Conclusion of VBA Review

Larry Caretto
Mechanical Engineering 309

Numerical Analysis of Engineering Systems

January 29, 2014

Outline

- Assessment quiz results
- Review
  - User-defined functions (UDF)
  - Choice (If) statements
  - Conditional and count-controlled looping
- Arrays
- Strings
- Debugging and Help
- First programming exercise

Assessment Quiz Results

- 23 Responses (Non-responses not shown)
- General computer skills:
  - 0 None
  - 0 Minimal
  - 10 OK
  - 0 Good
  - 4 Excellent
- Word processing skills:
  - 0 None
  - 0 Minimal
  - 8 OK
  - 14 Good
  - 1 Excellent
- Spreadsheet skills:
  - 0 None
  - 0 Minimal
  - 11 OK
  - 12 Good
  - 3 Excellent
- Programming language(s):
  - VBA(11), C/C++(5), Java(3), HTML(1), MATLAB(2), Python(1), EES(1) No Response

Assessment Quiz Results II

- Programming skills:
  - 1 None
  - 6 Minimal
  - 8 OK
  - 5 Good
  - 0 Excellent
- Highest math course:
  - Math 150B(4)
  - Math 250(7)
  - Math 280(12)
- Code output?
  - j = 0: For i = 0 To 5 'missing j = 0
  - If i = j Then Print #1 2 * i else j=j+2
  - Next i
  - Answer: 0, 4, 8
  - None correct with j = 0 missing

Assessment Quiz Results III

- Solution of equations:
  - 4x₁ + 2x₂ = 2, 3x₁ = 3
  - x₂ = 3/3 = 1; x₁ = [2 – 2(1)]/4 = 0
  - 19 Correct, 2 Partially correct, 2 incorrect
- Describe what the following code does
  - sum = 0.0 : For i = 1 to n
  - sum = sum + x(i) : Next i
  - result = sum / n
  - Computes average of the first n elements of the x array
  - 5 Correct, 11 Partially Correct, 7 incorrect

User-Defined Functions (UDF)

- The function has the following form
  - Function <name>(<arguments>) As <type>
  - <code to do computations>
  - <name> = <value from computations>
  - End Function
- Example
  - Function vCyl(R as Double, H as Double) As Double
  - vCyl = 4 * atn(1) * R^2 * H
  - End Function

ME 309 – Numerical Analysis of Engineering Systems
Using Your UDF

• Use with cell references or range names in worksheet
  =vCyl( B1, B2)
  =vCyl( 1, 20)
  =vCyl( radiusName, heightName)

• Call from other VBA procedures
  V = vCyl( radius, height)
  cylVol = vCyl( 1, 20)
  v10cyls = 10 * vCyl( rad, hgt)

Count Controlled Loop

For <counter> = <start> to <end>
  <statements>
Next <counter>
For <counter> = <start> to <end> _
  _Step <increment>
  <statements>
Next <counter>
Statements in loop repeated nTimes = (<end> - <start>) /<increment> + 1 (converted to integer)

Loop not executed if nTimes <= 0

Conditional Loop

Do <stmts> _ If <cond> _ Then Exit Do
Do While <cond> _ <stmts> _ Loop
Do Until <cond> _ <stmts> _ Loop
Do _ <stmts> _ Loop While _ <cond>
Do _ <stmts> _ Loop Until _ <cond>
Note tests before or after loop

Solutions to Looping Exercises

• Looping exercise given at end of class for ungraded homework
  – Two functions use Taylor sine series
    • mySine uses fixed number of terms
    • mySine2 uses fixed allowable relative error
  – Modify functions to allow user inputs for
    – Total number of terms in mySine
    – Relative error and maximum iterations in mySine2
  – Download solution from web site home page

Looping Solutions
mySine Code Revisions

Function mySine(x As Double) As Double
For k = 1 To 4
  Two original statements above modified as shown below
Function mySine(x As Double, nTerms As Long) As Double
For k = 1 To nTerms - 1

mySine2 Code Revisions

Function mySine2(x As Double, allowedError As Double, maximumTerms As Long) As Variant
Converged = Abs(term) <= allowedError * Abs(mySine2)
Loop Until Converged Or k > maximumTerms

Arrays

- Arrays can be visualized as data on an experimental variable
  - Could describe pressure data points mathematically as P₁, P₂, etc.
  - In VBA we can represent these data points as P(1), P(2), etc.
  - We call the numbers (1, 2, etc.) indices or subscripts
- We can use constants or variables for the subscripts: P(4), P(k), where k has a value

Two-dimensional Arrays

Consider an experiment where you vary the current over six levels, the voltage over four levels and measure the efficiency, e, of an electromechanical device. The data for each combination of current and voltage can be represented as shown below

<table>
<thead>
<tr>
<th>I(1)</th>
<th>I(2)</th>
<th>I(3)</th>
<th>I(4)</th>
<th>I(5)</th>
<th>I(6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e(1,1)</td>
<td>e(1,2)</td>
<td>e(1,3)</td>
<td>e(1,4)</td>
<td>e(1,5)</td>
<td>e(1,6)</td>
</tr>
<tr>
<td>e(2,1)</td>
<td>e(2,2)</td>
<td>e(2,3)</td>
<td>e(2,4)</td>
<td>e(2,5)</td>
<td>e(2,6)</td>
</tr>
<tr>
<td>e(3,1)</td>
<td>e(3,2)</td>
<td>e(3,3)</td>
<td>e(3,4)</td>
<td>e(3,5)</td>
<td>e(3,6)</td>
</tr>
<tr>
<td>e(4,1)</td>
<td>e(4,2)</td>
<td>e(4,3)</td>
<td>e(4,4)</td>
<td>e(4,5)</td>
<td>e(4,6)</td>
</tr>
</tbody>
</table>

Declaring Arrays

- Arrays must be declared as arrays by specifying the size of the array
  - The maximum size of the array must be specified in the Dim statement
- In VBA the lowest array subscript is zero by default
  - Can use Option Base 1 in declarations section to change default lowest subscript to one
- Can also set lowest subscript on each array

Dimensioning Arrays

- Declare **maximum array subscript**
  Dim I(1 to 6) as double
  Dim V(1 to 4) as double
  Dim e(1 to 4, 1 to 6) as double
- Size below depends on Option Base 0/1
  Dim I(6) as double  How many elements are in these arrays?
  Dim V(4) as double
  Dim e(4, 6) as double
  Option base 0: 7, 5, 35
  Option base 1: 6, 4, 24
Using Arrays

- Arrays components are referenced by their subscripts
- This is often done in a For loop
  
  \[ PI = 4 \times \text{atan}(1) \]
  
  For \( k = 0 \) to 100
  
  \[ x(k) = \sin(k \times PI / 100) \]
  
  Next k

- \( x \) is an array with 101 components giving \( \sin(x) \) for \( 0 \leq x \leq \pi \), with \( \Delta x = \pi/100 \)

Two-Dimensional Arrays

- Use nested for loops
  - Use example of current and voltages
  
  For \( k = 1 \) to 4
    
    For \( j = 1 \) to 6
      
      Power\((k,j) = I(j) \times V(k)\)
      
      Next \( j \)
    
    Next \( k \)

  Recall table:
  
  V was in rows
  
  I was in columns
  
  Power\((k,j)\) is Power(row, column)
  
  Are \( k \) and \( j \) indices correct?

Dynamic Arrays

- What if you do not know array size until program is actually running?
- Use Dim \( a() \) to tell compiler that \( a \) is an array then use ReDim with actual dimensions

Sub getArray( \( N \) as long) as Variant

Dim \( x() \) as Double : ReDim \( x(1 \) to \( N) \)

- Can go from Dim \( a() \) as Double to any size ReDim: ReDim \( a(1 \) to \( 10, 6 \) to \( 12) \)

Passing Arrays to Procedures

- Declare array in argument list with parentheses to indicate array

Sub mine( \( A() \) as double)

  'No dim statement for \( A \)
  
  A(2,3) =

- Calling program sets actual dimensions on array and uses only the following

  Dim \( B(1 \) to \( 10, 1 \) to \( 6) \) as double
  
  Call mine\((B)\)

Determining Array Bounds

- The UBound and Lbound functions determine the upper and lower bounds of unknown array dimensions
- For a two-dimensional array, \( A(m,k) \)
  
  - LBound\((A,1)\) is the lower bound of \( m \)
  
  - UBound\((A,1)\) is the upper bound of \( m \)
  
  - LBound\((A,2)\) is the lower bound of \( k \)
  
  - UBound\((A,2)\) is the upper bound of \( k \)

Worksheet Arrays to VBA

- Passed as a range of cells
- First step is to set a type variant variable equal to the input range variable
  
  - The variant variable is now a two-dimensional array
  
  - May have single row or single column, but is still a two-dimensional array
  
  - Lower bound is always one for arrays from worksheet
  
  - Can use UBound to get sizes
Worksheet Array Example

Function `getMean (Ain As Range) As Double`
Dim `A` as Variant, `m` as Long, `k` as Long
Dim `sum` as double, `cells` as Long
Dim `nRows` as Long, `nCols` as Long

`A = Ain`: `nRows = UBound(A, 1)`: `sum = 0`
`nCols = UBound(A, 2)`: `cells = nRows * nCols`
For `k = 1` to `nRows`
    For `m = 1` to `nCols`
        `sum = sum + A(k,m)`
    Next `m`
Next `k`

`getMean = sum/cells`: End Function

VBA Array to Worksheet

- VBA steps to return array to worksheet
  - Declare the function type as Variant
  - In the function or sub declare a working array for calculations
    - Use `application.caller` for dimensions
  - Write the code for values in working array
  - At end of function set `<function name> = <working array name>
- To use the function: select cells; enter function in formula bar; `Ctrl+Shift+Enter`

Function `array2wks(<arguments>) As Variant`
Dim `userRows` As Long
Dim `userColumns` As Long
Dim `workArray`() as Double

'Statements below determine rows and columns
`userRows = Application.Caller.Rows.Count`
`userColumns = Application.Caller.Columns.Count`
ReDim `workArray(1 to userRows, 1 to userColumns)`

'Place code here to compute all components of `workArray`
`array2wks = workArray`
End Function

Passing by Reference/Value

- Consider the following function call
  - Call `mySub(a, b)`
  - `Sub mySub(x as Long, y as Long)`
    - `x = 2 * x` **Answer:** The value of `a` will have the new value of `x` computed in `mySub`
    - `y = x / y`
  - End Sub
- What happens to the value of `a` in the calling program because of the `x = 2 * x`?

Passing by Reference/Value 2

- By default VBA passes memory locations of variables to procedures
  - This is known as pass by reference
- Alternative is pass by value
  - This simply sends the procedure the value stored in the memory location
  - To use pass by value enter the keyword `ByVal` before the variable in the header
  - `Sub mySub(Bval x as Long, y as Long)`

Strings

- Consider only variable length strings
- Use `Dim str as String`
- Use `& or +` as concatenation operator to join two strings
- `Len(str)` gives length of string
- `Left`, `Right`, and `Mid` give substrings in same manner as worksheet functions
- `InStr` function searches for substrings
Debugging

• Debugger allows you to step through a program and see intermediate values
  – Useful to find location of errors
• Items to use in debugger
  – Breakpoints stop execution at certain points
  – Step-by-step execution
  – Intermediate and Watch windows
  – Hover mouse over variable to get its value
  – Change statement to be executed next

Debugging Toolbar Icons

• Windows show values of variables during program execution
• Step commands allow you to step through individual statements

Help

• Help systems for Excel and VBA
• Search function does not always return what you are looking for
• If you know the keyword, type it, place the cursor in the keyword, and press F1
• Sometimes a Google search for “Excel VBA <subjectYouAreInterestedIn>” works better than Excel/VBA help

First Program

• Six different approaches to calculation of a trajectory, x(t), y(t) from 0 to t_max
  – Conventional cell formulas
  – Individual UDFs with “A1” cell references
  – Cell formulas with names
  – Individual UDFs with names
  – Array function
  – Macro
• Due Monday, February 3, 11:59 pm

Range Names

• Can assign meaningful names to cells
• Names are used in formulas and show in name box when cell is selected
• Example below shows change with names

Worksheet Using Names

Worksheet Without Names

Default cell names are absolute

Getting Range Names

• Select cells with names adjacent to cells to be named
• Choose Create from Selection in Defined Names group of Formula tab
• In resulting dialog box make sure name location is correct and click OK
Name Box
- The name box, to the left of the formula bar, shows the currently selected cell
- When range names are defined, this box becomes a pulldown menu that shows all the defined names
- Clicking on a name in this menu takes you to the named cell
- When entering formula, clicking on a named cell enters name into formula

Name Manager
- Main tool for managing range names
  - Choose Name Manager in Defined Names group of Formula tab
  - Create, delete or edit names in Manager
- By default names have scope of entire workbook
- Name manager allows you to make name scope one worksheet
  - More instructions on first exercise

Names to Existing Formulas
- Click the down arrow next to Define Names and select Apply Names from the resulting submenu
- Select names in Apply names dialog and click OK
  - Can leave settings for checkboxes at bottom of dialog as set by Excel

Array Formula
- Array expressions: formulas that are entered into multiple cells at the same time
- Array functions: functions whose values are entered into multiple cells as arrays
- When entering an array formula or using an array function you must press Control+Shift+Enter instead of Enter
- You are asked to modify VBA code to use an array formula for the trajectory

Assignment One Code Structure
- Array function to return trajectory
  Function trajectory(v0 As Double, _
  theta As Double) As Variant
- Macro to return trajectory
  Sub getTrajectory()
- Common calculation function for both procedures above
  Function calculateTrajectory(v0 As Double, _
  theta As Double, nRows As Long) As Variant
  ‘Comments not shown
  Dim results() As Variant
  ReDim results(1 To nRows, 1 To 3)
  ‘(1) Use “Dim” statements to declare variables: time, dTime and k
  results(1, 1) = "Time (s)"  ‘Comment ...
  results(1, 2) = "x (m)"
  results(1, 3) = "y (m)"
  You have to edit last function
What is nRows?

- nRows is the total number of rows including the header row
  - How is nRows related to the Number of time intervals?
  - Previously found time step as maximum time divided by Number of time intervals

Modify Calculate Trajectory II

'(2) Compute time step, dTime, from maxTime function and number of rows, nRows. How is nRows related to number of time steps (NTS)?

\[ dTime = \frac{\text{maxTime}}{\text{nRows}} \]

'Hint: On worksheet you found \( \Delta t = \frac{t_{\text{max}}}{\text{NTS}} \).

'(3a) Complete coding of loop. Set lower and upper limits for k, the loop index

For \( k = \) <What are my limits?>

\[ \text{time} = (k - 2) \times dTime \]

\[ \text{results}(k, 1) = \text{time} \]

Modify Calculate Trajectory III

results(k, 1) = time

(3b) Put statements here to compute x and y as additional columns in the results array. Use UDFs for x and y. \( \text{results}(k, m) \) is an array that has results for time = \((k-2)\)\( \times \)dtime in the first column and the corresponding x and y values for that time in columns 2 and 3

Next k

calculateTrajectory = results

End Function

Suggestion: Make good guess about dTime equation and loop limits and check results