Topics in Excel's VBA 2003: Functions and Add-Ins

Edward M. Rosen
© ***EMR Technology Group***

1. Introduction

The use of Excel 2003 and its macro language, Visual Basic for Applications (VBA), can be used to carry out many non-trivial engineering calculations. There are several attractive features associated with the use of VBA:

1. Excel is widely used in industry and academia.
2. There are a number of books describing the VBA language [1, 2, 3].
3. The use of VBA can simplify complicated spreadsheets and make them more understandable.
4. There are many add-ins available that greatly extend the computational capability of Excel.
5. Many other systems and programs can be called from VBA.

VBA is a rich object oriented type language. It does require some effort to learn the IDE (Integrated Development Environment) and the VBA language, but enough can be learned rapidly so that it can quickly become useful for performing engineering calculations.

2. Accessing the Visual Basic Editor (VBE) and Project Explorer

Coding in VBA is done in a procedure that resides in a module developed in the Visual Basic Editor. To access the editor use **Ctrl+F11** or use the menu item *Tools➔Macro➔Visual Basic Editor.*

Once in the editor, use **Ctrl+R** to gain access to Project Explorer. Project Explorer lists all of the workbooks that are currently open (including add-ins). Each workbook is known as a project (default name is VBA project).

A Project Explorer window (with nine projects) is shown in Figure 2.1

Among the open workbooks in Project Explorer is the current workbook (Book2). To add a module to a project (the current one or other projects), highlight the project and select *Inset➔ Module*. This will open a code window for the macro.

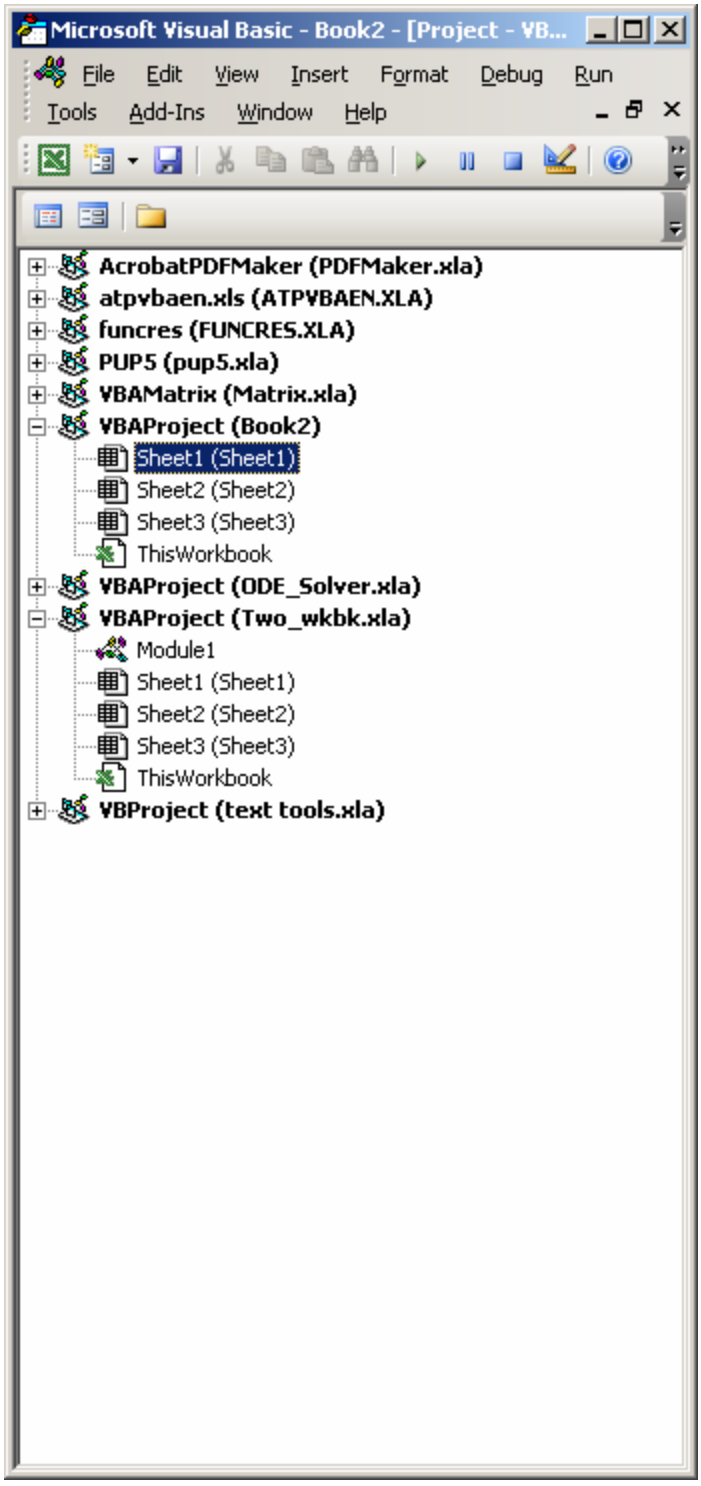


Figure 2.1 Project Explorer Window

3. Hidden Files

It is useful to show files that are hidden by Windows XP especially when working with add-ins.

To unhide these files:

From any folder window [4] :

Tools ➔ *Folder Options* ➔ *View*

Check the *Show Hidden files and folders* as shown in Fig 3.1.

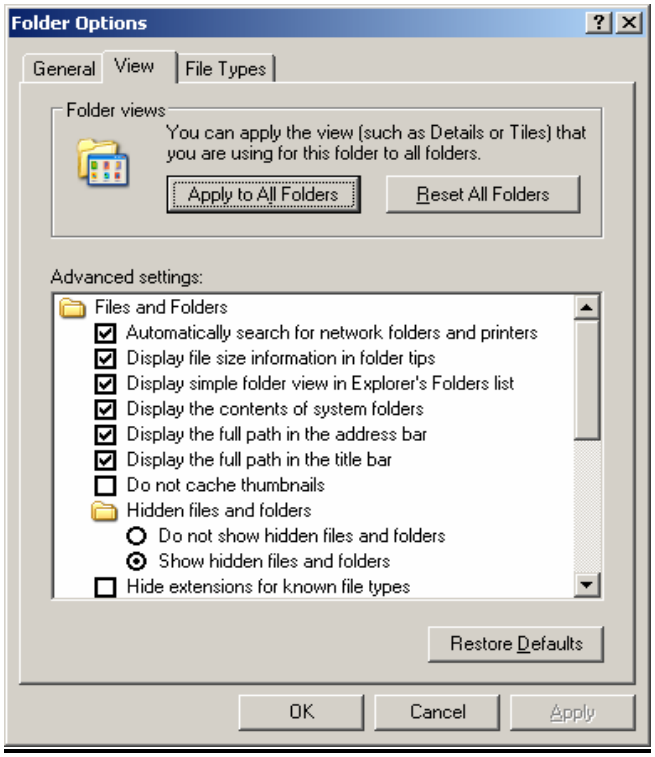


Figure 3.1 Folder Options Dialog Box

4. Excel Functions vs. VBA Functions

There a number of built-in and add-in functions available for use in the Excel spreadsheet and to the VBA programmer in the VBE. To make these functions available, however, they must be checked in the Add-Ins dialog box shown in Figure 4.1.

To access the Add-Ins dialog box:

Tools ➔ *Add-Ins*

Note that both the Analysis ToolPak (for Excel functions) and Analysis ToolPak Analysis ToolPak – VBA (for VBA functions) are checked:

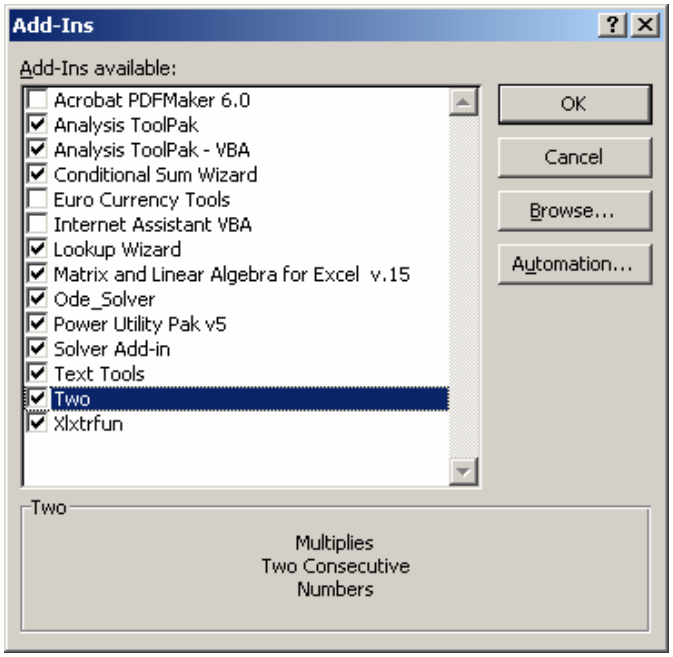
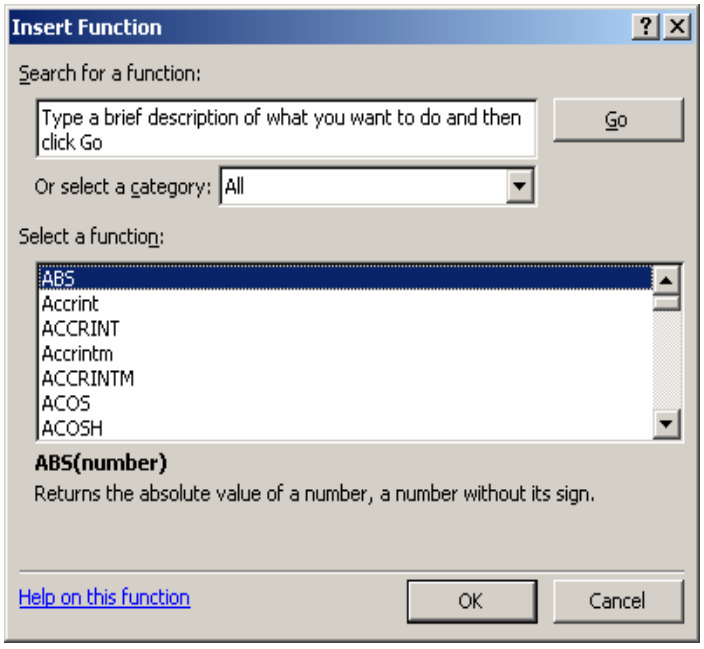


Figure 4.1 Add-Ins Dialog Box



The Excel functions which are available may be located by using the Insert (Paste) Function dialog box.

Access this dialog box by clicking the function button f_x on the formula bar (or selecting *Insert* ➔*Function*). The dialog box is shown in Figure 4.2.

Figure 4.2 Insert Function (Paste) Dialog Box

A list of the functions (and their descriptions) available to VBA (which may differ from the Excel functions) can be found by going to VBE and clicking on

? MicroSoft Visual Basic Help
MicroSoft Visual Basic Documentation
Visual Basic Language Reference
Functions

When coding a VBA macro, Excel functions may be used by specifying the function as Application.xxx where xxx is an Excel function.

5. Add-Ins, Commercial and Free

As can be seen by a search for 'Excel Add-Ins' on *Google*, there are an extraordinary number of Add-Ins available for Excel. These Add-Ins vary in price, quantity and quality.

Walkenbach [2] supplies a number of Excel Add-Ins with his book. To obtain the source code, however, there is a charge. Matrix.xla [5] has a very large number of Excel Add-Ins which are not only free, but whose source code is available. Most add-ins are password protected and the source is not available. Once loaded, the Matrix routines will show up in Project Explorer with other *.xla add-ins. The routines can be invoked using the Insert (Paste) Functions Dialog box.

Another free add-in is (xlxtrfun.xll) for interpolation, regression and curve fittings (2D and 3D) available at Reference [6]. This is a stand-alone DLL file, written in C and compiled. The add-in can be loaded by placing it in the XLSTART folder:

Documents and Setting
Owner
Application Data
MicroSoft
Excel
XLSTART

The routines behave in the same way as Excel functions and are shown in Excel's Add-Ins dialog box but do not show up in Project Explorer. They are accessed through the ' Insert (Paste) Function Dialog box and can be found in the Engineering category.

6. Custom Add-Ins

6.1 *Creating an Add-In*

A custom add-in may be created in the following way:

1) Open Excel and bring up an empty macro sheet (see 2. above).

2) Code the macro. Figure 6.1 is an example of a function procedure to multiply two numbers together (Function Two).

```
Option Explicit

Public Function Two(prm)

Application.Volatile True

Two = prm(1) * prm(2)


End Function
```

Figure 6.1 A Simple Function Procedure

The Option Explicit statement requires that all the variables be declared as to their type (integer, single, etc). In this case there are no variables be be declared.

The Public declaration indicates that the function is available to other spreadsheets

The Application .Volatile statement ensures that the function will be executed each time there is a change in the spreadsheet.

.

3) Return to the spreadsheet (**Alt+ F11**). A description of the macro function can be entered in the following way (it will be displayed when the function is selected in the Insert Function Dialog Box).

   a) Tools➔ Macro➔ Macros. This will display the Macro Dialog Box.
   b} Enter the name of the macro in the Macro name box (Two).
   c) Click on Options
   d) Enter the description of the macro in the Description box (maximum of three lines)

Multiples
Two Consecutive
Numbers

e) Click OK
f) Click on X (Exit)

4) Enter the file name *Two_wkbk*
Click on *Save as type* and go to the last entry:

MicroSoft Office Excel Add-In (.xla)*. Click on it.
Click *Save*

The workbook add-in file *Two_wkbk.xla* containing the function *Two* will be saved in the following location. (Note that hidden files need to be unhidden):

Documents and Setting
Owner
Application Data
MicroSoft
Addins

5) A window will now appear asking about the properties of the saved workbook file *Two_wkbk.xla* :

.

Click on the Summary tab.

In the comments enter up to four lines describing the workbook. This description will appear in the Add-Ins Dialog Box when the file *Two_wkbk* is Selected.

Click OK

6) Click File on the Menu and then Exit
Do not save the changes.

When Excel is next brought up and *Tools* ➔ *Add-Ins* is selected, an unchecked box with the file name *Two_wkbk* will appear. Check it to make the function *Two* available to the Excel spreadsheet.

6.2 *Deleting an Add-In from The Add-Ins Dialog Box*

Deleting an entry in the Add-Ins Dialog Box is rather awkward:

1) Locate the file *Two_wkbk.xla* . Note it was stored in *Documents and Settings* ➔*Owner* ➔*Application Data* ➔*MicroSoft* ➔*Add-Ins* Delete the file (or change its name).

2) When loading Excel a message will appear indicating that the file can't be found. Click *OK*.

3) Bring up the the Add-Ins Dialog Box and highlight the file *Two_wkbk* . A message will appear asking if the entry should be deleted. Enter *Yes* .

7. The VBE Reference

Procedures in other workbooks (*.xls) whether open or not may be used in your current workbook.

For example, assume that there is a workbook *Three_wkbk.xls* that contains Functions *Three* and *Four*. Function *Three* will multiple three consecutive numbers and Function *Four* will multiply four consecutive numbers, It is desired to use the function *Four* in a Sub procedure *TryNew* as shown in Figure 7.1

```
Option Explicit

Public Sub TryNew()

Dim X(1 To 4) As Single
Dim Prod2 As Single

X(1) = 3
X(2) = 4
X(3) = 5
X(4) = 12

Prod2 = Four(X)

MsgBox " Value of Prod2  = " & Prod2

End Sub
```

Figure 7.1 Sub Procedure TryNew

In Project Explorer, highlight the current project. Go to *Tools* ➔ *Reference* ➔*Browse*

After locating *Three_wkbk.xls* click *Open*. The References-VBAProject dialog box opens with VBAProject checked. Click OK.

There may be a problem at this point however. This is due to fact that all projects are named (by default) VBAProject (Figure 2.1). A message indicating there is a name conflict may be received.

If this happens click OK and go back to Project Explorer. Highlight VBAProject *Three_wkbk.xls* , right click and go to *Properties*. Rename the project *ThreeZ* (or something else).

Make sure the current project is highlighted. Go again to *Tools* ➔ *Reference* and check *Threez* in the Available References list. Click *OK*. The Sub procedure shown in Figure 7.1 will now execute properly.

Note a reference to *Three_wkbk.xls* is now associated with the current project in Project Explorer.

Functions *Three* and *Four* now can also be used in the Excel spreadsheet by referring to *Three* or *Four* or *Three_wkbk.xls!Three* and *Three_wkbk.xls!Four* using Insert Function.

8. A Free Add-In: Singular Value Decomposition

8.1 *An Excel Add-In*

One of the routines from the Matrix package [5] is for Singular Value Decomposition. It is used to analyze non square matricies.

The three functions SVD_U, SVD_D and SVD_V calculate respectively the U , D V matricies such that

$$A = U * D * V^T$$

where A (for example) is 3 x 2, U is 3 x 2, D is 2 x2 and V is 2 x 2. Each of the three routines are invoked separately. Figure 8.1 is a spreadsheet implementation.

The input matrix A is given in C9:D11. Each of the functions are function arrays which means a suitable output area must first be selected. For the matrix U the area F15:G17 is highlighted, then ' = SVD_U(C9:D11)' is entered into cell F15 a with **Ctrl+Shift+Enter.** Similar procedures are followed for the D and V matricies in F21:G22 and F24:G25.

	C	D	E	F	G	H	I	J
4	Singular Value Decomposition							
5	$A = U*D*V^T$		A(n,m)					
6	p = min (n,m)		U is orthonormal (n x p), D is a square Diagonal matrix (p x p)					
7			V is a orthogonal matrix (m x p)					
8	Input Matrix A							
9	1	1						
10	0	1						
11	1	0						
12								
13				Output			Equivalents	
14								
15			SVD_U	-0.8165	0.0000		-sqrt(6)/3	0
16				-0.4082	-0.7071		-sqrt(6)/6	-sqrt(2)/2
17				-0.4082	0.7071		-sqrt(6)/6	sqrt(2)/2
18								
19								
20								
21			SVD_D	1.7321	0.0000		sqrt(3)	0
22				0.0000	1.0000		0	1
23								
24			SVD_V	-0.7071	0.7071		-sqrt(2)/2	sqrt(2)/2
25				-0.7071	-0.7071		-sqrt(2)/2	-sqrt(2)/2
26								
27			Matrix Multipy U * D					
28								
29			-1.4142	0.0000				
30			-0.7071	-0.7071				
31			-0.7071	0.7071				
32								
33			V^T					
34			-0.7071	-0.7071				
35			0.7071	-0.7071				
36								
37								
38			A Matrix Generated = $U*D*V^T$					
39								
40			1.0000	1.0000				
41			0.0000	1.0000				
42			1.0000	0.0000				

Figure 8.1 Spreadsheet Using SVD Routines

To check the output, the A matrix is regenerated from the U, D avd V matrices using the Excel function MMULT.

8.2 *Use in a VBA Function*

Since the Matrix system [5] is not passward protected, the source may be copied and utilized in a VBA macro. Figure 8.2 is a macro with array Function TrySVD which calls the three routines SVD_U, SVD_D and SVD_V. The following procedures copied from the Matrix source are not shown in Figure 8.2:

```
Sub svd_decomp
Private Function pythag
Private  Function sign
Public Function SVD_U
Public Function SVD_D
Public Function SVD_V
Private Function Max
Private Function Min
Private Function svd_sort
```

The spreadsheet which invokes TrySVD is similar to Figure 8.1 except the number of rows, columns and a code is passed in a parameter vector (prm).

```
Public Function TrySVD(AA, prm)

'   AA is the input matrix
'   prm (1) is the number of rows
'   prm (2) is the number of columns
'   prm (3) is the code for the matrix desired

Dim m, n, i, j    As Integer

m = prm(1)
n = prm(2)
icase = prm(3)

ReDim A(1 To m, 1 To n)

ReDim U(1 To m, 1 To n)
ReDim D(1 To n, 1 To n) As Double
ReDim v(1 To n, 1 To n)

For i = 1 To m
  For j = 1 To n
    A(i, j) = AA(i, j)
  Next j
Next i

Select Case icase

   Case 1: U = SVD_U(A): TrySVD = U

   Case 2: D = SVD_D(A): TrySVD = D

   Case 3: v = SVD_V(A): TrySVD = v

End Select

End Function
```

Figure 8.2 TrySVD Array Function

9. Conclusions

Add-ins are for use in the Excel spreadsheet. However, if the add-in's source is available, the source may be copied to a VBE macro and be used in VBA. Procedures in other workbooks may also be used either in the spreadsheet or VBA by using the VBE References.

Much of the difficulty in using VBA may be in finding the way around the IDE rather than the VBA language. Once this difficulty is overcome, use of VBA can be very attractive.

10. References

1. Roman, Steven, *Writing Excel Macros with VBA* , O'Reilly, 2nd Edition June 2002

2. Walkenbach, John, *MicroSoft Excel 2000 Power Programming with VBA* IDG Books WorldWide, Inc, Foster City, CA 1999

3. Harris, Matthew, *Teach Yourself Visual Basic for Applications in 21 Days* Sams Publishing 1994

4. Pogue, David *Windows XP, Home Edition: The Missing Manual,* Pogue Press, O'Reilly (2002)

5. http://digilander.libero.it/foxes/matrix_review.htm

6. http://www.xlxtrfun.com/XlXtrFun/XlXtrFun.htm