A Guide to MATLAB
for Chemical Engineering Problem Solving
(ChE465 Kinetics and Reactor Design)

Kip D. Hauch
Dept. of Chemical Engineering
University of Washington

About this Manual

Matlab is a matrix-based mathematical software package that is used in several ChE classes including ChE465, Kinetics and Reactor Design, ChE480 Process Control& Laboratory, and ChE475 Computational Methods. It may also be useful in ChE310 as well as other ChE and other courses e.g. P-Chem. While Matlab is very powerful, many students often find it to be "unfriendly" and difficult to learn and understand; and frankly it is. This manual was compiled from several handouts that have been used previously in the above classes in an effort to make Matlab easier for you to understand and use. This manual demonstrates a select assortment of the common features and functions that you will use in your ChE classes. IT is NOT meant to be comprehensive, rather it is meant to supplement the published Matlab manual (*Student Matlab*, available at the UW Bookstore or with the purchase of the Student Matlab software.), and the on-line help available in Matlab (See p. 3) Another good reference is Engineering Problem Solving using Matlab, by D.M. Etter (Prentice Hall, 1993.)

This manual assumes that you are already familiar with the typical Macintosh operating system and the environment common to most Macintosh applications. Along with scalar variables, Matlab makes extensive use of vectors and matrices, and familiarity with the standard vector and matrix operations is very helpful in understanding how Matlab works.

This manual was compiled in Fall 1994 and includes material form Profs: Krieger-Brockett, Holt, Ricker, and Finlayson. If you find errors or wish to suggest changes or inclusions please contact your course instructor.

A Guide to MATLAB
for Chemical Engineering Problem Solving
(ChE465 Kinetics and Reactor Design)

I. General Introduction

What is MATLAB? (Matrix Laboratory), What is Simulink?

It is a powerful mathematical software package that you may use in solving some of the problems assigned in this course. MATLAB will likely be used again (more heavily) when you take ChE480 Process Control, and may also be helpful to you in other coursework or experimental work as well.

As with any software, it is only a tool that you may choose to apply to solve particular problems or tasks. It will not interpret problems for you; it will not guarantee that you get the 'right' answer. MATLAB IS only as smart (or as dumb) as the person using it. During your coursework you will encounter tasks such as numerical integration, and differential equation solving. MATLAB is not the only software tool that you may choose to apply to solve these tasks; other packages such as Mathematica, Maple V, Theorist, MathCAD and others may be adept at meeting your needs. In the future, as a fully employed process engineer you will be given certain mathematical tasks to solve, and you may be requested to adapt to using the software tools (and platforms) provided. At the UW we will make available the Macintosh version of Matlab for your use; but you should feel free to use other software tools or platforms if you are comfortable with them. We will, however, be unable to help you with other packages besides Matlab for Macintosh.

There are two easy ways to tell if a variable is a scalar, vector or matrix: 1) use the Who&Size command by typing `whos` *at the command line prompt, or 2) simply type the variable name and return. Matlab responds by displaying the variable and it's current value(s)*

Part of the power of Matlab comes from the fact that one can manipulate and operate on scalars, vectors and matrices with the same level of ease. However, therein lies one pitfall; the user must pay close attention to whether Matlab is assuming a particular variable to be a scalar, row vector, column vector, or matrix. Matlab does nothing to make this distinction immediately apparent.

Matlab also provides for a powerful high-level programming or scripting language. There exist hundreds of pre-written subroutines that accomplish simple to very high level mathematical manipulations, such as matrix inversion, ordinary differential equation solving, numerical integration, etc. In fact, most of the powerful commands that you invoke from within Matlab are actually separately written subroutines. You can (and will) write your own subroutines, as well as examine the ones the manufacturer has provided.

Simulink (previously known as Simulab) is a graphical interface for Matlab that links together blocks of complicated Matlab code to perform analysis, modeling, and simulation of dynamic systems. Simulink is used in the Process Control course for process control diagrams. At various times you may see Matlab referred to as: Matlab, Matlab/S, Matlab/Simulink, or just Simulink. Don't let this confuse you, in each case you are still using Matlab.

Where to use MATLAB? (Should I buy Student MATLAB?)

The Macintosh version of MATLAB is available for your use in Benson Hall Computing Lab, Room 125. This computer laboratory is for the use of students enrolled in ChE classes only; it is not open to the general campus. Our computer resources are limited, and the computer lab is reserved at certain times during the week for instructional use. Budget your time accordingly (i.e. plan ahead, work during non-peak hours). The MATLAB application

cannot be copied to your own machine.

The version of MATLAB available in the computing lab is a complete, full-featured

version of MATLAB (Matlab Professional vers. 4.2a). The publishers of Matlab have made available a somewhat limited version of the program, Student MATLAB, available for individual purchase at a reasonable cost. The biggest limitation is that the Student version is limited to working with variables (matrices) with less than 8K of elements (8192 elements or a 32 by 32 matrix). Student Matlab therefore, can handle only smaller problems, and may run more slowly. Also, some of the graphics and output routines may be more limited. It is likely that Student MATLAB will handle many, but not all of the problems you will want to tackle while here at UW ChE. As with any software I urge you to talk with other classmates who may have purchased Student MATLAB, and try the software for yourself. You will have to weigh many factors, such as the cost, the convenience to you of having your own copy, your own computer hardware and its performance, and the limitations of the Student version, before making your purchase decision.

(Student) MATLAB is also available on the MS-DOS platform as well as other workstation and mainframe platforms, however, you will be on your own regarding questions specific to these other platforms.

II. GETTING STARTED

HELP!!!
(Getting on-line Help)

MATLAB has simple and fairly extensive on-line help, although it is, at times, cryptic. You will be expected to **use** the on-line help to **first** learn about the syntax of a particular command or function, and to refresh your memory later. In this tutorial, you should first try to read through the on-line help for the applicable commands, then try the examples. If you are still stuck, re-read the on-line help, and then seek help from your instructor or TA.

On-line help is available by selecting About Matlab (or About Simulink) from the pulldown menu. Matlab also provides several demos here that you should explore.

On-line help is also available from the command prompt by simply typing:

```
» help function name
```

This is the easiest way to get help, and can be used at any time in the COMMAND window.

IMPORTANT STUFF ➨

Launching Matlab

All students are responsible for establishing an 'account' on the ChE UGrad Appleshare server, and abiding by the rules and regulations regarding the use of the computers and software. If you do not yet have such an account, or if you have forgotten how to use it, or if you have forgotten your password; go see the Department's Computer Engineer, Eric Mehan, in room B-007 immediately. The UGrad server provides you with access to a variety of applications including Word, Excel, DeltaGraphPro, as well as access to campus mainframes, e-mail etc. You are also provided with a small storage space on the server where you may store your own personal work files.

Never save your files to the Macintosh hard disk. Save your work frequently. Backup your work on floppy and take it with you.

THE FIRST TIME YOU LAUNCH MATLAB: Establish a connection to the UGrad server. COPY the file MATLAB from the Application Startup Documents folder on the Macintosh hard drive to your personal folder on the server. (Rename it Matlab Startup) You may now launch Matlab at any time by double clicking on this startup document in your folder. By launching Matlab in this manner, it will by default save your work files to your folder on the Server. After you have saved your work to your folder on the server, you may copy your files to a floppy for transport home, or just for use as a backup. You must pay careful attention to where Matlab is saving your files (which disk, server, directory, etc.). Matlab must be 'pointed' in the right direction, especially if you expect it to call a function or subroutine that you have written and saved in a particular location on the server. Also, you may lose your work if you accidently save to a folder or area to which you have no access. Most importantly: NEVER SAVE YOUR FILES ON THE MACINTOSH HARD DISK. As part of routine maintenance, the hard disks on the Macintoshs are frequently erased completely WITHOUT PRIOR WARNING.

The Workspace Environment Three types of Windows

+ TIP: Use the WINDOW *pull down menu to keep track of, and access open windows of all types.*

The Matlab environment provides three different types of windows: the COMMAND window, M-FILE editing windows and FIGURE windows. Each type of window is used for a different purpose, and it is important that you keep track of which window is your 'active' window. Use the WINDOW pull down menu to conveniently switch between any of the open windows. The startup document leads to an M-FILE window. You should simply close this window without saving any changes.

+ TIP: In the COMMAND Window, Use the Up arrow and Down arrow on the keyboard to scroll through your most recently issued commands.

In the COMMAND window, Matlab executes the commands on each line as you type them in at the command prompt, ». You will use this window to input values for variables and execute short series of commands. Matlab also displays most numerical results in this window. You may use the familiar Cut and Paste while in the COMMAND window as well as the mouse to perform editing.

Matlab outputs graphical data such as plots to a FIGURE window. A figure window will be created automatically when you issue a graphical output command, like `plot`. However, often the figure widow that is created is buried behind other windows. Plots can be copied and imported into other documents as graphics in the usual manner.

+ TIP: The first step in writing a script is to open a new M-file window.

Since typing even a handful of the same commands over and over again is tiresome, Matlab provides for powerful scripting of macros. The script file (called a M-file) is simply a list of commands. When the script file is executed, it is as if each of the commands was entered at the command prompt in the COMMAND window for you. The M-FILE Window is used to build, edit, and execute these scripts or programs. This window operates in the same manner as a simple text editor. Writing M-Files is discussed later in section IV.

Variables and Data entry

Once Matlab is launched you may begin defining variables at will. Each variable will remain stored in memory, with its assigned value until: it is reassigned a new value, it is manually cleared, or you quit Matlab. Although you can name variables almost anything, here are some tips. Matlab is case sensitive ('A' is not the same as 'a'). For this reason, you may find it more convenient to avoid using lots of capital letters. Stick to alphanumeric characters and the underbar. Keep your variable names short, but still long enough to be descriptive and easily distinguishable. (In scripts you should use comment lines to clearly spell out the meaning of the variables.) The default font used by Matlab is Monoco 12pt. In this font the capital letter 'Oh" and the Zero are identical: beware.

+ TIP: Matlab is case sensitive ('A' is not the same as 'a')

Assigning a scalar to a variable is straightforward:

```
»a = 5.348

a =

    5.3480

»
```

If you perform no other operations, Matlab responds by echoing back the variable with the value assigned.

Entering a vector or matrix is performed using a variable name and the square brackets. The individual elements may be separated by spaces or by commas. New rows may be indicated by returns or by semi-colons (;) within the brackets. Finally if no variable name is specified, Matlab assigns the input to the variable `ans` by default — you should avoid using `ans` as a variable name in your scripts.

Examples:

```
»a = [1 2 3]

a =

   1    2    3

»b = [1;2;3]

b =

   1
   2
   3
```

```
»[ 1 3 5
2 4 6
3 6 9]

ans =

   1    3    5
   2    4    6
   3    6    9
```

Exercise:

Input the following matrices::

$$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 3 & 8 \\ 2.3 & 5.6 & 10 \end{bmatrix}, \quad \begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 0 \\ 1 & 0 & 5 \end{bmatrix}$$

+ TIP: Assigning a range of values within a vector without typing each element.

There are several useful shortcuts for building more complex matrices. First the colon operator can be used to assign an evenly spaced range of values. The usage is:
[*starting value* : *increment* : *end value*]. If no increment is specified it is assumed to be one.

Example:

```
»time = [0: 0.1 :1.5]

time =

  Columns 1 through 7

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000

  Columns 8 through 14

    0.7000    0.8000    0.9000    1.0000    1.1000    1.2000    1.3000

  Columns 15 through 16

    1.4000    1.5000
```

+ TIP: Append a semi-colon (;) to the end of the line before the return to suppress this kind of lengthy output.

Individual elements or subsets of a matrix can be freely referred to by their indices
a(row, column).

Examples:

```
»a = [1 2 3; 4 5 6; 7 8 9]

a =

     1     2     3
     4     5     6
     7     8     9
```

```
»a(2,3)

ans =

     6

»a(1:2,3)

ans =

     3
     6
```

note the use of the colon operator to specify a range

```
»conc = [1 .9 .8 .7 .65 .63
2 1.9 1.8 1.6 1.5 1.43
2 .9 .85 .8 .75 .71]

conc =

    1.0000    0.9000    0.8000    0.7000    0.6500    0.6300
    2.0000    1.9000    1.8000    1.6000    1.5000    1.4300
    2.0000    0.9000    0.8500    0.8000    0.7500    0.7100
```

+ TIP: Extract a row or column from a data matrix

```
»conc(:,6)
```

says conc('all rows',column#6)

```
ans =

    0.6300
    1.4300
    0.7100
```

Finally, here are some special matrices that are often useful

```
»eye(3)
```

The identity matrix yields

```
ans =

     1     0     0
     0     1     0
     0     0     1
```

```
»ones(2,4)
```

Fills in a matrix of specified size with ones

```
ans =

     1     1     1     1
     1     1     1     1
```

```
»zeros(2,3)
```

likewise with zeros

```
ans =

     0     0     0
     0     0     0
```

Larger Matrices can be built from smaller ones.

Example:

```
a =

     1     2     3
     4     5     6
     7     8     9

»e = [[zeros(2,3);ones(1,3)] a]

e =

     0     0     0     1     2     3
     0     0     0     4     5     6
     1     1     1     7     8     9

»e = [e e]

e =

     0     0     0     1     2     3     0     0     0     1     2     3
     0     0     0     4     5     6     0     0     0     4     5     6
     1     1     1     7     8     9     1     1     1     7     8     9
```

Matrix Operations

```
»a = [1 2 3
4 5 6
7 8 9]

a =

     1     2     3
     4     5     6
     7     8     9


»a + a
ans =
     2     4     6
     8    10    12
    14    16    18

»a * a       matrix multiplication

ans =

    30    36    42
    66    81    96
   102   126   150
```

IMPORTANT ➨ *Placing a period in front of the operator causes it to be executed on a element-by-element basis.*

+ TIP: Pay close attention to whether your variables are row or column vectors

```
»a .* a

ans =

     1     4     9
    16    25    36
    49    64    81
```

There are two matrix division symbols in Matlab, / and \ . a/b = a*inv(b)
and a\b = inv(a)*b.

```
»a = [1 2 5
2 3 1
3 1 6]

a =

     1     2     5
     2     3     1
     3     1     6

»b = [1 1 5
4 1 2
6 4 1]

b =

     1     1     5
     4     1     2
```

```
     6     4     1

»a/b

ans =

    1.1569   -0.5686    0.3529
    0.3922   -0.9216    0.8824
    0.9216    0.7843   -0.1765

»a\b

ans =

    2.4722    1.1667   -1.6111
   -0.2500   -0.5000    1.5000
   -0.1944    0.1667    0.7222
```

And again, the element - by - element operator.

```
»a./b

ans =

    1.0000    2.0000    1.0000
    0.5000    3.0000    0.5000
    0.5000    0.2500    6.0000
```

The transpose is represented by the apostrophe.

```
»a = [1 2 3
4 5 6
7 8 9]

a =

     1     2     3
     4     5     6
     7     8     9

»a'

ans =

     1     4     7
     2     5     8
     3     6     9

»t = [0:8]

t =

 0  1  2  3  4  5  6  7  8

»t = t'

t =

     0
     1
     2
     3
     4
     5
     6
     7
     8
```

III. FUNCTIONS (log, exp, conv, roots)

Matlab is complete with a large number of useful, specialized, built-in functions. Descriptions of each function can be displayed using the on-line help. Here are some more commonly used functions:

*+ NOTE:: The natural logarithm is **log(x)** not ln(x)*

The natural logarithm in Matlab is performed using the command: `log(x)`

The base-10 logarithm is performed using the command

`log10(x)`, and the exponential is `exp(x)`.

Polynomial multiplication using convolve: (see also deconv)

What is:

$(3X^2 + 2X + 5) * (19X^2 - 7X - 13)$?

Solution:

`»a = [3 2 5]` (Put the polynomial coefficients into a

row vector in decreasing power)

```
a =

     3     2     5

»b = [19 -7 -13]

b =

    19    -7   -13

»conv(a,b)

ans =

    57    17    42   -61   -65
```

Answer:

$57X^4 + 17X^3 + 42X^2 - 61X - 65$

Exercise:

What is

$(3X^3 + 2X + 5) * (X^3 + 2X^2 - 2)$?

answer:

$3X^6 + 6X^5 + 2X^4 + 3X^3 + 10X^2 - 4X + 10.$

The roots of a polynomial can be found from its coefficients, e.g.:

What are the roots of:

$5X^2 + 17X + 6$?

```
»roots([5 17 6])

ans =

   -3.0000
   -0.4000
```

What are the roots of

$5X^2 + 6.5X + 19$?

```
»roots([5 6.5 19])

ans =

  -0.6500 + 1.8378i
  -0.6500 - 1.8378i
```

The roots are complex.

IV. MATLAB SCRIPTS AND FUNCTION FILES (M-FILES)

+ *NOTE: Both script files and function files MUST have the file extension .m appended to the filename.*

+ *TIP: Be careful! While fiddling with small changes in a script, it is all too easy to overwrite a script that you wanted to keep with one that you don't. Save your changes to a separate file, with a unique name first. THEN use the SAVE&EXECUTE command.*

Matlab Scripts

Matlab scripts, also known as macros, programs, code, M-files, are simply collections of commands. The code is constructed in the M-FILE editing window, which operates just like a text editor. In fact, an M-file is just a simple ASCII text file, and can be created and edited by any simple text editor, although, it is probably easier to use the editor in Matlab. Each line should be entered just as if you were to enter it at the command prompt in the COMMAND window. When you have finished editing, save your work, IN YOUR FOLDER OR ON YOUR DISK, as a M-file. In order to be recognized by Matlab as a valid M-file it MUST have the file extension .m appended to the file name.

To actually execute your code, use the Save and Execute command under the FILE pull down menu (E is the keyboard equivalent). Note that this command first saves your file to disk, overwriting the previous version of your script of that particular name!,(without even asking first!) It then runs the code.

Another important tool in writing Matlab scripts is the use of comment lines. Matlab will ignore all characters after the percent sign (%) on a given line. It is impossible for others to evaluate and modify your code if they can't understand what your variables stand for and what steps your code performs.

In order to receive full credit, any homework solution, solved using Matlab or any other computer code MUST include a printout of the code used. Comment lines should be used to provide definitions for all the variables used, and the appropriate units.

Example: Start with a fresh M-file editing window. Write a script to convert the temperature in Celsius into °F and then into °R for every multiple of 15 from 0-100°C. Combine the three results into one matrix and display.

```
% tc is temperature Celsius, tf is temp deg F,
% and tr is temp deg Rankin.
tc = [0:15:100];
tf = 1.8.*tc + 32;
tr = tf + 459.69;
% combine answer into one matrix
t = [tc;tf;tr]
```

Function files

Function files are a special kind of script file (M-file) that allow you to define your own functions for use during a Matlab session. You might think of them as subroutines that can be called from within a script, or even called directly from the command line. Many of the "built-in" functions in Matlab are actually stored as M-files as part of the Matlab package. Function files are created and edited in identically the same manner as the script files above, however they differ in two important ways.

1) When a function file is called and executed, certain arguments are passed from the main script to the function; thereafter, the variables defined and manipulated in the function file exist only temporarily, and they are deleted after the function returns its result.

2) The first line of a function file is special. The first word of the first line must be
`function` and this is followed by a statement of the output arguments and input arguments (in terms of the "local" or function variables) and function name:

```
function  outputarg = functnam(inputarg1, inputarg2, ... )
```

Next comes the code for the function, i.e. expressions that define the outputarg as mathematical expressions of the inputargs. In many cases these expressions will be matrix expressions, and the arguments passed matrices.

Save the function with the same file name as the functnam, (and with the proper extension *.m*). As long as the M-file file that defines the function exists, with the proper name, saved to your folder, Matlab can refer to it as a new function. You can then invoke it as a function from the command line, or from within a larger Matlab script.

+ TIP: To see what user-defined functions are currently available for Matlab to use, enter the command what *at the command prompt.*

Example:

```
function y = tconvert(x)
% this function converts temperature in deg C to  deg F
y =  1.8*x + 32;
```

+ TIP: Be careful when choosing names for your scripts and functions. The name should begin with an alpha character, and should NOT include other things such as: spaces, #, - / or other such characters. You can use the under bar _.

These three lines are saved as a function file named *tconvert.m* Note that when you go to save the file, Matlab has already peaked inside the file, and noticed that this is a function, and prompted you with the correct filename!

Now at the command prompt in the COMMAND Window:

```
»tc = [0:10:100]

tc =

    0   10   20   30   40   50   60   70   80   90  100

»tconvert(tc)

ans =

   32   50   68   86  104  122  140  158  176  194  212
```

We have now created a new built-in function.

More script writing hints:

The variables you create at the command prompt and in scripts (and their values) will stick around in memory long after you have run your macro. Sometimes it is useful to include a line at the very beginning of the macro that clears all the existing variables from the workspace, so there is no confusion. Just include the line:

```
clear
```

+ Common Mistake: Be sure to save any changes to scripts and functions before trying to use them!!!

at the beginning. Sometimes when editing several open windows at the same time, one can lose track of which changes have been saved, and which have not. Under the Window pulldown menu, the titles of the windows that have been edited, but NOT saved will appear underlined. If your script includes commands to create more than one graph than you will need to include the command pause after the plot commands so that you are given an opprtunity to actually view your graph before the script moves on and the graph is cleared.

V. PROBLEM SOLVING

USING SCRIPTS, USER-DEFINED FUNCTIONS AND BUILT-IN FUNCTIONS TO PERFORM CURVE FITTING, NUMERICAL INTEGRATION, ALGEBRAIC, AND DIFFERENTIAL EQUATION SOLVING: (POLYFIT, POLYVAL, POLYDER, QUAD, FSOLVE, ODE45)

This section presents some common problem solving examples. Often we will want to use our new user-defined functions in other Matlab scripts or built-in functions.

Polynomial Curve fitting, taking a derivative

Matlab has three related functions (`polyfit, polyval and polyder`) that are particularly useful for: fitting data to a polynomial curve (of specified order), evaluating a given polynomial over a specified rage of independent variable, and taking the derivative of a given polynomial.

Example:

Fit the following data describing the accumulation of species A over time to a second order polynomial. Using this polynomial, predict the accumulation over the range of 20 - 30 hours. Finally, calculate the time derivative of the accumulation over the period 0-10 hours.

Mass of A accumulated as a function of time:

Mass of A accumulated (arbitrary units)	Time (hours)
9	1
55	3
141	5
267	7
345	8
531	10

Solution:
First, input the data into vectors, let:

```
»a = [9 55 141 267 345 531]

a =

     9    55   141   267   345   531

»time = [1 3 5 7 8 10]

time =

     1     3     5     7     8    10
```

+ *TIP:Pay close attention to which variable is the independent and the dependent variable*

Now fit the data using
`polyfit(independent_variable,dependent_variable, polynomial order)`

```
»coeff = polyfit(time,a,2)

coeff =

    5.0000    3.0000    1.0000
```

So, Mass A = 5*(time)^2 + 3 * (time) + 1.

The coefficients of the polynomial fit are now stored in the row vector coeff. To evaluate this polynomial over the range of 20-30 hours we define a new time vector (the independent variable)

```
»newtime = [20:30]

newtime =

    20    21    22    23    24    25    26    27    28    29    30
```

Use the function polyval(coefficient_vector,independent variable) to evaluate this polynomial over this new range and store the result in the vector pred.

```
»pred = polyval(coeff,newtime)

pred =

   1.0e+03 *

  Columns 1 through 7

    2.0610    2.2690    2.4870    2.7150    2.9530    3.2010    3.4590

  Columns 8 through 11

    3.7270    4.0050    4.2930    4.5910
```

Next use the function polyder(coeff) to determine the coefficients of a new polynomial that is the derivative of the original polynomial. Store these new derivative coefficients in the vector dervcoef

```
»dervcoef = polyder(coeff)

dervcoef =

   10.0000    3.0000
```

Again, use the function polyval to evaluate this derivative polynomial over this desired range and store the result in the vector dervpred.

```
»dervpred = polyval(dervcoef,[0:10])

dervpred =

  Columns 1 through 7

    3.0000   13.0000   23.0000   33.0000   43.0000   53.0000   63.0000

  Columns 8 through 11

   73.0000   83.0000   93.0000  103.0000
```

Misc. Hints

If the data were collected as a function of time at REGULAR intervals, then use the colon operator to create an evenly spaced "time" vector, t:

```
»t = [0:1:10]

t =

     0     1     2     3     4     5     6     7     8     9    10
```

for data collected one per second for 10 seconds.

+ TIP: Use the trailing semi-colon to suppress lengthy output

There are times when it is distracting for Matlab to echo back the entire vector or matrix. For example if we had collected data once per second for 1 hour. Placing a semicolon after the expression and before the return suppresses the output.

```
»time = [0:1:3600];
»
```

To list all the variables currently in use in the workspace, use the command Who&Size by typing whos at the command prompt

```
» whos
                Name        Size            Total     Complex

                   T       1 by 5              5        No
                   a       1 by 6              6        No
                 ans       1 by 3              3        No
                   b       1 by 3              3        No
                conc       3 by 6             18        No
                   e       3 by 12            36        No
                   k       1 by 5              5        No
                   t       1 by 11            11        No
                time       1 by 3601        3601        No

Grand total is (3688 * 8) = 29504 bytes,

leaving 265296 bytes of memory free.
```

Numerical Integration

The function `quad('myfunct',a,b)` integrates the value of the function defined in `myfunct` over the range `a` to `b`. Note the function name is put in single quotes.

Example:

Given an estimate for the yearly U.S. deficit as a function of population,
and projections for the population growth over the next ten years, calculate the total nation debt amassed over the next five decades.

Given: yearly deficit ($) = $0.01*(\text{population})^2 + 2000 * (\text{population}) + 50$
and, population (millions) = 250*exp(t/25), where t (years).

Solution:

define a function called `deficit` and save it.

```
function  y = deficit(time)
% calculate the population for given time in millions
pop = 250*(exp(time/25));
% convert
pop = pop*1e6;
% calculate the deficit per year for given t in $/year
y = 0.01*(pop).^2 + 2000*(pop) + 50;
```

now write a Matlab script to integrate this function over 0-50 years.

```
% integrate deficit over specified time period
% tinit = initial time, tfin = final time
tinit = 0
tfin = 50
% integrate using quadrature debt in $
debt = quad('deficit',tinit,tfin)
```

Answer:

```
debt =

   4.1882e+17
```

Let's hope this is a fictitious example indeed.

Exercise:

Let the coefficients in the deficit equation be specified in a vector k = [x1 , x2 , x3]. Rewrite the code to solve for the debt.

Solving simultaneous algebraic equations (fsolve)

+ *TIP: To ABORT a lengthy script or function use Control-C.*

Example:

Find the solution for a,b,c that satisfies the following set of algebraic equations:

$$5a + 6 = 0$$
$$3a + 4b + 7 = 0$$
$$4b + c + 3 = 0$$

Solution:

First, let S be a vector such that s(1) = a, s(2) = b and s(3) = c
Now describe the set of equations as one matrix function

```
function f = myfunct(s)

f(1) = 5*s(1) + 6;
f(2) = 3*(s(1) + 4*s(2) + 7;
f(3) = 4*s(2) + s(3) + 3;
```

and save it. Note that the output argument of this equation is zero.
Next use `fsolve('functnam',guess)`, where `functnam` is the name of the function and `guess` is an appropriately sized vector of initial guesses for s.

```
»guess = [1 1 1]
```

```
guess =

     1     1     1

»fsolve('myfunct',guess)

ans =

   -1.2000   -0.8500    0.4000
```

So, a = -1.2, b = -0.85 and c = 0.4.
`fzero` can also be used to find a zero of a function of one variable.

Solution to (sets of) Ordinary Differential Equation (ode45)

Here is an example using the power of an ODE solver. Consider the system of reactions in a constant volume, constant temperature batch reactor.

A —–> D reaction 1
A + A ——> U reaction 2

where D is a desired product, U is undesired product and where k1 and k2 are the rate constants for reactions 1 and 2. Let k1, k2 be given parameters, and the initial concentration of A (ca0) be a design variable. The independent variable is time (tfin), and the dependent variables are the concentration of the species, ca, cd, and cu. Note that in most design situations k1 and k2 might be design parameters, adjusted via the temperature.

By writing the mass balance equations over the batch reactor, the system of differential equations is the following: they are not linearly independent.

$d(ca)/dt = -k1(ca) - k2(ca)^2$

$d(cd)/dt = k1\ (ca)$

$d(cu)/dt = k2\ (ca)^2$

Solving this problem in Matlab involves two parts. First, write a function file that describes the set of ODEs in terms of a single, combined matrix variable (the dependent variable). Next, in a second, (main) script invoke the ode solver, ode45. This script that might include other things like the initial conditions and other given parameters. (The solution to a single ODE is analogous, but the dependent variable is not a matrix).

First the function file. In this example, let C be a three column matrix, where C(1) is really ca, C(2) is cd and C(3) is cu, and t is the independent variable, time.

+ TIP: It is useful to have some convention for naming the ODE containing function files and the main scripts. Here the letters ode appear in the functionfile name while the script name is descriptive of the particular problem being worked.

```
function dC_dt = exampleode(t,C)
global k1 k2 % variables that we wish to share with the main script
dC_dt(1) = -k1*C(1) - k2*C(1)*C(1);
dC_dt(2) = k1*C(1);
dC_dt(3) = k2*C(1)*C(1);
```

Save this as a function in a function file called *exampleode.m*

Now write the main script. Start with a fresh M-file editing window.

```
clear
% Batch reactor, multiple reaction and multiple species
% requires the odes be in function 'exampleode'

% define as global any variables used by any
% and all subroutines, functions

global k1 k2

% Set parameters,
k1 = 2; k2 = 1;
ca0 = 2; % ca0 = initial concentration of species A
tfin = 1/3;
% all parameters in arbitrary units for purposes of demo
% t is time, tfin is final time,
% c will be a matrix with three columns, one for each species.
% each row will be for another time point.

% initial conditions (c0) is a vector of three initial conds.

c0(1) = ca0 ; c0(2) = 0 ; c0(3) = 0;

% integrate ODE's from 0 to tfin
% the equations are specified in the function 'exampleode'
% the time parameters are set, and the initial conditions
% are in the vector c0

[t,c] = ode45('exampleode',0,tfin,c0);

% concentrations of the three species as function of time, each is a
% column vector
ca = c(:,1)
cd = c(:,2)
cu = c(:,3)

% like to know the size of the result matrix c
last = size(c)

% extract the final value of the species out of c

caf = c(last(1),1)
cdf = c(last(1),2)
cuf = c(last(1),3)
```

Now save this as our script (any name, e.g. dualrxnprob), and execute

```
                    1.9793          1.4545
ca =                1.8248          1.3549
                    1.6875          1.2647
   2.0000           1.5648          1.1825
```

1.1075		
1.0389	cd =	cu =
0.9758		
0.9177	0	0
0.8641	0.0104	0.0103
0.8144	0.0896	0.0856
0.7684	0.1627	0.1498
0.7257	0.2304	0.2048
0.6907	0.2932	0.2523
	0.3517	0.2934
	0.4063	0.3291
	0.4572	0.3602
	0.5049	0.3875
	0.5496	0.4115
	0.5916	0.4326
	0.6310	0.4513
	0.6681	0.4678
	0.7031	0.4825
	0.7360	0.4955
	0.7671	0.5072
	0.7930	0.5163

(*this example shown in three columns just to save space)*

```
last =

    18     3


caf =

    0.6907


cdf =

    0.7930


cuf =

    0.5163
```

These are the final values of the species.

VI. INPUT AND OUTPUT IN MATLAB (INPUT, LOAD, PLOT, SUBPLOT, LABELS AND TITLES)

Getting a solution to the problem, isn't the end; you still have to present the results in an intelligible manner. The input and output routines in Matlab are perhaps the most likely to be different on different platforms, and are the most likely to change in the future.

Input

Unless you have more than 15-20 data points in a data vector, or expect to have to re-enter significantly different data over and over again, the best

solution is to enter data by specifying it directly in a variable assignment inside a script.

conversion = [0 0.23 0.25 0.27 0.45 0.78 0.79 0.81 0.91]

If you are investigating parameter sensitivity, it might be helpful to include a prompt for input from the keyboard as part of a script. For example:

```
k1 = input('Please input the rate constant, k1   ')
```

If you must import data, Matlab can be used to import tab-delimited text files into variables (vectors, matrices). See the help for the function *load.* Again pay attention to row versus column vectors.

Output

Many of the results you will generate can be displayed directly in the COMMAND window, and either printed directly or copied to your favorite word processor. The remainder, of course are graphs that are plotted in the FIGURE window. First, the FIGURE window is like the variables in the Workspace, it does not clear until you tell it too. If you don't clear, it will just plot over the previous graphs. The command to clear the FIGURE window is *clg*, and should be included in your macro code prior to the graphic commands.

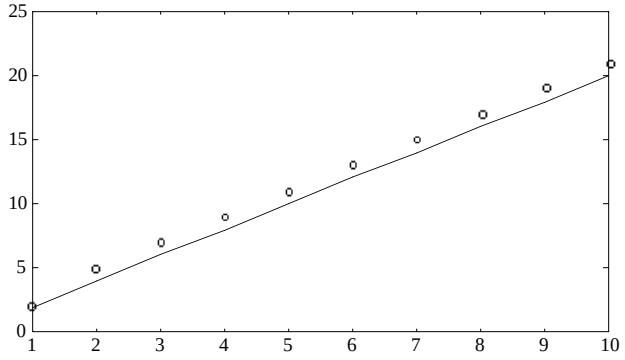
The syntax for the plot command is:

`plot (x,y)` , where x and y are vectors with the x data points and the y data points.

+ NOTE: Plotting two or more dependent variables against the same independent variable.

Where there are two sets of dependent data to plot against the same independent data set,
the form must still be of pairs of x and y data. To use a special symbol for plotting, tack on a 'o' to the plot command after the data pair. (see on-line help for details about plot symbols)

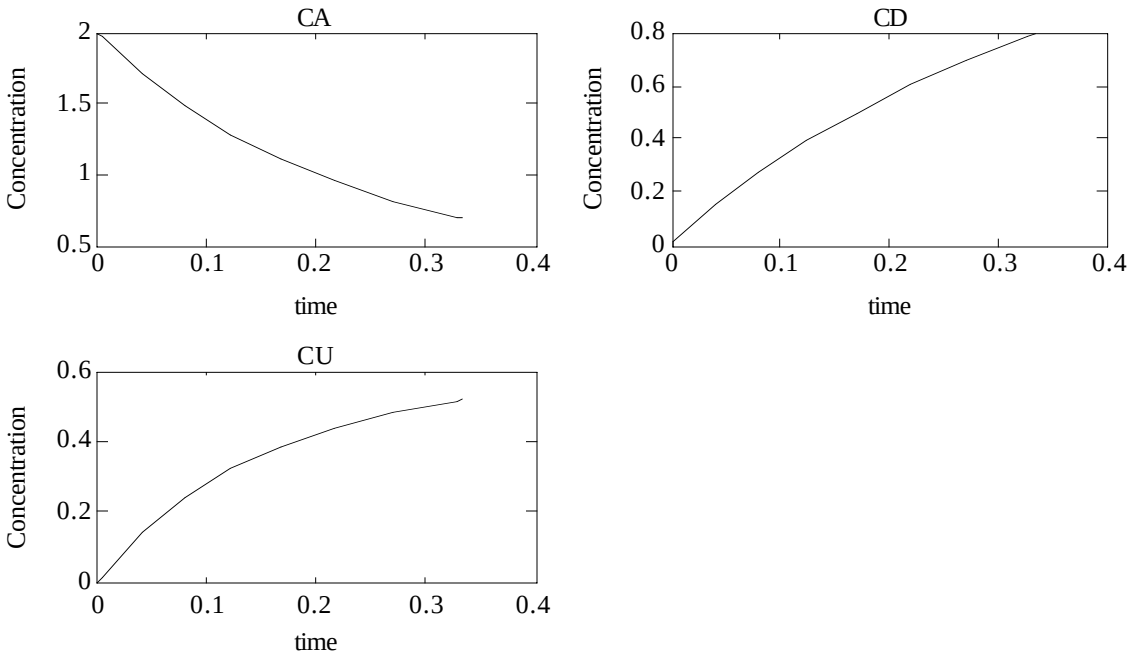
```
»t = [1:10];
»y1 = [2 4 6 8 10 12 14 16 18 20];
»y2 = [2 5 7 9 11 13 15 17 19 21];
»plot (t,y1,t,y2,'o')  % note t is repeated.
```



Matlab can provide one plot in the window, or stack two atop one another, or split the GRAPH window into quadrants and plot four graphs.

Let's return to the ODE example and plot the other data we generated.

```
clg
subplot(221) % splits into four plots, selects plot 1 to draw in
plot (t,c(:,1))  % plots all rows column 1 of matrix c
title('CA'), xlabel('time'), ylabel('Concentration')
subplot (222)  % advance to plot 2 and repeat
plot (t,c(:,2))  % plots all rows column 2 of matrix c
title('CD'), xlabel('time'), ylabel('Concentration')
subplot(223)
plot (t,c(:,3))  % plots all rows column 3 of matrix c
title('CU'), xlabel('time'), ylabel('Concentration')
```



You can copy and paste these graphs into your favorite word processing program

+ TIP: Pay attention to axis limits and graph size. Keep it sensible, and make it clear.

Finally, Matlab will normally autoscale the axis for you, but there may be times, in which you wish to draw attention to a particular feature of the curve that requires that you to override these pre-set axis limits. You may do this under the Graph pulldown menu, or in the code with the command Axis(V) where v is a vector containing your specified x and y min and max (see *Axis*). Consider the resolution of your output device —if the trend or result you wish to show is not evident after printing at reduced size on the crude Imagewriter, than you may not get full points for your work.

Exporting Data as a Tab-delimited text file.

Matlab is incapable of creating double-y axis plots. If you need this type of plot (for example to present and compare two curves along the same independent variable when the absolute value of the data in the two curves differs greatly), or if you prefer to create graphs using another program such as DeltaGraph Pro, Kaleidagraph or Excel, then consider using this technique for exporting your results as a tab-delimited file.

First, as part of your script, create a new matrix, result that comprises all the data you wish to export. For the above example that would be t, the independent variable, and c our solution matrix.

`result = [t c];` (keep track of which columns are which)

Next include the following line in the script to prompt for a filename for your stored data.

```
filename = input('filename please?   ', 's')
```

And finally, include this line *exactly as typed here* as the last line in your script

```
eval(['save ', filename, ' result /ascii /tabs'])
```

VII. SIMULINK

ChE480 Process Control and Laboratory makes use of several Matlab functions and features The functions `impulse( )` and `step( )` calculate and plot the response of a dynamic system to an impulse and step input. The user defines the system by specifying the polynomials that describe the numerator and denominator of the system's transfer function.

Simulink is a more advanced and powerful graphical interface for simulating more complicated dynamic systems built from various system transfer function blocks. Simulink is invoked at the Matlab command prompt by typing `simulink` and has its own windowed interface along with controls for starting and stopping the simulation and viewing and printing the results.