

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	1
I. General Introduction.....	2
<i>What is Matlab? (Matrix Laboratory), What is Simulink?</i>	2
<i>Where to use Matlab? (Should I buy Student Matlab?)</i>	2
II. Getting Started.....	3
<i>HELP!!!</i>	3
<i>Launching Matlab</i>	3
<i>The Workspace Environment Three types of Windows</i>	4
<i>Variables and Data entry</i>	4
<i>Matrix Operations</i>	7
III. Functions (log, exp, conv, roots).....	8
IV. Matlab Scripts and function files (M-files).....	10
<i>Matlab Scripts</i>	10
<i>Function files</i>	10
<i>More script writing hints</i>	
V. Problem Solving.....	11
<i>Polynomial Curve fitting, taking a derivative</i>	12
<i>Misc. Hints</i>	13
<i>Numerical Integration</i>	14
<i>Solving simultaneous algebraic equations (fsolve)</i>	15
<i>Solution to (sets of) Ordinary Differential Equation (ode45)</i>	16
VI. Input and Output in Matlab.....	18
<i>Input</i>	18
<i>Output</i> 18	
<i>Exporting Data as a Tab-delimited text file</i>	20
VII. Simulink.....	21

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 from 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 accidentally 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 Macintoshes 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.

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

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 window that is created is buried behind other windows. Plots can be copied and imported into other documents as graphics in the usual manner.

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

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

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 Monaco 12pt. In this font the capital letter 'O' and the Zero are identical: beware.

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::

```
[ 1 2 3
5 3 8
2.3 5.6 10 ] , [ 1 2 3
1 0 0
1 0 5 ]
```

+ *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
```

```
»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
```

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*

```

        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

$\log_{10}(x)$, 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)

Matlab Scripts

+ *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, 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.

Example:

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

These three lines are saved as a function file named *tconvert.m*. Note that when you go to save the file, Matlab has already peeked 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
```

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 opportunity to actually view your graph before the script moves on and the graph is cleared.

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

+ *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 underbar `_`.*

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

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 range 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:

<u>Mass of A accumulated (arbitrary units)</u>	<u>Time (hours)</u>
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(coeff, 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:

$$\begin{aligned} 5a + 6 &= 0 \\ 3a + 4b + 7 &= 0 \\ 4b + c + 3 &= 0 \end{aligned}$$

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

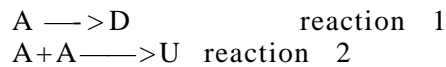
```

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.



where D is a desired product, U is undesired product and where k_1 and k_2 are the rate constants for reactions 1 and 2. Let k_1, k_2 be given parameters, and the initial concentration of A (ca_0) be a design variable. The independent variable is time (t_{fin}), and the dependent variables are the concentration of the species, ca , cd , and cu . Note that in most design situations k_1 and k_2 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.

$$d(ca)/dt = -k_1(ca) - k_2(ca)^2$$

$$d(cd)/dt = k_1 (ca)$$

$$d(cu)/dt = k_2 (ca)^2$$

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 *clf*, and should be included in your macro code prior to the graphic commands.

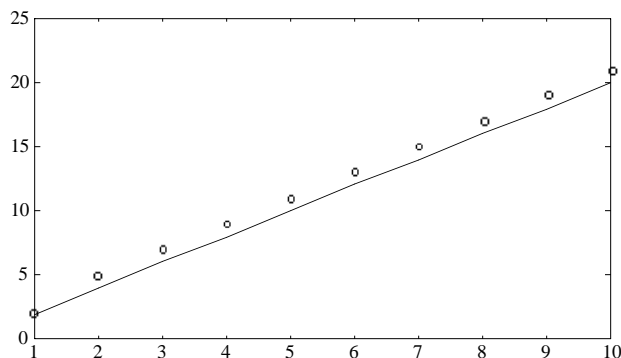
The syntax for the plot command is:

`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.
```



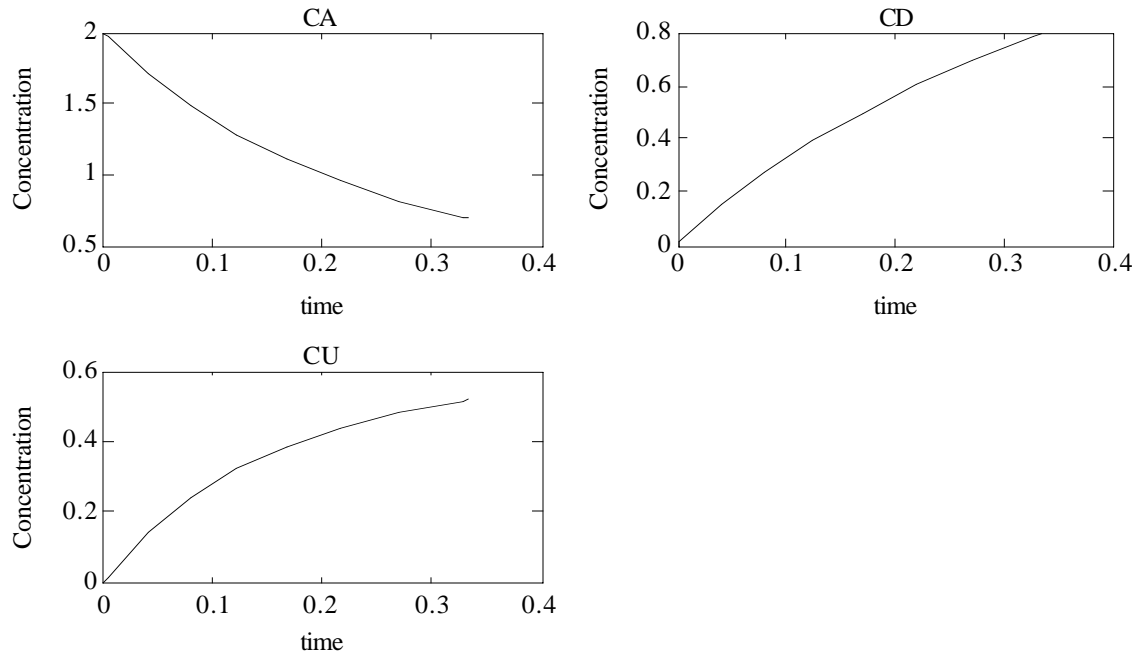
Matlab can provide one plot in the window, or stack two atop one another, or split the GRAPH window into quadrants and plot four graphs.

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 `impz` 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.

MATLAB 입문

CHAPTER 10

Symbolic processing with MATLAB

OUTLINE

- **Symbolic expressions and algebra(10)**
- **Algebraic and transcendental equations(3)**
- **Calculus (9)**
- **Differential Equations (6)**
- **Laplace Transforms (2)**

Symbolic expressions and algebra(1/10)

■ The *sym* function

```
Ex >>x = sym ('x')
    >>x = sym ('x', 'real')
    >>syms x y u v
    >>pi=sym('pi')
    >>fraction=sym('1/3')
    >>sqrt2=sym('sqrt(2)')
```

■ Symbolic expressions

```
ex >>syms x y
    >>s = x + y
    >>r = sqrt(x^2+y^2)

ex >>symbolic matrix A
    >>n=3
    >>syms x;
    >>A=x.^((0:n)^(0:n))
```

```
A=
    [1,1,1,1]
    [1,x,x^2,x^3]
    [1,x^2,x^4,x^6]
    [1,x^3,x^6,x^9]
```

Symbolic expressions and algebra(2/10)

■ Manipulating expressions

- collect (E)

```
ex >>syms x y
>>E=(x-5)^2+(y-3)^2;
>>collect (E)

ans =
      x^2-10*x+25+(y-3)^2

>>collect(E,y)
ans =
      y^2-6*y+(x-5)^2+9
```

- expand (E)

```
ex >>syms x y
>>expand((x+y)^2)
                                % applies algebra rules
ans =
      x^2+2*x*y+y^2

>>expand(sin(x+y))
                                % applies trig identities
ans =

>>sin(x)*cos(y)+cos(x)*sin(y)

>>simplify(6*((sin(x))^2+(cos(x))^2))
                                % applies another trig identity
ans =
      6
```

Symbolic expressions and algebra(3/10)

□ factor (E)

```
ex >>syms x y
    >>factor(x^2-1)
ans =
      (x-1)*(x+1)
```

□ simplify (E)

```
ex >>syms x y
    >>simplify(x*sqrt(x^8*y^2))
ans =
      x*(x^8*y^2)^(1/2)
```

□ simple (E)

```
[r, how] = simple (E)
```

```
Ex> >>syms x y
    >>E1=x^2+5; % define two expressions
    >>E2=y^3-2;
    >>E3=x^3+2*x^2+5*x+10; % define a third
    expressions
    >>S1=E1+E2 % add the expressions
S1 =
      x^2+3+y^3
    >>S2=E1*E2 % multiply the expressions
S2 =
      (x^2+5)*(y^3-2)
    >>expand(S2) % expand the product
ans =
      x^2*y^3-2*x^2+5*y^3-10
    >>S3=E3/E1 % divide two expressions
S3 =
      (x^3+2*x^2+5*x+10)/(x^2+5)
    >>simplify(S3) % see if some terms cancel
ans =
      x+2
```

Symbolic expressions and algebra(4/10)

□ [num den] = numden (E)

ex >> syms x

>> E1=x^2+5;

>> E4=1/(x+6);

>> [num, den]=numden(E1+E4)

num =

$x^3+6x^2+5x+31$

den =

$x+6$

□ double (E)

ex >> sqrt2=sym('sqrt(2)');

>> y=6*sqrt2

y =

$6 \cdot 2^{1/2}$

>> z=double(y)

z =

8.4853

Symbolic expressions and algebra(5/10)

□ `poly2sym (p)`
`poly2sym (p, 'v')`

ex `>> poly2sym([2,6,4])`

`ans =`
 $2x^2+6x+4$
`>> poly2sym([5,-3,7],'y')`
`ans =`
 $5y^2-3y+7$

□ `sym2poly (E)`

ex `>> syms x`
`>> sym2poly (9*x^2+4*x+6)`
`ans =`
 $9 \quad 4 \quad 6$

□ `pretty (E)`

ex `>> E5=1/(x+6);`
`>> pretty(E5)`

$$\frac{1}{x + 6}$$

□ `subs (E,old,new)`

ex `>> syms x y`
`>> E=x^2+6*x+7;`
`>> F=subs(E,x,y)`
`F =`
 $y^2+6*y+7$

ex `>> syms t`
`>> f=sym('f(t)');`
`>> g=subs(f,t,t+2)-f`
`g =`
 $f(t+2)-f(t)$

Symbolic expressions and algebra(6/10)

```
ex (n-1)!  
>> kfac=sym('k!');  
>> syms k n  
>> E=subs(kfac,k,n-1)
```

E =

(n-1)!

```
>> expand(E)
```

ans =

n!/n

To compute a numeric factorial

```
ex >> 5!  
>> factorial(5)  
>> prod(1:5)
```

```
ex >> syms a b x  
>> E=a*sin(b);  
>> F=subs(E,{a,b},{x,2})
```

F =

x*sin(2)

Symbolic expressions and algebra(7/10)

■ Evaluating expressions

```
ex >> syms x
    >> E=x^2+6*x+7;
    >> G=subs(E,x,2)
```

```
G =
```

```
    23
```

```
>> class(G)
```

```
ans =
```

```
    double
```

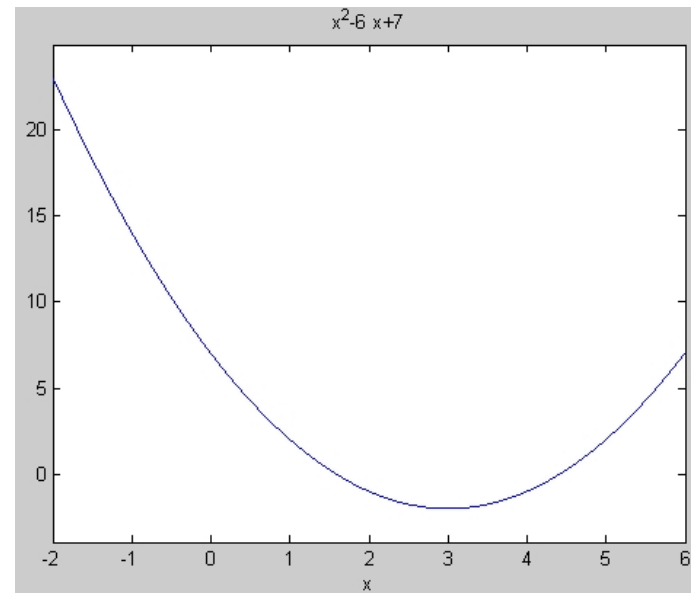
- digit (d) to change the number of digits MATLAB uses for calculating and evaluating expressions
- vpa (E) to compute the expression E to the number of digits specified by the default value of 32 or the current setting of digits
- vpa (E,d) to compute the expression E using d digits

Symbolic expressions and algebra(8/10)

■ Plotting expressions

- `ezplot (E)`
`ezplot (E, [xmin xmax])`

ex `>> syms x`
`>> E=x^2-6*x+7;`
`>> ezplot(E,[-2 6])`



Symbolic expressions and algebra(9/10)

■ Order of precedence

```
ex  syms x
    E=x^2-6*x+7;
    F=-E/3
```

```
F = -1/3*x^2+2*x-7/3
normally F = -(x^2-6x+7)/3
```

Functions for creating and evaluating symbolic expressions

Command	Description
class (E)	Returns the class of the expression E .
digits (d)	Sets the number of decimal digits used to do variable precision arithmetic. The default is 32 digits.
double (E)	Converts the expression E to numeric form.
explot (E)	Generates a plot of a symbolic expression E , which is a function of one variable. The default range of the independent variable is the interval $[-2\pi, 2\pi]$ unless this interval contains a singularity. The optional form explot (E,[xmin xmax]) generates a plot over the range from xmin to xmax .
findsym(E)	Finds the symbolic variables in a symbolic expression or matrix, where E is a scalar or matrix symbolic expression, and returns a string containing all the symbolic variables appearing in E . the variables are returned in alphabetical order and are separated by commas. If no symbolic variables are found, findsym returns the empty string.
findsym(E,n)	Returns the n symbolic variables in E closest to x , with the tie breaker going to the variable closer to z .
[num den]=numden(E)	Returns two symbolic expressions that represent the numerator expression num and denominator expression den for the rational representation of the expression E .
x=sym ('x')	Creates the symbolic variable with name x . typing x=sym ('x','real') tells MATLAB to assume that x is real. Typing x=sym ('x','unreal') tells MATLAB to assume that x is not real.
syms x y u v	Creates the symbolic variables x,y,u, and v . when used without arguments, syms lists the symbolic objects in the workspace.
vpa (E,d)	Sets the number of digits used to evaluate the expression E to d . typing vpa(E) causes E to be evaluated to the number of digits specified by the default value of 32 or by the current setting of digits.

Symbolic expressions and algebra(10/10)

Functions for creating and evaluating symbolic expressions

Command	Description
collect (E)	Collects coefficients of like powers in the expression E .
expand (E)	Expands the expression E by carrying out powers.
factor (E)	Factors the expression E .
poly2sym (p)	Converts a polynomial coefficient vector p to a symbolic polynomial. The form poly2sym(p,'v') generates the poly
pretty (E)	Displays the expression E on the screen in a form that resembles typeset mathematics.
simple (E)	Searches for the shortest form of the expression E in terms of number of characters. When called, the function displays the results of each step of its search. When called without the argument, simple acts on the previous expression. The form [r, how] = simple (E) does not display intermediate steps, but saves those steps in the string how . The shortest form found is stored in r .
simplify (E)	Simplify the expression E using Maple's simplification rules.
subs (E,old,new)	Substitutes new for old in the expression E , where old can be a symbolic variable or expression, new can be a symbolic variable ,expression ,or matrix, or a numeric value or matrix.
sym2poly (E)	Converts the expression E to a polynomial coefficient vector.

Algebraic and transcendental equations(1/3)

■ The solve Function

(equation $x+5=0$)

```
ex >> eq1='x+5=0';
    >> solve(eq1)
    ans =
        -5
```

```
ex >> solve('x+5=0')
    >> syms x
    ans =
        -5
```

```
ex >> solve(x+5)
    ans =
        -5
```

```
ex >> syms x
    >> x=solve(x+5)
    x =
        -5
```

(equation $e^{2x} + 3e^x = 54$)

```
ex >> solve('exp(2*x)+3*exp(x)=54')
    ans =
        log(-9)
        log(6)
```

```
ex >> eq2='y^2+3*y+2=0';
    >> solve(eq2)
    ans =
        [-2]
        [-1]
```

```
ex >> eq3='x^2+9*y^4=0';
    >> solve(eq3) % Note that x is presumed to be the unknown
    variable
    ans =
        [3*i*y^2]
        [-3*i*y^2]
```

Algebraic and transcendental equations(2/3)

■ solve (E, 'v')

ex `>> solve('b^2+8*c+2*b=0')` % solves for c because it is closer to x
 ans =

$$-1/8*b^2-1/4*b$$

ex `>> solve('b^2+8*c+2*b=0','b')` % solves for b
 ans =

$$\begin{bmatrix} -1+(1-8*c)^{1/2} \\ -1-(1-8*c)^{1/2} \end{bmatrix}$$

■ [x ,y] = solve (eq1,eq2)

ex `>> eq4='6*x+2*y=14';`
`>> eq5='3*x+7*y=31';`
`>> solve(eq4,eq5)`
 ans =
 x : [1x1 sym]
 y : [1x1 sym]

```
>> x=ans.x
```

```
x =
```

```
1
```

```
>> y=ans.y
```

```
y =
```

```
4
```

ex `>> [x ,y] = solve(eq4,eq5)`

```
>> S=solve(eq4,eq5)
```

```
S=
```

```
x : [1x1 sym]
```

```
y : [1x1 sym]
```

```
>> S.x
```

```
ans=
```

```
1
```

```
>> S.y
```

```
ans =
```

```
4
```

Algebraic and transcendental equations(3/3)

Functions for solving algebraic and transcendental equations

Command	Description
solve (E)	Solves a symbolic expression or equation represented by the expression E . if E represents an equation, the equation's expression must be enclosed in single quotes. If E represents an expression, then the solution obtained will be the roots of the expression E ; that is, the solution of the equation $E=0$. You need not declare the symbolic variable with the sym or syms function before using solve .
solve (E1,...,En)	Solves multiple expressions or equations
S= solve (E)	Saves the solution in the structure S .

Calculus (differentiation) (1/9)

■ diff (E)

ex $\frac{dx^n}{dx} = nx^{n-1}$

```
→ >> syms n x y
>> diff(x^n)
ans =
    x^n*n/x
>> simplify(ans)
ans =
    x^(n-1)*n
```

ex $\frac{d \ln x}{dx} = \frac{1}{x}$

```
→ >> diff(log(x))

ans =
    1/x
```

ex $\frac{d \sin^2 x}{dx} = 2 \sin x \cos x$

```
→ >> diff((sin(x)^2)

ans =
    2*sin(x)*cos(x)
```

ex $\frac{d \sin y}{dy} = \cos y$

```
→ >> diff((sin(y))

ans =
    cos(y)
```

Calculus (differentiation) (2/9)

ex $f(x, y) = \sin(xy)$

$$\frac{df}{dx} = y \cos(xy)$$

→ `>> diff(sin(x*y))`

ans =
cos(x*y)*y

ex x^2

1) `>> diff(sin(x*y))`

ans =
cos(x*y)*y

2) `>> E='x^2';`
`>> diff(E)`

ans =
2*x

3) `>> syms x`
`>> diff(x^2)`

ans =
2*x

■ **diff (E, v)**

ex $\frac{\partial [x \sin(xy)]}{\partial y} = x^2 \cos(xy)$

→ `>> syms x y`
`>> diff(x*sin(x*y),y)`

ans =
x^2*cos(x*y)

■ **diff (E, n)**

ex $\frac{d^2(x^3)}{dx^2} = 6x$

→ `>> syms x`
`>> diff(x^3,2)`

ans =
6*x

■ **diff (E, v, n)**

ex $\frac{\partial^2 [x \sin(xy)]}{\partial y^2} = -x^3 \sin(xy)$

→ `>> syms x y`
`>> diff(x*sin(x*y),y,2)`

ans =
-x^3*sin(x*y)

Calculus (integration) (3/9)

■ int (E)

ex >> syms x
 >> int(2*x)
 ans =
 x^2

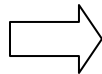
ex

$$\int x^n dx = \frac{x^{n+1}}{n+1}$$

$$\int \frac{1}{x} dx = \ln x$$

$$\int \cos x dx = \sin x$$

$$\int \sin y dy = -\cos y$$



```
>> syms n x y
>> int(x^n)
ans =
      x^(n+1)/(n+1)
>> int(1/x)
ans =
      log(x)
>> int(cos(x))
ans=
      sin(x)
>> int(sin(y))
ans=
      -cos(y)
```

Calculus (integration) (4/9)

- **int (E, v)** **ex** $\int x^n dx = \frac{x^{n+1}}{n+1}$

→ >> syms n x
 >> int(x^n,n)

 ans =
 1/log(x)*x^n

- **int (E, a, b)** **ex** $\int_2^5 x^2 dx = \frac{x^3}{3} \Big|_2^5 = 39$

→ >> syms x
 >> int(x^2,2,5)

 ans =
 39

Calculus (integration) (5/9)

■ int (E, v, a, b)

ex $\int_0^5 xy^2 dy = x \frac{y^3}{3} \Big|_0^5 = \frac{125}{3}x$

→ `>> syms x y`
`>> int(x*y^2,y,0,5)`

ans =
 125/3*x

ex $\int_a^b x^2 dx = \frac{b^3}{3} - \frac{a^3}{3}$

→ `>> syms a b x`
`>> int(x^2,a,b)`

ans =
 1/3*b^3-1/3*a^3

■ int (E, m, n)

ex $\int_1^t x dx = \frac{x^2}{2} \Big|_1^t = \frac{1}{2}t^2 - \frac{1}{2}$

$$\int_t^{e^t} \sin x dx = -\cos x \Big|_t^{e^t} = -\cos(e^t) + \cos t$$

→ `>> syms t x`
`>> int(x,1 t)`

ans =
 1/2*t^2-1/2
`>> int(sin(x),t,exp(t))`
 ans =
 -cos(exp(t))+cos(t)

ex $\int \frac{1}{x-1} dx = \ln|x-1|$

→ `>> syms x`
`>> int(1/(x-1))`
 ans =
 log(x-1)
`>> int(1/(x-1),0,2)`
 ans =

NaN :singularity at x=1

Calculus (Taylor series) (6/9)

■ Taylor's theorem

$$f(x) = f(a) + \left(\frac{df}{dx}\right)\Big|_{x=a}(x-a) + \frac{1}{2}\left(\frac{d^2f}{dx^2}\right)\Big|_{x=a}(x-a)^2 + \cdots + \frac{1}{k!}\left(\frac{d^k f}{dx^k}\right)\Big|_{x=a}(x-a)^k + \cdots + R_n, \quad R_n = \frac{1}{n!}\left(\frac{d^n f}{dx^n}\right)\Big|_{x=b}(x-a)^n$$

ex $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, -\infty < x < \infty$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, -\infty < x < \infty$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots, -\infty < x < \infty$$

■ taylor (f, n, a)

ex

```
→ >> syms x
>> f=exp(x);
>> taylor(f,4)
ans =
    1+x+1/2*x^2+1/6*x^3
>> taylor(f,3,2)
ans =
    exp(2)+exp(2)*(x-2)+1/2*exp(2)*(x-2)^2    =( e^2[1+(x-2)+1/2(x-2)^2] )
```

Calculus (sums) (7/9)

■ symsum (E)

$$\sum_{x=0}^{x-1} E(x) = E(0) + E(1) + E(2) + \cdots + E(x-1)$$

■ S = symsum (E, a, b)

$$\sum_{x=a}^b E(x) = E(a) + E(a+1) + E(a+2) + \cdots + E(b)$$

ex $\sum_{k=0}^{10} k = 0+1+2+3+\cdots+9+10 = 55$

$$\sum_{k=0}^{n-1} k = 0+1+2+3+\cdots+n-1 = \frac{1}{2}n^2 - \frac{1}{2}n$$

$$\sum_{k=1}^4 k^2 = 1+4+9+16 = 30$$

ex

```
>> syms k n
>> symsum(k,0,10)
ans =
    55
```

```
>> symsum(k,0,n-1)
ans =
    1/2*n^2-1/2*n
```

```
>> factor(ans)
ans =
    1/2*n*(n-1)
```

```
>> symsum(k^2,1,4)
ans =
    30
```

Calculus (limits) (8/9)

■ **limit (E, a)** ($\lim_{x \rightarrow a} E(x)$)

■ **limit (E) : limit as $x \rightarrow 0$**

ex $\lim_{x \rightarrow 0} \frac{\sin(ax)}{x} = a$

→

```
>> syms a x
>> limit(sin(a*x)/x)
ans =
    a
```

■ **limit (E, v, a) : limit as $v \rightarrow a$**

ex $\lim_{x \rightarrow 3} \frac{x-3}{x^2-9} = \frac{1}{6}$

$$\lim_{h \rightarrow 0} \frac{\sin(x+h) - \sin(x)}{h}$$

→

```
>> syms h x
>> limit((x-3)/(x^2-9),3)
ans =
    1/6
>> limit((sin(x+h)-sin(x))/h,h,0)
ans =
    cos(x)
```

■ **limit (E, v, a, 'right')**

■ **limit (E, v, a, 'left')**

ex $\lim_{x \rightarrow 0^-} \frac{1}{x} = -\infty$

$$\lim_{x \rightarrow 0^+} \frac{1}{x} = \infty$$

→

```
>> syms x
>> limit(1/x,x,0,'left')
ans =
    -Inf

>> limit(1/x,x,0,'right')
ans =
    Inf
```

Calculus (9/9)

Symbolic calculus functions

Command	Description
diff (E)	Returns the derivative of the expression E with respect to the default independent variable.
diff (E,v)	Returns the derivative of the expression E with respect to the variable v .
diff (E,n)	Returns the n th derivative of the expression E with respect to the default independent variable.
diff (E,v,n)	Returns the n th derivative of the expression E with respect to the variable v .
int (E)	Returns the integral of the expression E with respect to the default independent variable.
int (E,v)	Returns the integral of the expression E with respect to the variable v .
int (E,a,b)	Returns the integral of the expression E with respect to the default independent variable over the interval [a,b] , where a and b are numeric quantities.
int (E,v,a,b)	Returns the integral of the expression E with respect to the variable v over the interval [a,b] , where a and b are numeric quantities.
int (E,m,n)	Returns the integral of the expression E with respect to the default independent variable over the interval [m,n] , where m and n are symbolic expressions.
limit (E)	Returns the limit of the expression E as the default independent variable goes to 0 .
limit (E,a)	Returns the limit of the expression E as the default independent variable goes to a .
limit (E,v,a)	Returns the limit of the expression E as the variable v goes to a .
limit (E,v,a,'d')	Returns the limit of the expression E as the variable v goes to a from the direction specified by d , which may be right or left.
dymsum (E)	Returns the symbolic summation of the expression E .
taylor (f,n,a)	Gives the first n-1 terms in the Taylor series for the function defined in the expression f , evaluated at the point x=a . If the parameter a is omitted, the function returns the series evaluated at x=0 .

Differential Equations (1/6)

■ Solving a Single Differential Equation

- The `dsolve` function's syntax: `dsolve('eqn')` → returns a symbolic solution of the ODE specified by the symbolic expression `eqn`.
- The uppercase letter D → the first derivative, D2 → the second derivative.
(ex) Dw → dw/dt.
- Cannot use uppercase D as symbolic variable when using the `dsolve` function.

(example)

$$\frac{dy}{dt} + 2y = 12 \quad \Rightarrow \quad \text{Analytic solution: } y(t) = 6 + C_1 e^{-2t}$$

```
>> dsolve('Dy+2*y=12')
```

```
ans =
```

```
6+exp(-2*t)*C1
```



A symbolic solution using the `dsolve` function.

Differential Equations (2/6)

■ Solving Sets of Equations

- The appropriate syntax: `dsolve('eqn1', 'eqn2', ...)` → returns a symbolic solution of the set of equations specified by the symbolic expressions `eqn1` and `eqn2`.

(example)

$$\frac{dx}{dt} = 3x + 4y$$

$$\frac{dy}{dt} = -4x + 3y$$



Analytic solution:

$$x(t) = C_1 e^{3t} \cos 4t + C_2 e^{3t} \sin 4t$$

$$y(t) = -C_1 e^{3t} \sin 4t + C_2 e^{3t} \cos 4t$$

```
>> [x,y]=dsolve('Dx=3*x+4*y','Dy=-4*x+3*y')
```

```
x =
```

```
-exp(3*t)*(C1*cos(4*t)-C2*sin(4*t))
```

```
y =
```

```
exp(3*t)*(C1*sin(4*t)+C2*cos(4*t))
```

Differential Equations (3/6)

■ Specifying Initial and Boundary Conditions

- The appropriate syntax: `dsolve('eqn','cond1','cond2')` → returns a symbolic solution of the ODE specified by the symbolic expression `eqn`, subject to the conditions specified in the expressions `cond1`, `cond2`, and so on.

(example)

$$\frac{d^2 y}{dt^2} = c^2 y, y(0) = 1, \dot{y}(0) = 0 \quad \longrightarrow \quad \text{Analytic solution: } y(t) = (e^{ct} + e^{-ct}) / 2$$

```
>> dsolve('D2y=c^2*y', 'y(0)=1', 'Dy(0)=0')
```

```
ans =
```

```
1/2*exp(c*t)+1/2*exp(-c*t)
```

Differential Equations (4/6)

■ Plotting the Solution (1)

- The `ezplot` function can be used to plot the solution

(example)

$$\frac{dy}{dt} + 10y = 10 + 4 \sin(4t), y(0) = 0$$

Analytic solution:

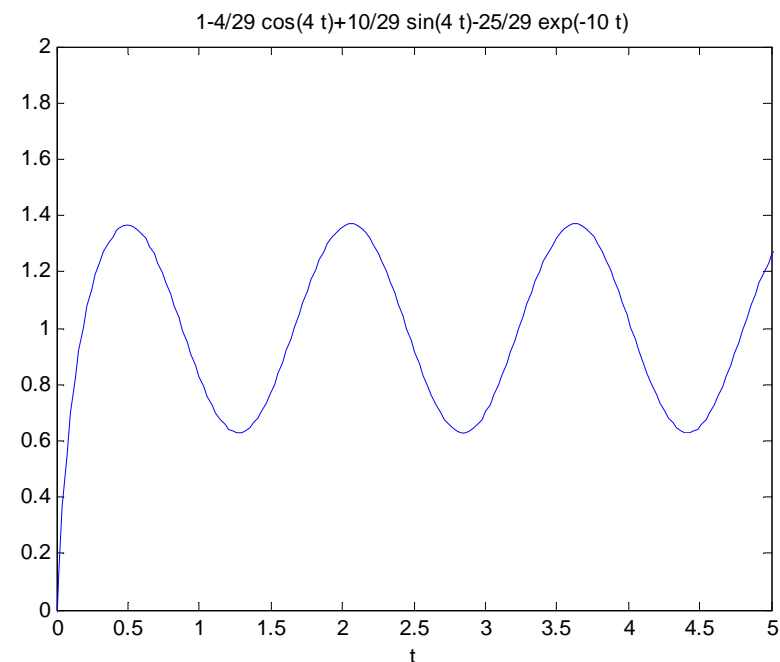
$$y(t) = 1 - \frac{4}{29} \cos(4t) + \frac{10}{29} \sin(4t) - \frac{25}{29} e^{-10t}$$

```
>> y=dsolve('Dy+10*y=10+4*sin(4*t)', 'y(0)=0')
```

```
y =
```

```
1-4/29*cos(4*t)+10/29*sin(4*t)-25/29*exp(-10*t)
```

```
>> ezplot(y),axis([0 5 0 2])
```



Differential Equations (5/6)

■ Plotting the Solution (2)

- Sometimes the `ezplot` function uses too few values of the independent variable and thus does **not** produce a smooth plot.
- To override the spacing chosen by the `ezplot` function, you can use the `subs` function to substitute an array of values for the independent variable.

```
>> syms t  
>> x=[0:0.05:5];  
>> P=subs(y,t,x);  
>> plot(x,P),axis([0 5 0 2]),xlabel('t')
```

Define t to be a symbolic variable.

Differential Equations (6/6)

(Table 10.4-1) The `dsolve` function

Command	Description
<code>dsolve('eqn')</code>	Returns a symbolic solution of the ODE specified by the symbolic expression <code>eqn</code> . Use the uppercase letter D to represent the first derivative; use D2 to represent the second derivative, and so on. Any character immediately following the differentiation operator is taken to be the dependent variable.
<code>dsolve('eqn1','eqn2',...)</code>	Returns a symbolic solution of the set of equations specified by the symbolic expressions <code>eqn1</code> , <code>eqn2</code> , and so on.
<code>dsolve('eqn','cond1','cond2',...)</code>	Returns a symbolic solution of the ODE specified by the symbolic expression <code>eqn</code> , subject to the conditions specified in the expressions <code>cond1</code> , <code>cond2</code> , and so on. If <code>y</code> is the dependent variable, these conditions are specified as follows: $y(a) = b$, $Dy(a) = c$, $D2(a) = d$, and so on.
<code>dsolve('eqn1','eqn2',...,'cond1','cond2',...)</code>	Returns a symbolic solution of a set of equations specified by the symbolic expressions <code>eqn1</code> , <code>eqn2</code> , and so on, subject to the initial conditions specified in the expressions <code>cond1</code> , <code>cond2</code> , and so on.

Laplace Transforms (1/2)

■ Direct Method (1)

(example) solve the equation with $f(t) = \sin t$, in terms of an unspecified value for $y(0)$.

$$a \frac{dy}{dt} + y = f(t)$$

(Symbolic processing with MATLAB)

```
>> syms a L s t
>> y=sym('y(t)');
>> dydt=sym('diff(y(t),t)');
>> f=sin(t);
>> eq=a*dydt+y-f;
>> E=laplace(eq,t,s)
E =

a*(s+laplace(y(t),t,s)-y(0))+laplace(y(t),t,s)-1/(s^2+1)
>> E=subs(E,'laplace(y(t),t,s)',L)

E =

a*(s+L-y(0))+L-1/(s^2+1)
```

(to be continued...)

The steps of solving a equation by using direct method.

1. Define the symbolic variables, including the derivatives that appear in the equation. Note that $y(t)$ is explicitly expressed as a function of t in these definitions.
2. Move all terms to the left side of the equation and define the left side as a symbolic expression.
3. Apply the Laplace transformation to the differential equation to obtain an algebraic equation.
4. Substitute a symbolic variable, here L , for the expression `laplace(y(t),t,s)` in the algebraic equation. Then solve the equation for the variable L , which is the transform of the solution.
5. Invert L to find the solution as a function of t .

Laplace Transforms (2/2)

■ Direct Method (2)

```
>> L=solve(E,L)

L =

(a+y(0)*s^2+a*y(0)+1)/(a*s^3+a*s+s^2+1)

>> l=simplify(ilaplace(L))

l =

(-a*cos(t)+sin(t)+exp(-t/a)+y(0)*a^2+exp(-t/a)+y(0)+exp(-t/a)*a)/(a^2+1)

>> l=collect(l,exp(-t/a))

l =

1/(a^2+1)+(-a*cos(t)+sin(t)+(y(0)*a^2+y(0)+a)/(a^2+1))*exp(-t/a)
```

➔
$$y(t) = \frac{1}{1+a^2} \left\{ \sin t - a \cos t + e^{-t/a} [y(0) + a^2 y(0) + a] \right\}$$

(Table 10.5-1) Laplace transform functions

Command	Description
<code>ilaplace(function)</code>	Returns the inverse Laplace transform of function.
<code>laplace(function)</code>	Returns the Laplace transform of function.
<code>laplace(function,x,y)</code>	Returns the Laplace transform of function, which is a function of x, in terms of the Laplace variable y.

MATLAB® Applications in Chemical Engineering

James A. Carnell
North Carolina State University

MATLAB is a powerful code-based mathematical and engineering calculation program. It performs all calculations using matrices and vectors in a logical programming environment. This guide is a brief introduction to MATLAB in chemical engineering, and in no way attempts to be a comprehensive MATLAB learning resource. This guide is a starting point for the new MATLAB user, and as such no prior MATLAB experience is necessary. Further help can be found in the MATLAB help files or at Mathworks website at www.mathworks.com and in the help section at <http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml>

Table of Contents

Section I: How MATLAB Works

- » Basic MATLAB: The Language
- » Mfiles (Function files)
- » Dealing with Functions
 - The 'plot' function
 - The 'fzero' function
 - Solving linear equations in MATLAB
 - The 'fsolve' function

Section II: Numerical and Symbolic Integration

- » Numerical Integration; Quadrature
 - The Simpson's Rule and Lobatto Quadrature
- » Symbolic Integration and Differentiation

Section III: Numerically Solving Differential Equations and Systems of Differential Equations

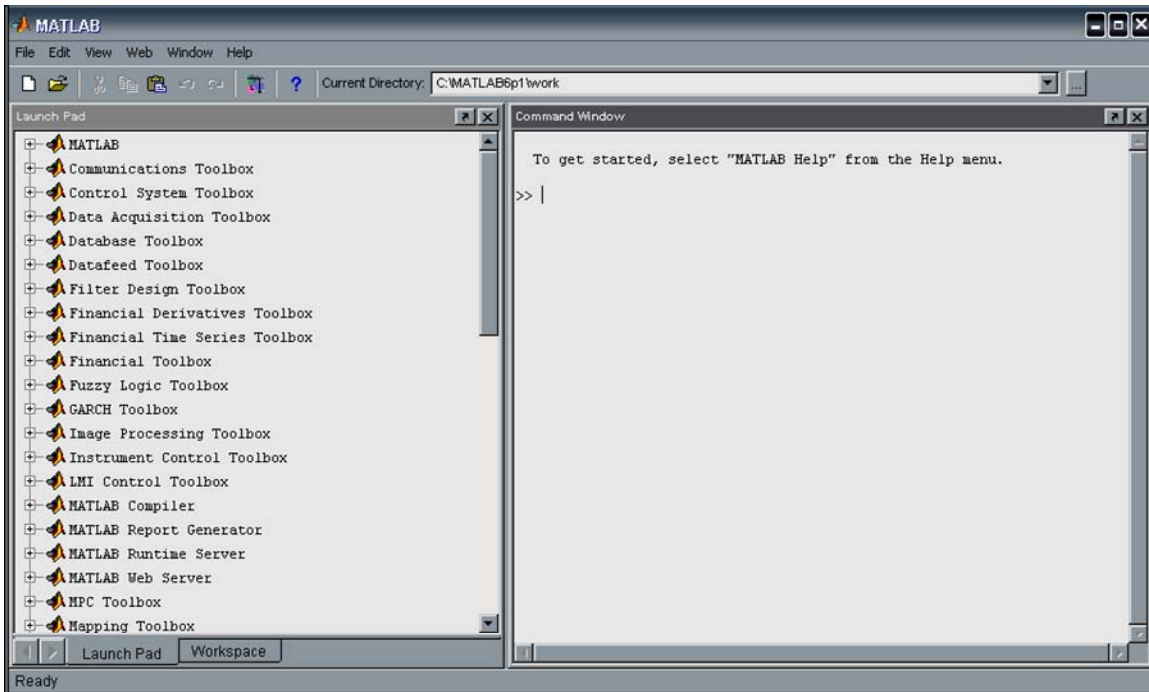
- » First order ode's
- » Higher order ode's

Appendix

- » Glossary of Commands, Parts I, II and II
- » Line Markers

Section I: How MATLAB works

Most MATLAB commands can be run from the command window (shown below, on the right hand side of the interface). MATLAB commands can also be entered into a text file labeled with the '.m' extension. These files are known as 'm-files'. These commands can be broken down into scripts and programming. Scripts can be thought of as commands that instruct MATLAB to execute a particular function or pre-made program, and programming can be thought of as the raw code required to construct functions and programs within MATLAB. Generally, all programming must be contained within a file used by MATLAB (called an m-file), but script can be entered either in an m-file or directly into the command window. An image of the MATLAB interface is shown below.



MATLAB contains many ready-made programs or functions that are conveniently arranged into different toolboxes. When using MATLAB, these toolboxes and their functions can be called upon and executed in any MATLAB script. In the above image, the toolbox selection or launch pad is shown (at the left hand side of the interface).

Basic MATLAB: The language

MATLAB uses a language that is somewhat similar to that of Maple¹. The scripts or calling functions have a particular name and argument that must be entered into the function execution call. For example, to plot the sine function in MATLAB between 0 and 6 using the *fplot* command, the following code can be entered directly into the command window, or into an m-file:

```
fplot('sin(x)', [0,6])
```

¹ Maple is a registered trademark of Waterloo Maple, Inc.

(One can define the function $\sin(x)$ in an m-file and replace the *fplot* command to be *fplot('filename',[0,6])*)

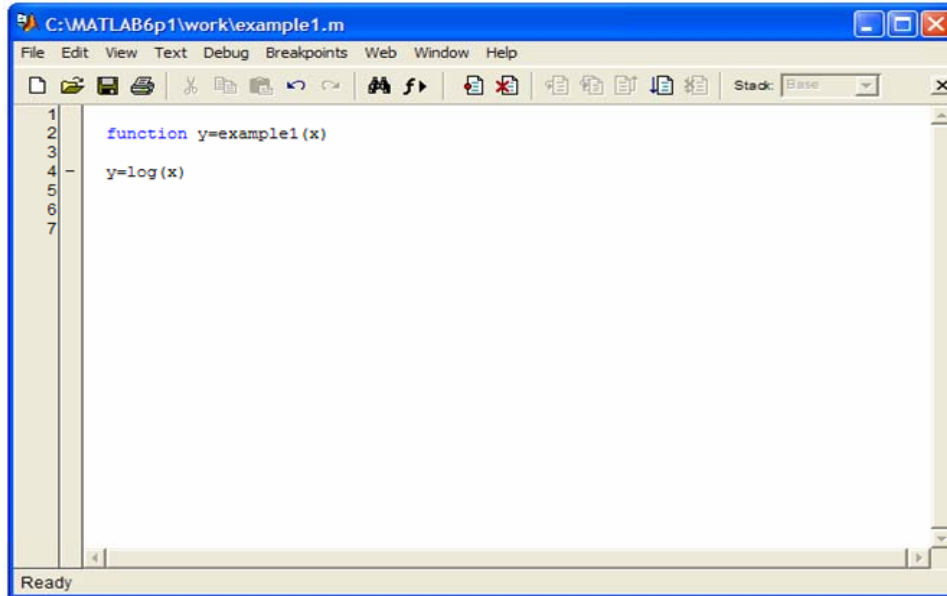
Before going much further, an understanding of the structure of a MATLAB simulation or execution must be developed.

M-files

M-files contain programming, scripts, equations or data that are called upon during an execution. If the m-file is a function definition, then the most important part of this type of m-file is the first line. The first line **must** contain the function definition so that MATLAB can find those m-files that are called upon. These types of m-files are called function m-files or function files. The code used to define the function file is as follows:

```
function z=file_name(x,y)
```

'file_name' is simply the name of the m-file (the filename *must* be the same in the definition and the file-name), z is the dependant variable, and x and y are the independent variables. (Of course, one can have less or more independent variables depending upon the complexity of the problem and the equations involved.) The next few lines of script in the m-file can define the function or functions and label any required variables. The following is an example of an m-file used to plot the *natural* logarithm² function.

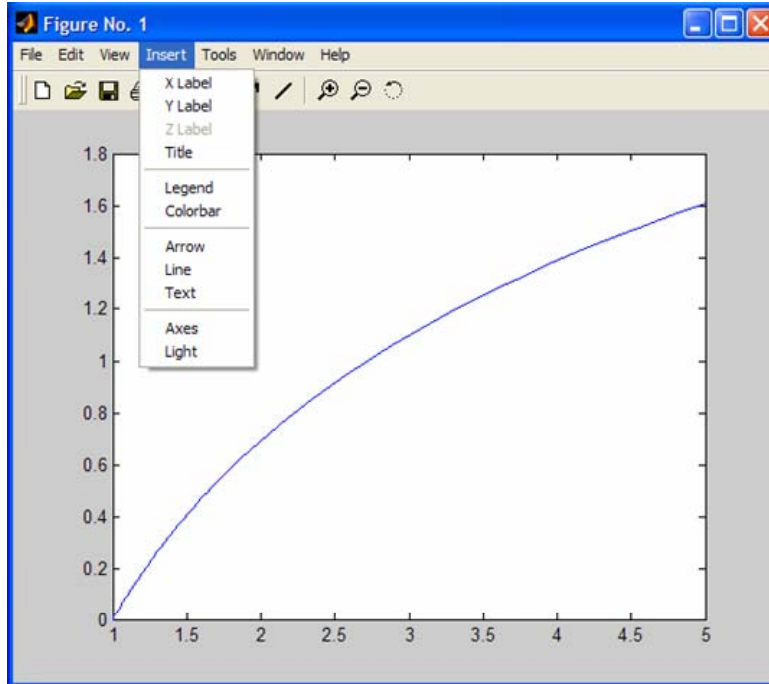


To produce a plot of this function, the following code is entered into the command window:

```
fplot('example1',[1,5])
```

This yields a plot of $\ln(x)$ between $x= 1$ and $x= 5$.

² MATLAB uses 'log' as the natural logarithm function, and 'log10' as logarithm base ten.



Using the 'insert' menu one can add a title, x and y axis titles, and if necessary a legend. One can also use commands nested within the *fplot* command to title the chart, add axis titles, or decide upon curve characteristics such as line color or marker.

Dealing with functions

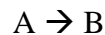
Standard functions such as the sine, cosine, logarithmic, exponential, and user-defined functions within MATLAB will now be covered. *fplot* has already been introduced; now *plot*, *fzero* and *fsolve* will be introduced

The 'plot' function

The plot function produces a 2-D plot of a y-vector versus either its real index or a specified x-vector. The plot function can be used for data, standard functions, or user-defined functions. Let's look at a simple example of using *plot* to model data.

Example 1.1

The following reaction data has been obtained from a simple decay reaction:

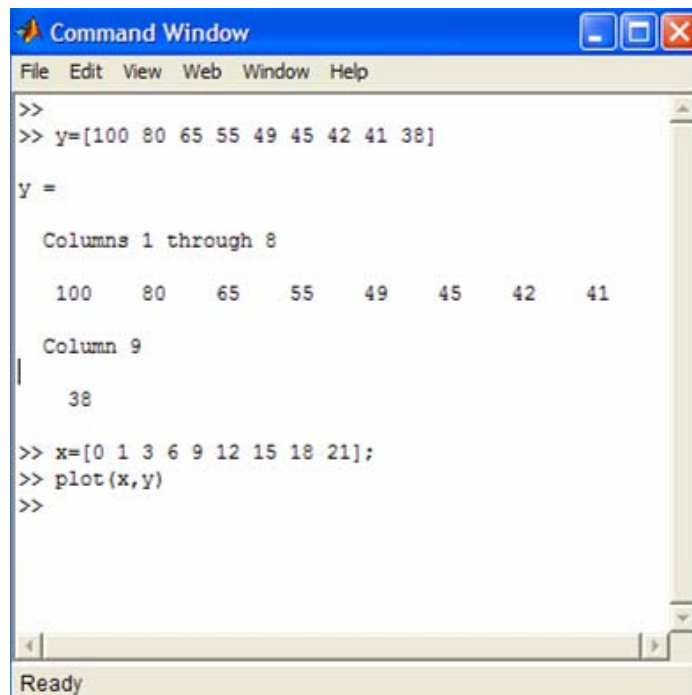


Use MATLAB to plot the concentration of component A in mol/L against the reaction time, t, in minutes. Title the plot, label the axes, and obtain elementary statistics for the data.

Time (Minutes)	Concentration (Moles/Liter)
0	100
1	80
3	65
6	55
9	49
12	45
15	42
18	41
21	38

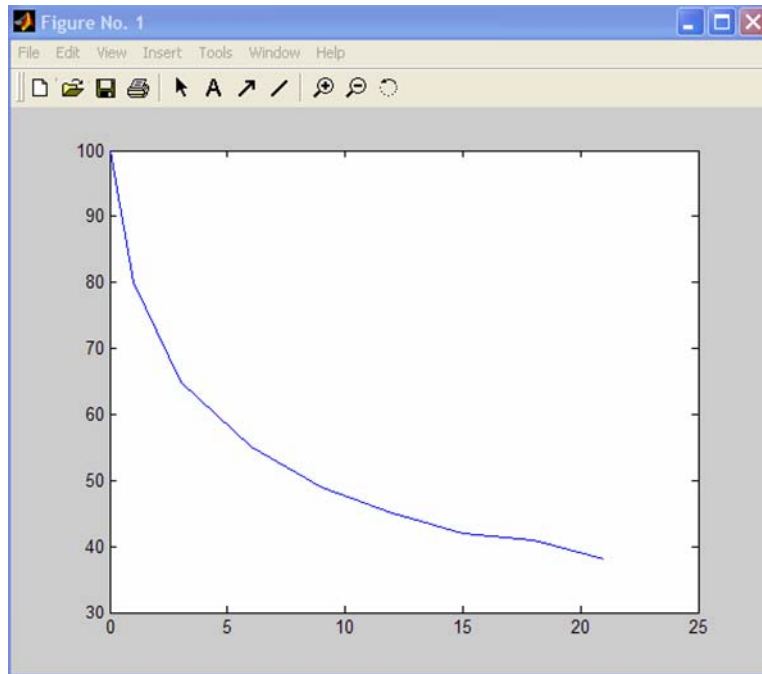
Solution

First, the data must be entered into MATLAB as two vectors. The vectors x and y are defined in the command window, followed by the command to plot the data. The following graph is displayed:



```
Command Window
File Edit View Web Window Help
>>
>> y=[100 80 65 55 49 45 42 41 38]
y =
Columns 1 through 8
    100    80    65    55    49    45    42    41
Column 9
     38
>> x=[0 1 3 6 9 12 15 18 21];
>> plot(x,y)
>>
```

Ready

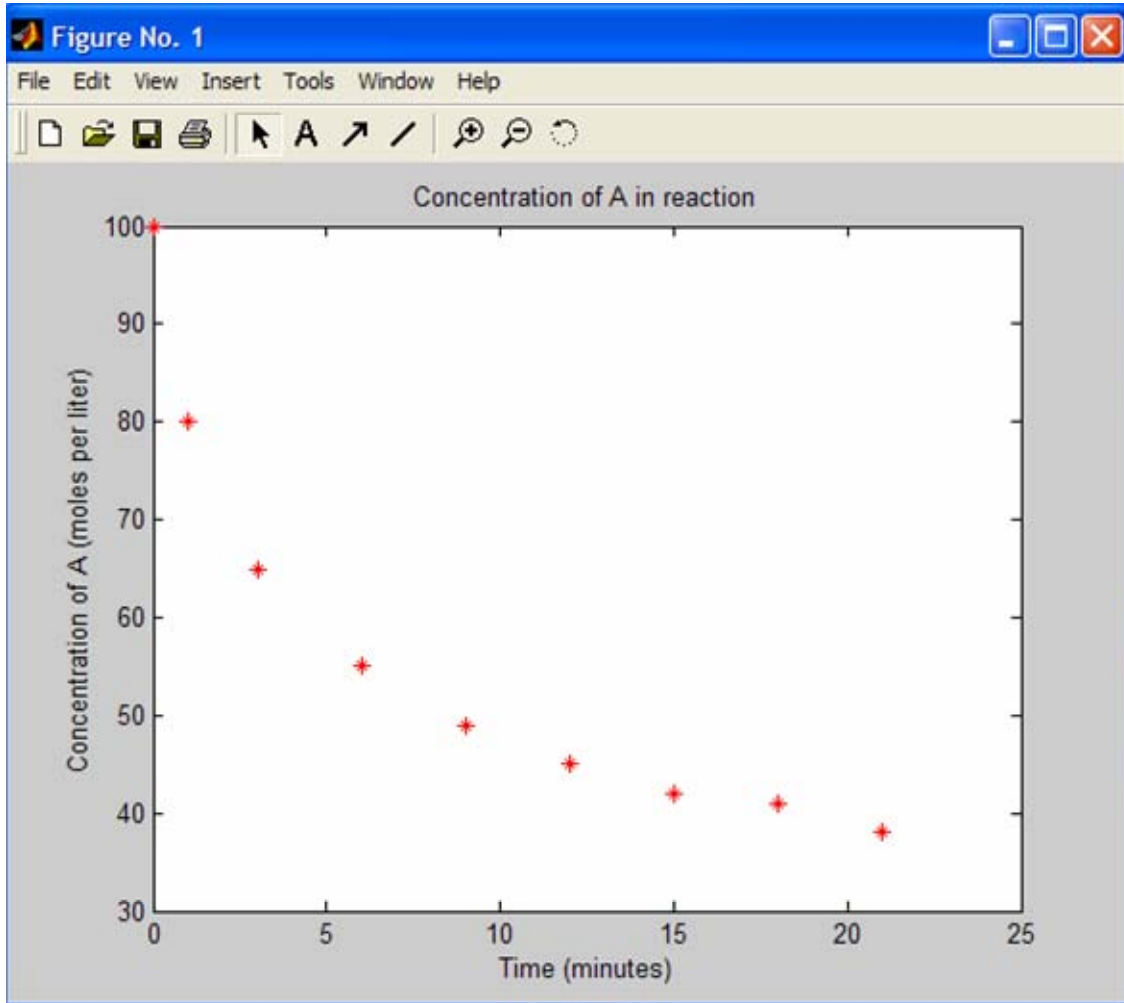


The 'x' row matrix (or vector*) has its display output suppressed using the ';' at the end of the line.

Syntax can be used to specify the title and labels, but an easier GUI (Graphical User Interface) based approach is to simply edit the figure.

- Select the 'Edit Plot' command found in the 'Tools' menu, or click the north-west facing arrow icon on the figure.
- Double click on the white-space in the graph. This enables the property editor. Now the title and axes can be inserted under the 'labels' command.
- Now click directly on the line, and the line property editor will come up. Now the lines color, pattern, or the markers can be edited. The final curve is presented below:

*A single row matrix or column matrix can represent vectors within MATLAB.



Solution curve to example 1.1

NOTE: These figures can be exported in .bmp or .jpg format so they can be made part of a document. Find these under the 'File→Export' menu.

To display simple statistics of the data, follow the path, 'Tools→Data Statistics' and the minimum, maximum, mean, median, standard deviation, and range of x and y will be displayed. Within this box, each statistic can be added to the curve as a data point/line.

To plot functions using *plot*, an m-file can be created defining the function, or the definition can be specified in the command window. This is done in a similar fashion as seen in '*fplot*'. The following code entered into the command window yields a plot of the exponential function between 0 and 10, at intervals of 1.

```
>> x = [0:10]
x =
Columns 1 through 8
    0    1    2    3    4    5    6    7
Columns 9 through 11
    8    9   10
>> y = exp(x);
>> plot(x,y)
>>
```

There are many other 2-D plotting functions in Matlab, but *fplot*, *plot* and *ezplot* are the most useful. ‘ezplot’ is another way to quickly plot functions between specified or non-specified ($-2\pi < x < 2\pi$) intervals. The syntax key for *ezplot* is shown below.

```
ezplot(f)
ezplot(f, [min,max])
ezplot(f, [xmin,xmax,ymin,ymax])
ezplot(x,y)
ezplot(x,y, [tmin,tmax])
```

This can be used to plot $f(x)$ or $f(x(t),y(t))$.

The ‘fzero’ function

The *fzero* function is a convenient function used to find local zeros of a one variable function. The general syntax that can be used for *fzero* (entered into the command window) is

```
x=fzero('function',initial guess)
```

Matlab will then return the solution or a NaN result. An NaN result is ‘not a number’ and represents a ‘no solution’ for the function. Below are a few examples relating to *fzero*.

Example 1.2

Find the zero of the function $y(x) = x^2 - 1$ using an m-file and *fzero*.

Solution

Create the m-file, ‘myfunction.m’

```
function y = myfunction(x)
y = x^2-1
```

In the command window, use *fzero* to find the zero of the function ‘myfunction’

```
x = fzero('myfunction',3)
```

Matlab returns a few iterations and then produces the result ‘x = 1’

Example 1.3

Find the zero of the function $y(x) = \sin(x)$ closest to $x=3$ using the `fzero` command directly in the command window. Do not use an m-file for this example.

Solution

Enter into the command window `x = fzero('sin(x)', 3)` Matlab returns a value of 3.1416. This is the zero of $\sin(x)$ closest to our initial guess of 3.

Solving linear equations in Matlab

Matlab has the ability to easily solve linear equations directly on the command window. The following example will demonstrate how to do this in Matlab.

Example 1.4

Solve the system of linear equations given below using Matlab.

$$\begin{aligned} 3.5u + 2v &= 5 \\ -1.5u + 2.8v + 1.9w &= -1 \\ -2.5v + 3w &= 2 \end{aligned}$$

Solution

Enter the following code directly into the clear Matlab command window. (If the window is not clear, enter the command 'clear' to clear the memory of any variables). The matrix 'a' contains the coefficients of u, v and w respectively. Each row in the matrix (there are three) corresponds to the coefficients of u, v and w in that order. Matrix 'b' contains the solution to each row or equation, 5, -1 and 2. The matrix 'x' that is divided out can be thought of as containing the values for u, v and w.

```
>> a = [3.5 2 0; -1.5 2.8 1.9; 0 -2.5 3];
>>
>> b = [5 -1 2]

b =

     5     -1     2

>> a\b'

ans =

     1.4421
    -0.0236
     0.6470
```

The solutions to the equations are displayed as u, v and w as they are listed in the matrix. The command `a\b'` performs reverse matrix division. The `'` command transposes matrix b, this is required to perform matrix division. It is a property of matrices. Essentially, the

matrix is in the form $Ax=B$, the command $a\b'$ can be thought of as 'dividing out' the values of x . (Alternatively, the command $'b/a'$ will also yield the correct solution set.)

The 'fsolve' function

The 'fsolve' function will probably be the most useful function for simple chemical engineering problems. It is essentially a numeric solver, capable of solving systems of non-linear *continuous* equations.

For the system of n continuous equations (f_1 -- f_n) with n unknown variables (x_1 -- x_n) given by

$$\begin{aligned} f(1) &\equiv f_1(x_1, x_2, x_3, \dots, x_n) = 0 \\ f(2) &\equiv f_2(x_1, x_2, x_3, \dots, x_n) = 0 \\ f(3) &\equiv f_3(x_1, x_2, x_3, \dots, x_n) = 0 \\ &\dots\dots\dots \\ f(n) &\equiv f_n(x_1, x_2, x_3, \dots, x_n) = 0 \end{aligned}$$

Matlab's solver can be used to determine the unknown variables, x_1 through x_n . The following example will illustrate the use of the *fsolve* function.

Example 1.5

Solve the system of equations below using *fsolve*.

$$\begin{aligned} 2a - b - e^{-a} &= 0 \\ 2b - a - e^{-b} &= 0 \end{aligned}$$

Solution

The most efficient way to solve these types of systems is to create a function m-file that contains the equations.

```
function f=example15(x)

f = [2*x(1)-x(2)-exp(-x(1)); -x(1)+2*x(2)-exp(-x(2))];
```

The equations are separated within the matrix 'f' using the semi-colon operator. Now the 'fsolve' command must be used to solve the system.

```
>>
>> x0 = [-5 -5]; %Initial guess, arbitrary
>>
>> fsolve('example15',x0,optimset('fsolve')) %Command calls m-file to solve
Optimization terminated successfully:
  Relative function value changing by less than OPTIONS.TolFun

ans =

    0.5671    0.5671
```

The solutions to the system are $a = 0.5671$ and $b = 0.5671$. Using the 'fsolve' command, much more complicated systems can be easily solved in Matlab. NOTE: 'optimset' is used to change the default solver in Matlab to the optimization toolbox solver (fsolve, 2.0 onwards)

Section II: Numerical and Symbolic integration

Numerical Integration-Quadrature

MATLAB can perform in-depth numerical integrations or quadrature effortlessly and accurately. The first part of this section will look at some of the simple methods of numerical integration within MATLAB.

The Simpson's Rule and Lobatto Quadrature

In MATLAB, the functions 'quad' or in more recent versions, 'quadl' perform numerical integration based on the Simpson's rule and the adaptive Lobatto quadrature respectively. The syntax for the Simpson's based approximation and the Lobatto quadrature are the same. The difference is in each function's name. The syntax below is for the Lobatto quadrature (quadl), and it is the same for the Simpson's quadrature (quad).

```
q = quadl(fun,a,b)
q = quadl(fun,a,b,tol)
q = quadl(fun,a,b,tol,trace)
q = quadl(fun,a,b,tol,trace,p1,p2,...)
[q,fcnt] = quadl(fun,a,b,...)
```

The functions can be defined in function m-files or as inline functions (those functions entered directly into the command). MATLAB also has a function called 'trapz' which enables numerical integration using the trapezoid rule. More information about 'trapz' can be found in the MATLAB help files.

Symbolic Integration and differentiation

Matlab has the ability to perform symbolic integration and differentiation thanks to the Maple® engine located in the symbolic toolbox. Matlab can solve differential equations symbolically, providing general or unique solutions. First, let's look at symbolic integration and differentiation.

Using the 'diff' command, symbolic differentiation of a function can be achieved, and analogously, using the 'int' command symbolic integration of a function can be achieved. Example 2.1 demonstrates use of the 'int' and 'diff' commands.

Example 2.1

Integrate ax^2+bx , with respect to x , (where a and b are constant) and then differentiate the solution obtained with respect to x to regain the initial function.

Solution

To begin with, the symbolic variables within the expression must be defined within MATLAB as symbolic variables. This is done using the syms command. In the

expression, a, b, and x are the symbolic variables that must be defined. The following code is used directly at the command prompt to obtain the solution:

```
>> syms a b x
>> diff(a*x^2+b*x)

ans =

2*a*x+b

>> int(2*a*x+b)

ans =

a*x^2+b*x
```

The syms command is used to define a, b and x as variables for symbolic purposes. The expression is differentiated to obtain the solution $2ax+b$, which upon integration with respect to x yields ax^2+bx as expected.

MATLAB can solve ordinary differential equations symbolically with or without boundary conditions or initial value parameters. The 'dsolve' command is used for this purpose. Within dsolve, the letter 'Dij' is used to indicate a derivative, where i is the order of differentiation, and j is the dependent variable. 'D' implicitly specifies first order derivative, 'D2' signifies a second order derivative and so on. The letter t is the default independent variable for dsolve. So D2y is analogous to $\frac{d^2y}{dt^2}$. The following example illustrates the use of dsolve.

Example 2.2

Solve the ode $\frac{d^2y}{dx^2} = 6y - \frac{dy}{dx}$, where y(x) using dsolve

Solution

```
>> dsolve('D2y=6*y-Dy','x')

ans =

C1*exp(-3*x)+C2*exp(2*x)
```

Where C1 and C2 are constants of integration.

The next example shows how to solve an initial value problem for a second order ode.

Example 2.3

Solve the ode $\frac{d^2m}{dx^2} + m = 0$, $m(0)=2$ and $\frac{dm}{dx}(0) = 3$, where $m(x)$.

Solution

```
>> dsolve('D2m+m=0','m(0)=2','Dm(0)=3','x')  
  
ans =  
  
3*sin(x)+2*cos(x)
```

Within dsolve, $m(0)=2$ and $Dm(0)=3$ define the initial conditions of the ode.

In the assignment of the initial conditions, labeled variables could have been inserted rather than numeric values. For example, if the initial conditions for example 2.3 were $m(0)=a$ and $\frac{dm}{dx}(0) = b$, then the command chain

```
dsolve('D2m+m=0','m(0)=a','Dm(0)=b','x')
```

 could have been used to yield the solution $b*\sin(x)+a*\cos(x)$.

Section III: Numerically Solving Differential Equations and Systems of Differential Equations

First order ode's

One of the best engineering uses of MATLAB is its application to the numeric solution of ordinary differential equations (ode's). MATLAB has multiple different ode solvers that allow ode's to be solved accurately and efficiently depending on the stiffness of the ode. Stiffness is the relative change in the solution of a differential equation. A stiff differential equation is one that the solution changes greatly when close to the point of integration, but need not change significantly over the duration of the integration. For this type of solution, a numerical method that takes small integration intervals rather than large intervals would be required. Stiffness and solver selection are mainly a matter of efficiency. In solving ode's, selecting a solver that takes the largest step, yet still maintains an accurate solution is the key to increased efficiency.

There are different ways to set up and execute the ode solvers, but for this guide, a system that uses multiple m-files per each ode solution will be employed. The main two m-files that are needed are a run file and a function file. For the solution of ode's in MATLAB *all* ode's *must* be defined in a function m-file. When entered into the function file, the ode's must have the first order form $\frac{dy}{dx} = f(y,x)$. The function file must contain

- i) The function definition e.g. `function dmdt=file_name(t,m)`, where t is the independent variable and m is the dependent variable of first order

- ii) If global variables are used, the global command must be inserted after the function definition
- iii) The ode or ode's in the form described above e.g. `dmdt=f(m,t)`

The filename, variables (m and t) and dmdt are arbitrary. 'dmdt' is also arbitrary and can equivalently be called 'A' or any other non-reserved name. One stipulation that exists is that the definition within the 'function `dmdt=file_name(t,m)`' MUST be the same as that defined when the ode's are listed. For example, the following function m-file is incorrect

```
function dmdt=myfile(t,m)
A=4*m
```

But, the following function file is correct:

```
function dmdt=myfile(t,m)
dmdt=4*m
```

(It may appear that from this MATLAB is unable to solve any more than a first order ode, but all ode's of second order or higher can be written in the form of multiple ode's, each of first order. This will be covered once an introduction using first order ode's has been accomplished).

Once a suitable function file has been created, a run-file or executable file is created that is used to solve the ode or ode's that are within the function file. The run file must contain the following items:

- i) If global variables are used, the global command must be inserted at this point.
- ii) Depending on how the tspan, the integration interval, is defined, the lower and upper limits can be defined here. E.g. `tspan=[t0 tf];`, where t0 and tf are predefined vectors for the integration limits. Forward or reverse integration can be used; t0 does not have to be less than tf in the case of reverse integration
- iii) The initial conditions must be defined as vectors or single direction matrices.
- iv) The ode solver must then be called. The following is the syntax for the solver, and ode45 is the solver that will be used. It is a good place to start as a general solver:

```
[independent, dependent]=ode45('file_name',tspan,initial_condition_vector)
```

- v) The solution can now be plotted using the 'plot' command and then formatted either using the GUI interface or by the commands 'legend', 'xlabel', 'ylabel', 'title' and 'axis'. E.g.

```
plot(independent, dependent(:,n),independent,dependent(:,n+1)),
```

where n denotes the 1st dependent variable, then second and so on.

The following example demonstrates the setup and execution for the solution of a set of differential equations.



Example 3.1

A fluid of constant density starts to flow into an empty and infinitely large tank at 8 L/s. A valve regulates the outlet flow to a constant 4L/s. Derive and solve the differential equation describing this process over a 100 second interval.

Solution

The accumulation is described as input – output, so the ode describing the process becomes $\frac{d(\rho V)}{dt} = (8 - 4)\rho$. Since density is constant, then $\frac{dv}{dt} = 8 - 4 = 4$ in liters per second. The initial condition is that at time $t=0$, the volume inside the tank =0. The following function file 'ex31' is used to set up the ode solver.

```
function dvdt=ex31(t,v)
dvdt=4
```

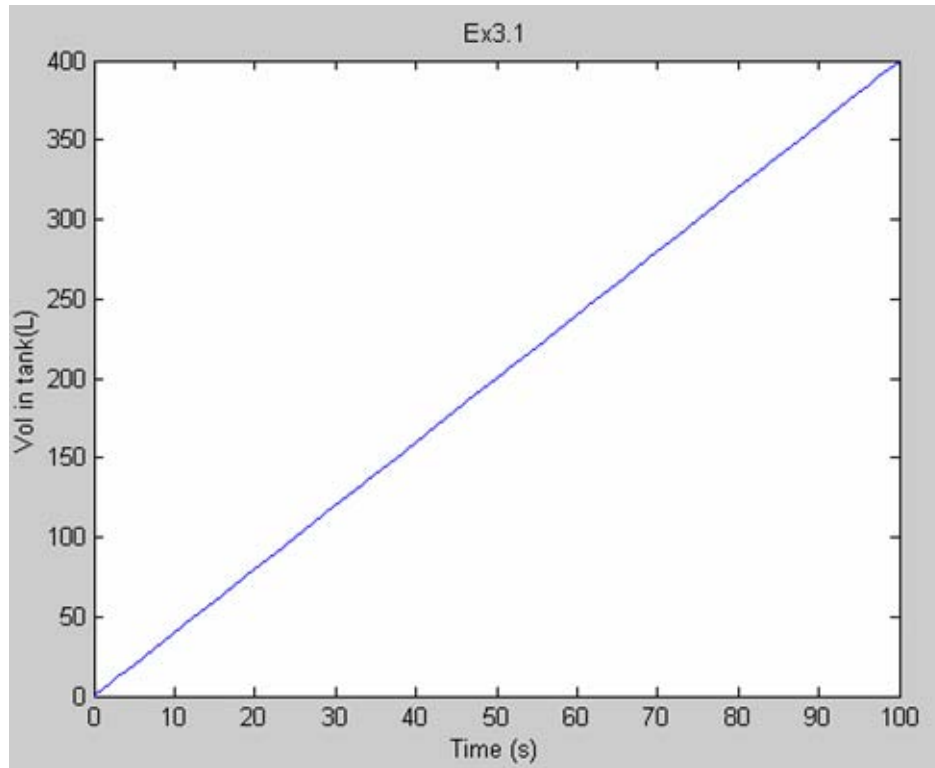
The file ex31run is used to execute the solver. The code for this file is overleaf.

```
t0=0;
tf=100;
tspan=[t0 tf]; %Integration interval
v0=0 %Initial condition

[t,v]=ode45('ex31',tspan,v0)

plot(t,v(:,1))
xlabel('Time (s)')
ylabel('Vol in tank(L)')
title('Ex3.1')
```

The plot produced is



As expected, a constant volume increase of 4L/s is described by the curve.

Systems of ode's can also be easily solved in MATLAB using the same setup as described for a single ode. Example 3.2 demonstrated how to solve a system of simultaneous ode's. In example 3.2 the 'global' command is used to define certain variables as 'shared' or 'in common' between the run file and the function file. Global variables are not necessary, but they are convenient. Using the 'global' command variables can be defined once in the run file and the global command will link them to the function file. If used, the global command *must not* contain the independent or dependent variables.

Example 3.2

The following set of differential equations describes the change in concentration three species in a tank. The reactions $A \rightarrow B \rightarrow C$ occur within the tank. The constants k_1 , and k_2 describe the reaction rate for $A \rightarrow B$ and $B \rightarrow C$ respectively. The following ode's are obtained:

$$\frac{dC_a}{dt} = -k_1 C_a$$

$$\frac{dC_b}{dt} = k_1 C_a - k_2 C_b$$

$$\frac{dC_c}{dt} = k_2 C_b$$

Where $k_1=1 \text{ hr}^{-1}$ and $k_2=2 \text{ hr}^{-1}$ and at time $t=0$, $C_a=5 \text{ mol}$ and $C_b=C_c=0 \text{ mol}$. Solve the system of equations and plot the change in concentration of each species over time. Select an appropriate time interval for the integration.

Solution

The following function file and run file are created to obtain the solution:

```
function dcdt=Ex32(t,c)
%c(1)=ca, c(2)=cb, c(3)=cc
global k1 k2
dcdt=[-k1*c(1); k1*c(1)-k2*c(2); k2*c(2)];
```

C_a , C_b and C_c must be defined within the same matrix, and so by calling C_a $c(1)$, C_b $c(2)$ and C_c as $c(3)$, they are listed as common to matrix c .

```
clc
clf
clear
global k1 k2
k1=1;
k2=2;

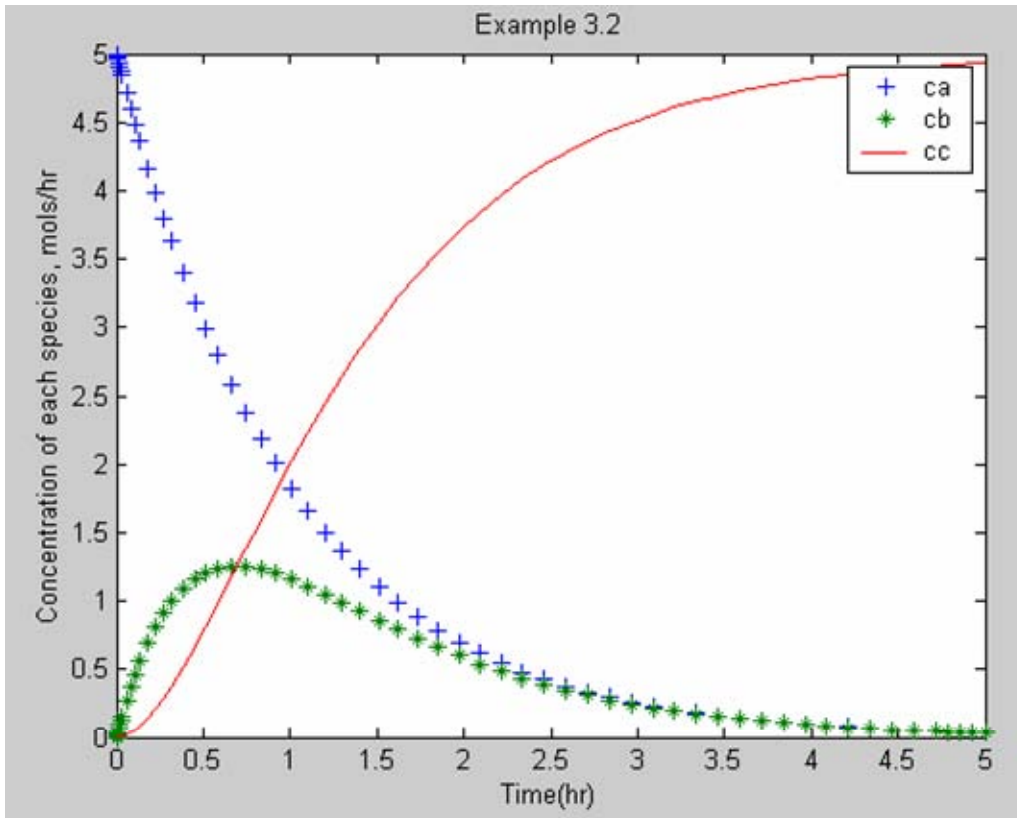
tspan=[0 5];

c0=[5 0 0];

[t,c]=ode45('Ex32',tspan,c0)

plot(t,c(:,1),'+',t,c(:,2),'*',t,c(:,3))
legend('ca','cb','cc')
xlabel('Time(hr)')
ylabel('Concentration of each species, mols/hr')
title('Example 3.2')
```

Notice that the constants k_1 and k_2 are defined (only once) in the run file, and using the 'global' command they are linked to the function file. The following curve is produced upon execution. In the 'plot' command, the + and * change the line markers so that they can be easily distinguished using a non-color printer. The appendix contains a list of available markers for plotting.



Higher order ode's

To be able to solve higher order ode's in MATLAB, they must be written in terms of a system of first order ordinary differential equations. An ordinary differential equation can be written in the form

$$\frac{d^n y}{dx^n} = f(x, y, y', y'', y''', \dots, y^{n-1})$$

and can also be written as a system of first order differential equations such that

$$y_1 = y, y_2 = y', y_3 = y'', \dots, y_n = y^{n-1}.$$

From here, the system can then be represented as an arrangement such that

$$y'_1 = y_2, y'_2 = y_3, \dots, y'_{n-1} = y_n, \text{ where } y'_n = f(x, y_1, y_2, \dots, y_n).$$

Example 3.3 demonstrates this technique.

Example 3.3

Solve the following differential equation by converting it to a system of first order differential equations, then using a numeric solver to solve the system. Plot the results.

$$Y'' + Y' + Y = 0, Y(0)=1 \text{ and } Y'(0)=0$$

To convert this 2nd order ode to a system of 1st order ode's, the following assignment is made:

$$Y = y_1 = y(1) \text{ and } Y' = y_2 = y(2),$$

then the ode can be written as the first-order system:

$$\begin{aligned} Y &= y(1) \\ \frac{dY}{dt} &= Y' = y(2) \\ \frac{d^2Y}{dt^2} &= Y'' = -(y(2) + y(1)) \end{aligned}$$

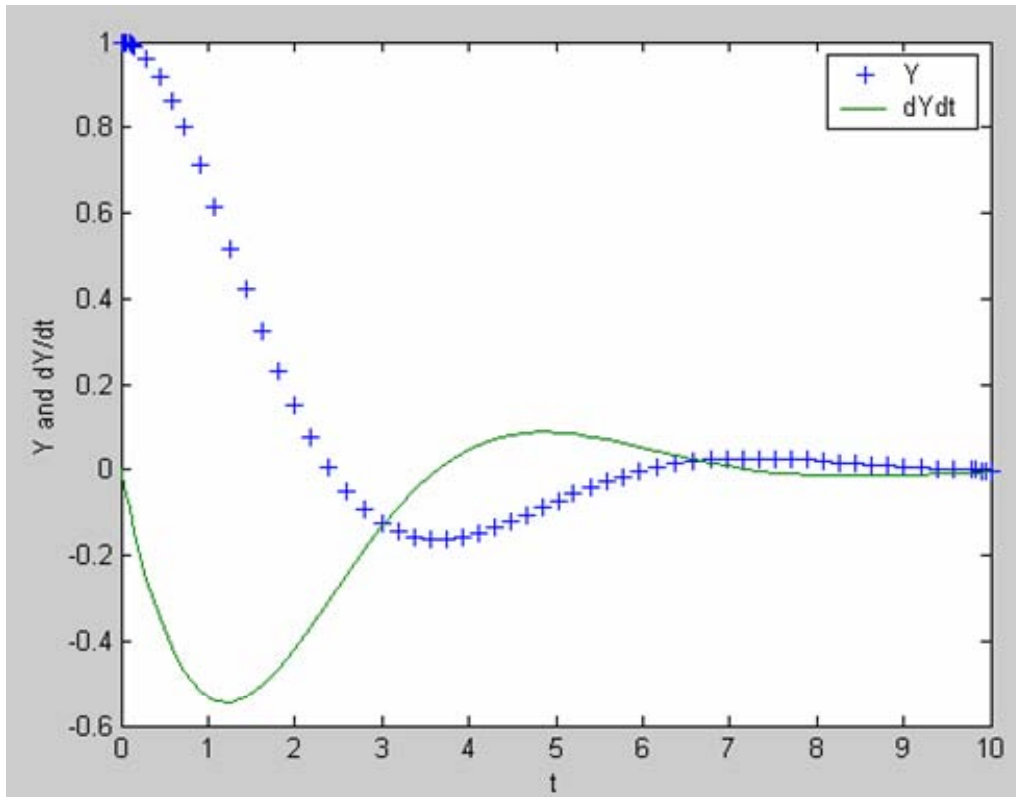
The function file containing this system can now be created and solved. The function file and run file are shown below:

```
function dmdt=Ex33(t,y)
%Solving Y'' + Y' + Y = 0
%Assign Y=y(1), Y'=y(2), Y''=y(3)
dYdt=y(2);
dY2dt=-(y(2)+y(1));
dmdt=[dYdt; dY2dt];
```

The variable dmdt is a 'dummy' variable that is used to describe the system as a whole,

```
clc
clf
clear
tspan=[0 10];
y0=[1 0];
[t,Y]=ode45('Ex33',tspan,y0)
plot(t,Y(:,1),'+',t,Y(:,2))
legend('Y','dYdt')
xlabel('t')
ylabel('Y and dY/dt')
```

The following curve is produced



The appendices of this guide are designed to be used as 'cheat-sheets' to aid in using MATLAB without simply manipulating the examples herein. Successful use of MATLAB comes from understanding how and why it works the way it does. MATLAB can be used for almost anything that requires calculation and so frequent use for simple tasks will aid in the competency of the package.

Appendix

MATLAB Glossary of commands (Part 1)

- Matrix commands (A is an arbitrary matrix)
 - » `sum(A)` → Summation of the rows of matrix A
 - » `A'` → Transposes matrix A
 - » `diag(A)` → Takes the diagonal values of square matrix A
 - » `fliplr(A)` → Flips matrix A from left to right
 - » `1:n:10` → Fills sequence of numbers, 1 to 10 with spacing n (if n is not stipulated, then MATLAB will assume n=1)

- Command Window
 - » `clear` → Clears memory of variables
 - » `clc` → Clears command window (clf will clear the figure window)
 - » `...` → Wraps text or code if required, (followed by return or enter)
 - » up arrow → Recalls last line
 - » `;` → Output suppression (eg. Does not SHOW output)
 - » `iskeyword` → Displays reserved MATLAB keywords

- Functions and other commands
 - » `plot(independent, dependent)` → A graphing command. Independent variables must be defined before using the plot command.
 - » `fplot('function',[xmin, xmax, ymin, ymax])` → Graphs a function either from an m-file or direct command, the function entered can be a file-name linking to an m-file or it can be the function itself e.g. 'sin(x)'
 - » `ezplot('function',[xmin, xmax, ymin, ymax])` → As above
 - » `fsolve('m-file-name',initial_guess,options)` → Solves non-linear continuous systems of equations. Initial guess usually in the form of a matrix, predefined. Usual option is 'optimset('fsolve')'
 - » `x=fzero('function',initial_guess)` → Numerically finds the zero of the function close to the initial guess. The function can be in an m-file or a direct command (similar to fplot). The initial guess can be entered directly, or in the form of a predefined vector. The in-line function must be in terms of x only. It cannot contain any predefined constants.
 - » `m=input('m =')` → Requests that an input variable be entered into MATLAB, which is then saved as 'm' or any other arbitrary name. Used in m-files
 - » `display('text to be displayed')` → Displays 'text to be displayed' in a matlab execution. Used in m-files

In all of the above function definitions, the ' notation must be included in the MATLAB command execution. For example, the fzero command to find the zero of x^3 , one would use the notation: `x=fzero('x^3',1)`, where 1 is the initial guess.

MATLAB glossary of commands (Part II)

Symbolic integration and differentiation

- » `syms v1 v2` → Declares variable v1 and v2 as symbolic variables, used for symbolic integration and differentiation
- » `int(exp1, independent)` → Performs symbolic integration of exp1 with respect to independent variable 'independent'
- » `diff((exp1, independent)` → Performs symbolic differentiation of exp1 with respect to independent variable 'independent'
- » `dsolve('ode1,ode2,oden', 'bc1,bc2,bcn', 'IV')`
→ Performs symbolic ode solving, where ode1, ode2 and oden are systems of ode's and bc1, bc2 and bcn are their respective boundary conditions. IV is the common independent variable. The default IV is t, so if no IV is specified, then t is assumed to be the independent variable. To indicate a derivative the capital letter 'D' is used, followed by the order then the dependent variable. So D2y represents $\frac{d^2 y}{d(IV)^2}$. For non-boundary condition problems, the 'bc1,bc2,bc3' term can be negated and the solution will be returned containing integration constants.
- » `simplify(A)` → Simplifies expression A

MATLAB glossary of commands (Part III)

The sequence of this sheet hints towards the correct sequence for solving odes in matlab

1. The different ODE solvers:

Solver	Solves These Kinds of Problems	Method
ode45	Nonstiff differential equations	Runge-Kutta
ode23	Nonstiff differential equations	Runge-Kutta
ode113	Nonstiff differential equations	Adams
ode15s	Stiff differential equations and DAEs	NDFs (BDFs)
ode23s	Stiff differential equations	Rosenbrock
ode23t	Moderately stiff differential equations and DAEs	Trapezoidal rule
ode23tb	Stiff differential equations	TR-BDF2

2. `global v1 v2 vi`

Defines global variables v1, v2, and vi. When required, the `global` command is used in all m files to declare common variables.

3. `tspan [to tf]`

Sets integration interval. `to` and `tf` must be defined prior to `tspan`. `tspan` is an arbitrary name for the row vector containing `to` and `tf`, which are also arbitrarily named.

4. `yo=[a0, a1, a2 ...]`

Initial conditions for each dependent variable in a 1st order differential equation. `yo` is an arbitrary name. A single 1st order DEQ requires only one initial condition, and each independent DEQ thereafter requires another.

5. `[Indepent, dependentbase] = ode#('filename', tspan, y0)`

Syntax to instruct MATLAB to use ODE solver called `ode#` to solve the function defined in file called `filename`, using the initial conditions described in `y0`. “#” is the number for a particular solver; generally `ode45` works as an initial try.

6. `plot (t, y(:,1), t, y(:,2), t, ...)`

Invokes `plot` command to plot `t` versus variables defined in function as `y(1)`, `y(2)`, etc.

7. Plot formatting commands

```
xlabel('X axis title')  
ylabel('Y axis title')  
axis([xmin xmax ymin ymax])  
title('Plot title')
```

8. Flow commands

Flow commands are used to change functions between different sets of specified conditions. For example, equations describing the amount of heat required by a system used to boil cold water would change once the temperature reached T_b , the boiling temperature:

```
if t<100
    Q=m*cp*(T-Tref)
else
    Q=m*heatvap
end
```

The operator 'elseif' can also be used. It acts in a similar fashion to the 'else' command except that it requires a condition to be true. For example, the following set of differential equations apply to the specified intervals

$$\frac{dm}{dt} = 4, \text{ when } m \text{ is less than or equal to } 10$$
$$\frac{dm}{dt} = 2, \text{ when } m \text{ is between } 10 \text{ and } 20$$
$$\frac{dm}{dt} = 6, \text{ when } m \text{ is greater or equal to } 20$$

In MATLAB, the 'if', 'else' and 'elseif' commands can be used to specify this system.

```
if m<=10           %If this statement is true, then
    dmdt=4         %MATLAB executes the ode dmdt=4
elseif m<20       %If the first statement is not true, I.E. we
                  %are at a point where m is greater than 10,
                  %then the 'elseif m<20' dictates that if m is
                  %less than 20, then dmdt=2, and this is the ode
                  %MATLAB executes
    dmdt=2
elseif 20<=m      %If neither of the first statements are true,
dmdt=6             %but the third 'elseif 20<=m', or if 20 is less
end               %than or equal to m, then MATLAB executes the
%               %ode dmdt=6
```

Allowable operators for the “if” statement:

>	greater than
<	less than
<=	less than or equal to
>=	greater than or equal to
==	equals
~=	does not equal

Line Markers

The following table describes the available line markers that can be used for plotting in MATLAB.

Keyboard key or symbol	Marker
+	Plus sign +
*	Multiplication sign *
o	Circle o
.	Dot .
x	Cross x
s	Square
d	Diamond
v	Downward pointing triangle
^	Upward pointing triangle
>	Right pointing triangle
<	Left pointing triangle
p	Five point star
h	Six point star