Advanced Spreadsheet Use to Improve Engineering Education

John H. Ristroph, Ph.D., P.E.
Professor Emeritus, University of Louisiana at Lafayette

Abstract

This paper illustrates methods that can be used across disciplines by applying Excel to engineering economics. It first discusses intrinsic functions, and then it shows how to use VBA to create custom functions that use notation familiar to a student. Next it covers how to produce diagrams and graphics via the drawing toolbar and custom cut-and-paste libraries, as well as how to show all formulas and logic rather than just numeric results. Then it describes hiding and protecting answers, and it addresses common notational issues such as subscripts, superscripts, and Greek symbols. Finally, it opens the door to further development by explaining objects and user forms.

Introduction

Students are encouraged to use spreadsheets via numerous examples in textbooks, but what about professors? Do they use spreadsheets to solve routine problems, such as those given for homework, and in turn encourage students to do so? Probably not. One obstacle is the differences between spreadsheet notation and that of textbook or manual solutions. Similarly, drawing and labeling diagrams can be time consuming. Another major concern is that of security: Copying and pasting computer solutions is even easier than the time honored tradition of manual cheating.

This paper examines the foregoing issues by first exploring simple solutions and then progressing to more advanced methods. The presentation is based on Microsoft's widely used Excel software [1], but other systems can be used as long as they support user-written routines known as macros to extend their capabilities. For example, OpenOffice.org's free Calc [2] system has most of the capabilities of Excel, but it uses a different dialect of Basic for its macro language. EIOffice [3] is an integrated office system with a spreadsheet very similar to Excel that uses Java to program its macros. Most engineers are familiar with engineering economics, so examples are given for that field. The approach, however, is applicable to other engineering fields, and several references [4-14] allow extending the material presented below, including offering a numerical methods course using Excel and VBA [15].

Factors and Functional Notation

Spreadsheets have a wealth of built-in or intrinsic functions. Excel's financial functions are computationally similar to factors used in textbooks, but notational differences sometimes slow learning. The Visual Basic for Applications (VBA) component of Excel allows custom or user-written functions that correspond clearly to factors. For example, Table 1 shows how easily the challenging (P|A,g,i,n) factor is implemented as an electronic factor or Efactor.

Use the Tools|Macro|Security menu to allow macros by setting security at a medium or low level. Then click Tools|Macro| Visual Basic Editor (or key Alt + F1 1) and click the editor's Insert|Module menu shown in Figure 1 to create a module in which to store functions and subrou-

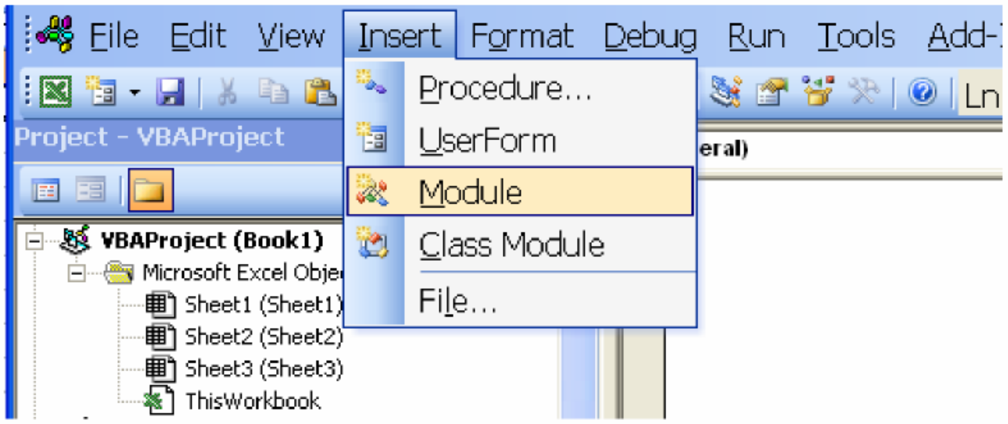


Figure 1. Creating a Module

Table 1. PAG(i, n) EFactor

```
Function PAG(GRate As Double, IRate As Double, n As Integer) as Double
  If n <= 0 Then
   MsgBox "Last parameter must be positive."
   Exit Function
  End If
  If GRate = IRate Then
   PAG = n / (1 + IRate)
  Else
   PAG = (1 - ((1 + GRate) / (1 + IRate)) ^ n) / (IRate - GRate)
  End If
End Function
```

tines. By default the module is named Module1, but that can be changed by editing its property labeled (Name) shown in Figure 2. Once the module is created, type in the code, such as that given in Table 1. Then return to the spreadsheet interface by pressing Alt + F 11 again. A custom function is used just like Excel's intrinsic functions such as Sum, so an entry into a cell such as that shown at the bottom-right of Figure 3,

= 100*PAG(5%, 8%, 5-0) , evaluates

the desired equivalent.

Figure 3 shows the results of applying this technique to each commonly used factor and also how each computation would be performed using intrinsic functions. The similarity of factors and Efactors allows students to focus on learning the material as presented in a textbook and worry about notation later.

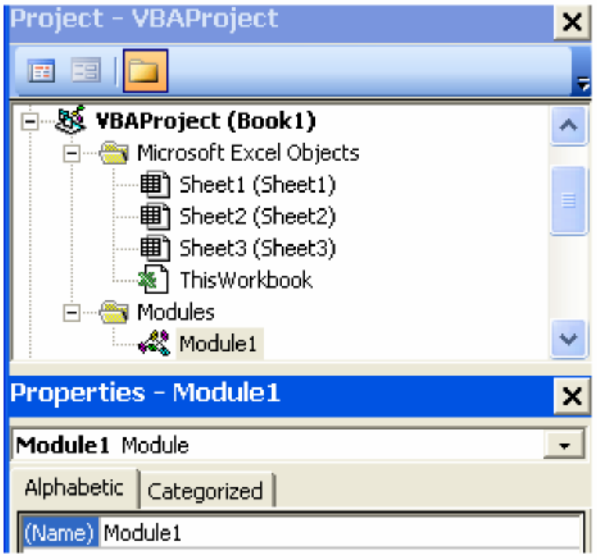
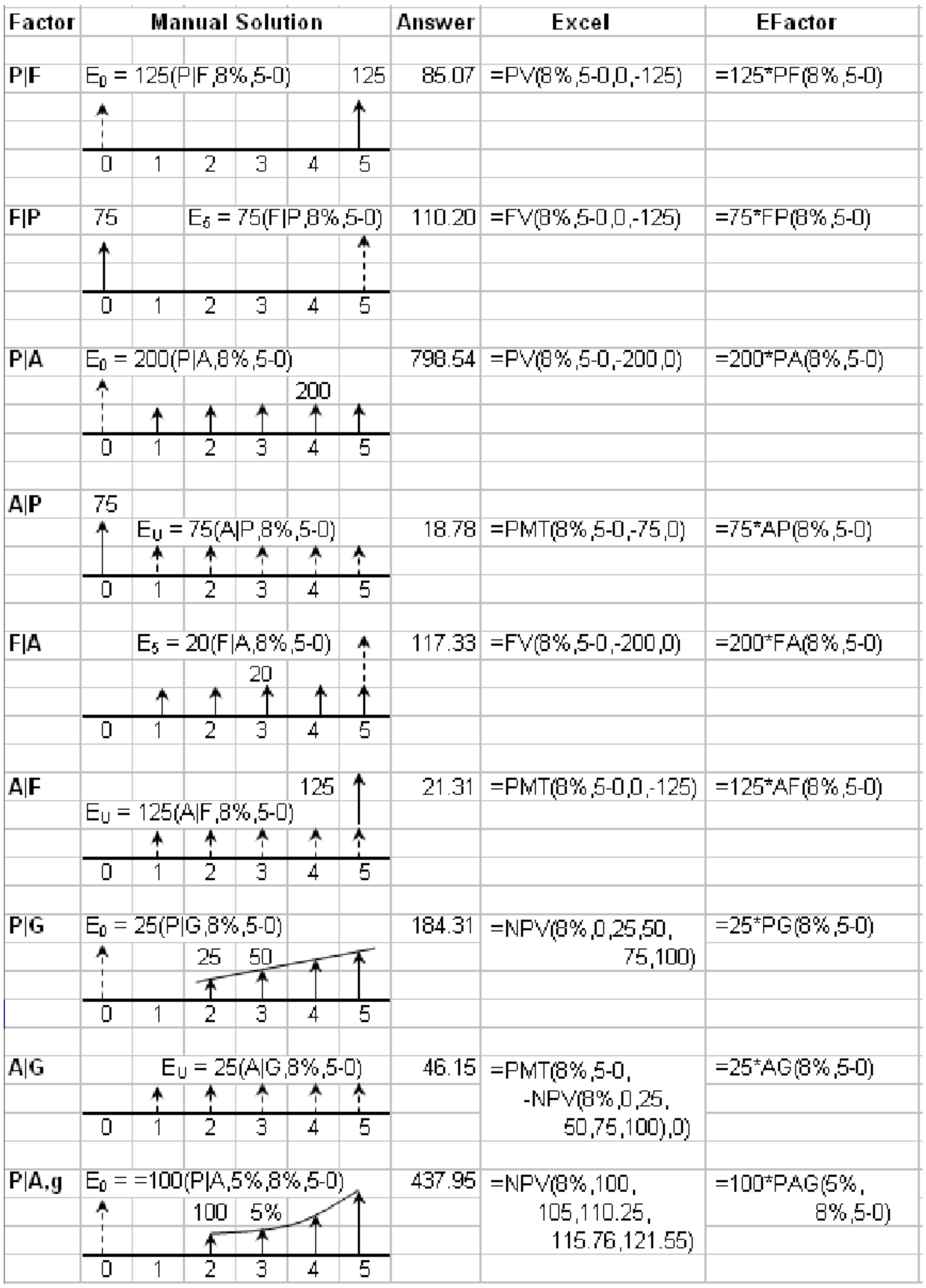


Figure 2. Naming a Module



Factor	Manual Solution	Answer	Excel	EFactor
P|F	E_0 = 125(P|F,8%,5-0) 125 0 1 2 3 4 5	85.07	=PV(8%,5-0,0,-125)	=125*PF(8%,5-0)
F|P	75 E_5 = 75(F|P,8%,5-0) 0 1 2 3 4 5	110.20	=FV(8%,5-0,0,-125)	=75*FP(8%,5-0)
P|A	E_0 = 200(P|A,8%,5-0) 200 0 1 2 3 4 5	798.54	=PV(8%,5-0,-200,0)	=200*PA(8%,5-0)
A|P	75 E_U = 75(A|P,8%,5-0) 0 1 2 3 4 5	18.78	=PMT(8%,5-0,-75,0)	=75*AP(8%,5-0)
F|A	E_5 = 20(F|A,8%,5-0) 20 0 1 2 3 4 5	117.33	=FV(8%,5-0,-200,0)	=200*FA(8%,5-0)
A|F	125 E_U = 125(A|F,8%,5-0) 0 1 2 3 4 5	21.31	=PMT(8%,5-0,0,-125)	=125*AF(8%,5-0)
P|G	E_0 = 25(P|G,8%,5-0) 25 50 0 1 2 3 4 5	184.31	=NPV(8%,0,25,50, 75,100)	=25*PG(8%,5-0)
A|G	E_U = 25(A|G,8%,5-0) 0 1 2 3 4 5	46.15	=PMT(8%,5-0, -NPV(8%,0,25, 50,75,100),0)	=25*AG(8%,5-0)
P|A,g	E_0 = =100(P|A,5%,8%,5-0) 100 5% 0 1 2 3 4 5	437.95	=NPV(8%,100, 105,110.25, 115.76,121.55)	=100*PAG(5%, 8%,5-0)

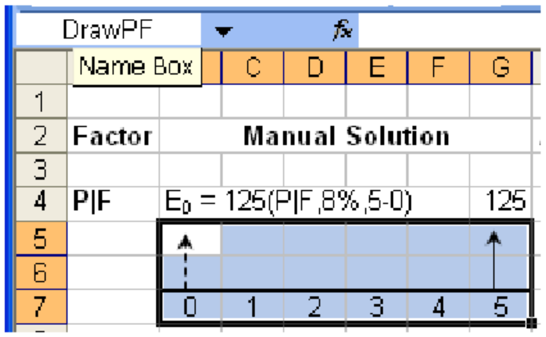
Figure 3. Intrinsic Excel Functions and EFactors

Basic Graphics
Excel provides a drawing toolbar that can be accessed via the View|Toolbars|Drawing menu. It has buttons for lines, arrows, their styles, and a Draw menu that allows grouping, ordering, and other options. It is fairly intuitive, for example, a line is drawn by clicking the line button and then dragging the line. However, there is one trick worth noting: lines and arrows can be made precisely horizontal or vertical by depressing the Shift key while dragging.

Creating shapes is not difficult, but it can be time consuming. One very easy way to facilitate this process is to create a basic set of cash flow diagrams, such as those shown in Figure 3. Figure 4 illustrates selecting the range around each diagram, typing its name into the name box, and pressing Enter. Copy these named ranges along with their shapes by

- selecting the target cell in the upper-left corner of where the range is to be placed,
- clicking the name box drop-down to list all names,
- selecting the desired name,
- keying Ctrl + C to copy it,
- keying Ctrl + G and then Enter to return to the target cell,
- and keying Ctrl + V to paste it.

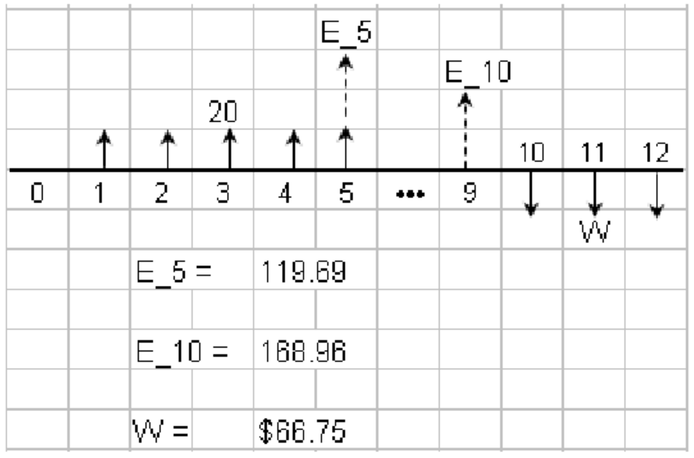
Scaling is maintained if the receiving range has cell sizes equal to those of the source range. Additional editing, copying, and drawing frequently are necessary to achieve the desired final result. A later section explains how macros can greatly simplify creating graphics.



1						
2	Factor	Manual Solution				
3						
4	P|F	E_0 = 125(P|F,8%,5-0)				125
5						
6						
7		0	1	2	3	4 5

Figure 4. Naming a Range

Showing Logic and Naming Equivalents
Figure 5 shows a solution created using the simple techniques explained above. Graphics corresponding to F|A and to P|A were copied, pasted, and edited. Keying Ctrl + (accent grave, usually above the Tab key) shows the logic of the solution by placing the display in formula mode, as shown in Figure 6. Notice that the formulas for E_10 and W reference E_5 and E_10, respectively, as named ranges. For example, the cell containing the result 119.69 is named by selecting it and typing SheetName!E_5 into the name box, where SheetName is the name of



E_5 = 119.69

E_10 = 168.96

W = $66.75

Figure 5. Solution Using Simple Techniques

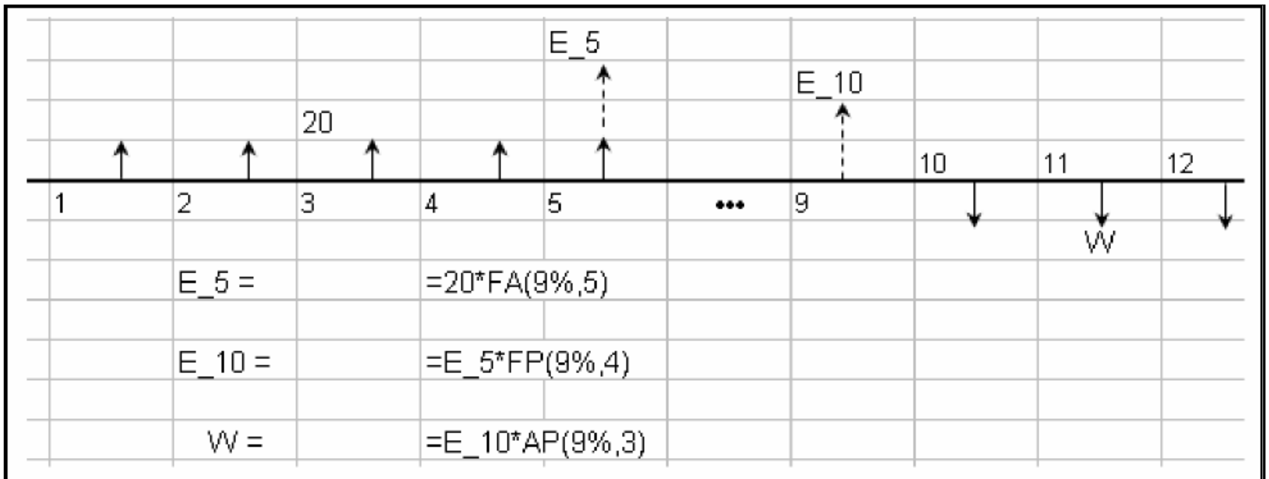


Figure 6. Formula Mode

the current sheet. The sheetname is included as part of the range name so that name can be used on other sheets, if needed, but references in expressions on the same sheet do not need to include the sheetname. Keying Ctrl + a second time toggles the display back to results mode.

Checking Solutions

Students usually want feedback regarding the correctness of their solutions. Table 2 shows a custom function that determines if a student's numeric response is within a given tolerance of the target solution. Allowable AE (absolute error) and APE (absolute percent error) values are input, and if either one is satisfied, then IsClose is true; otherwise it is false. An exact match is required for non-numeric responses.

Students can be required to put solutions within specific bordered cells, as shown in Figure 7. If the bordered cell is H12, then the check cell to its right contains:

Table 2. IsClose Function

```
Function IsClose(Response As Variant, Target As Variant, AE As Double, _
       APE As Double) As Boolean
  'Returns True if Response is close enough to Target; else returns False.
  On Error GoTo IsCloseFail
  If IsNumeric(Response) Then
   If (Abs(Response - Target) <= AE) Then
     IsClose = True
    Exit Function
    End If
     If (Abs((Response - Target) / Target) <= APE) Then
     IsClose = True
   End If
  ElseIf Response = Target Then
   IsClose = True
  End If
IsCloseFail:
End Function
```

=IF(ISBLANK(H 12),"", IsClose(H 12,66.75,0.1,0.01)) This results in its displaying nothing if H12 is blank. Otherwise, it checks to see if the answer is 66.75 within an AE of 0.1 and an APE of 0.01, and its value becomes TRUE or FALSE accordingly. Checking solutions can be extended to automated grading by storing solutions on sheets that have been declared via VBA code as deeply hidden so that neither is the sheet visible nor does its name appear in any menu. Then a VBA procedure can be written to compare a student's result with the correct one, return a score, and store it.

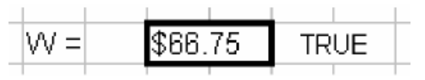


Figure 7. Checking an Answer

Professors can make cheaters work harder by protecting the check cell. First select all cells via Ctrl + A, and use the Format|Cells|Protection menu to unlock and unhide all cells by removing all check marks in the dialog box. Then select the check cell and use the same menu to set it to locked and hidden. Next use the Tools|Protection|Protect menu to allow users to do everything *but* select insert columns, so students cannot move the check cell away from the answer cell and replace it with one of their choosing. Making the checked cell both locked and hidden prevents its formula from being seen. If students are permitted to look at the formula to determine the answer, then do not make the check cell hidden.

Professors can make cheating even more difficult by using random number generators to provide problems for each student that have the same logical structure, but different parameters. For example, one student's problem might have an initial investment of $100,000, whereas another student's might have a cost of $103,000 and a slightly different project life. Students still can mimic the logic of someone who has gotten his or her question correct, but this requires a more sincere cheater than a simple copier, and some learning might accidentally occur. Implementing individual questions is clearly feasible, but doing this in a comprehensive, effective manner ultimately requires logistical considerations best addressed in a separate paper.

Toolbars and Forms

The foregoing procedures enhance the use of Excel in the classroom, but the large number of seemingly minor activities, such as editing a shape or entering names or using smaller fonts for time line values, still take time and discourage use. One solution to this is to automate those tasks with custom toolbars or forms. The primary difference that an end-user perceives is that a toolbar's buttons are graphics whereas form buttons generally are text-based. The positioning of toolbars is more flexible, but it is easier to implement forms such as the one shown in Figure 8.

The initial steps in creating a form are similar to those presented for a module. Press Alt + F 11 to open VBA, and use the Insert menu shown in Figure 2, but this time select UserForm. This causes a blank form to be displayed. Editing its (Name) property, and clicking the Toolbox icon and dragging buttons onto the form eventually results in the display shown in Figure 9. Double clicking a button opens the code view of the form. For example, double clicking the help button (with a question mark) shows the code that is

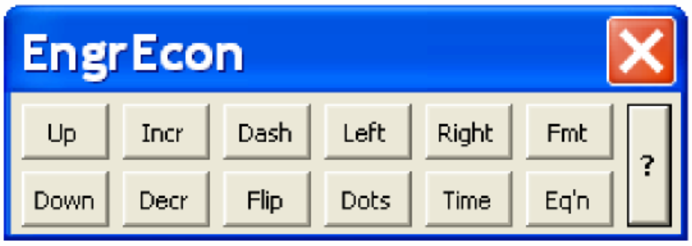


Figure 8. EngrEcon Form

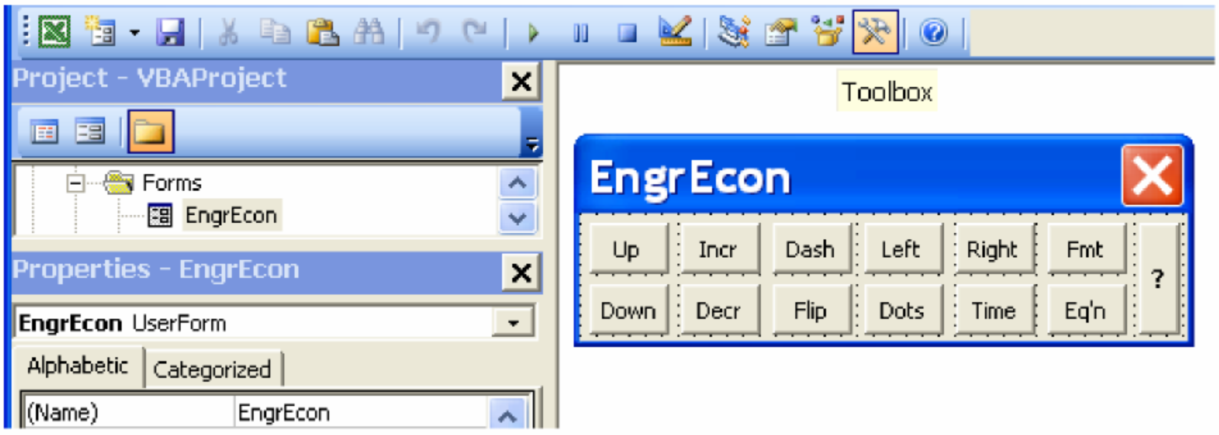


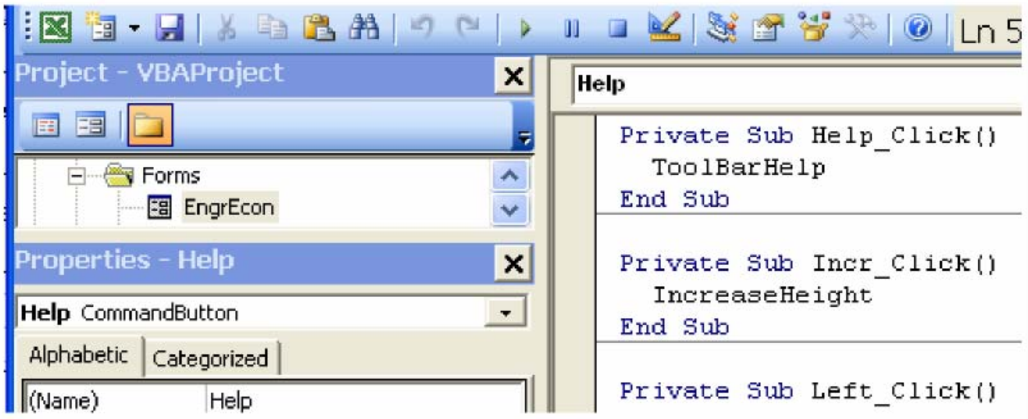
Figure 9. EngrEcon Form in Design Mode

executed when the help button is clicked from the spreadsheet interface. In this case, it simply calls another user-written subroutine named ToolBarHelp as shown in Figure 10. Writing VBA routines to create arrows and perform various formatting chores is not as simple as creating custom functions to evaluate factors, since it requires a knowledge of VBA's object structure for Excel. The following section provides a brief introduction to that topic, and then the next section shows how the EngrEcon form uses Excel's objects.

Objects

Most professors have programming experience, but many are not familiar with object-oriented programming. One of the basic problems in this regard is learning what objects are in a quick, efficient manner so that one feels comfortable with using them. That is the topic of this section.

Figure 2 shows how to create code modules used for Efactors and forms. A code module consists of variables used to store data and of procedures (subroutines and functions). Modules are used to organize a large program, much like directories arrange files on a hard drive. Variables that can be accessed in other code modules are declared with a Public statement, which allocates space like the Fortran Common statement. Variables accessible only by routines within their



```
Private Sub Help_Click()
  ToolBarHelp
End Sub

Private Sub Incr_Click()
  IncreaseHeight
End Sub

Private Sub Left_Click()
```

Figure 10. EngrEcon Form in Code View

code module are declared with a Private statement. Similarly, Public and Private statements can control how procedures are used by other modules. Within a procedure, Dim statements serve a role similar to the Fortran Dimension statement for local variables.

Another option in Figure 2 is to create a class module. Just like a code module, a class module contains storage declarations using Public, and Private statements, and these declarations are followed by functions and subroutines. Unlike a code module, a class module's Public and Private declarations do not actually allocate storage; they only describe its type, such as Integer or String, or the potential number of array elements. The actual allocation of storage occurs when an object is created by code within a code module such as:

```
Dim MyArrow as Shape
```

Excel has an intrinsic class module named Shape that has storage allocations such as Height, and procedures, such as Flip. Declaring MyArrow to be an object of class Shape creates a new set of variables named MyArrow with individual elements accessible via references such as MyArrow. Height. For example, the following statement changes the value of MyArrow's height:

```
MyArrow. Height = 20
```

Such accessible variables are referred to as *properties.*

VBA records that MyArrow was based on the class module Shape, so Shape's procedures can access MyArrow's data. For example, the syntax for reversing the direction of MyArrow is:

```
MyArrow.Flip
```

Such procedures are referred to as *methods*. Each set of variables and their related procedures is called an *object*. Multiple objects can be created, such as You rArrow, as shown in Figure 11. Some class modules, such as Shape, contain methods for creating new objects. For example,

```
ActiveSheet. Shapes.AddLine(FromX, FromY, ToX, ToY)
```

refers to the active sheet and creates a new line object on it.

Professors developing computer-aided-instruction using Excel as a platform can regard object oriented programming basically as a naming device and a convenient way to use Excel's extensive subroutine libraries that are stored as class modules. One of the best ways to learn which objects exist and how to use them is to click Tools|Macro| Record New Macro from an Excel spreadsheet, perform several actions, and then examine the resulting subroutine. VBA's help menu is another source of information, but sometimes Internet searches using the keyword VBA followed by others words result in quicker finds. Another good method is to click VBA's Object Browser icon as shown in Figure 12, scroll through it (or use its search facility) until the desired method or property is found, and then press F 1.

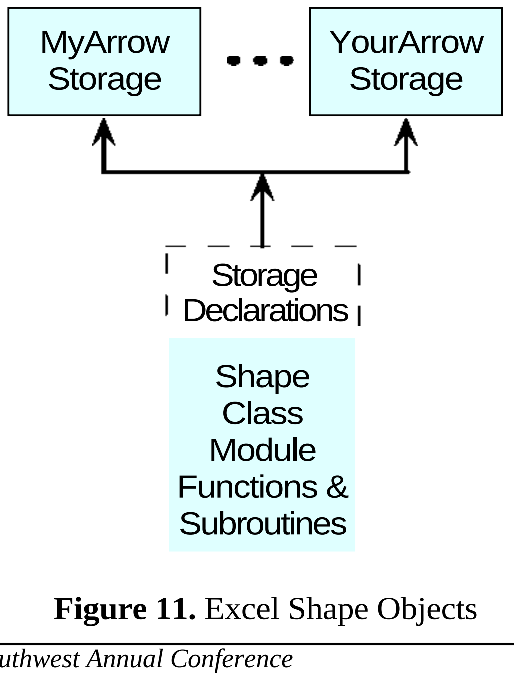


Figure 11. Excel Shape Objects

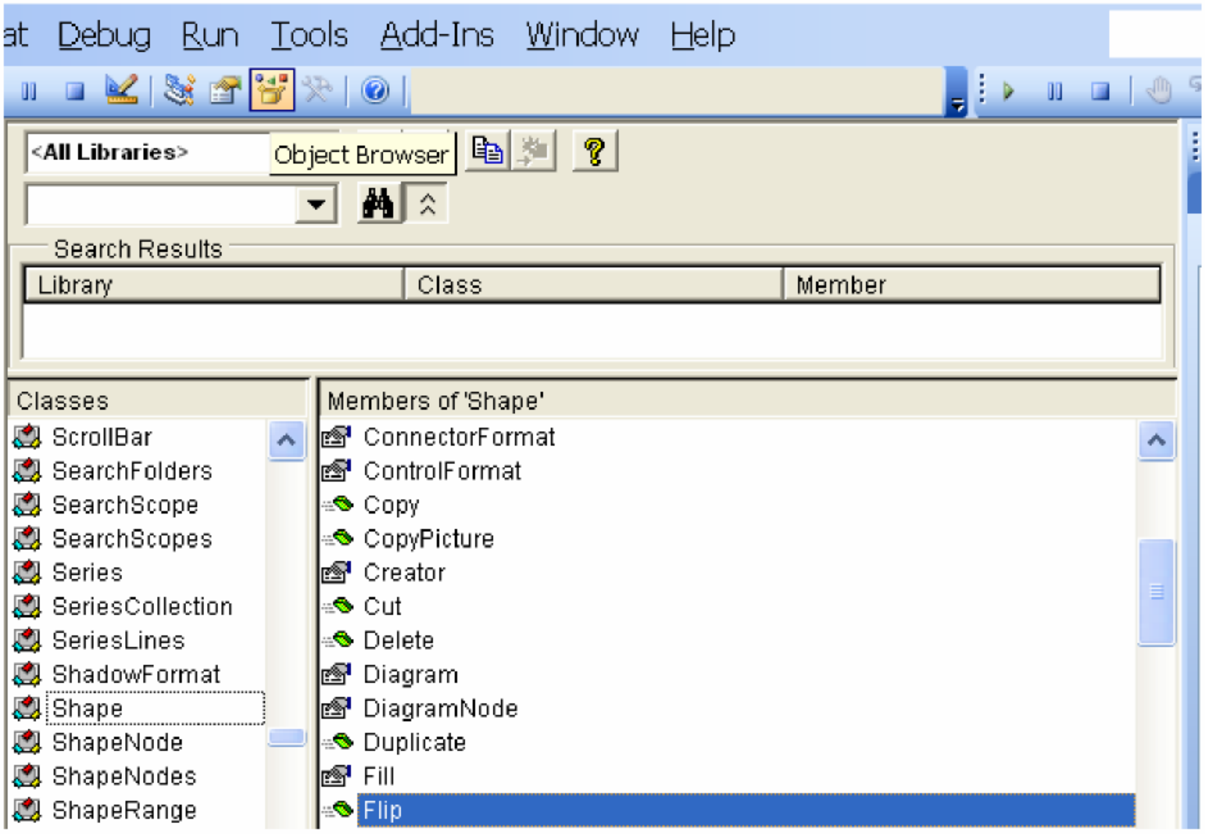


Figure 12. VBA's Object Browser

EngrEcon Form
The EngrEcon form uses subroutines that access Excel's intrinsic objects. It works by first selecting either a range of cells or one or more shapes, and then clicking the appropriate button. For example, Figure 13 shows drawing down arrows by first selecting E5:H5 and then clicking EngrEcon's Down button; the resulting arrows are shown at the bottom of the figure. Shapes usually are left in a selected state immediately after they are created, so simply clicking the Incr or Decr buttons increases or decreases the length of the arrows in convenient increments. Clicking elsewhere deselects the arrows, and then clicking the third arrow and keying the Delete key removes it. Multiple shapes can be selected by shift-clicking them.

Figure 14 shows the completed cash flow diagram and solution, as well as providing explanatory notes on the function of other buttons. The Fmt button used for cell M7 also can create Greek letters, and it assigns a currency format to numbers. The Eq'n (Equation) button used for cell F9 also provides superscripts and Greek. In this example it accesses a user-defined substitution list to find FA(in cell F9 and replace it with (F|A in cell J9. Clicking the Help button displays the additional information shown in Figure 15.

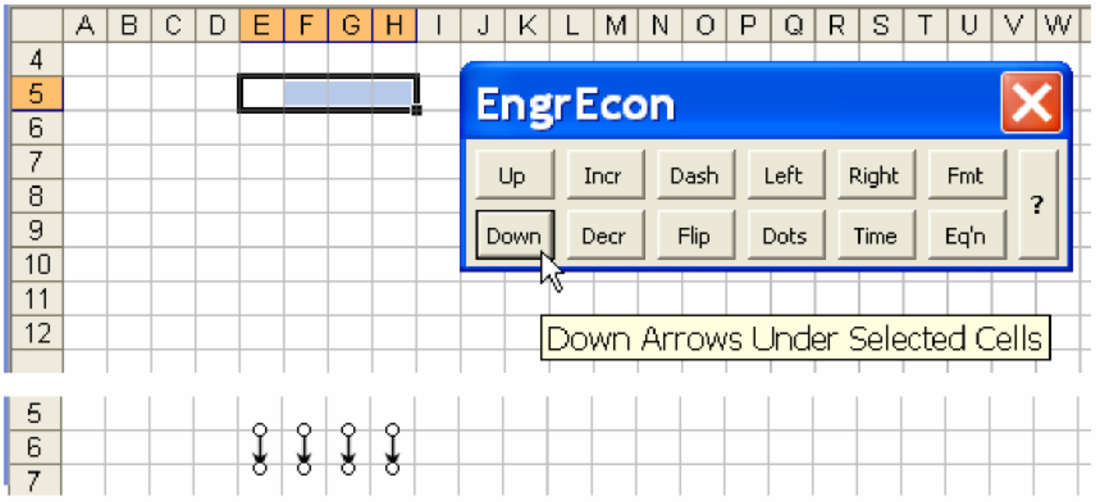
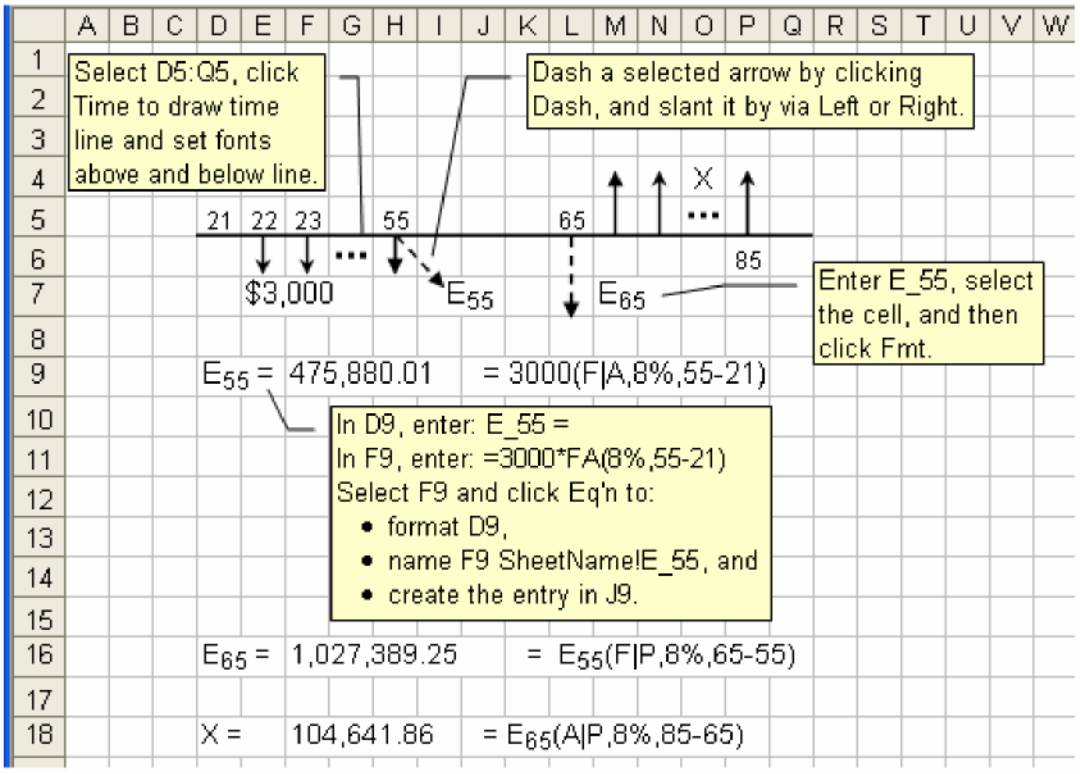


Figure 13. Down Arrows

Conclusion

Calculators have enhanced engineering education by allowing both professors and students to explore and solve problems more quickly. Some of the far greater potential offered by spreadsheets has been incorporated into the classroom, but they remain a largely unexploited tool in part due to problems with formatting, notation, and graphics. This paper offers suggestions to



Select D5:Q5, click Time to draw time line and set fonts above and below line.

Dash a selected arrow by clicking Dash, and slant it by via Left or Right.

Enter E_55, select the cell, and then click Fmt.

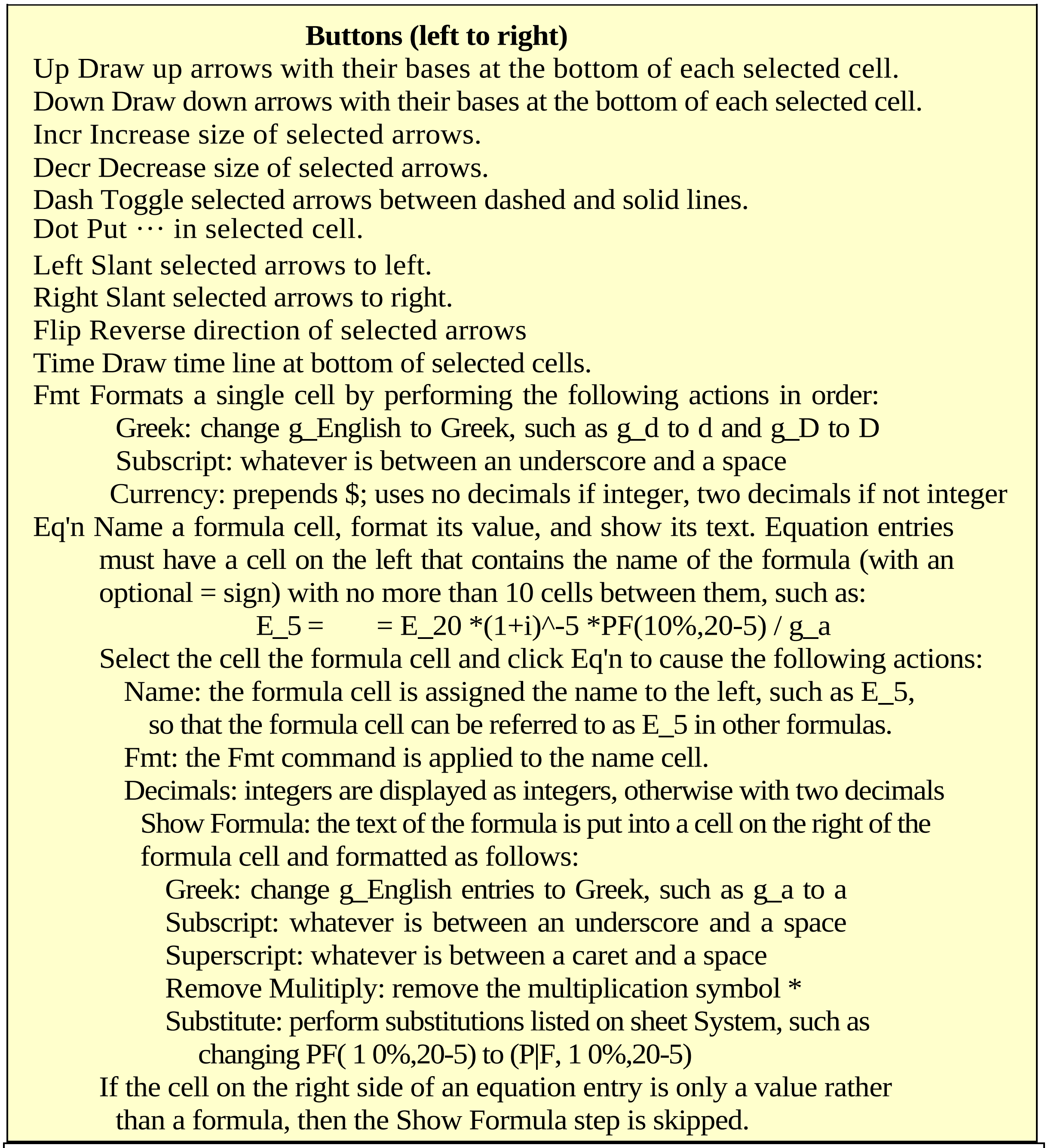
E_{55} = 475,880.01 = 3000(F|A,8%,55-21)

In D9, enter: E_55 =
In F9, enter: =3000*FA(8%,55-21)
Select F9 and click Eq'n to:
- format D9,
- name F9 SheetName!E_55, and
- create the entry in J9.

E_{65} = 1,027,389.25 = E_{55}(F|P,8%,65-55)

X = 104,641.86 = E_{65}(A|P,8%,85-65)

Figure 14. Completed Diagram and Solution



Buttons (left to right)

Up Draw up arrows with their bases at the bottom of each selected cell.
Down Draw down arrows with their bases at the bottom of each selected cell.
Incr Increase size of selected arrows.
Decr Decrease size of selected arrows.
Dash Toggle selected arrows between dashed and solid lines.
Dot Put ··· in selected cell.
Left Slant selected arrows to left.
Right Slant selected arrows to right.
Flip Reverse direction of selected arrows
Time Draw time line at bottom of selected cells.
Fmt Formats a single cell by performing the following actions in order:
  Greek: change g_English to Greek, such as g_d to d and g_D to D
  Subscript: whatever is between an underscore and a space
  Currency: prepends $; uses no decimals if integer, two decimals if not integer
Eq'n Name a formula cell, format its value, and show its text. Equation entries must have a cell on the left that contains the name of the formula (with an optional = sign) with no more than 10 cells between them, such as:
  E_5 =   = E_20 *(1+i)^-5 *PF(10%,20-5) / g_a
  Select the cell the formula cell and click Eq'n to cause the following actions:
   Name: the formula cell is assigned the name to the left, such as E_5, so that the formula cell can be referred to as E_5 in other formulas.
   Fmt: the Fmt command is applied to the name cell.
   Decimals: integers are displayed as integers, otherwise with two decimals
   Show Formula: the text of the formula is put into a cell on the right of the formula cell and formatted as follows:
    Greek: change g_English entries to Greek, such as g_a to a
    Subscript: whatever is between an underscore and a space
    Superscript: whatever is between a caret and a space
    Remove Mulitiply: remove the multiplication symbol *
    Substitute: perform substitutions listed on sheet System, such as changing PF(1 0%,20-5) to (P|F, 1 0%,20-5)
  If the cell on the right side of an equation entry is only a value rather than a formula, then the Show Formula step is skipped.

Figure 4 15. Help

facilitate using spreadsheets by applying Excel to engineering economics. It presents basic intrinsic features and then extends them via macros, and it is demonstrates how a discipline specific form or toolbar can remove many of the problems associated with spreadsheets.

Extensions to this basic system already are underway to modularize graphic toolkits so that disciplines other than engineering economics can be served. Other advances include procedures for giving each student unique homework assignments, automating grading, developing interactive tutorials, and authoring classroom presentations. Computer-aided-instruction is improving, but the key word is *aided*. Professors dedicated to engineering education must continue to help students reach their potential, and enlightened administrators must encourage and facilitate the pursuit of excellence in teaching.

Bibliography

1. http://office.microsoft.com/en-us/excel/default.aspx
2. http://www.openoffice.org/product/calc.html
3. http://www.evermoresw.com/
4. http://peltiertech.com/Excel/excellinks.html
5. http://www.anthony-vba.kefra.com/
6. http://www.bettersolutions.com/excel.aspx
7. http://www.cpearson.com/excel.htm
8. http://www.digdb.com/
9. http://www.exceltip.com/category.html
10. http://www.exceltip.com/category.html
11. http://www.functionx.com/excel/
12. http://www.microsoft.com/technet/prodtechnol/office/office2000/proddocs/opg/part2/ch09.mspx
13. http://www.mrexcel.com/articles.shtml
14. http://www.aiche.org/uploadedFiles/CareersEducation/Education/DepartmentUploads/CEI Catalog Spring09.pdf
    See CH200 and CH201 course descriptions on page 4.
15. Chambers, Terrance L., "Teaching Engineering Analysis Using VBA for Excel," *Computers in Education Journal*, ASEE, April-June, 2008, p. 71-78.
    http://www.asee.org/asee/publications/divisions/coedPapers/display.cfm?pdf=16148 1 .pdf

Note: All web sites were last visited and functional on December 2, 2008.