Excel Programming Tutorial 1
Macros and Functions

by Dr. Tom Co

Department of Chemical Engineering
Michigan Technological University
(8/31/07, 11/11/07)

Excel Version: Excel 2007

Basic Concepts:

- A macro is a user-programmed routine that can be accessed by shortcut-keys.
- Visual Basic for Applications (VBA) is a programming language used by Microsoft Office Programs to allow the user to develop customized routines and functions.
- Functions are routines that return values.

Features:

1. One approach for developing macros is by recording manual steps
2. Another approach is by loading Visual Basic files
3. The macros can be modified and enhanced by using the Visual Basic Editor.
4. Properties, such as name and shortcut keys, of the macro can be changed using the **[Macro Toolbar]**
5. Shortcut keys of the macro can also be included in the **[Quick Access Toolbar].**

Example 1: Develop a macro to calculate the average of five cells to the left of a selected cell.

   Step 0. Initialize a spreadsheet and fill five vertical cells with numbers, similar to Figure 1.

Figure 1.

   Step 1. Select the cell to the right (cell **C2** in Figure 1).

Step 2. Start macro-recording by selecting **[View]→[Macro]→[Record Macro…]** and modify macro properties (see Figure 2).

Figure 2.

Step 3. Input formula in cell **C2** and then stop macro-recording by selecting **[View]→[Macro]→[Stop Recording].** (see Figure 3)
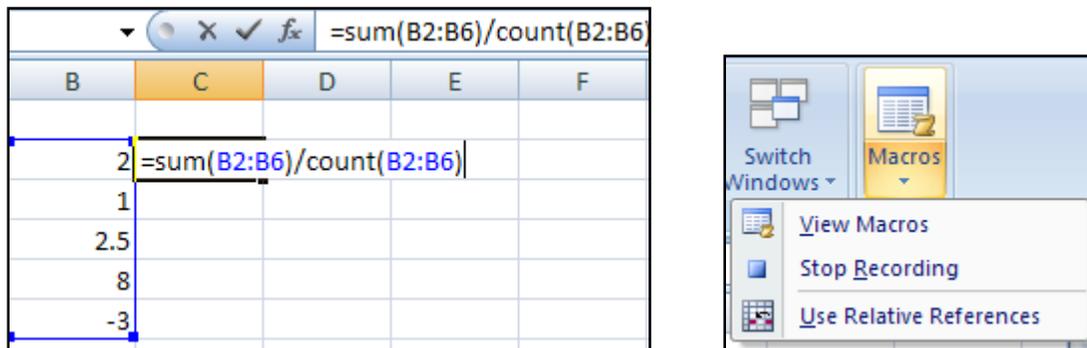
=sum(B2:B6)/count(B2:B6)

Figure 3.

Alternatively, you can click on the [Stop] button, usually located at the bottom left corner of the Excel window (see Figure 4).
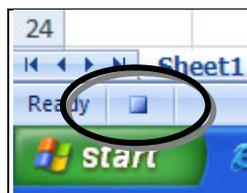
Figure 4.

Step 4. Test the macro by filling other five cells with numbers, then select the cell to the right and then click shortcut-key. (based on Figure 2, we had chosen **[Ctrl-a]**)

| B8 | | f_x | =SUM(A8:A12)/COUNT(A8:A12) |

	A	B	C	D	E	F	G
8	1	0.6					
9	2						
10	3						
11	-3						
12	0						

Figure 5. After clicking shortcut-key while cell **B8** was selected.

Step 5. Click [**Alt-F11**] to access VBA module as shown in Figure 6.


```vba
(General)                                              MyAverage

Sub MyAverage()
'
' MyAverage Macro
' Averages a five vertical cell-range on the left
'
' Keyboard Shortcut: Ctrl+a
'
    ActiveCell.FormulaR1C1 = "=SUM(RC[-1]:R[4]C[-1])/COUNT(RC[-1]:R[4]C[-1])"
    Range("C3").Select
End Sub
```

Figure 6.

Remarks:

1. The single-quote marks the start of comments (characters to the right are ignored).

2. Remove the line with [`Range("C3").Select`] if it exists. Otherwise, the macro will always return to cell **C3.**

Step 7. Save the macro as a Visual Basic file: select **[File]→[Export File..]** and save as a Basic [*.bas] file. *(Note: make sure* **Module1** *is highlighted*).
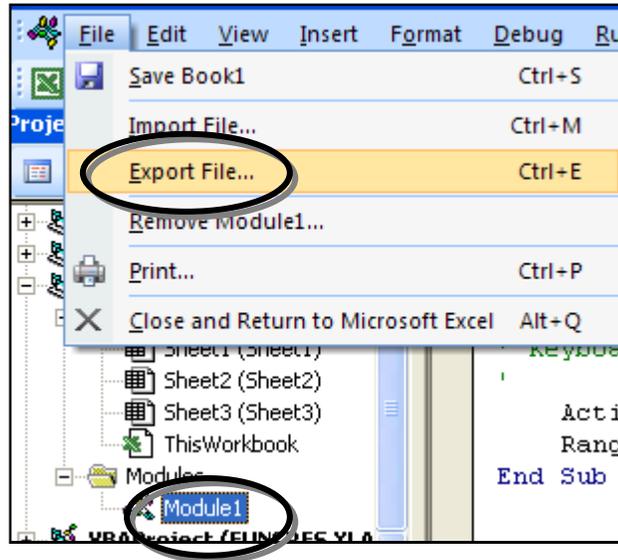
Figure 7.

Example 2: Loading a macro that was previously saved.

> **Step 0**. Start with a new spreadsheet.

> **Step 1**. Click **[Alt-F11]** to open the Visual Basic Editor, then select **[File]→[Import File..]**to import the basic file.

> **(Remark**: This is one method to access previously-saved macros that will not be affected too much with the security-handling issues of Excel 2007. Another method is to build a customized add-in.)

Example 3: Saving and loading macro-enabled worksheets.

> This is in case you want to save the spreadsheet together with the macros and functions you had built for that spreadsheet.

> **Step 1**. Save the worksheet containing the macros by selecting to save it as a "macro-enabled workbook" (see Figure 8.). Close the workbook to test how to load it.
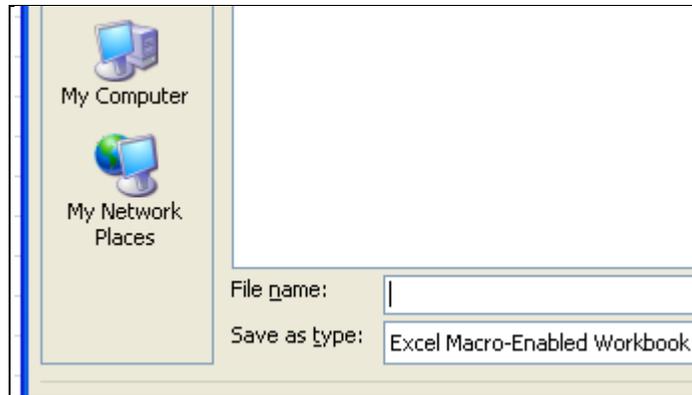
Figure 8.

Step 2. Open the workbook. The **[Security Warning]** tab should appear on the top part. Click on **[Options...]**.
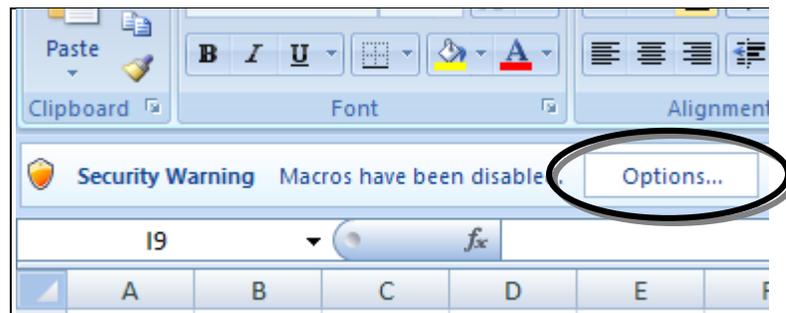

Figure 9.

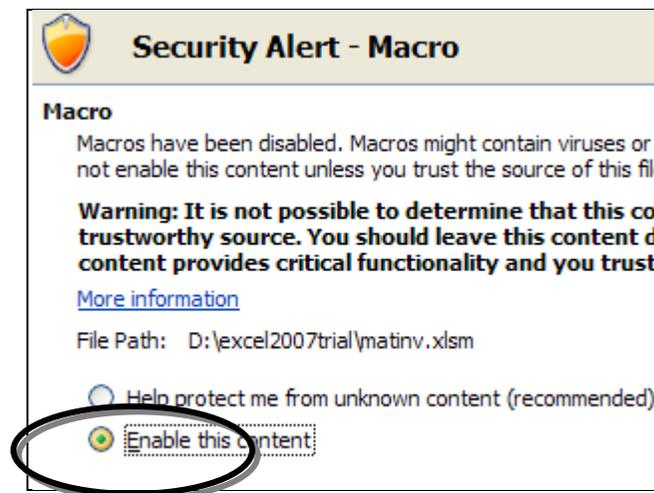**Step 3**. Select the option that enables the macro.

Figure 10

TIPS:

1. In case you have forgotten the different shortcut keys, you can view, run or edit the macros you built by selecting **[View]→[Macros]→[View Macros],** or click **[Alt-F8].**

2. Another approach is to add the macro into the Quick Access Toolbar. To do so, click on the customization button (see Figure 7) and select [**More Commands…**].
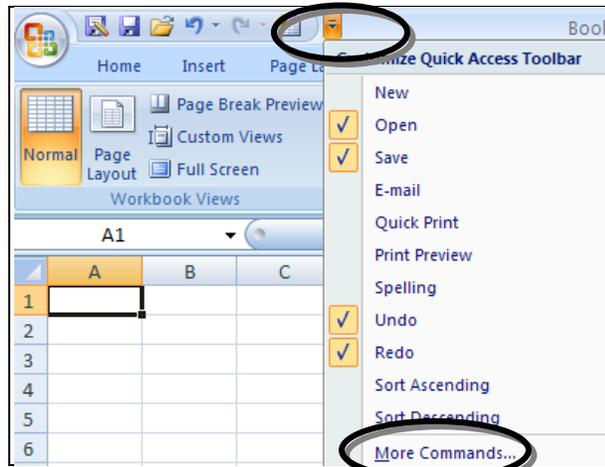

Figure 7.

Then select the desired macro from the **[Macros]** list and click **[Add>>]** button. (See Figure 10). After clicking [OK], you should notice that addition of the macro button. (see Figure 11).
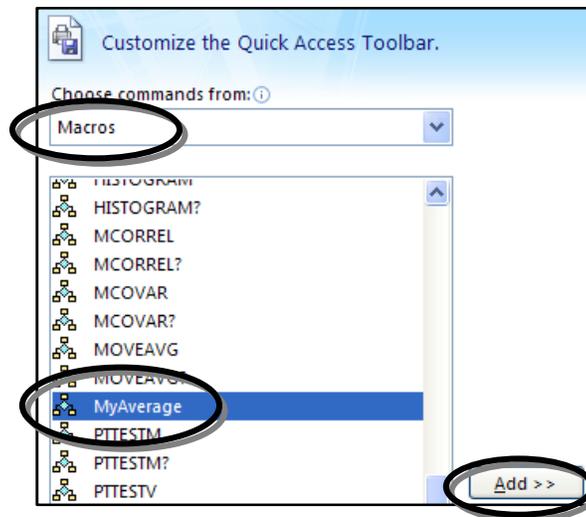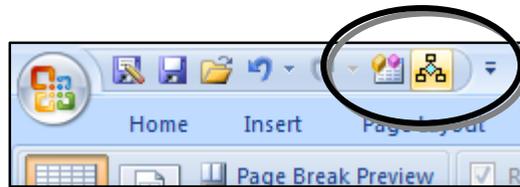

Figure 10.

Figure 11.

Example 3: Writing a subroutine to solve a set of linear equations while allowing the user to input cell-range.

Step 0. Start with a new spreadsheet.

Step 1. Click **[Alt-F11]** to open the Visual Basic Editor.

Step 2. Select **[Insert]→[Module].** (see Figure 12)

Figure 12.

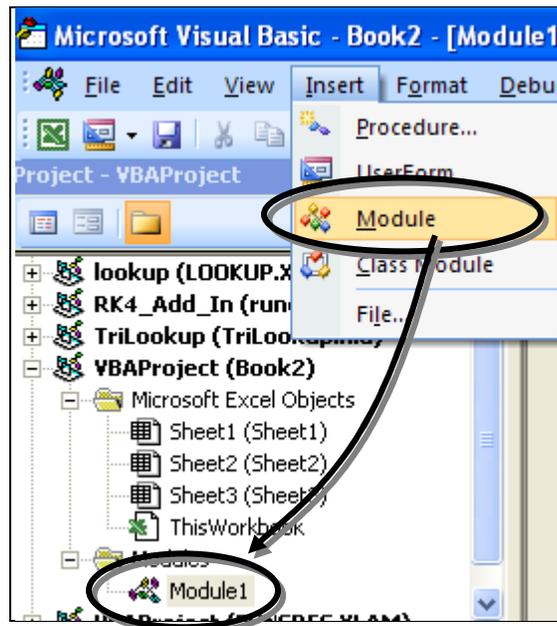**Step 3**. Type in the following code:

```
(General)                                            ▼  │ SolveAxb

    Sub SolveAxb()

        Set A = Application.InputBox("Matrix A", Type:=8)
        Set b = Application.InputBox("Vector b", Type:=8)
        Set x = Application.InputBox("Vector x", Type:=8)

        A.Name = "A"
        b.Name = "b"
        x.Name = "x"

        x.FormulaArray = "=MMULT(MINVERSE(A),b)"

    End Sub
```

Figure 13.

Step 3. Go back to the Excel workbook by clicking **[Alt-F11]** once more. Then click **[Alt-F8]** to change the macro options and select shortcut key (plus add description if desired).
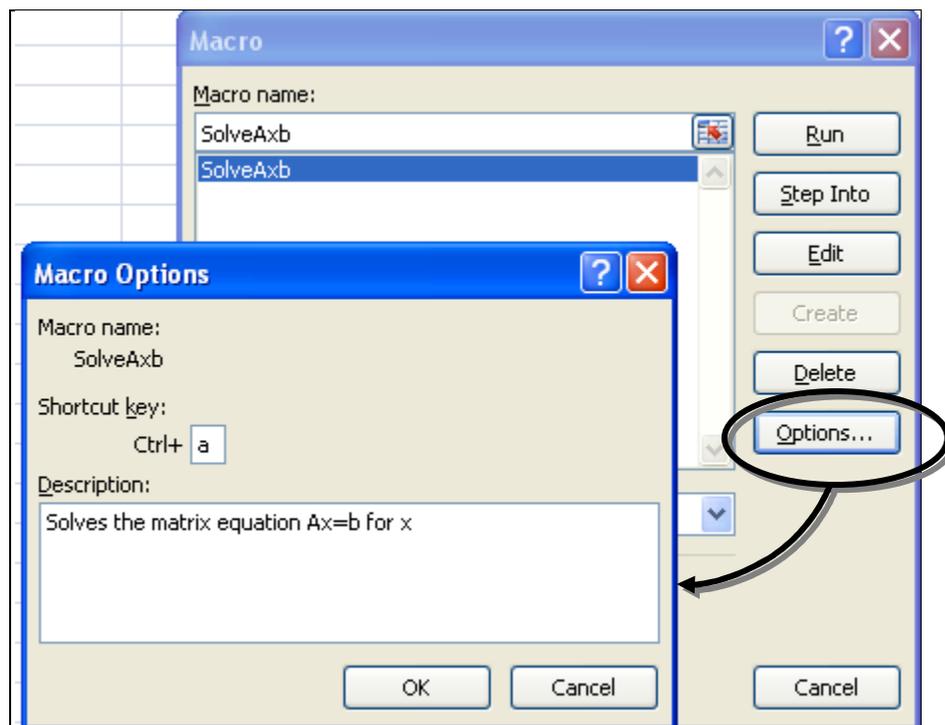

Figure 14.

Step 4. Test the macro. First prepare the spreadsheet with a matrix which will be treated as matrix A and column of cells to be treated as vector b. Then run the macro.

	A		B		X
	1	2	3		
	3	3	6		

Figure 15.

Functions

A function is different from subroutines (or macros) in that they return values.

1. They are bounded by the **Function ...** statement and **End Function** statement.
2. They can take several inputs with different input types
3. Typically, functions are used when iteration loops and decision blocks are needed. Example of iteration loops:

```
a)  x=0
    For i = 1 to 10
            x=x+i^2
    Next i
b)  y=0
    Do
        i=i+1
        y = y+i^2
    Loop While y<100
```

Example of decision loops:

```
a)  If  (A>B) then
        C = sqrt(A-B)
    Elseif (A<B) then
        C = sqrt(B-A)
    Else
        C = 0
    End If

b)  Select Case n
        Case 1
         A=1
        Case else
         A=A+B
    End Select
```

Example 4: Writing a function to evaluate the log mean of two values.

Step 0. Repeat the steps 0 to 2 of Example 3.

Step 1. Type the following code:

```
(General)

    Function logmean(t1, t2)

        If t1 = t2 Then
            logmean = t1
        Else
            logmean = (t2 - t1) / Log(t2 / t1)
        End If

    End Function
```

Figure 16.

Step 2. Test the function.

	A	B	C
1	120	130	124.933
2	110	110	110

=logmean(A1,B1)

=logmean(A2,B2)

Example 4: Writing a function for solving the average of a range of cells.

```
Function newaverage(x As Range)

    n = x.Count
    Sum = 0
    For i = 1 To n
        Sum = Sum + x.Cells(i)
    Next i
    newaverage = Sum / n

End Function
```

Example 5: Writing a function that evaluates the real roots of a cubic equation

For the cubic equation:

$$ax^3 + bx^2 + cx + d = 0$$

One of the real roots can be found using Newton's method:

$$x^{[k+1]} = x^{[k]} - \frac{f(x^{[k]})}{df/dx\,(x^{[k]})}$$

Where the iteration is repeated until the difference between $x^{[k+1]}$ and $x^{[k]}$ is below a specified tolerance.

11

After this first root, x_1, has been found, two real roots will exist if the discriminant Δ is non-negative, where

$$\Delta = \alpha^2 - 2\alpha x_1 - 3x_1{}^2 - 4\beta$$
$$\alpha = \frac{b}{a}$$
$$\beta = \frac{c}{a}$$

If $\Delta \geq 0$, the other two real roots are given by

$$x_2 = \frac{-(\alpha + x_1) + \sqrt{\Delta}}{2}$$
$$x_3 = \frac{-(\alpha + x_1) - \sqrt{\Delta}}{2}$$

A function to implement this is given in Appendix 2 as **realCubeRoot(a,b,c,d,n)**.

Appendix 1. Some shortcut-keys for Excel VBA

Key Combination	Effects
[Alt-F8]	View Macro
[Alt-F11]	Toggle between VBA editor and Excel worksheet

Appendix 2. VBA code for the function realCubeRoot(a,b,c,d,n):

```
Function realCubeRoot(a, b, c, d, n)
'
'    computes the nth real root of the cubic equation
'
'        a x^3 + b x^2 + c x + d = 0
'
'    =================================================

    xold = 1
    iter = 0
    Do
        iter = iter + 1
        f = a * xold ^ 3 + b * xold ^ 2 + c * xold + d
        df = 3 * a * xold ^ 2 + 2 * b * xold + c
        xnew = xold - f / df
        Err = xnew - xold
        xold = xnew
    Loop While (iter < 1000) And (Abs(Err) > 0.000001)
    If n = 1 Then
        realCubeRoot = xnew
    Else
        aa = b / a
        bb = c / a
        real = -(aa + xnew) / 2
        Disc = (-3 * xnew ^ 2 - 2 * aa * xnew + aa ^ 2 - 4 * bb)
        If Disc < -0.0000001 Then
            realCubeRoot = "NA"
        Else
            Disc = Abs(Disc)
            If n = 2 Then
                realCubeRoot = real + Disc ^ (1 / 2) / 2
            Else
                realCubeRoot = real - Disc ^ (1 / 2) / 2
            End If
        End If
    End If
End Function
```