TEACHING ENGINEERING ANALYSIS USING VBA FOR EXCEL

Terrence L. Chambers
Department of Mechanical Engineering
University of Louisiana at Lafayette
PO Box 44170
Lafayette, LA 70504-4170

Abstract

When teaching computer programming to engineering students there is always some amount of controversy regarding the language which should be taught. Traditional choices were Fortran or C, but more modern choices are C++, Java, and Visual Basic. This paper describes an Engineering Analysis class that is taught primarily using Visual Basic for Applications (VBA) for Excel. The paper outlines the reasons for choosing that language, illustrates how the course is organized, and describes how well the course has worked in practice. Certain topics of interest with regard to VBA for Excel are explored, and sample programs are included.

Introduction

In many engineering programs a course exists to teach computer programming to the engineering student. Sometimes this course is a standard introductory-level programming course taught by the Computer Science Department. However, many schools have decided to teach this class within the College of Engineering, since the homework programs in a CS class tend to have nothing to do with engineering. When the course is taught within the College of Engineering, the programming course is often combined with a Numerical Methods for Engineers course, since the student can then learn the numerical methods and programming skills in parallel.

This paper will describe an Engineering Analysis course taught at the University of Louisiana at Lafayette that teaches numerical methods that are useful to engineers in conjunction with the VBA for Excel programming language. First, the reasons for choosing VBA for Excel will be explored. Then, the structure of the class will be described, with particular attention to the order in which the topics are presented, so that the students progress at an equal rate in both topics, numerical methods and computer programming. A simple VBA for Excel Program is provided to illustrate the use of that language.

Reasons for Choosing VBA for Excel

The major alternative languages that might have been chosen for use with this class were C++ and Java, but a great deal of legacy engineering code is written in Fortran and C, so there would be some justification for using one of these languages as well. Table 1.0 below compares the above languages plus VBA with regard to several attributes.

Based on Table 1.0, Fortran and C were eliminated early because they are not in such common usage anymore, and because they did not have as many capabilities as the more modern languages. The choices therefore, for this class, came down to C++, Java, and Visual Basic. C++ was a likely candidate since at this time most commercial applications are written in that language. However, it is very hard to

Table 1.0 – Comparison of Programming Languages.

	Fortran	C	C++	Java	Visual Basic
Programming style	Procedural	Procedural	Object-oriented or procedural	Object-oriented only	Procedural and object-oriented
Current usage	Rare	Fairly common	Very common	Common	Common
Engineering calculations	Easy	Fairly easy	Fairly easy	Fairly easy	Easy
Development of graphical user interfaces	Hard	Hard	Hard	Fairly easy	Very easy
Used as a macro language	No	No	No	No	Yes
Web scripting	No	No	No	Yes	Yes
Compiler Needed	Yes	Yes	Yes	No	No

write programs that have a graphical user interface using that language. As a result, it was felt that if C++ were selected it would take two semesters worth of training to get the student proficient enough to write practical programs with a modern Windows-based graphical user interface. Since we did not have two semesters available for this topic in our curriculum, C++ was eliminated. It is worth noting that Mason and Cornwell[1] have recently addressed this problem by creating a C++ programming shell to simplify the development of C++ graphical user interfaces. That is another possible approach.

Java and Visual Basic stack up fairly well one against the other. One characteristic of Java is that it is purely object-oriented. Although we wanted to expose the students to object-oriented programming, we did not want to force that rather advanced programming concept upon them too early. In Java you have to understand objects before you can create your very first program. Visual Basic has both procedural and object-oriented features, but the object-oriented features that are forced upon the student early on are "visual," and therefore very easy to use. Both Java and Visual Basic can boast of features that are not available in the other languages, such as a scripting language (either Javascript or VBscript) that can be used in web page development, and the fact that both languages can be used either with or without a compiler. We finally chose Visual Basic. The main factor that caused us to select VB over Java was the fact that it is used as the macro programming language for all the Microsoft Office products, including Excel, as well as many engineering programs such as the Solidworks CAD program. It was felt that the leverage that one gains by learning a language that is used so extensively as a macro language would be very beneficial to the student. Naraghi[2] and Thomas et al.[3] recently reported the use of VBA/Excel in Freshman Engineering classes. In those cases, VBA/Excel was chosen for many of the same reasons cited here.

Once the decision was made to use Visual Basic, a further choice existed between the standalone programming environment, such as that contained in VB 6, VB.Net, and VB 2005,

or the embedded Visual Basic for Applications (VBA) environment that comes standard in all Office products. We chose VBA for Excel for two reasons. First, the standalone language has been undergoing major revisions lately, with a big difference existing between the VB 6 version and the VB.Net versions, while the VBA environment has been quite stable. The second reason is that the standalone development environment is not always available to the student after leaving school, whereas it is considered very likely that the graduate will have access to Microsoft Excel, which contains VBA.

Structure of the Class

The course is organized as a 3 credit hour class with two hours of lecture and three hours of lab. Normally, a new numerical method is taught each week during the lecture periods and the programming topic for the week is taught during lab. Steven Chapra has written both a numerical methods book and a programming text that work well with this class[4, 5]. After the programming topic has been introduced during the lab period, the students write a program to implement the numerical method that has been discussed in the lecture periods.

A complete syllabus for the course is shown in Appendix A. The important feature to notice, however, is that the course is designed so that the student can write a very basic program during the very first lab class, and that the order of the numerical methods topics have been arranged so that the programs the student will write to implement those topics will increase slowly in complexity. This insures that as each week's numerical method assignment is given, the student will have the programming skills necessary to complete the assignment.

The first two weeks of the course vary from the standard format so that the student can get a slight head start on their programming skills. The first lab period is used to refresh the student's knowledge of Excel, and to introduce VBA in its two simplest applications, as a recorded macro, which requires no programming at all, or as a user-defined Excel function, neither of which require a graphical user interface. Figure 1 below shows the user-defined function which the students write during the first lab period. The function is called Cube, and it calculates the value of x^3.

```
Private Function Cube(x as Double) as Double
Cube = x*x*x
End Function
```

Figure 1. Sample User-Defined Function.

During the second lab period the students are introduced to the way that VBA is used to create graphical user interfaces. In this lesson they learn how to create a basic window (called a form), how to create text boxes and command buttons, how to connect a subroutine to the command button, and how to get data into and out of text boxes on the form. This information is all put together into a simple program that adds two numbers together. This simple program becomes the basis of all the other programs in the class. A screen capture of that program is shown in Figure 2, while the source code is given in Appendix B.

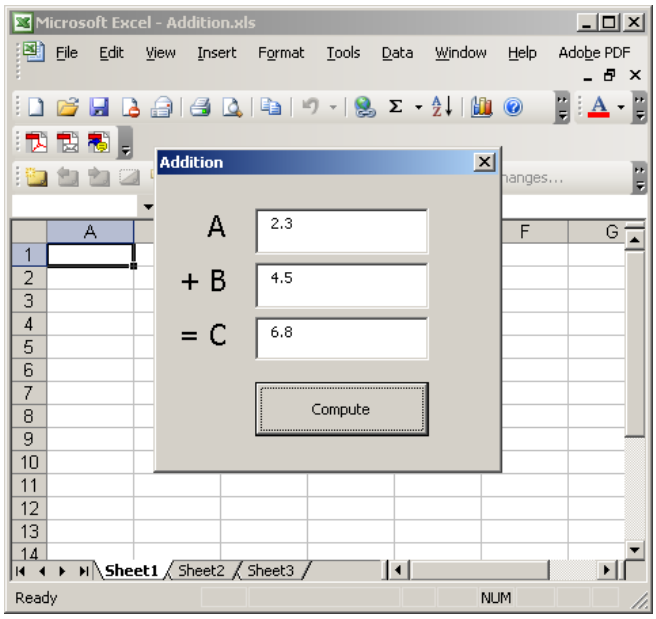

Figure 2 – VBA for Excel Addition Program.

The next week of class begins the normal rotation of two lectures on a numerical method and one lab period where the students are introduced to whatever additional programming techniques are necessary to write the program of the week. The first topic in numerical methods is Computer Precision. The programming topics for the week include a discussion of data types and of strings, which are used for the first time in the Computer Precision program, as shown in Figure 3 below.

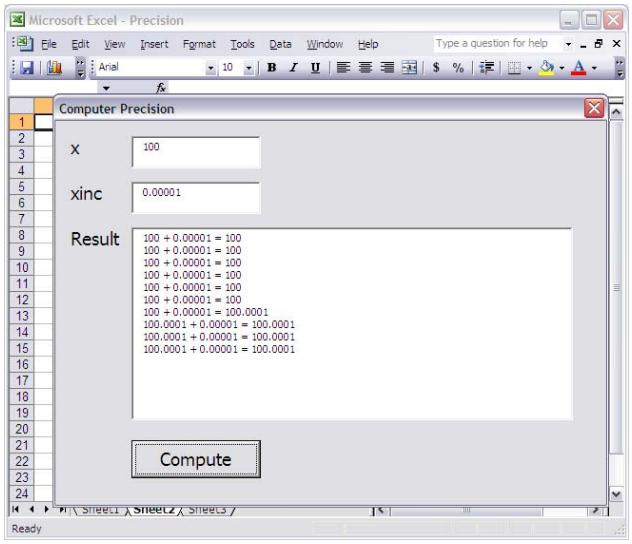

Figure 3. VBA for Excel Computer Precision Program.

In this program, the student learns how to create, update, and manipulate strings, and how to create multi-line output in a text box. This program is a modification of the previous week's Addition program. In addition, the student learns about Computer Precision by conducting an experiment using the program shown above.

The next week the students are introduced to the very important concept of how to communicate information back and forth between the program and the spreadsheet. One of the most pleasing features of using VBA for Excel as the programming language is that formulas can be entered on the spreadsheet rather than hard-coded into the program. For example, the next numerical method discussed in this class is Numerical Derivatives (See Figure 4 for a screenshot, and Appendix C for the source code). In other languages, the function that will be differentiated normally resides in a hard-coded subroutine or function, requiring the user to modify the program (and usually to re-compile it as well) every time a new equation needs to be differentiated. With VBA for Excel, however, the equation can be entered as a formula on the spreadsheet. A subroutine called, "PutData" is created which will take a double precision variable from the program and put it into a designated cell on the spreadsheet (see cell B1 in Figure 4.0 below). This causes the spreadsheet formula to re-calculate, and the new function value is then read into the program from the cell on the spreadsheet that contains the formula (see cell E1) using a function called, "Func." The key VBA for Excel programming concept is the use of RefEdit boxes on the form and corresponding Range type variables in the program to create pointers in the program to cells on the spreadsheet. In Figure 4 below, the RefEdit boxes are boxes designated as, "x Cell," "h Cell," and "f(x) Cell."

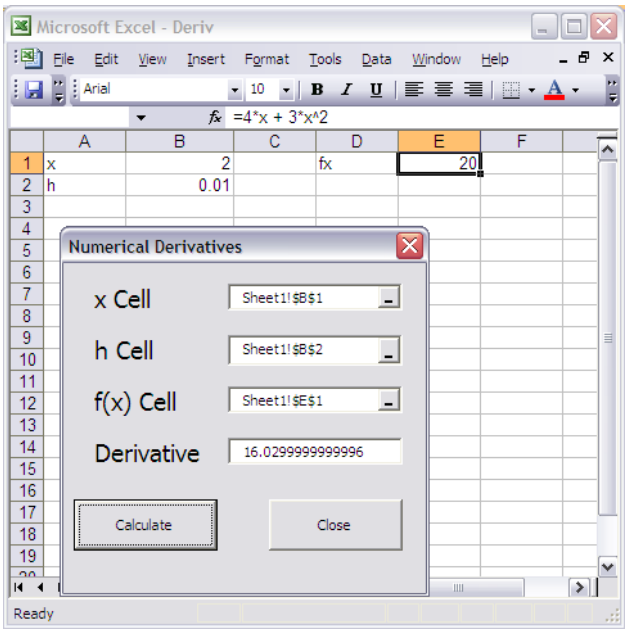

Figure 4. VBA for Excel Numerical Derivatives Program.

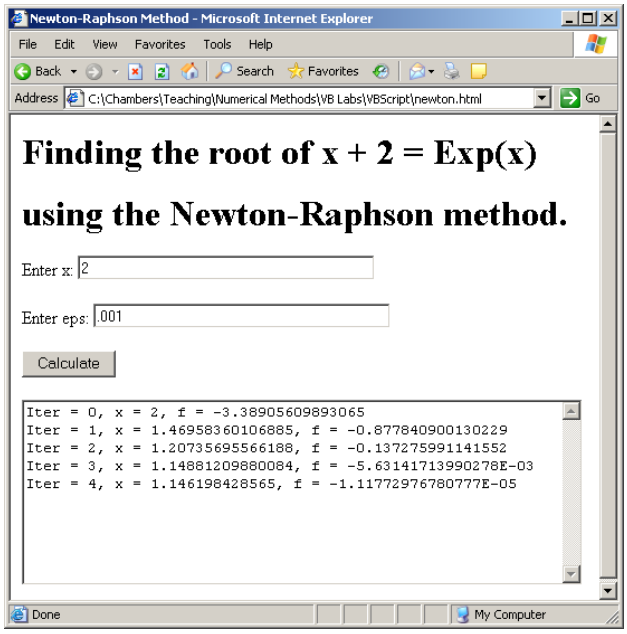

Figure 5. VBScript Newton-Raphson Program.

After the Numerical Derivatives program, the students have most of the user interface programming skills that they will need to complete the course, although their skills will continue to improve as they learn to use dynamic memory allocation, read and write matrices to and from the spreadsheet, create an array function for use directly from a cell in Excel (a single step simultaneous equation solver), how to create and write to new sheets in the Excel workbook, and how to create programs that incorporate multiple subroutines and functions in multiple code modules.

During week 10 the students are also introduced to web programming, HTML, and VBScript, a subset of Visual Basic that can be used for client side processing when using Internet Explorer. Figure 5 shows a screenshot of the VBScript version of the Newton-Raphson program to solve for the root of an equation. Appendix D contains the VBScript source code.

Results

The Engineering Analysis class at the University of Louisiana at Lafayette has been taught using Visual Basic for several years now. The switch to VBA for Excel, however, has improved the course in two ways. First, the language is stable, and second, since the function to be used can be programmed onto the spreadsheet, the program does not need to be modified every time the user wants to operate on a different function. The students rate the course very highly, and indicate the workload is about right for a 3 credit-hour course. There has always been a high rate of loss of programming skills in subsequent semesters unless the students are forced to use their newly found programming skills in later classes to cement the knowledge into long term memory. However, because the students continue to use Excel in other classes, the students seem to maintain the programming skills learned in this class more than if they had to use another language, such as Java or C++.

Conclusion

The choice of VBA for Excel has been shown to be an effective choice as a programming language for engineering students. The choice is advantageous because: 1) the language is easy for the students to master in one semester, 2) the students can easily create graphical user interfaces for their programs, 3) the language can be used as a scripting language to create interactive web pages, 4) the language can be used as a macro programming language in all the Microsoft Office products, as well as several engineering software packages, such as Solidworks, and 5) the language is embedded in Excel, which will likely be available in the workplace, thus making it more likely that the student will actually make use of the programming skills after graduation.

The students have responded well to the change to this programming language and have demonstrated an ability to become quite competent in its use in one semester.

References

1. Mason, G., Cornwell, R., 2007, "A C++ Programming Shell to Simplify GUI Development in a Numerical Methods Course," *Computers in Education Journal*, Vol. XVII, No. 2, April – June, 2007, pp 66 – 73.

2. Naraghi, M. H. N., 2007, "VBA/Excel: An alternative Computer Programming Tool for Engineering Freshman," *Computers in Education Journal*, Vol. XVII, No. 2, April – June 2007, pp 74 - 80.

3. Thomas, G. E., Minnick, M. V., Gang, D., 2006, "Evolution of a Freshman Software Tools Class," *Computers in Education Journal*, Vol. XVI, No. 3, July – September 2006, pp 40 - 49.

4. Chapra, S., *Power Programming with VBA/Excel*, Prentice-Hall E-Source, 2003.

5. Chapra S., Canale, *Numerical Methods for Engineers*, 5th ed., McGraw-Hill Book Company, New York, 2003.

Biographical Information

Dr. Chambers currently serves as an Associate Professor of Mechanical Engineering at the University of Louisiana at Lafayette. His research interests include engineering design and optimization, artificial intelligence, genetic algorithms and genetic programming, engineering software development, and numeric and symbolic solutions to engineering problems. Prof. Chambers is a registered Professional Engineer in the state of Louisiana.

Appendix A

MCHE 301 Class Schedule
(Note: PP refers to reference [4], and NM refers to reference [5]).

Week	Monday Lecture (Reading)	Wednesday Lecture (Reading)	Wednesday Lab (Reading)
1	Martin Luther King Holiday	Intro to Programming (PP Chap 1)	Excel Tutorials Basic VBA/Excel (PP Chap 2 – 4)
2	Modular Programming (PP Chap 5)	Object Oriented Programming and Debugging (PP Chap 6 - 7)	Addition Lab Custom Dialog Boxes (Chap 15)
3	Computer Precision (NM Chap. 3)	Data Types (PP Chap 8)	Computer Precision Lab Computations and Strings (PP Chap 9 - 10)
4	High Accuracy Numerical Differentiation Methods (NM Sec. 23.1)	High Accuracy Numerical Differentiation Methods (NM Sec. 23.1)	Num. Derivatives Lab Structured Programming Decisions (PP Chap 11)
5	Roots of Equations Bracketing Methods (NM Chap. 5)	Roots of Equations Open Methods (NM Chap. 6)	Newton-Raphson Lab Structured Programming Loops (PP Chap 12)
6	Numerical Integration (NM Sec. 21.1 - 21.2)	Numerical Integration (NM Sec. 21.1 - 21.2)	Simpson's 1/3 Rule Lab Arrays (PP Chap 13)
7	Mardi Gras	Mardi Gras	Mardi Gras
8	Ordinary Differential Equations. Euler's Method (NM Sec. 25.1, 2)	Ordinary Differential Equations. Runge-Kutta Method (NM Sec. 25.3)	Runge-Kutta Lab Files (PP Chap 14)
9	Linear Algebra	Linear Algebra	Midterm Exam
10	Linear Algebra	HTML, VBScript	VBScript Lab
11	LU Decomposition (NM Chap. 10)	LU Decomposition (NM Chap. 10)	LU Decomposition Lab Linear Algebra HW
12	Curve Fitting (NM Chap. 17)	Curve Fitting (NM Chap. 17)	Curve Fitting Lab
13	Easter	Easter	Easter
14	Simultaneous Non-linear Equations (NM Sec. 6.5)	Simultaneous Non-linear Equations (NM Sec. 6.5)	Multiple Newton-Raphson Lab
15	Higher Order ODE's (NM Sec. 25.4)	Higher Order ODE's (NM Sec. 25.4)	Multiple Runge-Kutta Lab

Appendix B

Sample VBA for Excel Program

Code for Addition Program in Figure 2.0

```
Private Sub cmdCompute_Click()
    Dim a As Double, b As Double, c As Double

    If (IsNumeric(txtA.Text)) Then
        a = txtA.Text
    Else
        MsgBox "Please enter a numeric value for A"
        Exit Sub
    End If

    If (IsNumeric(txtB.Text)) Then
        b = txtB.Text
    Else
        MsgBox "Please enter a numeric value for B"
        Exit Sub
    End If

    c = a + b
    txtC.Text = c
End Sub
```

Appendix C

Sample VBA for Excel Program

Code for Numerical Derivative Program in Figure 4.0

```
Private Sub cmdCalculate_Click()
    Dim x As Double, h As Double, dfdx As Double, f As Double
    Dim rngVar As Range, rngH As Range, rngFx As Range

    If (frmDeriv.refX.Value = "") Then
      i = MsgBox("Must select a cell for the x variable.", vbOKOnly And
      vbApplicationModal, "Error")
        Exit Sub
    End If
    Set rngVar = Range(frmDeriv.refX.Value)

    If (frmDeriv.refH.Value = "") Then
      i = MsgBox("Must select a cell for the h variable.", vbOKOnly And
      vbApplicationModal, "Error")
        Exit Sub
    End If
    Set rngH = Range(frmDeriv.refH.Value)

    If (frmDeriv.refFx.Value = "") Then
      i = MsgBox("Must select a cell for the function.", vbOKOnly And
      vbApplicationModal, "Error")
        Exit Sub
    End If
    Set rngFx = Range(frmDeriv.refFx.Value)

    Call GetVar(rngVar, x)
    Call GetVar(rngH, h)


    dfdx = (Func(x + h, rngVar, rngFx) - Func(x, rngVar, rngFx)) / h
    txtFx.Text = dfdx

    Call PutVar(x, rngVar)
End Sub

Private Sub PutVar(x As Double, rngVar As Range)
    rngVar.Value = x
End Sub

Private Sub GetVar(rngVar As Range, x As Double)
    x = rngVar.Value
End Sub

Private Function Func(x As Double, rngVar As Range, rngFunc As Range) As
Double
    Call PutVar(x, rngVar)
    Func = rngFunc.Value
End Function
```

Appendix D

Sample VBA for Excel Program

Code for VBScript Newton-Raphson Program in Figure 5.0

```
<HTML>
<title>Newton-Raphson Method</title>
<!--Comment-->
<head>
<script language="VBScript">
<!--
Sub cmdCalculate_OnClick()
   Dim x, eps, f, i, output

   if (IsNumeric(txtX.Value)) then
      x = txtX.Value * 1.0
   else
      MsgBox "Enter a numeric value for x."
      Exit Sub
   end if

   if (IsNumeric(txtEps.Value)) then
      eps = txtEps.Value * 1.0
   else
      MsgBox "Enter a numeric value for eps."
      Exit Sub
   end if

   i = 0
   f = Func(x)
   output = "Iter = " & i & ", x = " & x & ", f = " & f & vbCrLf
   Do
      i = i + 1
      x = x - f / Deriv(x)
      f = Func(x)
      output = output & "Iter = " & i & ", x = " & x & ", f = " & f & vbCrLf
      If (i > 20) Then
         MsgBox "Does not converge after 20 iterations."
         Exit Do
      End If
   Loop While (Abs(f) > eps)

   MsgBox = output
   'txtOutput.Value = output
End Sub

Function Func(x)
    Func = x + 2 - Exp(x)
End Function

Function Deriv(x)
   dim h
```

```
   h = 0.0001
   Deriv = (Func(x + h) - Func(x)) / h
End Function
-->
</script>
</head>

<body>
<h1>Finding the root of x + 2 = Exp(x)<p>
using the Newton-Raphson method. </h1>
Enter the function (as shown below):<p> <textarea name=txtOutput rows=10
cols=60>
Function Func(x)
    Func = x + 2 - Exp(x)
End Function
</textarea><p>
Enter x: <input type=text size=40 ID=txtX Value="2"><p>
Enter eps: <input type=text size=40 ID=txtEps Value="0.001"><p>
<input type=Button Value="Calculate" ID=cmdCalculate><p>
Results:<p>
<textarea name=txtOutput rows=10 cols=60>
</textarea>
</body>
</html>
```