

APM 153 LECTURE SEVENTEEN – Visual Basic for Applications (VBA)

Introduction

(1) VBA is a programming language built into Microsoft Excel. VBA is a modern version of the BASIC programming language developed at Dartmouth in the 1960's.

(2) VBA allows you to program **more sophisticated algorithms** than would be possible using the multi-step method we performed in Excel for Assignment Seven.

The VBA Toolbar

(3) Before we can use VBA, we make sure it is available. On the Excel toolbar, click on **Tools** and choose **Customize**.

(4) Scroll down until you see **Visual Basic** and make sure the box is **checked**.

(5) Once Visual Basic has been checked, the VBA toolbar, or at least some part of it, should be visible on the Excel toolbar or on the workspace.

(6) There are a total of seven “buttons” on the VBA toolbar. The first two are icons that look similar to the buttons on a VCR or DVD player.

(7) The first icon is a triangle and the second is a circle. The triangle icon means **Run Macro** and is like the **Play** button on your DVD.

(8) The circle means **Record Macro**.

(9) The next thing on the VBA toolbar is the word **Security**. Clicking on this word allows you to set or change the security settings on the computer you are using.

Setting Security Levels

(10) Setting the proper security level in Excel is very important. If the security level is set to High, you won't be able to run your VBA programs.

(11) You must set the security level on the computer you are using to Medium or Low.

Using the VBA Editor

(12) After the word “Security”, the next icon looks like a white box with some red, yellow, and blue doodads at the top. This is the icon for the VBA Editor.

(13) Click once on the icon to open the VBA Editor.

Modules

(14) There are two types of VBA programs: functions and subroutines. Both types of programs are **inserted** into the Excel program as a **Module**.

(15) To start a new VBA program in the VBA Editor, click on the word **Insert** and select **Module**.

(16) It is important that you **do not forget** to write your programs as **modules**. If you do, your programs will not work!

Functions versus Subroutines in VBA

(17) Just like script files and functions in Matlab, there are some real and important differences functions and subroutines in VBA.

(18) Subroutines get their input from assigned cells in the worksheet. Functions get their input from the cells entered as input arguments.

(19) Subroutines can produce more than one result at a time. Functions produce a single value (as far as I can tell).

(20) Subroutines write their output to assigned cells in the worksheet. Functions write their single result to the cell in which they are “called”

(21) Subroutines must be “run” like a program. Functions can be called just like any of the built-in functions such as ABS, SQRT, etc.

(22) Subroutines are more flexible in what kind of output they can generate, but are less flexible about where they get their input data and where they write out their output than functions.

TABLE 1. Subroutines versus Function in VBA

	Subroutines	Functions
Input Data	get data from specific cells	get data from “relative cells”
Output Location	output is to specific cells	output is to the cell from where the function was “called”
Output Number	can output more than one value	can output only one value at a time
How to use	must be “run” using the Run Macro button	must be “called” like other Excel functions such as SQRT or ABS
Flexibility	More flexible about what kind of data they generate, less flexible about where data is placed	More flexible about where they are used and get their data, can only produce one value at a time.

Writing Functions

(23) Take a look at the VBA function on the next page based on the Newton-Raphson equation. By Friday, you should type this function into a module in a Excel spreadsheet named CALCROOTS.xls.

- (24) Start by opening up a new spreadsheet in Excel.
 Make sure that the VBA toolbar is available.
 Open the VBA Editor
 Insert a new Module
 Type the function into the module as it appears on the next page

Function roots(a, b, c, d, x)

' This function solves for the roots of a cubic equation
' using the Newton-Raphson method

```
xnplus1 = x - (a*x^3 + b*x^2 + c*x + d) / (3*a*x^2 + 2*b*x + c)
```

```
While Abs(xnplus1 - x) > 0.00001
```

```
    x = xnplus1
```

```
    xnplus1 = x - (a*x^3 + b*x^2 + c*x + d) / (3*a*x^2 + 2*b*x + c)
```

```
Wend
```

```
roots = xnplus1
```

```
End Function
```

(25) You do not need to save the function or the module. You don't even need to name the function. Instead, when you are finished working, you **save the spreadsheet**.

(26) Notice that a, b, c, and d and the estimate x are the **input arguments**.

(27) Also notice the **syntax** for writing the “do-while” loop. The loop starts with the word “While” and ends with the word “Wend” which means “while end”

(28) To use this function, you would first type the coefficients and the initial estimate x into cells in an Excel spreadsheet.

8 6 4 2 100

(29) You then would click on the next cell, and type the following into the function window in the toolbar at the top of the screen,..

= roots (A1, B1, C1, D1, E1)

where A1, B1, etc are the cells that contain the coefficients and the initial estimate x .and Then press Enter . The resulting value should be (but isn't always!) the root nearest to the estimate x.

(31) For example, given the coefficients -2000, -54, 100, and -2, using our root3.m function in Matlab, we found three real roots (0.1990, 0.0204, and -0.2464).

(32) To find these roots in Matlab we used the estimates 1, 0, and -1. The root nearest to 1 is 0.1990. The root nearest to 0.0204 is 0. And the root nearest to -1 is -0.2464.

(33) However, if you use these estimates for x the VBA function generates only the first and last roots (0.1990 and -0.2464). To generate the third root (0.0204) you must enter an estimate of x that is almost exactly equal to the root (in this case 0.02)

(34) And, if you enter an estimate of the root that is exactly equal to the root, the function returns the nonsense answer #VALUE. This problem points to some limitations in VBA.

(35) Furthermore, VBA cannot handle imaginary or complex numbers easily. But as we saw in Matlab, there are ways around that problem. Take a look at the following code.

```
Sub Quad_roots()
```

```
a = Sheet2.Cells(1, 1)
b = Sheet2.Cells(1, 2)
c = Sheet2.Cells(1, 3)
discriminant = (b ^ 2) - (4 * a * c)
If discriminant > 0 Then
    x1 = (-b + Sqr(discriminant)) / (2 * a)
    x2 = (-b - Sqr(discriminant)) / (2 * a)
    Sheet2.Cells(5, 2) = "x1 = "
    Sheet2.Cells(5, 3) = x1
    Sheet2.Cells(5, 4) = "x2 = "
    Sheet2.Cells(5, 5) = x2
Elseif discriminant = 0 Then
    x1 = -b / (2 * a)
    Sheet2.Cells(8, 2) = "x1 = "
    Sheet2.Cells(8, 3) = x1
    Sheet2.Cells(8, 4) = "x2 = "
    Sheet2.Cells(8, 5) = x1
Elseif discriminant < 0 Then
    real_part = -b / (2 * a)
    imag_part = Sqr(Abs(discriminant)) / (2 * a)
    Sheet2.Cells(11, 2) = "x1 and x2 = "
    Sheet2.Cells(11, 3) = real_part
    Sheet2.Cells(11, 4) = " +/- "
    Sheet2.Cells(11, 5) = imag_part
    Sheet2.Cells(11, 6) = "* i"
End If
End Sub
```

36) This subroutine uses the algorithm from the Matlab textbook for the quadratic equation and calculates real roots and the real numbers for complex roots without having to use imaginary numbers.

(37) By this I mean that if one of the roots is the complex number $10 + 9i$, the algorithm generates the 10 and the 9, and we just print out the “i”.

(38) Notice that in VBA, the proper syntax for If, ElseIf, End If, all use capital letters. One of the nice features of VBA however, is that it automatically capitalizes **reserved words**, such as If, End If, While, and Wend.

(39) Notice also how the subroutine specifies where it gets data and where it writes output by pointing to a specific sheets and cells in the workbook..

(40) That is why the parentheses next to the words Sub Quad_Roots () are empty. No input or output arguments!

(41) In the case of this subroutine that input and output data are in specific cells on Sheet2. To run the subroutine the input data must already exist on Sheet2.

(42) Click on the Sheet2 tab at the bottom of your Excel workbook and enter the values 2, 4, 6 in the first three cells of row 1.

(43) Open the VBA Editor and Insert a new Module. Then type in the subroutine above.

(44) Go back to Sheet2, and run the subroutine by clicking on the **triangle** icon. A dialog box will appear and allow you to choose which subroutine. Click on Quad_Roots.

(45) The roots of the equation are complex, so the output should appear on row 11.

(46) Test your subroutine with the following values.

(a) $x^2 + 5x + 6 = 0$

(b) $x^2 + 4x + 4 = 0$

(c) $2x^2 + 4x + 8 = 0$

(47) You will turn the output from this **subroutine** as part of Assignment Eight.