AC 2007-2524: BIOMEDICAL ENGINEERING SIMULATION USING VISUAL BASIC MACROS IN MICROSOFT EXCEL

Lanny Griffin, California Polytechnic State University

Lanny Griffin received his Ph.D. from the University of California at Davis in Materials Science and Engineering. He also has a BS degree in Mechanical Engineering from California Polytechnic State University in San Luis Obispo. Currently, he is a Professor of Biomedical Engineering at California Polytechnic State University in San Luis Obispo. He is also on the Mechanical Engineering faculty of the US Military Academy at West Point as an Army Reserve Officer. Dr. Griffin's research interests are in bone mechanics and biomaterials and has been the Principal Investigator of several projects from the Army, DOD, and NIH.

Robert Crockett, California Polytechnic State University

© American Society for Engineering Education, 2007

BIOMEDICAL ENGINEERING SIMULATION USING VISUAL BASIC MACROS IN MICROSOFT EXCEL

I. Introduction

Biomedical engineering analyses are often more complex than typically found in other engineering disciplines due to the inherent variation and uncertainty associated with living systems. Attempting to find the optimum solution to complex problems almost always is done using numerical analysis techniques, such as finite element or finite difference solutions.

Educationally, distance learning classes offer unique challenges to teaching of numerical analysis for engineers. While it is very common to use programs such as MATLAB for teaching numerical analysis, licensing this type of program is expensive and therefore not always available at the distance learning site. Using more conventional programming languages such as C or FORTRAN are also not as useful for distance learning for the same reason that compliers are expensive and not generally accessible to industrial distance learning sites and freeware compilers are not always robust enough for class work where time is limited. The need for a robust programming platform for industrial partners at distance learning sites is critical to avoid unnecessary learning curves. This is where using Microsoft Excel has a distinct advantage over other programming platforms as a tool for teaching numerical analysis. First of all, Excel is familiar. Secondly, Excel is ubiquitous. Lastly, Excel is capable of programming using Visual BASIC, for which the learning curve is not excessive as long as an individual has had a structured programming course.

For a reasonably complete course in numerical analysis, topics such as the solution of linear systems, non-linear systems, curve fitting, numerical integration, and solutions to ordinary and partial differential equations are usual topics. In order to create a course that is interesting and relevant to biomedical engineering students, having projects that are based on bioengineering problems is necessary. Fortunately, there are numerous problems that can be addressed which are well-posed.

Finding a textbook can be somewhat challenging since most numerical analysis books are written for MATLAB or C. Most of the Excel textbooks are not well-suited to a formal numerical analysis class because they are written to address typical spreadsheet usage rather than customization by macros [1]. We ultimately settled on using a newly published book entitled, Numerical Analysis for Biomedical Engineers [2] even though it was written for MATLAB. In this case, MATLAB script could be treated as pseudo-code and easily ported to Visual BASIC in Excel.

II. Learning Objectives and Outcomes

The numerical engineering class was designed for juniors to first year graduate students. The class is designed to be delivered both in a distance and local classroom setting since approximately 25 percent of the class is made up of engineers obtaining their Master's degree off-site. To pique student interest, all numerical lab exercises and homework were derived from

numerical examples taken from the authors' research experience and the engineering textbook. The terminal learning objectives were that students would have sufficient programming knowledge to: (1) develop code to numerically solve problems that would be commonly encountered in biomedical engineering. (2) Translate pseudo code into a functioning code. (3) To routinely use Excel or Visual BASIC to solve problems encountered in biomedical engineering. Our primary outcome was simply that the students become proficient enough to use Excel to solve problems in subsequent classes or at their work.

III. Course Design

The numerical analysis course for biomedical engineers covers nine topics with embedded case studies distributed over one quarter (Table 1). The use of Excel has evolved over several years through experience with distance learning. The primary problem in teaching a distance learning course in numerical analysis has been accessibility to programming software.

It is expected that students have taken at least one course in structured programming. The students are provided with a primer which introduces the Visual BASIC syntax, looping structures, conditionals and several detailed programming examples (One such programming example is provided in as an appendix). Excel Visual BASIC has a relatively short learning curve and students do not realize how much functionality can be added by using the macro language.

Table 1: Course structure of numerical analysis for biomedical engineers.

Topic	Case Study	Evaluation Type
Linear Equations (Matrix Inversion)	Joint reaction forces	Laboratory and Homework
Multiple linear regression	Fitting creep data to a linearizible model	Laboratory
Non-linear regression	Composite materials	Homework
Systems of non-linear equations	Fractional occupancy of ligand binding sites	Laboratory
Splines	Heart rhythm interpolation	Homework
Monte Carlo Simulation	Diffusion	Laboratory
Numerical Integration	Non-reacting tracer concentration in a chemical reactor	Homework
Ordinary Differential Equations	Nerve cell potentials	Laboratory
Partial Differential Equations	2-D Transient Diffusion	Laboratory and Homework

For all numerical exercises, we required that students interact with the program both through user forms and the spreadsheet using the Visual BASIC command:

```
worksheets(a).cells(i,j).value
```

where "a" is a reference to a worksheet and "i" and "j" are the row and column indices, respectively. This enables the students to work from familiar (the worksheet) to the unfamiliar (the macro).

We favor the use of Excel for several reasons, such as ease of programming, and availability – particularly in a distance learning context – which is probably the most compelling reason for using Excel to teach numerical analysis. Several of the labs are elaborated in the following sections.

A. *Joint Reaction Forces.*

This is the first formal programming exercise for the course. It is a simplified, statically determinate system for the forearm with a single muscle force (biceps) and a weight in the hand moving the forearm from the resting position till the forearm is parallel with the ground. The unknowns to be solved for are the x and y joint reaction forces and the muscle force. This problem is simple enough to be solved without a matrix solver; nevertheless, since the matrix solver is the basis of many numerical schemes, we felt that it is necessary to minimize the difficulty of the problem so that the students could easily verify results. The use of an angular quantity as an independent variable necessitated a looping structure to repeatedly call the matrix solver. The code was to write the coefficient matrix values to a spreadsheet page, then write the right hand side vector to a different sheet, and finally, write a Gauss-Jordan linear solver to place the solution on a separate sheet and then plot the data. While it is not necessary to write the data to different sheets, the instructive point is to have the students learn how to put the data where they want without having to format the data for plotting in order to minimize the overall outlay of work. Also, writing coefficient and right hand side matrices to separate sheets facilitates debugging.

One of the requirements for the matrix solver is a *matrix inversion routine*, which was used in subsequent labs. Because of the unfamiliarity of students with programming, many of the students (particularly distance learning students) solved the problem using the spreadsheet rather than trying to program the solution. This created more difficulty for those students because subsequent labs are built on the linear solver. In retrospect, it may have been better to create a more difficult problem for the students (one which would be intractable on a spreadsheet without macros) and would force the use of a matrix solver.

B. Fitting a Creep Model to Data.

This exercise was by far the most demanding. For this laboratory, students were required to find the fitting coefficients for a set of creep data that was dependent on stress and temperature. The model was a standard power-law creep equation

$$\dot{\varepsilon} = A\sigma^n \exp\left(\frac{-Q_c}{RT}\right). \quad (1)$$

Creep of polymers, particularly in load bearing applications, is extremely important. In this case, the students were given creep data for UHMWPE, commonly used in artificial joints.

The students linearized Equation 1 and fit the model to the data with a multiple linear regression tool that they developed. Details for regression analysis using matrices along with equations for confidence and prediction intervals can be found in most statistics textbooks [e.g. 3]. While Excel does have a regression solver, the solver the students created was more akin to a statistics program, such as MINITAB. The program output was to be highly formatted on the worksheet and included confidence and prediction intervals on the predicted means and coefficients. The students are required to determine the value of the t-statistic which is the basis of finding the confidence and prediction intervals. Calculating the t-statistic was done using a worksheet function, TDIST, and the GoalSeek function found under the Tools menu. Rather than have the students "invent" the method, they were given a subroutine:

```
Sub Goal(DOF,Alpha)

 With Worksheets("Sheet1")
   .Range("A2").Value = 2.5       ' T-Stat Seed Value
   .Range("A3").Value = DOF       ' Degrees of Freedom
   .Range("A4").Value = 2         ' Number of tails on t-dist
   .Range("A1").FormulaR1C1 = "=tdist(R2C1,R3C1,R4C1)"
   .Range("A1").GoalSeek Goal:= Alpha,
       ChangingCell:=Worksheets("sheet1").Range("A2")
 End With

End Sub
```

On the worksheet, the function TDIST will be placed in cell "A1" upon calling the subroutine Goal using the conventional syntax:

```
Call Goal(DOF,Alpha)
```

The seed value of the t-distribution is placed in "A2", as well as the degrees of freedom in "A3", and the number of tails on the t-distribution "A4". What the subroutine Goal does is vary the value of the t-statistic ("A2") until the level of confidence (Alpha) is met. The resulting value in "A2" will then be used to calculate confidence and prediction intervals. This particular part of the exercise is very instructive since it allows the students to access some of the built-in functionality of Excel within their own programs.

A fair amount of teaching effort is put forward in developing the mathematics so that the students can do the programming since it may have been a long time since they had a formal course in statistics (some may never had such a course). The students were encouraged to create a "form interface" (Figure 1) which has some of the nice data gathering features, such as the RefEdit box.

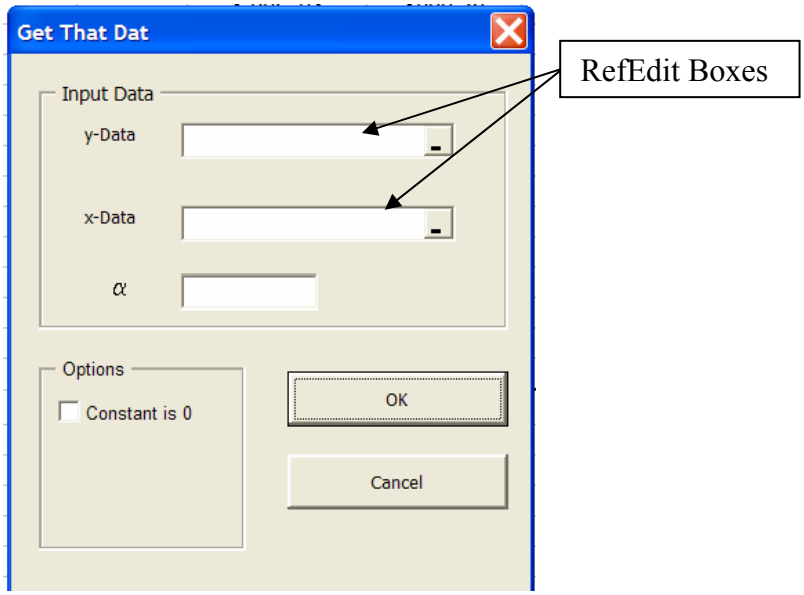

Figure 1: Example of a data collection form for the multiple linear regression model.

The equations for regression analysis require a matrix multiplier and matrix transposer in addition to the Gauss-Jordan solver they coded and validated in lab 1. Students were given two weeks to complete this lab because it requires a great deal of coding.

C. Systems of Non-linear Equations.

In class, we determined the receptor occupancy during receptor-ligand binding for two selective species. This was an example from the textbook [2] we used. For the laboratory, we developed a three-species ligand binding problem. Since cell membrane receptors bind to specific ligands, the gist of this problem was to determine the fractional occupancy of trimeric, dimeric, and monomeric receptors at equilibrium. From these considerations given that:

C_A = concentration of trimeric receptors
C_B = concentration of dimeric receptors
C_C = concentration of monomeric receptors
C_L = concentration of the ligand
C_D = concentration of all receptor-bound ligands

$$\begin{aligned} A + 3L \Leftrightarrow D \quad & K_{eq,1} = \frac{C_D}{C_A C_L^3} \\ B + 2L \Leftrightarrow D \quad & K_{eq,2} = \frac{C_D}{C_B C_L^2} \\ C + L \Leftrightarrow D \quad & K_{eq,3} = \frac{C_D}{C_C C_L} \end{aligned} \tag{2}$$

The students were given the conditions:

$K_{eq,1}$ = 2.1e-14
$K_{eq,2}$ = 6.00e-10
$K_{eq,3}$ = 4.60e-09

C_{L0} = 90000 $cell^{-1}$
C_{A0} = 5000 $cell^{-1}$
C_{B0} = 15000 $cell^{-1}$
C_{C0} = 22000 $cell^{-1}$

Under these conditions, using the Newton-Raphson method, the students were to find the fractional occupancy of the three receptor populations (x_1, x_2, x_3), at equilibrium. For this problem, convergence was assumed if the change in the magnitude of x_i was ≤ 0.0001. The iteration was to be programmed into Excel by repeatedly solving the matrix set of equations (a while-loop construction) using the matrix solver (GaussJ) from lab 1.

Additionally, students were to plot the error (residual) versus iteration number in order to demonstrate quadratic convergence if the Newton-Raphson method is properly programmed. Students were required to clearly state the number of iterations required to reach convergence. In general, this lab is not difficult since the textbook provides a MATLAB script which students ported to Excel and modified for the existing problem.

D. Diffusion and Monte Carlo Simulation.

Monte Carlo is a powerful method that can be used for simulation or integration. As a statistically-based method, it is ideally suited to stochastic processes such as diffusion. Monte Carlo exercises in this course topic were purposely kept simple, such that the underlying principles could be well understood. All models were created by the students from scratch, using Excel Visual Basic scripts. Homework exercises introduced students to the concepts of random number generation and impact of number of simulation runs, using Monte Carlo techniques to approximate the area of a complex shape and the "Buffon's Needle" problem to converge on the value of pi. Subsequent homework, combined with laboratory demonstrations, allowed students to model a one-dimensional random walk process, an activated diffusion process, and a chemical gradient process, comparing them to observed physical phenomena. Little formal guidance was provided upfront, and students were expected, through the use of multiple plots of simulation results, to interpret the success and limitations of the Monte Carlo simulation under various simulation parameters.

E. Systems of Ordinary Differential Equations.

In this exercise, students were asked to develop the solution for the dynamics of membrane and nerve cell potentials. This problem was fully developed in the textbook, example 7.5 [2] for MATLAB using the intrinsic ODE solver ODE45. For Excel, there is no ODE45, so students were required to develop a function that would solve a system of ordinary differential equations using a fourth-order Runge-Kutta scheme. The beauty of this approach was that

students had a completely solved problem that they could verify their solutions against, and it gave them practice at porting code from one language to another as well as coding a solution scheme for ordinary differential equations. This also helped them to understand the importance of time-step size on the stability of the solution. Overall, at this point in the quarter, the students were getting weary of programming and debugging, but actually were quite accomplished and less dependent on the instructor for help with problems.

IV. Discussion

Having taught several courses in numerical analysis and modeling in both local and distance learning contexts, it has become increasingly clear – particularly for distance learning sites in a professional Master's program – that access to programming compilers is challenging as companies mete out resources in accordance with the needs of the engineers. MATLAB is a wonderful tool, but since MATLAB is not as ubiquitous as Excel, many of the off-site engineers complained that they had no access to the program and used this as an excuse for not getting the assignments done. There are student versions, of course, but the engineers may strictly not qualify for this license as they are not really full-time students and having the software on company computers may also violate licenses. Freeware compilers for FORTRAN or C are not robust and there are those who again used the excuse that they were unable to get the compiler functioning. This is where Excel was perfect. It is ubiquitous and every engineer and student was at least familiar with it (although most do not realize how capable an analysis tool it could be).

What was remarkable about this experience with Excel is the amount of positive feedback after the course stating how it is used with everyday work. One student wrote:

> ENGR 550 (*the numerical analysis course*) was one of the most valuable courses I've had at Cal Poly, due in large to the programming introduction I received. It was in some ways responsible for a summer employment with Olympus Microsystems America. The programming labs of ENGR 550 are what affected me greatly. Prior to taking ENGR 550, I had only enrolled in the mandatory programming course for my major (FORTRAN), from which I retained very little.
>
> While the learning curve was huge, learning VB was worth every frustrating moment. I fought through much of my programming fear and in doing so realized the value in grasping Visual Basic:
>
> - VB is a basic (pun intended) programming language. This is ideal for beginners because there are no shortcuts, and all the elements of coding are appreciated and better understood.
> - Virtually everyone can read and understand VB code because of its simplicity, so it's very common in industry. It provides an excellent foundation upon which C, C++, Java etc can be absorbed.

- VB is the basis for writing macros in Microsoft Office. Macros are particularly valuable in Excel to increase the efficiency of data handling.

- From ENGR 550, I received an excellent introduction to VB through the writing of Excel macros. It was this practical application of VB that cemented this skill into my brain. From this single class, I received enough introduction (sic) to write a complete VB program while working for Olympus Microsystems. This program communicates with a test stand to gather and analyze micro-mirror data used for telecommunications. The VB code communicates with motion controllers and data acquisition through the PC. Data is crunched in Excel with several macros written specifically for each sheet of the Excel file. All of my coding ability stems from the brief introduction I received in ENGR 550. VB is extremely powerful because it's so basic; this simplicity lends itself nicely to the classroom and because it's used by Microsoft Office, virtually every student already has Visual Basic at their fingertips for homework and lab assignments.

Another student wrote that the primary factor for obtaining an internship for the summer was the experience and fluency with Visual BASIC in Excel. Until now, no student has commented that they use FORTRAN or C for their work. This is a huge plus and we feel that more room should be made for integrating Excel as a numerical analysis tool.

V. Conclusions

Learning objectives were evaluated by laboratory performance, examinations, and student feedback. For students who took the class locally, the performance on laboratories and exams was not that different from year-to-year when using Excel, MATLAB, or FORTRAN as programming platforms. The comparison between local and distance learning students among the various years is confounded by the fact that different content was covered between distance and local sites because we accommodated the distance learning students. Using Excel provided the first time when the content in both modes was identical; which we believe is a success. Despite the difficulties with any past situation, the distance learning students have been extremely motivated in all cases (they are practicing engineers pursing a Master's degree); however, we believe that the accessibility to Excel as a numerical analysis tool provides for an equivalent experience. Statistically, there was no difference in performance between the local and distance learning student's final scores when we used Excel. This is the first time we can assert this because, as previously noted, we have had to modify the content for the distance site in the past. Furthermore, we have never received any feedback (positive or negative) previously from students stating that they are using the tools we taught them in this class. We also have evidence that students use Excel macros in homework from follow-on classes.

Visual BASIC macros are a powerful and under-utilized method for maximizing the use of Excel. It has enabled us to provide an equivalent laboratory experience in numerical analysis to a distance learning site when compared to more traditional tools, such as using MATLAB or C. Our primary outcome we have informally assessed from the class was that students have gained the ability to solve problems numerically and continue to use the Excel beyond the class experience. Also, since there was no statistically significant difference in grades between local and distance learning classes, we believe the experience is equivalent.

While Excel is only one of many data analysis tools that engineers routinely use, it does have limitations – particularly with respect to the size of the problem that can be tackled. We do not believe that Excel is the panacea for all numerical problems. This is particularly true when analyzing very large sets of data or approaching large-scale problems. Yet, for what Excel is, it can be an extremely useful analysis tool and can effectively be used to teach engineering students formal numerical analysis principles – especially when dealing with students in a distance learning situation.

VI. References

1. Dunn, S.M.; Constantinides, A., Moghe, P.V. (2006) Numerical Methods in Biomedical Engineering. Academic Press, Elsevier, NY.

2. Gotterfried, B.S. (2007) Spreadsheet Tools for Engineers Using Excel, 3rd Ed. McGraw Hill, NY.

3. Walpole, R.E.; Myers, R.H. (1993) Probability and Statistics for Engineers and Scientists, 5th Ed. Mcmillian, NY

Appendix: A Simple Linear Regression Tool Example.

Taken from the Excel Primer.

Given an (x,y) data set of arbitrary length, a line of the form:

$$y = a + bx \tag{1}$$

can be fit to the data using the method of least squares. It is not difficult to show that

$$a = \left(\sum_{i=1}^{n} y_i - b \sum_{i=1}^{n} x_i \right) \Big/ n \tag{2a}$$

$$b = \left(n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \right) \Big/ \left(n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right) \tag{2b}$$

Create a User Form

In the VB editor (ALT+F11 from the worksheet), insert a UserForm. This is done from the insert menu item in the Visual BASIC editor. This form will be the basis of our regression tool. Using the toolbox in the VB editor, add four (2) textboxes, two (2) command buttons, as shown in Figure 2. Label each using the caption and labels as appropriate. For the text boxes, change the names (*it is the first item in the properties window*) as follows:

Textbox1 → a This is the intercept
Textbox2 → b This will be the slope

Each of the textbox names will represent variables on the form that we can work with.

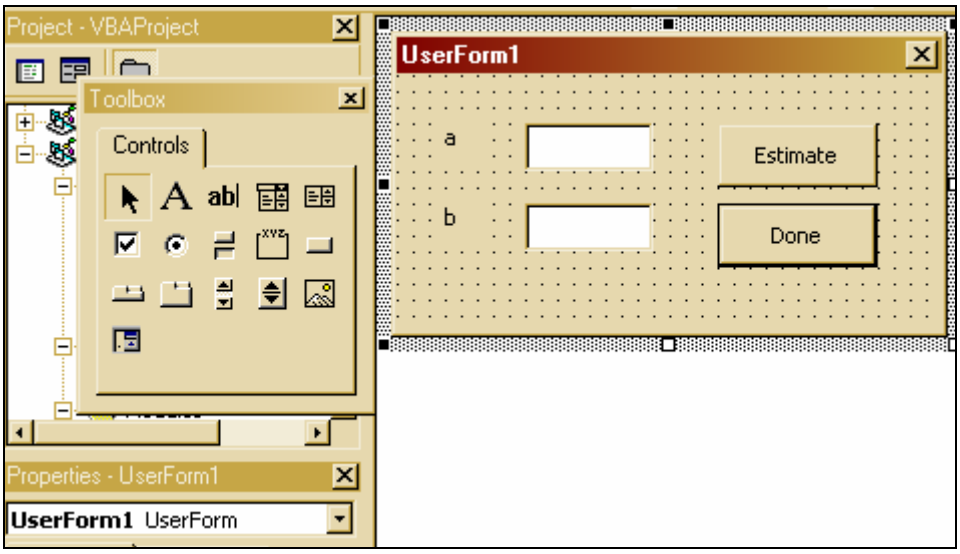

Figure 2: User form for a linear regression application.

We need to read in data from a set of cells on the worksheet. The reference on the worksheet will be x data begins in row 3, column 1 (cell A3), y data begins in row 3, column 2 (cell B3). As we read the data in, we will simply sum the data and perform the calculations as we go.

Set up a means to terminate the process (Done button)

Enter the code for the done button. Right click on the estimate button and select the view code option, and then fill in the code as shown. You are telling Excel that when this button is pressed, hide the form and return to the spreadsheet.

```
Private Sub CommandButton2_Click()
      Userform2.hide
End sub
'
```

Code the Equations

Enter the code for the estimation button. Right click on the estimate button and select the view code option.

```
Private Sub CommandButton1_Click()
'
'estimate the parameters a and b
'
i = 3   'This is a row identifier that says my data will begin
        'at row 3 on the worksheet.
```

```
'  Initialize the sums
'
sumx = 0.       '  Note that 0. becomes 0# which means double precision
Sumxsq = 0.
sumy = 0.
sumxy = 0.
'
' x - data on the worksheet begin in row 3, column 1.
' y - data on the worksheet begin in row 3, column 2.
'
'loop until no more data are found in the worksheet
While Worksheets("sheet1").Cells(i, 1).Value <> ""
    sumx = sumx + Worksheets("Sheet1").Cells(i, 1).Value
    Sumxsq = Sumxsq + Worksheets("Sheet1").Cells(i, 1).Value ^ 2
    sumy = sumy + Worksheets("Sheet1").Cells(i, 2).Value
    temp = 0.
    temp = Worksheets("Sheet1").Cells(i, 1).Value
    sumxy = sumxy + Worksheets("Sheet1").Cells(i, 2).Value * temp
    i = i + 1
Wend
n = i - 3    'The total number of data points are i - 3
'
'  Solve for b and a
'
b = (n * sumxy - sumx * sumy) / (n * Sumxsq - sumx ^ 2)                ' slope
a = (sumy - b * sumx)/n                                              ' intercept
'
end sub
```

Using the macro in Excel

Now assign the userform to an event on the spreadsheet: A button press. On the VB toolbar, (Fig 3) notice the wrench and hammer icon. This is the control toolbox. Add a command button to the spreadsheet by depressing the command button tool and using the mouse to construct the button on the worksheet.

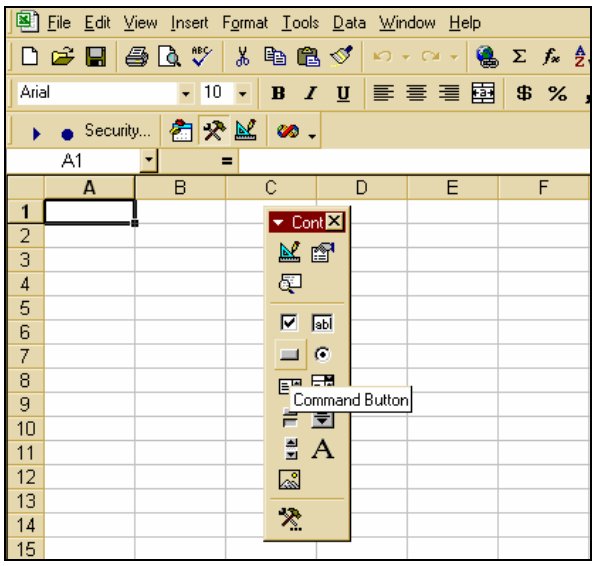

Figure 3: The control toolbox and the command button tool.

After you have added the command button to the spreadsheet, then place the cursor over the button and right click the mouse for the popup menu (Fig 4). Select view code as the option (Fig 4). You will be taken to the visual basic editor. In the view code option, type the code:

```
Private Sub Commandbutton1_Click()

      Userform1.show  <<Insert this line>>

End sub
```

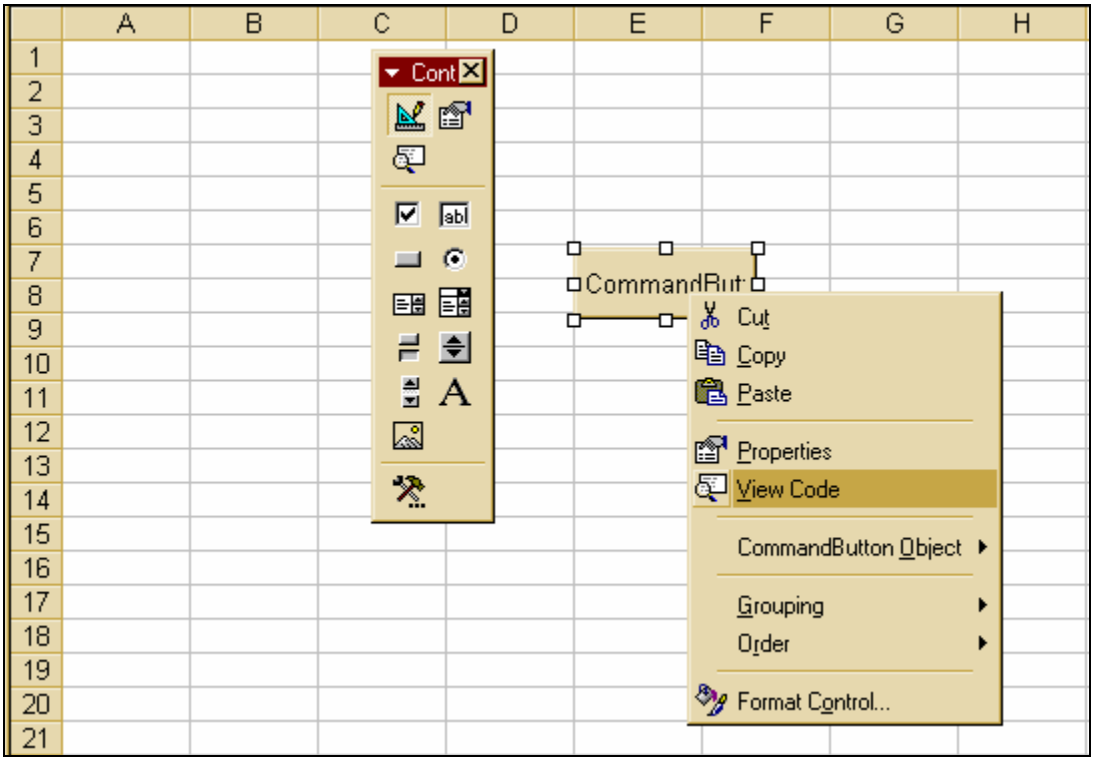

Figure 4: Assigning the subroutine to the button press event.

This is all that is required. Compare the results to those you can get in Excel already through the charts using the trendline.

While there are certainly more efficient ways to accomplish this simple program (it could easily be a function call), we chose to use the "forms interface" to familiarize students with an alternative method.