VBA Primer – Excel in Office 2013
R. Russell Rhinehart – June 2016 – www.r3eda.com

To Start:

- Open an Excel Worksheet (and save it).
- Set Macro Security Setting to Medium. Click on the Office Button in the upper left of the menu bar, Excel Options button in bottom of window, Trust Center in left hand menu, Trust Center Settings in middle right of window, Choose your security preference ("disable with notification", probably). Click OK.
- Press ALT-F11 to open the VBA editor. Alternately, use the "Launch VBA" icon on the Developer Toolbar. [To install the Developer Toolbar, click on the "Office Button" (upper left), click on "Excel Options" (lower border of new window), in the "Popular" set, check the "Show Developer Tab in the Ribbon" box. Close the "Office Button" window, and you should see the "Developer" tab on the upper ribbon.]
- In the new open window "VBA Editor" you should see three windows (Project, Properties, and Immediate) and a grayed space for another window. If the "Project" window does not appear, use the "View" menu item to add "Project Explorer"
- Right-click in the Project window, then click on "Insert", then "Module". This will open a window in the grayed space for writing VBA code. Alternately, you can double-click on the spread sheet name in the Project window to open a code window. Either way allows you to write VBA code. However, code in a code window attached to a spread sheet (for instance "Sheet1"), can only interface with that sheet.
- There are two categories of code – subroutine and function – with attributes similar to that of any language. To create a subroutine, type "SUB", spacebar, subname, and "()". For example "Sub Practice()". Then hit enter, and "End Sub" will appear. Your code goes in between these two lines. To create a function, type "FUNCTION", spacebar, functionname, "(", argument list, and ")". For example "Function F_Multiply(x,y)". Then hit enter, and "End Function" will appear. Your code goes in between these two lines. Subs do not need an argument list, but can take one. There can be no spaces in the name of either the subroutine or the function, but you can use the underscore. Such as "Sub Help_Me()".

General:

- There are no rules about the use of columns (unlike Fortran columns 1-6, 7, and 8-72).
- To continue a long statement on the next line, break it with a blank_underscore at the end of the line. " _" acts like a hyphen in English.
- Names for variables, functions, and subroutines cannot include spaces or characters, and must start with a letter. However, you can use the underscore to join two_words into a single name.
- Some combinations of letters are reserved for functions or values. These include log, sin, sqr, rand, name, TRUE, single, etc. You cannot use these for variable names.
- The assignment symbol is simply "=", not ":=".
- There is no end-of-line symbol, like ";".
- Unless the user declares "Option Explicit", variables types do not need to be explicitly declared.
- Comments are indicated by the single quote mark, " ' ", and whether starting a new line or following code on the same line, anything on the line after the " ' " mark is ignored. The term "Rem", short for "Remark", does the same.
- Run a program by pressing the F5 key, or by clicking on the "Run arrow", which looks like a sideways triangle, on the VBA toolbar.

- If an Excel cell is in the edit mode, you cannot run a VBA program. Press the "Esc" key or click on another cell to close the cell in edit mode.

I/O to Excel Cells:

- Cells are addressed by the row and column number, as Cells(4,2) representing the B4 cell. The operator name is "Cells", not "Cell". For your convenience, switch the Excel display from the default "A1" notation (column-as-a-letter row-as-a-number) to "R1C1" notation (row-as-a-number column-as-a-number). To do this, open an Excel workbook, click on the "Office Button" then "Excel Options" at the bottom of the window, then "Formulas" in the left hand meny, then check the R1C1 notation box in the "Working with Formulas" group of items. OK-out of the sequence.
- Like an array, "Cells" is the name (not "Cell") and the row and column numbers are the indices. The VBA assignment statement "A = Cells(5,6)" reads the value in cell "F5" (R5C6) and assigns it to the variable "A". "Cells(i,j) = q" assigns the value of "q" for display in the cell in the i^{th} row and j^{th} column.

Variable Types and Declarations:

- Variables which store text are called "string" variables. Those storing integer values are called "integers". Double precision integers are called "long". Variables storing single precision real values are called "single". And double precision reals are called "double". They do not have to be explicitly declared, but it is good practice. They are declared using a "dimension" statement, "Dim" for short, which sets up storage space as follows:
  - Dim Name As String ‘sets a location called "Name" for text
  - Dim I As Integer
  - Dim J As Long
  - Dim a As Single
  - Dim b As Double
- Dimensions of vectors and arrays have to be declared, but the variable type does not. Examples include:
  - Dim List_One(25) as String ‘declares both number and type
  - Dim Table(10, 10) ‘only declares the array size
  - Dim Intensity(20, 20, 20)
  - Dim Population(100, 3) As Double
- Assignment statements for String Variables require double quotes. The others are conventional. Examples include:
  - List_One(k) = "Paul Taylor"
  - Table(I , J) = I * J
- It is good practice to organize all declarations in the header of the program, above all subroutine and function declarations, but VBA does not require this structure. If you declare variables within each sub-program, they are locally used, but variable values are forgotten when exiting the sub-program. Declaration in the header space above all sub-programs makes the variable commonly available, with its value retained when control is switched from one sub-program to another.
- The statement "Option Explicit" requires every variable to be declared with a "Dim" statement. This is good for subsequent users, if you include the definition and units as a comment line. It is also good to be sure that you did not type JELLO for JELL0, or line for 1ine, or dimond2 for diamond2.

Operations:

- Precedence of mathematical operators is the same as Fortran. Parenthesis first, in-to-out, then left-to-right. The parenthesis symbols are "(" and ")" whether nested or not. Exponentiation second, in-to-out then left-to-right. The VBA symbol for exponentiation is "^", in Fortran it is "**". Followed by multiplication and division, left-to-right. Then addition and subtraction left-to-right. These symbols are "*", "/", "+" and "-".
- Functions have parenthesis, and return the value that results from operating on their argument value. For example:
    - A = fun_fun(b)
- VBA can concatenate. The operator symbol is " & " (space-&-space). For example the statements

```
Name2 = "Rhinehart"
Name1 = "Russ"
Cells( 5, 9 ) = Name2  &  ",  "  &  Name1
N = 3
Cells( 4, N ) = "R^"  &  N
```

would write "Rhinehart, Russ" in cell I5, and "R^3" in cell C4.

Loops:

- The loop type closest in function to the Fortran Do loop is the FOR-NEXT loop. The loop is started with a FOR statement and an initialization of the loop index, an extreme limit, and an optional step increment. The loop ends with a NEXT statement, at which place the loop index is incremented and tested whether it is beyond the extreme to either exit the loop or re-enter. Here is an example of a loop that counts by 2s.

```
For k = 1 to 20 Step 2
        Cells(k, 1) = k
Next k
```

If the "Step" is not defined the default is "Step 1". "Step -5" would decrement.

Conditionals:

- The conditional statement closest in function to the Fortran IF is the IF-THEN-ELSE conditional. It starts with "IF" followed by the condition and the word "THEN". If the condition is true, code following the THEN is executed, other wise not. The <, =, and > keyboard symbols are used for the comparison. Here are two examples, one with, and one without an "ELSE"

```
IF name < "C" THEN Cells (17, I) = name
IF Abs(x) > = Abs(y) THEN
        Cells(k, 1) = "Great! x is at least as large as y"
ELSE
        Cells(k, 1) = "Sorry, x had a smaller magnitude than y."
END IF
```

- Compound antecedent tests require each comparison to be explicitly stated, and joined with either the "AND" or "OR" or other conjunction operator. For example, to test if the value of x is between "a" and "b", as, "Is a<x<b?" you would write:

```
IF  a < x   AND  x < b  THEN …
```

- DO WHILE or DO UNTIL. These are preferred over the GOTO statement. The end of the WHILE or UNTIL range is the statement LOOP. The form is:

```
Constraint = "INDETERMINED"
DO UNTIL Constraint = "PASS"
        x = Rnd()                    'Assigns a random value to variable x
        Constraint = Constraint_Test(x)  'Calls function
LOOP
```

Debugging:

- If VBA detects an error during the compile stage (actually an interpreter but it does look at the entire code for syntax errors such as FOR-without-NEXT, or a GOTO-Label-not-defined) (whether pre-execution or during your edit stage), or if it encounters an impossible operation during the execution stage, it will make a "BONK" sound (so that all of your friends can hear) and open a error message window. If you click "debug" it will highlight the problem line in yellow. This is termed being in "Break Mode". If the break mode occurs during execution, you can place the cursor on any variable in the VBA editor window and it will display its value. This is a great convenience for debugging.
- Click on the square "Reset" button on the toolbar to exit "Break Mode". You must do this to be able to run.
- If you want execution to stop at a particular line so that you can mouse-over variables to see their values, then click in the slightly gray column just to the left of the programming text area. It will create a red dot. The program will stop, in break mode, when it gets to that line.
- Alternately, you can step through the program during execution, line-by-line, by pressing the F8 key. VBA will highlight each line that is about to be executed. In "Break mode" you can observe variable values from formerly executed lines by a mouse-over (placing the cursor over) the variable symbol. Use the Escape key or the reset button to exit this "Step Into" mode.
- Instead of printing intermediate values to the Excel cells for display, you can add a watch window to the VBA editor. Use the "Debug" drop down menu, click on "Add Watch" and follow the directions.

Run Buttons (Commands):

- You can place a button on the Excel spreadsheet to run a subroutine. In Excel, open the "Developer" toolbar, and click on the "Insert" icon in the "Controls" category. Then click on the "Button" icon in the upper left. Move the cursor to the location on the spreadsheet where you want the button (the cursor will have a + shape) and left-click. It will open a window that provides a list of subroutines, and create a button in the grayed-boundary edit mode. Choose the sub that you want the button to start, and edit the button name. If the button is not in the active edit mode, right-click the button.

Objects and Properties:

- The Excel Workbook is an object. Objects contain objects, and objects have properties (attributes). For example consider "The red car has a flat tire". Tire is an object of the car object. The car object has a property (color) and a property value (red). The tire object has a property (air pressure) and a value (0 psig). The Workbook object contains worksheet objects, which

contain cell objects, which might contain a number. The number object has a numerical value, but it also has other properties such as font type, font size, and font color.

- You can use VBA commands to do any formatting thing that you can do using the Excel toolbar items – change font, create cell borders, set display characteristics, set font and background colors, set cell dimensions, etc. Use the VBA help menu for details.

Keystroke Macros

- Often it is easier to perform an operation such as sort, plot, clear contents, change font and color, with keystroke/mouse operations in an Excel Worksheet than in VBA. You can record the keystroke/mouse sequence in VBA, and then call it from VBA programs.
- Click on the "Developer" tool tab on the Excel ribbon; then click on "Record Macro". This opens a window. Fill in the name you want to assign to the keystroke sequence, and location to write it. I usually accept the default name and location. Perform your keystroke/mouse operations, then click on "Stop Recording" where you found the "Record Macro". The VBA code that the keystroke/mouse sequence represents will be in a new module in the VBA project window.
- Your VBA code can call that Macro subroutine.
- You can read the Macro code to see what the VBA instructions are.
- You can copy/paste that code, or appropriate sequences from it, into your VBA program.

External File I/O

- To open a file for input or output, state "Open" the file path including name, the purpose, then the number you choose for the file. If the path is not explicitly defined the default is the directory of the open Excel Workbook. The numbers are your choice. For example:

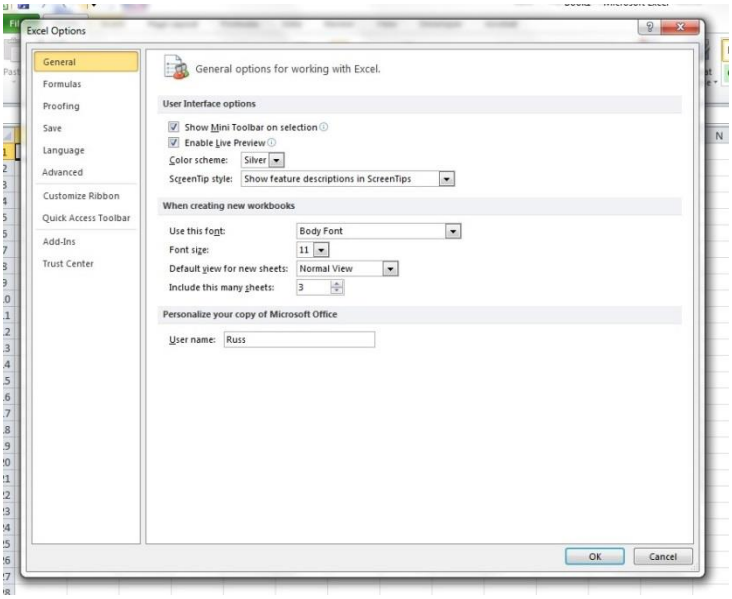
```
OPEN "C:/My Documents/VBA Programs/filename.txt" FOR OUTPUT AS #4
OPEN "source data.txt" FOR INPUT AS #1
```

- You should close a file, when through, with the CLOSE #n, statement, for example CLOSE #7.
- INPUT and PRINT read and write from and to txt files. Specify the file number followed by a comma, then the variable list. When the Input or Print list is complete, the next read or write "call" starts at the beginning of the next line in the file. Examples are:

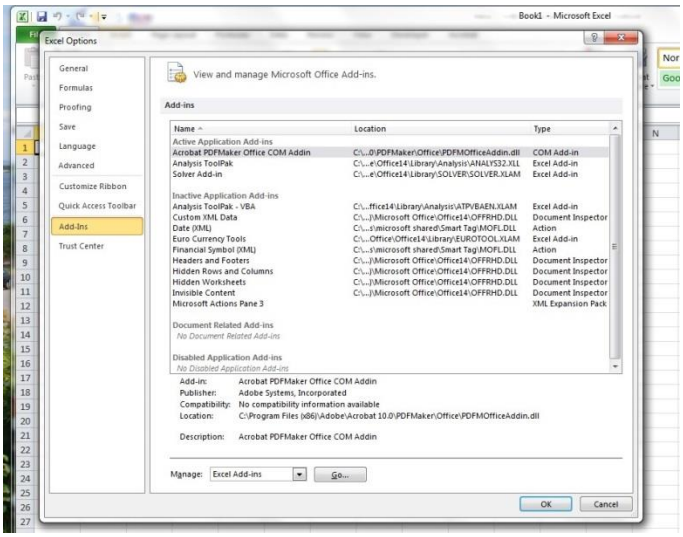
```
INPUT #8, a, b, c(2)
PRINT #2, Name2 & ",  " & Name1, age
```

Solver Add-In

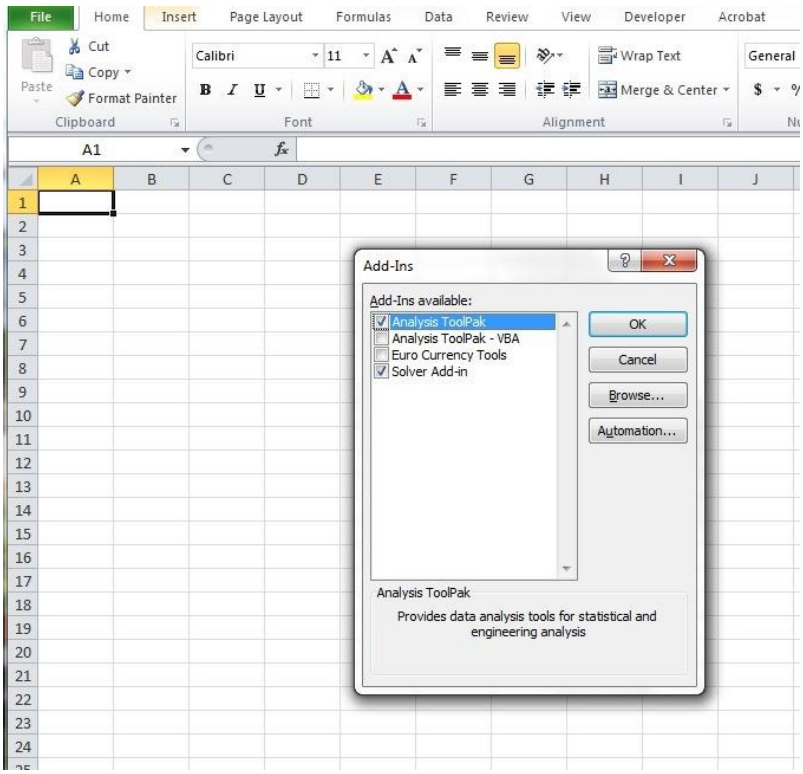
Solver is an optimization add-in to Excel. I'm currently running Office 2013, but it is very similar in earlier versions. To install Solver, open Excel, click on file, then options. This window should open:



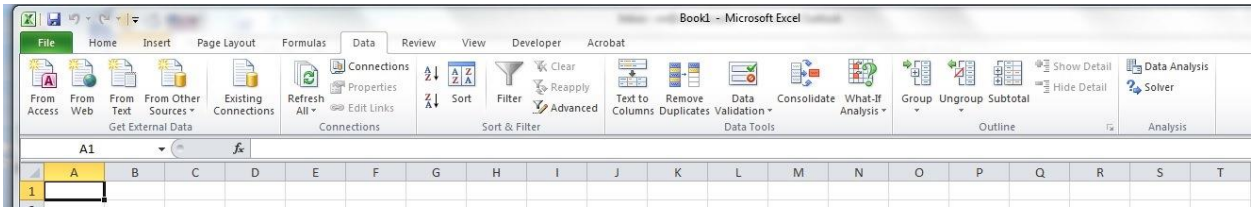
Click on Add-Ins in the left column and the view changes to this:



In the lower section, there is a “Manage” window, choose the “Excel Add-Ins” and click on the “Go” button. This opens a selection list.



Click on the Solver box to check it, then click "OK". Now Solver should appear on the far right of the Excel Data menu.



To use Solver, click on the menu icon. Then select the cell to be optimized (choose objective), choose min or max or value of, and choose the decision variable cells with the values that are adjusted to optimize the objective.

Solver has several optimization algorithms. In 2013 I prefer GRG Nonlinear. If you click on options you can change the convergence criterion and choice of forward or central difference derivatives. GRG stands for "Generalized Reduced Gradient". I think it is a Levenberg-Marquardt approach that, when it encounters a DV constraint violation, reduces the number of DVs by one. I think it uses L-M on the reduced number of DVs and calculates the other DV to remain on the constraint. I think if it finds it is free of the constraint, it returns to the original L-M with all DVs. I suspect all of this because it is a conventional approach, and because I am guessing at what Excel does. I can't get answers from MS.

Calling Solver from VBA

To Call Solver in VBA:

1. Make sure the Solver add-in is installed to VBA. In VBA editor, go to Tools, then References Menu, and add solver as a reference to the project. See Solver help in VBA editor for more details.

2. Record a Solver Macro from a keyboard implementation of Solver.

3. Modify the arguments for Solver

```
‘       A typical call in the recorded macro will appear as:
SolverOk SetCell:="$J$" & nI & "", MaxMinVal:=3, ValueOf:=0, _
ByChange:="$D$" & nI & "", Engine:=1, EngineDesc:="GRG Nonlinear"

‘       The following two lines are needed to close Solver and to save results.
‘       You need to add these two lines.
SolverSolve UserFinish:=True
SolverFinish KeepFinal:=1
```