Please note that there are a some Mathcad and MATLAB solutions described at the end of this document

EXCEL PROGRAMS

Introduction

The spreadsheet program Excel is a very good general purpose calculating program that allows easy plotting of results. Many users are familiar with the cability of placing formulae in cells. Although this is useful capability, its utility becomes limited as the complexity of the task increases.

Many users know that Excel can record macros from a series of key strokes or mouse clicks. These actions are translated by Excel into a Visual Basic for Applications (hereafter VB) subroutine that can be called by the user. The next step is to learn some of the features of the VB language and write your own programs.

The major reason for the Excel programs that are being provided here is that almost anyone who has a computer also has Excel installed. The reader of the text can try out analysis of fluid power systems without needing to invest in other mathematical programs such as Mathcad or MATLAB. It is not suggested that Excel replaces such mathematical programs for serious design work, but a student can use Excel to gain familiarity with analytical concepts covered in the text before moving on to the more powerful (and expensive) tools.

Running VB Programs

A standard activity is generating a series of independent and dependent values and plotting these. The Excel program intro.xls in this directory starts with a blank spreadsheet, generates $y = x^{1.5}$ and $y = x^2$ for 20 numbers between zero and 20. Descriptive headings are also written to the worksheet. The data generated by the program are then written to the worksheet. A subroutine is then called that plots the data and alters the visual appearance of the plot somewhat from the Excel defaults. The authors feel that the altered appearance looks better when printed, but that is obviously a matter of opinion!

It is sometimes important to recognize what form of sheet is currently in use. There are *worksheets* that have matrices of cells and *charts* that contain graphics.

Users often set up Excel to perform autosaving, it is not possible to save to a CD, so the first move should be to copy intro.xls to a folder on the user's hard drive. The user should then open Excel and use the Open option in a standard Windows manner from the top menu bar.

Let us first use a shortcut approach for running the program embedded in intro.xls. Enter <ALT><F8>. A window will appear titled Macro. In this window, you will see intro in the white pane. Highlight the program name, if it is not already highlighted and then do one left click on the

Run button. The Macro window will close, and information will be written to the columns on the worksheet. After a modest display of pyrotechnics while Excel is changing the chart appearance, a chart should appear on the screen. Now repeat the <ALT><F8> sequence and this time click on the Edit button. The screen will change to the VB editor and the code will be displayed on the screen.

The previous approach was called a shortcut. The Macro window can also be reached from the top menu bar via the Tools entry.

This collection of programs cannot provide a detailed VB tutorial, but it is hoped that reading through the VB programs that have been provided will help the reader gain sufficient expertise that he/she can begin writing programs for themselves.

A further comment on the mechanics of VB. Microsoft decided that it would be easier for users if VB came to an abrupt halt when an error was encountered. A window will appear in which there are Help and Debug buttons. The first move should be to click on the Help button. It should be admitted that sometimes the Help messages are difficult to interpret, but clicking on Help first is generally desirable because you cannot go back to Help after clicking on Debug. On the other hand, clicking on Debug will highlight in yellow the line in which Excel thinks that there is an error. Assuming that you know how to fix the error, do so. Often Excel will inform you that it is Reset-ing your program. If you see this message, switch back to the worksheet window and try running the program again. On the other hand, if the line remains highlighted after making your correction, you must Reset the program manually. Reset-ing can only be done from the VB window. Go to the top menu bar and click on Run. The drop down menu will contain an entry Reset. Click on this and you are ready to continue debugging the program.

As with any other programming language, there are certain good practices:

- Decide on an indenting scheme so you can follow nested loops more easily.
- Excel is well provided with if-then-else, do loops, and for-next constructs. This means that you should use an absolute minimum of goto statements.
- It is possible to enter Breakpoints in the code in the VB window. Once the code stops at a breakpoint, put the cursor over a variable and the value will appear in a flag. Unfortunately this procedure does not work for arrays, but it is often possible to put in a dummy scalar variable equated to an array element and the scalar's value can be read.
- Work in segments. Once a block of code is working, consider putting the block in as a subroutine. The ideal is a main program that only makes a series of subroutine calls.
- Write your code to perform mathematical operations. If you need to do Excel type things, for example examine the subroutine plot_xy in intro.xls, record a macro and use the editor to transfer the needed parts.
- Use the Help menu to find the functions available in VB.
- If you cannot remember what a function does exactly, type the name in a line in the VB window, place the cursor somewhere in the middle of the word, and press <F1>. If you have spelled the command correctly, a window of information will appear.

- In a long program, it is often handy to stop prematurely when debugging or adding such items as message boxes. If you place the command End in the code, the run will stop at this line and exit from the program.
- Message boxes (msgbox) and input boxes (inputbox) are useful VB features. The message box simply displays a message on the screen and the input box allows the user to pass information into the program, for example a chart sheet name, without needing to write the information in a cell. Message boxes are often useful for debugging programs. Note that multiline messages can be written by incorporating a string variable, e.g., crlf, that has been defined as crlf = Chr(13) & Chr(10).

A Review of the Excel Programs

Chap4\ch4_ex2.xls

This program is the most complex program in the collection and is probably not the best program to start analyzing. The program simulates the operation of a walking beam feedback unit described in Ch. 4 Section 4.4.2. It turns out that this is a stiff system and cannot be integrated with a Runge Kutta 4th order adaptive stepsize routine. Two backward Euler routines, Bulirsch Stoer and Kaps Rentrop, translated from FORTRAN routines given in Numerical Recipes 2nd ed. are provided and the user can choose any one of the three routines. The RK4 routine fails because the stepsize becomes vanishingly small. The BS and KR routines are both capable of integrating the differential equation set, but you will notice that the results are not identical. The greatest discrepency is near the beginning of the simulation.

This program will load as a blank sheet. The user should run the program using <ALT><F8>. The program uses a variant variable to detect if column 1 is empty, if so the program writes a data set that is known to integrate satisfactorily. The user is requested to select an integration routine, it is suggested that KR is used initially. Data is written to the worksheet and graphs of displacement, velocity, and load pressure are produced. The user may try altering values in column 1 and rerunning the program. Observe that the plotting routine, plot_xy, obeys the Excel convention. The plotting routine senses if the named chart sheet is already present, if so, the plotting routines are bypassed. Excel automatically updates the curves on a chart if the data used to draw the curve changes.

The user may like to try the effect of lowering the bulk modulus values. Do so in stages or the integration process may be very slow.

The reader may wish to examine the subroutine derivs to see how the differential equation is structured in VB. This subroutine is relatively complicated because it attempts to account for the effect of the piston hitting the ends of the actuator, the effect of the servovalve switching from positive to negative displacement, and pressure reversal across an orifice. As an aside, computers are now so fast that it is probably better to introduce extra variables to clarify what is being done rather than attempting to reduce calculation effort by using very complicated equation formulations.

chap6\frq_rsp.xls

This program is not as user friendly as chap4\ch4_ex2.xls because subroutines to sense empty columns and write a sample data set are missing. The program generates amplitude ratio vs. frequency and phase angle vs. frequency (Bode plots) for the ratio of two polynomials. The user can delete the tabular data in columns 7 to 9 and the charts, but leave columns 1 to 6 intact. Rerunning the program will restore the data columns and the two charts. As before, the program can be run as many times as desired after the first time and the charts will update automatically. The number of polynomial coefficients and the values of the coefficients may be changed as desired.

The program introduces a new VB feature. A programmer can define his/her own VB functions. This capability has been used to define a complex variable, which is missing in VB. Several subroutines are provided that multiply complex numbers, find the quotient of two complex numbers, find the amplitude of a complex number and find the phase angle.

chap9\ptrans_smpl.xls

This program performs a very simplified analysis of the pressure experienced in the cylinder of an axial piston pump as it passes across top dead center. There is only one equation, that in pressure, so the problem can be solved using a non stiff Runge Kutta method.

The reader should examine the VB code in conjunction with Fig. 9.2 to see the meanings of the variables. Several figures in Ch. 9 were produced using this program. It is regarded as simplified because the flow conditions through the orifices were treated as sharp edged orifices. Undoubtedly in a real pump, the flow phenomenon would be much more complicated.

This program was written during the early stages of developing the book when the benefits of adaptive stepsize algorithms had not been fully appreciated. A fixed step size algorithm is easy to program, but may produce thousands of data sets. The reader may wish to examine the code to see how the data stream was sampled periodically to obtain a data set to write to the spreadsheet. The programming is admittedly clumsy, but it is fairly simple and easy to implement.

The program has a feature of dubious utility. The user can click anywhere on the spreadsheet and invoke <ALT><F8>. The data is then written starting at the selected cell. Because the user might wish to prepare several data sets with different input data, the user is asked to supply a name for the chart.

chap10\hydrsttrans.xls

This program simulates a soil bin being accelerated up to a constant speed, being held at constant speed during tool engagement, and being decelerated. The program is fairly user friendly and its structure resembles chap4\ch4_ex2.xls. There are routines for writing a sample data set and plotting is performed automatically. Although there are 3 state variables, the system was not stiff enough to cause trouble integrating with a fixed step size Runge Kutta.

The data provided are for the initial design using flexible hoses that had a large contained oil volume and a low effective bulk modulus. The reader may wish to try altering the oil volume and bulk modulus to see what improvement can be made in tracking the desired velocity conditions. This is an open loop system, i.e., no feedback.

Comments

None of the programs are for very complicated systems. We felt that this was a good philosophy

for a text designed primarily for senior andergraduates. On the other hand, the reader should be able to take the code for the formulation of the differential equations and extend this to many more state variables as the system being analyzed becomes more complex.

Choosing an integration algorithm may be a problem. Our limited experience suggests that the Kaps Rentrop algorithm may be the best general purpose algorithm for use with VB. The Gear algorithms may be more sophisticated and better able to handle more problems. Such algorithms are more difficult to program, which is why the Gear method does not appear in this text. It should be stated clearly that the object of introducing VB is to give the student experience formulating equations and seeing the simplifications that must be made. Any engineer working on fluid power problems for a living should graduate from VB to programs like MATLAB or Mathcad.

The authors should admit that they could not use either Bulirsch Stoer or Kaps Rentrop for the flow divider valve in Ch. 13. After much frustration, this problem was solved for the figures in this text using FORTRAN and ODEPACK.

MATHCAD AND MATLAB SOLUTIONS

chap11\section11_2.mcd

This file solves the pressure relief valve problem in Ch. 11 using Mathcad.

chap11\section11_2.mdl

This file is an alternative solution for the relief valve using MATLAB.

chap13\chap13.mcd

This file solves the flow divider valve problem in Ch. 13 using Mathcad.

chap14\chap14.mcd

This file solves the tractor pump attenuator problem in Ch. 14 using Mathcad.

chap14\chap14.mdl

This file is an alternative solution for the tractor pump attenuator problem using MATLAB.