2D Plotting with Excel using COM

INTRODUCTION
The purpose of this document is to describe two scripts which demonstrate the generation of a Gaussian dataset and a data plotting routine that routes data from FRED to Excel directly via COM. Although the primary intention of this article is to demonstrate how to plot data in Excel directly from FRED, keep in mind that the COM connection is generic and one can access all of Excel’s functionality for spreadsheet manipulations in addition to accessing the plotting routines.

The FRED document associated with this document contains two embedded scripts:

1. excelPlotting – This script contains a single subroutine which generically accepts data and labels for plotting.
2. generateGaussianPlot – This script generates a Gaussian dataset and then calls the excel plotting method from the excelPlotting embedded script as an example of how to implement the plotting routine.

EXPLANATION
We start with the embedded script, generateGaussianPlot. This script contains some simple user-parameters on lines 15-17 which designate a start and ending value range and number of samples within the value range. The data points that are generated within this range will be sent over to Excel for plotting. In the example below, we will generate 51 data points between the values -10 < x < 10.

```
12 'USER SETUP
13 Dim x_start As Double, x_end As Double
14 Dim num_samps As Long
15 x_start = -10
16 x_end = 10
17 num_samps = 51
```

The script then calls a custom subroutine called “linspace” in order to generate an evenly spaced set of sample points within the designated range.

```
19 'create a linear array of numbers from min to max
20 Dim x_vals() As Double
21 ReDim x_vals( num_samps-1 )
22 linspace( x_start, x_end, num_samps, x_vals() )
```
After generating the evenly spaced samples, we loop over the sample points and calculate the value of a Gaussian function which is defined by the custom “gauss” routine. The returned function values are stored in an array.

```vba
Sub linspace( ByVal d_start As Double, _
               ByVal d_end As Double, _
               ByVal num_samps As Long, _
               ByRef ret_vals() As Double )
    Dim cur_ind As Long
    cur_ind = 0
    Dim cur_samp As Double
    For cur_samp = d_start To d_end+0.0001 Step (d_end-d_start)/(num_samps-1)
        ret_vals( cur_ind ) = cur_samp
        cur_ind = cur_ind + 1
    Next cur_samp
End Sub
```

After generating the evenly spaced samples, we loop over the sample points and calculate the value of a Gaussian function which is defined by the custom “gauss” routine. The returned function values are stored in an array.

```vba
' populate functions to plot
Dim x_func() As Double
Dim cur_x As Long
ReDim x_func( num_samps-1 )
For cur_x = 0 To UBound( x_vals, 1 )
    x_func( cur_x ) = gauss( x_vals( cur_x ), 1, 2 )
Next cur_x
```

```vba
Function gauss( x_val, amp, width ) As Double
    gauss = amp * Exp( -PI* ( x_val / width )^2 )
End Function
```

Next, we display the sample points and Gaussian function values in the FRED output window.

```vba
'dump plot data to output window
Print "X Val" & Chr(9) & " Y Val"
For cur_x = 0 To UBound( x_vals, 1 )
    Print x_vals( cur_x ) & Chr(9) & x_func( cur_x )
Next cur_x
```

Then, at last, we create an object out of our embedded script, “excelPlotting”, so that we can call the “excelPlot2DScatter” routine that is defined within it. We pass our x,y data into this routine, which handles all the creation of the Excel COM object and plotting.
An Object works in the following way for the case of our excelPlot2DScatter routine. We have an embedded script called “excelPlotting” that we wish to access from our current script, “generateGaussianPlot”.

If we open up the “excelPlotting” script we see the following:

```vba
Option Explicit

' This script is used to plot arbitrary sets of data on an excel scatter plot. Call the subroutine and pass it the following items:
' 1) xData() = vector of data points for the x-axis
' 2) xTitle = string name for x-axis data
' 3) yData() = vector of data points for the y-axis
' 4) yTitle = string name for the y-axis data
' 5) cTitle = string name for the chart title
' 6) seriesName = string name for the legend entry

Sub excelPlot2DScatter( ByVal xData() As Double, _
                        ByVal xTitle As String, _
                        ByVal yData() As Double, _
                        ByVal yTitle As String, _
                        ByVal cTitle As String, _
                        ByVal seriesName As String )

  ' Objects to be used in connection with Excel
  Dim excelApp As Object
  Dim excelWB As Object
```

What we are doing, then is wrapping up the entire “excelPlotting” embedded script into an Object, and then reaching into the object and calling the “excelPlot2DScatter” subroutine.
Okay, so now let's take a closer look at the excelPlot2DScatter subroutine. The subroutine takes six arguments when it is called:

- **xData()**: An array of doubles which are the x-data points
- **xTitle**: A string title for the x-axis of the plot
- **yData()**: An array of doubles which are the corresponding y-data points
- **yTitle**: A string title for the y-axis of the plot
- **cTitle**: A string title for the entire chart
- **seriesName**: A string name for the legend of the chart

```vba
Sub excelPlot2DScatter(xData() As Double, xTitle As String, yData() As Double, yTitle As String, cTitle As String, seriesName As String)
```

Next, we define four Objects which will allow us to use Excel's functionality from FRED. As described above, we are wrapping up various Excel components into objects (COM objects!) and then using the "dot" notation on them to call their functions and subroutines.

```vba
Dim excelApp As Object
Dim excelWB As Object
Dim excelRange As Object
Dim excelChart As Object
```

In the above chain of events, we start by creating the “Excel Application” object on line 31, then from the application object (excelApp variable in the FRED script) we use dot notation to access the...
Workbooks methods and specifically call the Add method. This then returns to us a new object, which is stored in excelWB (line 32). Lastly, from the excel workbook object we create an Excel Range object by calling the Cells method of the Active Sheet of the workbook (I know, I know, levels within levels of notation here).

Using the Range object we can modify the values of the active spreadsheet. In the code below, we print a header row and then loop over our x and y data values and print them to the cells of the spreadsheet.

```vba
'Print headers
excelRange.Cells(1,1).Value = xTitle
excelRange.Cells(1,2).Value = yTitle

'Print data
Dim curRow As Long
For curRow = 2 To UBound(xData, 1) + 2
    excelRange.Cells(curRow,1).Value = xData(curRow-2)
    excelRange.Cells(curRow,2).Value = yData(curRow-2)
Next curRow

'size the excel columns
excelWB.Worksheets("Sheet1").Columns("A:B").AutoFit
```

At this point, we have populated the spreadsheet with data and we would like to make the entire Excel application object visible to us. So, we fit the cells in the spreadsheet to their respective data values and then make the application visible.

```vba
'size the excel columns
excelWB.Worksheets("Sheet1").Columns("A:B").AutoFit
'make Excel visible
excelApp.Visible=True
```

Finally, we get to the code which actually does the plotting. We start by initializing our Chart object by calling the Add method from the Charts group of the Workbook object (excelWB.Charts.Add). The methods that allow us to modify the chart are now available to us through the excelChart object. In the code below, we specify the chart type, titles, data ranges, etc.
The last step is to “release” all of the Excel objects that we have initialized so that they are no longer bound to FRED.

```
'Empty the excel objects
Set excelRange=Nothing
Set excelWB=Nothing
Set excelApp=Nothing
```

The outcome of this entire process is that by running the generateGaussianPlot embedded script you should see a copy of Excel open up automatically and generate a chart from the FRED data.
CLOSING REMARKS

The excelPlotting script can be copied into any FRED file and called in the same manner described in this document. Note that the excelPlot2DScatter has been written generically to accept any type of x-y scatter data.