97.35751	96.66502	99.89471	
20	100.753	100.9786	100.6821	99.76045	103.8418	103.4986	102.3397	96.65734	95.82978	100.009	
21	100.5326	101.5665	101.2623	99.56597	104.8089	102.8419	101.8189	96.21249	96.13906	100.2429	
22	101.1491	100.759	102.5509	99.2354	104.5626	102.355	101.3579	95.29484	95.82594	101.6796	
23	102.3545	100.405	101.8632	99.1995	103.962	101.1657	101.6459	95.39161	95.35405	101.8289	
24	103.6876	100.1237	102.5118	99.74254	104.2554	100.6827	100.3215	95.83654	95.23378	101.8321	
25	102.2882	100.0709	102.8531	99.10391	104.6089	101.0467	100.141	95.07747	95.10316	102.2644	
26	102.1275	100.0675	104.388	98.58018	105.261	100.8473	100.4583	94.82713	94.90815	101.8036	
27	102.7133	100.1177	104.2656	98.07637	105.5147	100.8005	100.7453	94.59364	95.08268	101.7603	
28	102.1541	100.5132	103.9927	98.02962	105.9764	101.0819	100.2491	95.68207	95.43119	101.5454	
29	102.2695	99.76872	104.1189	98.5843	105.6365	101.1983	100.3013	93.82269	95.19471	102.214	
30	102.5249	98.84898	104.3235	98.77096	105.9967	101.2707	100.198	93.42314	94.88759	100.5416	
31	103.0033	98.37518	103.6694	97.94652	105.6619	100.0281	99.60972	94.31382	95.48049	99.27536	
32	102.9798	98.32519	103.9214	98.24973	105.2798	101.0535	100.0175	94.16127	94.88377	98.50698	

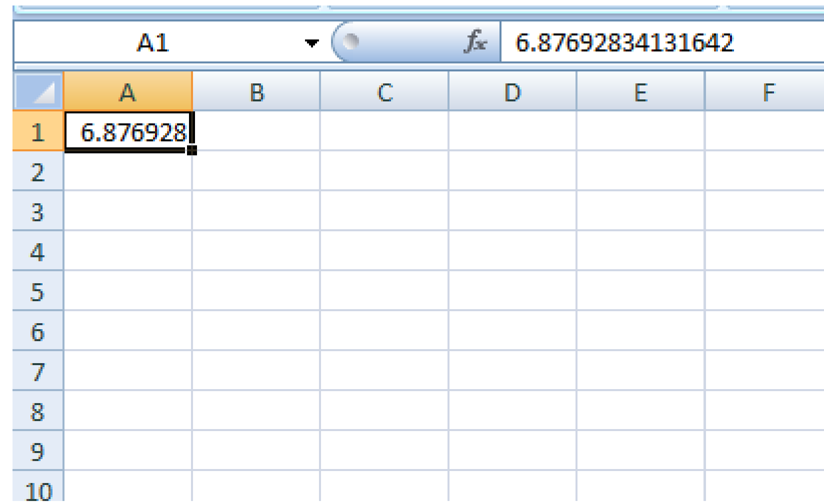
Paths Chart | Input | **Output-Paths** | Output-Price

Ready

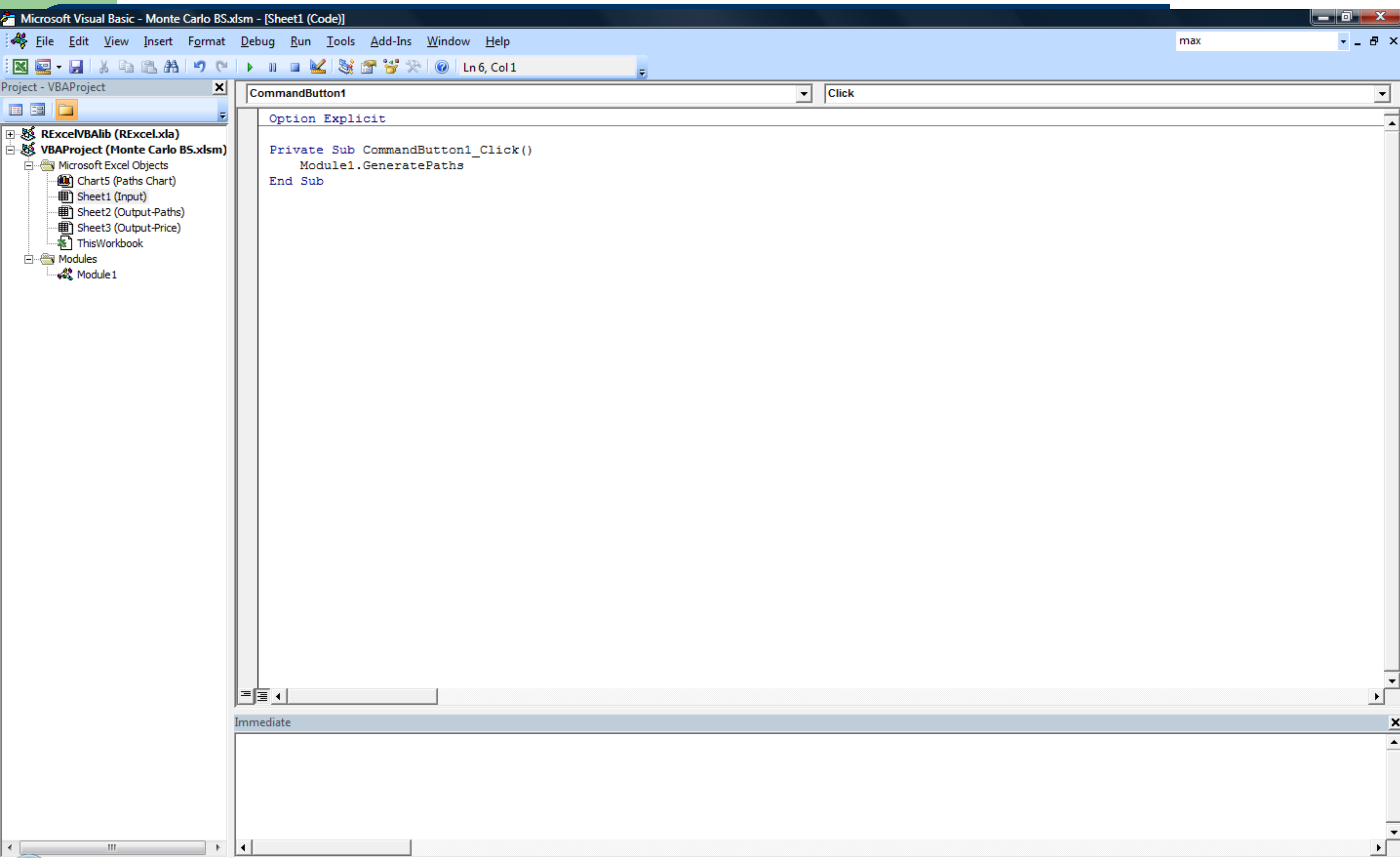
Paths Chart



Output-Price



VBA code behind button



Module1.GeneratePaths

```
Option Explicit

Sub GeneratePaths()
    Dim S0, r, sigma, T, dt, s, price, K As Double
    Dim n, m, i, j As Integer
    Dim chrt As Chart

    ' Capture parameters
    S0 = Sheets("Input").Cells(1, 3)
    r = Sheets("Input").Cells(2, 3)
    sigma = Sheets("Input").Cells(3, 3)
    T = Sheets("Input").Cells(4, 3)
    n = Sheets("Input").Cells(5, 3)
    m = Sheets("Input").Cells(6, 3)
    K = Sheets("Input").Cells(7, 3)

    ' Compute convenience parameters
    dt = T / n

    ' Clear Output sheets
    Sheets("Output-Paths").Cells.Clear
    Sheets("Output-Price").Cells.Clear

    ' Fill Output sheets
    price = 0
    For i = 1 To m
        s = S0
        For j = 1 To n
            Sheets("Output-Paths").Cells(j, i) = s
            s = s * (1 + r * dt + sigma * Sqr(dt) * WorksheetFunction.NormSInv(Rnd))
        Next j
        Sheets("Output-Paths").Cells(n + 1, i) = s
        price = price + WorksheetFunction.Max(s - K, 0)
    Next i
    price = price / m * Exp(-r * T)
    Sheets("Output-Price").Cells(1, 1) = price

    ' Create chart
    If Charts.Count = 0 Then
        ' Supposed to be created after Output sheet...apparently doesn't work
        Set chrt = Charts.Add(, Sheets("Output-Paths"))
        chrt.Name = "Paths Chart"
    Else
        Set chrt = Charts(1)
    End If
    chrt.ChartType = xlLine
    chrt.HasLegend = False
    chrt.SetSourceData Source:=Sheets("Output-Paths").Range("A1").CurrentRegion

End Sub
```

Excel Connectivity with PyWin32

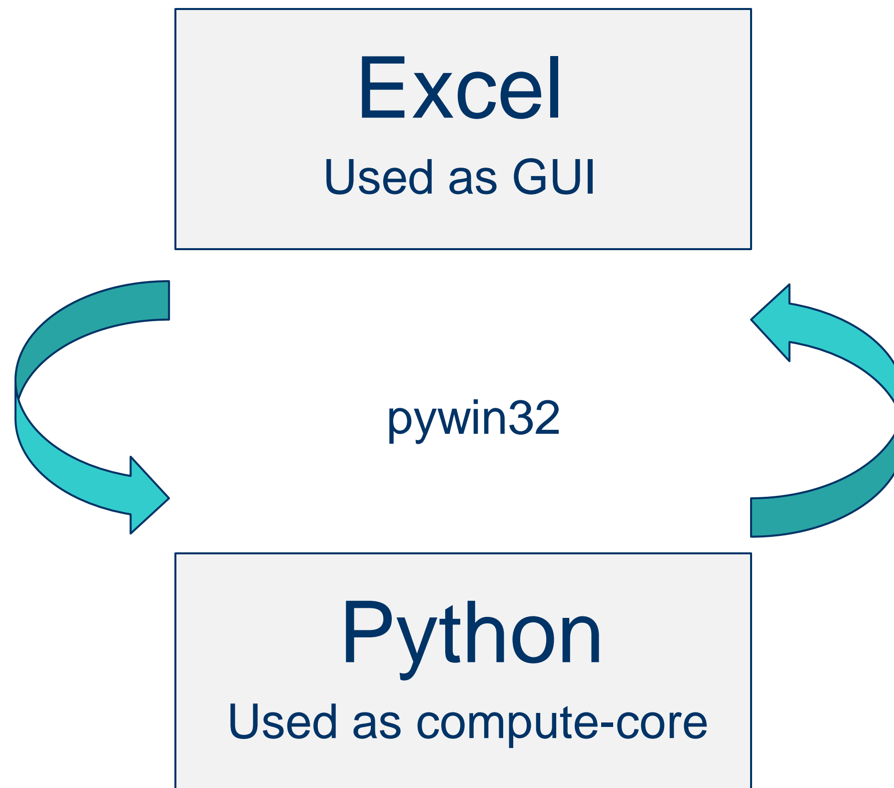
The case for extending Excel

- Excel – basically a glorified two-dimensional array
  - But financial information resides in high dimensional spaces!

- VBA is a clumsy language
  - OOP features weak to non-existent
  - Few freely available supporting libraries (particularly for numerical/statistical computations) except for a few add-ins
  - Validity of in-built mathematical functions is questionable

- However: Excel is perhaps the most widely used piece of software in the (finance) industry
  - Forsaking it is turning our backs against lots of folks

Strategy



This is what happens:
Python <-> PyWin32 <-> MSFT COM Technology <-> Excel Object Model

Caveat: Resize method of Range object does not work from pywin32

A brief introduction to PyWin32

- The win32com package is Python's interface with the Windows operating system

The win32com package

To facilitate an orderly framework, the Python "ni" module has been used, and the entire package is known as "win32com". As is normal for such packages, win32com itself does not provide any functionality. Some of the modules are described below:

- **win32com.pythoncom - core C++ support.**
  This module is rarely used directly by programmers - instead the other "helper" module are used, which themselves draw on the core pythoncom services.
- **win32com.client package**
  Support for COM clients used by Python. Some of the modules in this package allow for dynamic usage of COM clients, a module for generating .py files for certain COM servers, etc.
- **win32com.server package**
  Support for COM servers written in Python. The modules in this package provide most of the underlying framework for magically turning Python classes into COM servers, exposing the correct public methods, registering your server in the registry, etc.
- **win32com.axscript**
  ActiveX Scripting implementation for Python.
- **win32com.axdebug**
  Active Debugging implementation for Python
- **win32com.mapi**
  Utilities for working with MAPI and the Microsoft Exchange Server

The pythoncom module

The pythoncom module is the underlying C++ support for all COM related objects. In general, Python programmers will not use this module directly, but use win32com helper classes and functions.

This module exposes a C++ like interface to COM - there are objects implemented in pythoncom that have methods "QueryInterface()", "Invoke()", just like the C++ API. If you are using COM in C++, you would not call a method directly, you would use pObject->Invoke(…, MethodId, argArray…). Similarly, if you are using pythoncom directly, you must also use the Invoke method to call an object's exposed method.

There are some Python wrappers for hiding this raw interface, meaning you should almost never need to use the pythoncom module directly. These helpers translate a "natural" looking interface (eg, obj.SomeMethod()) into the underlying Invoke call.

Source: http://timgolden.me.uk/pywin32-docs/html/com/win32com/HTML/package.html

What is COM?

- COM is a Microsoft technology aimed at component-based programming for the purpose of code reusability
- It's been superseded by .NET technology; but it will be there to stay for awhile because lots of existing technology are COM object: e.g. Word, Excel
- The bare minimum about COM
  - Comprises objects and interfaces
  - Special interface - IDispatch – an interface that connects COM objects to high-level programming languages such as C++
  - Usage of Idispatch can be by early-binding or late-binding
    - Type information is available for early-binding
    - Binding at last possible moment upon calling is late-binding

Source: http://oreilly.com/catalog/pythonwin32/chapter/ch12.html

COM and Python

The interface between Python and COM consists of two discrete parts:
the pythoncom Python extension module and the win32com Python package.
Collectively, they are known as PythonCOM.

The pythoncom module is primarily responsible for exposing raw
COM interfaces to Python. For many of the standard COM interfaces,
such as IStream or IDispatch, there is an equivalent Python object that
exposes the interface, in this example, a PyIStream and PyIDispatch object.
These objects expose the same methods as the native COM interfaces
they represent, and like COM interfaces, do not support properties.
The pythoncom module also exposes a number of
COM-related functions and constants.

The win32com package is a set of Python source files that use the
pythoncom module to provide additional services to the Python programmer.
As in most Python packages, win32com has a number of subpackages;
win32com.client is concerned with supporting client-side COM
(i.e., helping to call COM interfaces), and win32com.server is concerned with
helping Python programs use server-side COM (i.e., implement COM interfaces).
Each subpackage contains a set of Python modules that perform various tasks.

Source: http://oreilly.com/catalog/pythonwin32/chapter/ch12.html

Talking to COM objects

```
import win32com.client

o = win32com.client.Dispatch("Outlook.Application")
mapi = o.GetNamespace("MAPI")

# Say mapi.Folders[3].Name reads "Mailbox - TAN Chong Hui"
for x in mapi.Folders:
    print x.Name

mapi = None
o = None
```

Connecting to Outlook

Connecting to R

```
from win32com.client import Dispatch
R = Dispatch('StatConnectorSrv.StatConnector')
R.Init('R')

R = None
```

```
import win32com.client

xlapp = win32com.client.Dispatch('Excel.Application')
xlwb = xlapp.Workbooks('Monte Carlo Simulation')

xlapp = None
```

Connecting to Excel

Monte Carlo Pricing of Black-Scholes Option on Excel with Python Compute-Core

About try_excel.py

- The code on the following slides replaces the VBA code in Monte Carlo BS.xlsm
- It may look longer – however, the lengthiness is for connectivity with Excel and recreation of some Excel convenience methods which are one-off
- The compute-core is the part that essentially changes with applications
- To use, start Monte Carlo BS.xlsm, then invoke "python try_excel.py" in the same folder

The Pythonic replacement for VBA

```
import math
import numpy as np
import win32com.client

# The 3 functions below are helper methods
# for addressing ranges in Excel

def cellstr_to_coord(cellstr):
    """
    Input: $B$1
    Output: 1, 2 (row, col)
    """
    tmparr = cellstr.strip().split('$')
    x = int(tmparr[2])
    y = 0
    for ch in tmparr[1]:
        y = 26 * y + ord(ch) - 64
    return x, y
```

```
def coord_to_cellstr(x,y):
    """
    Input: 1, 2
    Output: $B$1
    """
    digit = []
    while y > 0:
        digit.append(y % 26)
        y = y / 26
    tmp = ''.join([chr(64 + z) for z in digit[::-1]])
    return '$' + tmp + '$' + str(x)

def range_string(cellstr,rowsize,columnsize,rowoffset,columnoffset):
    """
    cellstr is of the form $A$1
    """
    r, c = cellstr_to_coord(cellstr.strip())
    r1 = r + rowoffset
    c1 = c + columnoffset
    r2 = r1 + rowsize - 1
    c2 = c1 + columnsize - 1
    return coord_to_cellstr(r1,c1) + ':' + coord_to_cellstr(r2,c2)
```

The Pythonic replacement for VBA

```
# The following code mimics what VBA does
# on Monte Carlo Simulation.xlsm

if __name__ == '__main__':
    # Initialization
    xlapp = win32com.client.Dispatch('Excel.Application')
    xlwb = xlapp.Workbooks('Monte Carlo BS')
    xlws_in = xlwb.Sheets("Input")
    xlws_out = xlwb.Sheets("Output-Paths")
    xlws_outprice = xlwb.Sheets("Output-Price")

    # Extract parameters
    s0 = xlws_in.Range("C1").Value
    r = xlws_in.Range("C2").Value
    sigma = xlws_in.Range("C3").Value
    t = xlws_in.Range("C4").Value
    n = int(xlws_in.Range("C5").Value)
    m = int(xlws_in.Range("C6").Value) # Number of paths
    k = float(xlws_in.Range("C7").Value)

    # Clear sheets
    xlws_out.Cells.Clear()
    xlws_outprice.Cells.Clear()

    # Compute-Core using numpy
    dt = float(t) / n
    price = 0
    mat = np.zeros((n+1,m))
    mat[0,] = np.ones((1,m)) * s0
    for i in range(1,n+1):
        mat[i,] = mat[i-1,] * ( 1 + r * dt + sigma * math.sqrt(dt) * \
            np.random.standard_normal((1,m)) )
    price = np.sum(np.maximum(mat[n,]-k,0)) / m * math.exp(- r * t)

    # Put matrix to Output sheet
    rngstr = range_string('$A$1',n+1,m,0,0)
    xlws_out.Range(rngstr).Value = mat
    xlws_outprice.Range("A1").Value = price

    # Crate chart
    if xlwb.Charts.Count == 0:
        xlchrt = xlwb.Charts.Add(xlws_in)
        xlchrt.Name = "Paths Chart"
    else:
        xlchrt = xlwb.Charts[0]
    xlchrt.ChartType = 4 # Check help in VBE for the value of xlline
    xlchrt.HasLegend = False
    xlchrt.SetSourceData(Source=xlws_out.Range("A1").CurrentRegion)

    # Clean up
    xlapp = None
    del xlapp
```

Preparing Excel for Bloomberg Options/Futures Data

Bloomberg Data

- Bloomberg's servers are polled for data
- Bloomberg supports an Excel add-in for this purpose
- User proceeds by filling in a Data Wizard:

Bloomberg Data Wizard – what it does

- Securities tickers (e.g. SPX Index), along with data fields (e.g. LAST_PRICE), frequency of data-points (e.g. daily, minute-bins, tick-level), period (start datetime – end datetime) are specified

- These are laid out on the Excel spreadsheet in a form similar to this:

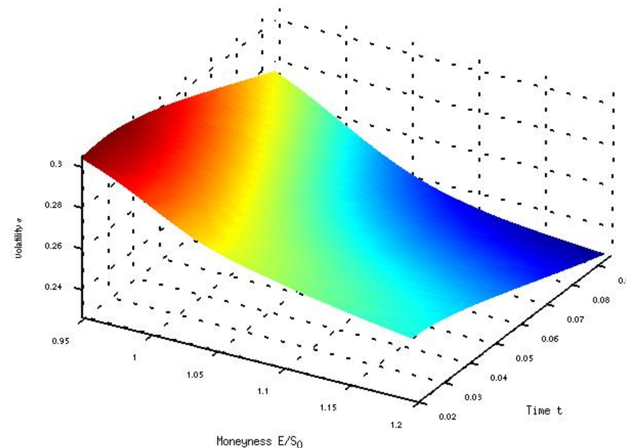
A4 =BDH($A$1,$B$3:$G$3,"1/1/1980 9:00:00 AM","",$A$2,"BarSz=1","Dir=V","Dts=S","Sort=A","Quote=C","UseDPDF=Y","CshAdjNormal=N","CshAdjAbnormal=N","CapChg=N")

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
1	SZU US 04/17/10 C400 Index								SZU US 04/17/10 C450 Index								SYU US 04/17/10 C500 Index					
2	BarTp=T								BarTp=T								BarTp=T					
3	Date	OPEN	HIGH	LOW	LAST_PRIC	NUMBER_	VOLUME		Date	OPEN	HIGH	LOW	LAST_PRIC	NUMBER_	VOLUME		Date	OPEN	HIGH	LOW	LAST_PRIC	NUI
4	#NAME?								#NAME?								#NAME?	618.8	618.8	618.8	618.8	
5																						
6																						

- The sheet is automatically refreshed (re-calculated) and the formula invokes the functionality of an add-in to poll data from Bloomberg data servers

Problem

- Getting data manually for one or two securities is fine, but not feasible for many securities
- An example where many securities are needed is when one is analyzing a volatility surface – this requires the prices of all options in an option class



An opportunity for automation

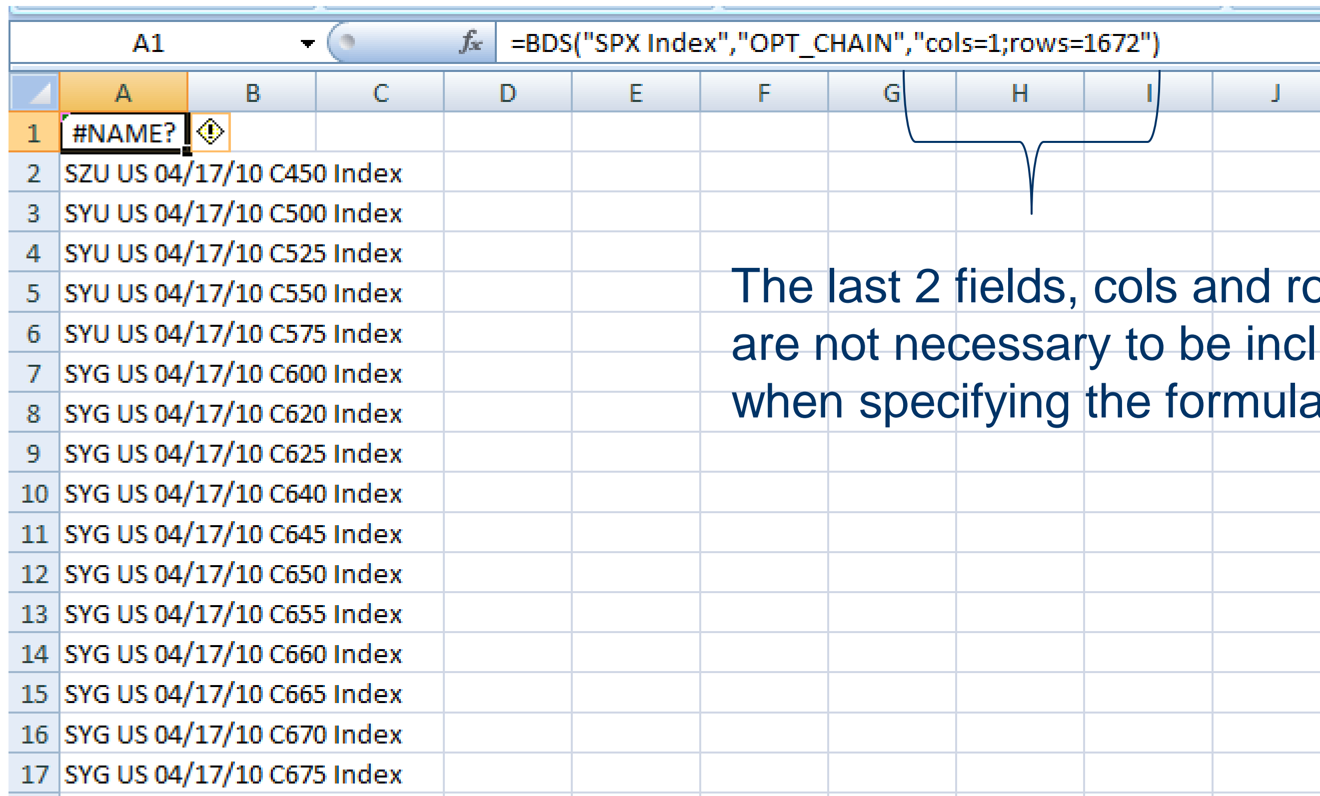
- What one basically needs to do is to create this layout

	A	B	C	D	E	F	G	H
1	SZU US 04/17/10 C400 Index							
2	BarTp=T							
3	Date	OPEN	HIGH	LOW	LAST_PRIC	NUMBER_	VOLUME	
4	#NAME?							
5								
6								

in a non-overlapping fashion horizontally across the worksheet, modifying the content of A1 (ticker of the option), and the formula of A4 so that it refers to the appropriate cells

Polling for tickers of an option class

- All the tickers belong to a class may be obtained as follows:



The code

```
import datetime
import win32com.client
import pysg.lib.xllib as xlb

def intraday_lay0(path, wbname, wsname, symbollist,
                fieldlist=['Date','OPEN','HIGH','LOW','LAST_PRICE','NUMBER_TICKS','VOLUME'],quote_type='T'):
    """
    Choice can be made between quote_type = T, B or A

    Start date is assumed to be 1/1/1980
    """
    s0 = '=BDH('
    s1 = ',"1/1/1980 9:00:00 AM","",'
    s3 = ',"BarSz=1","Dir=V","Dts=S","Sort=A","Quote=C","UseDPDF=Y",\
         "CshAdjNormal=N","CshAdjAbnormal=N","CapChg=N")'

    xlwb = win32com.client.GetObject(path + "\\" + wbname)
    xlws = xlwb.Sheets(wsname)

    for i, sym in enumerate(symbollist):
        ln = len(fieldlist)
        list0 = [None] * (ln+1)
        list0[0] = sym
        list1 = [None] * (ln+1)
        list1[0] = 'BarTp=' + quote_type
        list3 = [None] * (ln+1)
```

The code

```
        str0 = xlb.range_string('$A$1',1,1,0,i*(ln+1)).split(':')[0]
        str1 = xlb.range_string('$A$1',1,ln-1,2,1+i*(ln+1))
        str2 = xlb.range_string('$A$1',1,1,1,i*(ln+1)).split(':')[0]
        list3[0] = s0 + str0 + ',' + str1 + s1 + str2 + s3
        rangelist = [list0, list1, fieldlist + [None], list3]

        while 1:
            try:
                xlws.Range(xlb.range_string('$A$1',4,ln+1,0,i*(ln+1))).Value = rangelist
                break
            except:
                pass

    xlws = None
    xlwb = None
    del xlws
    del xlwb


def getlist_abtcell(path, wbname, wsname, cellstr):
    """
    Returns a list of values around a cell

    E.g.
    x = getlist_abtcell("c:\\users\\tanchonghui","Apr13A.xlsx","Sheet1","A2")
    """
    xlwb = win32com.client.GetObject(path + "\\" + wbname)
    xlws = xlwb.Sheets(wsname)
```

The code

```
    tmp = xlws.Range(cellstr).CurrentRegion.Value

    xlws = None
    xlwb = None
    del xlws
    del xlwb

    return [x[0] for x in tmp]


if __name__ == '__main__':
    # Get the data around the cell (assumed a row or a column) into a list
    x = getlist_abtcell('c:\\documents and settings\\chonghuitan\\my documents','Book1.xlsx','Sheet1','A1')

    # Use the list of ticker symbols found - typically with a command like =bds("SPX Index","OPT_CHAIN")
    # and populate the worksheet with the necessary strings to poll BB data
    # Click Refresh Worksheet on BB add-in to populate with data
    intraday_lay0('c:\\documents and settings\\chonghuitan\\my documents','Book1.xlsx','Sheet2',x)
```

Exercises

Exercises

- 1) Modify the Python Compute-Core example so that the prices of vanilla put options are computed. (Recall that the payoff of a put option is Max(K – S(T), 0).)

- 2) Generate a pair a correlated Brownian Motions by following the prescription below. (Display the numerics and price paths on Excel a la the example here.)

  Theoretical Background

  A pair of correlated Brownian Motions may be defined as follows:

$$Z_t^1 = W_t^1$$

$$Z_t^2 = \rho W_t^1 + \sqrt{1-\rho^2}\, W_t^2 \qquad (W_0^1 = W_0^2 = 0)$$

Exercises

where W^1 and W^2 are independent Brownian Motions and \rho is a number between -1 and 1.

The resulting Brownian Motions Z^1 and Z^2 are correlated with correlation coefficient \rho.

$$dZ_t^1 dZ_t^2 = \rho dt$$

<u>Implementation</u>

Discretize the equations above as follows:

$$Z^1_{(i+1)\delta} - Z^1_{i\delta} = W^1_{(i+1)\delta} - W^1_{i\delta}$$

$$Z^2_{(i+1)\delta} - Z^2_{i\delta} = \rho\left(W^1_{(i+1)\delta} - W^1_{i\delta}\right) + \sqrt{1-\rho^2}\left(W^2_{(i+1)\delta} - W^2_{i\delta}\right)$$

References

Ref.

- Excel Worksheet Functions
  - http://office.microsoft.com/en-us/excel/HP100791861033.aspx
- Excel
  - http://www.dailydoseofexcel.com/
- Excel Object Model
  - http://msdn.microsoft.com/en-us/library/wss56bz7%28VS.80%29.aspx

Ref.

- PyWin32
  - http://docs.activestate.com/activepython/2.4/pywin32/PyWin32.HTML
  - http://timgolden.me.uk/pywin32-docs/PyWin32.html