The reasonable man adapts himself to the world: the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.

— GEORGE BERNARD SHAW

Appendix B

An introduction to MATLAB

B.1 Introduction

MATLAB is a powerful, *interactive* software tool for numerical computations normally encountered in engineering and science. It is available on several platforms including personal computers using DOS, workstations, mainframes and supercomputers using UNIX. It brings together a large collection of powerful numerical algorithms (from LINPACK, EISPACK etc) for solving a variety of linear algebra problems and makes them available to the user through an *interactive* and easy-to-use interface. Very little programming effort is required to use many of its standard functions. Yet, an experienced programmer can write advanced functions and even develop entire tool boxes for specific applications like control system design and signal processing. In fact several such tool boxes already exist.

MATLAB is available in the micro computer lab in room CME 244 on nine of the IBM RS/6000-220 workstations and on the SunOS servers *ugrads1.labs* and *ugrads2.labs* running under SunOS operating system. These machines are on the internet and hence are accessible through a variety of means. Note that a student edition of MATLAB is available from the Book store. If you have a PC at home the software and the manual is a great buy.

Since MATLAB is interactive, you are encouraged to try out the examples as you read this manual. After each step, observe the outcome carefully. Since computers are programmed to respond in predictable

manner, the key to mastering them is to be very observant.

Familiarity with the basic concepts of the operating system and the networked environment are assumed. In this notes you will be introduced to some of the basic numerical and graphical capabilities of the MATLAB. In particular the following will be explored.

- Starting a MATLAB session
- Using built in HELP, DEMO features
- data entry, line editing features of MATLAB.
- Summary of some of the built in functions in MATLAB for solving problems in
  - ▷ linear algebra
  - ▷ root finding
  - ▷ curve fitting
  - ▷ numerical integration
  - ▷ integration of initial value problems
  - ▷ nonlinear equations and optimization
  - ▷ basic plotting capabilities
  - ▷ Writing MATLAB functions and scripts - the m-file

For the adventurous, here are some of its advanced features. Explore them on your own! The package provides features (or tool boxes) for signal processing, control system design, identification and optimization through what are called m-files. The graphic features support include 3D and contour plotting as well as device drivers for a variety of output devices including Postscript and meta file capabilities for producing high quality plots (not just screen dumps!). It also provides facilities for developing ones own tool boxes as well as facilities for interfacing with other high level languages such as FORTRAN or C and invoke such routines from within MATLAB.

B.2 Starting a MATLAB session

B.2.1 Direct access on AIX machines

Find a free station in room CME 244 the range of numbers 31 to 39. These are the AIX machines. Signon by entering your *userid* and *password. i.e.*,

```
Console login: userid
password: password
```

Next start the X-Window interface by entering

```
user@machine:dir> xinit
```

You may copy the files "**.mwmrc**" and "**.Xdefaults**" from the directory "**/afs/ualberta.ca/home/k/u/kumar/**". These files customize the X-windows environment when the X-server is started with the command "**xinit**". Several windows will be started up and one of them will be named "Aixterm". Normally this would be the shell "ksh" and all the paths to application program will be setup correctly for you to start running application programs. To start MATLAB simply enter

```
user@machine:dir> matlab
```

If MATLAB does not start, seek help from the system administrator!

B.2.2 Remote access from OS/2 machines or home computer

The procedure for connecting from an OS/2 machine to AIX machine via X-windows was outlined in section A.5.2. This will provide a full X-window based access to MATLAB. If you want to use MATLAB from a home computer, only VT100 based emulation support is available, unless you have X-windows client on your home computer. Follow the steps outlined in section A.5.3 to connect to an AIX machine from home using **kermit**. In either case, after a successful connection has been established, enter,

```
user@machine:dir> matlab
```

to start a MATLAB session. The current version of MATLAB is 4.0a. It has advanced 2-D and 3-D graphics capabilities. The graphics features, however, rely heavily on X-windows. Hence you must invoke MATLAB under X-windows in order to see the graphs and images on the screen. If you start MATLAB from a home computer, or a VT100 terminal on campus. you are limited to seeing the textual output only on the screen. You can still generate graphs with appropriate MATLAB commands, save them on to a file or print them, but you cannot see them on the screen.

B.3 MATLAB basics

Once you start MATLAB successfully, you should see the following prompt on your screen.

```
                    < M A T L A B (tm) >
          (c) Copyright 1984-92 The MathWorks, Inc.
                    All Rights Reserved
                       Version 4.0a
                        Dec 11 1992


Commands to get started: intro, demo, help help
Commands for more information: help, whatsnew, info, subscribe

»
```

This provides you with an interactive workspace in which you can define any number of variables and invoke any function. To exit MATLAB at any time enter

» **quit**

The commands that you enter within MATLAB are acted upon immediately. As soon as you enter a line like,

» *fname*

MATLAB checks if "fname" is a valid MATLAB command or a built in function. If so it will be executed immediately. If not MATLAB searches the path to look for a external function or a file by the name "fname.m". Such a file is called a m-file, as its file extension is "m". If such a file is found it will execute the contents of that file. If not, MATLAB will generate an appropriate error message. m-files can be either scripts (i.e. a series of valid MATLAB commands that are executed often and hence stored in a file) or they can be used to define entirely new MATLAB functions of your own. More on m-files later.

While in MATLAB, if you have the need to execute a UNIX shell command, you can do so with the escape character ! - *e.g.*, try

» **!emacs**

to invoke the emacs editor, or

When you exit the editor using ctrl-x ctrl-c, you will return to MATLAB

» **!ls -al**

to look at a list of all the files in your directory.

B.3.1 Using built in HELP, DEMO features

MATLAB provides extensive online help using commands like **help, demo, type, lookfor, whatsnew**. They are not only useful for checking the syntax of a particular function, but also for exploring and learning about new topics. Since the **help** command often generates lots of text that tend to scroll by very quickly, it is useful to enable a feature called "more" with the command,

» **more on**

When this is enabled, you will be shown one screen full of information at a time. Note that this is also UNIX feature that you can use with any program that generates lots of scrolling text. To get started with the online help, first get a list of help topics using

» **help**

Table B.1 provides a list of help topics which should give you some idea about the broad scope of MATLAB. You can obtain a list of functions under each topic (or directory) by entering **help** *topic*. For example to get a listing of general purpose commands (the first item in the above table) enter,

» **help** *general*

The list so produced is given in Table B.2 to serve as a reference material. Many of the functions that will be useful in a numerical methods course are listed in subsequent sections of this chapter. One way to become proficient in MATLAB is to use this HELP feature liberally - *i.e.*, when ever you are in doubt call on the HELP!

There is also a built in DEMO feature. To invoke this feature simply enter

Try some graphics demos. In Version 4.0, this works only under X-windows

» **demo**

It will provide you with a menu of items. Select the ones that interest you most. You can also search by keywords using the command **lookfor**. Try,

directory/topic	Brief description
matlab/general	General purpose commands
matlab/ops	Operators and special characters
matlab/lang	Language constructs and debugging
matlab/elmat	Elementary matrices and matrix manipulation
matlab/specmat	Specialized matrices
matlab/elfun	Elementary math functions
matlab/specfun	Specialized math functions
matlab/matfun	Matrix functions & numerical linear algebra
matlab/datafun	Data analysis and Fourier transform functions
matlab/polyfun	Polynomial and interpolation functions
matlab/funfun	Function functions & nonlinear numerical methods
matlab/sparfun	Sparse matrix functions
matlab/plotxy	Two dimensional graphics
matlab/plotxyz	Three dimensional graphics
matlab/graphics	General purpose graphics functions
matlab/color	Color control and lighting model functions
matlab/sounds	Sound processing functions
matlab/strfun	Character string functions
matlab/iofun	Low-level file I/O functions
matlab/demos	Demonstrations and samples
toolbox/control	Control System Toolbox
toolbox/ident	System Identification Toolbox
toolbox/local	Local function library
toolbox/optim	Optimization Toolbox
toolbox/signal	Signal Processing Toolbox
simulink/simulink	SIMULINK model analysis and construction functions
simulink/blocks	SIMULINK block library
simulink/simdemos	SIMULINK demonstrations and samples

Table B.1: List of MATLAB Ver 4.0 help topics

Function	Brief description
	Managing commands and functions
help	On-line documentation
what	Directory listing of M-, MAT- and MEX-files
type	List M-file
lookfor	Keyword search through the HELP entries
which	Locate functions and files
demo	Run demos
path	Control MATLAB's search path
	Managing variables and the workspace
who	List current variables
whos	List current variables, long form
load	Retrieve variables from disk
save	Save workspace variables to disk
clear	Clear variables and functions from memory
pack	Consolidate workspace memory
size	Size of matrix
length	Length of vector
disp	Display matrix or text
	Working with files and the operating system
cd	Change current working directory
dir	Directory listing
delete	Delete file
getenv	Get environment value
!	Execute operating system command
unix	Execute operating system command & return result
diary	Save text of MATLAB session
	Controlling the command window
cedit	Set command line edit/recall facility parameters
clc	Clear command window
home	Send cursor home
format	Set output format
echo	Echo commands inside script files
more	Control paged output in command window
	Starting and quitting from MATLAB
quit	Terminate MATLAB
startup	M-file executed when MATLAB is invoked
matlabrc	Master startup M-file

Table B.2: General purpose MATLAB Ver 4.0 commands

» **lookfor** *inverse*

which will scan for and print out the names of functions which have the keyword "inverse" in their help information. The result is reproduced below.

```
INVHILB Inverse Hilbert matrix.
ACOS    Inverse cosine.
ACOSH   Inverse hyperbolic cosine.
ASIN    Inverse sine.
ASINH   Inverse hyperbolic sine.
ATAN    Inverse tangent.
ATAN2   Four quadrant inverse tangent.
ATANH   Inverse hyperbolic tangent.
ERFINV  Inverse of the error function.
INVERF  Inverse Error function.
INV     Matrix inverse.
PINV    Pseudoinverse.
IFFT    Inverse discrete Fourier transform.
IFFT2   Two-dimensional inverse discrete Fourier transform.
UPDHESS Performs the Inverse Hessian Update.
```

B.3.2 Data entry, line editing features of MATLAB

The basic variables in MATLAB are treated as matrices. Vectors and scalar are special cases of a general matrix data structure. Similarly MATLAB handles complex variables and numbers in a natural way. Real variables, then are, special cases. Note that MATLAB is *case* sensitive.

- MATLAB remembers the previous command lines that you have entered. You can recall them by simply using the up and down arrow keys (or **ctrl-p** and **ctrl-n** key combinations) and then edit them and reenter the edited command as a new command. Basically, it supports the following emacs key definitions for command line editing.

Function	Key sequence
Previous line	ctrl-p
Next line	ctrl-n
One character left	ctrl-b
One character right	ctrl-f
One word left	esc b, ctrl-l
One word right	esc f, ctrl-r
Cursor to beginning of line	ctrl-a
Cursor to end of line	ctrl-e
Cancel line	ctrl-u
Delete character	ctrl-d
Insert toggle	ctrl-t
Delete to end of line	ctrl-k

- To *assign* a value to a variable use the assignment operator "=". For example,

  » A = [1 2 3;4 5 6;7 8 9]

  will result in a 3x3 matrix. Note that there is no need to explicitly declare the dimension of an array. Since MATLAB is case sensitive you have defined only "A " and "a" remains undefined. Similarly

  ```
  » x=[2+4*i, 3+5*i]
  ```

  will generate a complex vector with two elements. If you want to add another element enter

  what would the value of x(3) be?

  ```
  » x(4)=5+6*i
  ```

  Note that the dimension of the vector x is now automatically increased to 4. Observe that the square brackets are used in forming vectors and matrices. Semicolon is used to separate rows in a matrix. Comma is used to separate individual elements of a vector (or matrix). Parentheses are used to identify individual array elements. (Try **help punct** and **help paren**)

- After you have defined the variables A and x as above, go through the following exercise and make sure you understand the result.

  » **A(2:3,1:2)**

  Observe the use of () and : to select a sub block of A. Next, try

  What might happen if the size of sub-blocks are different?

» **B(4:5,2:3)=A(2:3,1:2)**

This demonstrates how to extract a sub-block matrix of A and assign it to another sub-block of B. Next, try,

What would be the value of x after you execute this command? Why?

» **x(4:-1:1)**

which reverses the order of elements of x. Next, try the command,

» **p=[1 3]; x(p)**

Observe that there are two commands, separated by semicolon. This example also demonstrates a powerful way of selecting specific elements of a vector. This is easily extended to matrices also. Well, try,

» **q=[2 3]; A(p,q)**

I hope you get the idea.

- To *examine* the value of a variable simply enter the name of the variable. All the variables that you define during a MATLAB session are stored in the workspace (*i.e.*, in computer memory) and they remain available for all subsequent calculations during the entire MATLAB session *i.e.*, until you "quit" MATLAB.

- You can declare any variable to be *global* in nature using,

  » **global** *A*

  If the same variable is also declared as *global* in several functions, then all those functions share the same value. To check if a variable is *global* use,

  » **isglobal(A)**

  A value of 1 is returned if it is global.

- To examine the list of variables currently defined in your workspace and the attributes of those variables, use one of the two commands "`who`" and "`whos`".

  » **whos**

- To generate a set of equally spaced values in a simple manner follow the example below:

  » $x = 0 : 0.05 : 1.0$

  will generate $x = [0\ 0.05\ 0.1\ 0.15\ 0.2\ \cdots\ 1.0]$. (Try `help colon`).

- To suppress the automatic echoing of any line that you enter from keyboard, terminate such a line with a semi-colon ";". For example

  » $x = 0 : 0.05 : 1.0;$

  will define x as before, but will not echo its value. (Try `help punct`).

- To continue the entry of a long statement onto the next line use an ellipsis consisting of three or more dots at the end of a line to be continued. For example

  » $s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 \cdots$
  » $-1/8 + 1/9 - 1/10 + 1/11$

- Anything that follows a % sign is treated as a comment. For example the following is a valid command line.

  » $I = 1 : 1 : 20$ %generating a set of integers from 1 to 20

- The numeric display format is controlled by the "format" command. Use

  » **format long**

  for 14 digits display. (Try `help format`)

- You can save the contents of a workspace with the "save" command. Try,

  » **save jnk**

  Try the command !ls jnk* Observe that the extension `.mat` has been added

  In the next few statements examine the currently defined variables, clear the workspace and load a previously saved workspace.

  » **whos**
  » **clear**

» **whos**
» **load jnk**
» **whos**

- The following matrix operations are available in MATLAB. You can use help on each of them to find out more precise information on them.

  + addition, *e.g.*, $C = A + B \Rightarrow C_{ij} = A_{ij} + B_{ij}$
  − subtraction, *e.g.*, *e.g.*, $C = A - B \Rightarrow C_{ij} = A_{ij} - B_{ij}$
  ∗ matrix multiplication, *e.g.*, $C = A * B \Rightarrow C_{ij} = \sum_k A_{ik} B_{kj}$
  ˆ Matrix power. Z = Xˆy is X to the y power if y is a scalar and X is square. If y is an integer greater than one, the power is computed by repeated multiplication. For other values of y the calculation involves eigenvalues and eigenvectors. (try **help arith**).
  ′ Matrix transpose. X' is the complex conjugate transpose of X. $X.'$ is the non-conjugate transpose. (try **help punct**).
  \ left division. $A\backslash B$ is the matrix division of A into B, which is roughly the same as **inv(A)*B** , except it is computed in a different way. If A is an N-by-N matrix and B is a column vector with N components, or a matrix with several such columns, then $X = A\backslash B$ is the solution to the equation $A * X = B$ computed by Gaussian elimination. (try **help slash**)
  / right division. B/A is the matrix division of A into B, which is roughly the same as **B*inv(A)**.

  Note that the dimensions of the matrices must be compatible for the above operations to be valid; if you attempt matrix operations between incompatible matrices an appropriate error message is generated.

- The following relational operators are available in MATLAB. Try **help relop** for additional details.

  < Less than relational operator
  > Greater than relational operator
  <= Less than or equal
  >= Greater than or equal

== equal

~= not equal

They are applied element-by-element between matrices of the same size, producing a resultant matrix consisting of 0's and 1's.

- Element-by-element multiplicative operations are obtained as follows:

operator	example	index notation
.*	C = A.*B	$C_{ij} = A_{ij}B_{ij}$
.^	C = A.^B	$C_{ij} = A_{ij}^{B_{ij}}$
./	C = A./B	$C_{ij} = A_{ij}/B_{ij}$
.\	C = A.\B	$C_{ij} = B_{ij}/A_{ij}$

B.3.3 Linear algebra related functions in MATLAB

A list of all advanced matrix related functions in MATLAB is given in Table B.3 Use the help command on each of these functions to find out more about the function and its exact syntax.

Work through the following exercise to become familiar with the usage of some of the linear algebra functions and refresh some of the results from a first year linear algebra course.

Exercise - *review of 1st year linear algebra*

- Define the matrix, A and a vector, b as

  » **A = [1 0 0.307; 0 1 0.702; -2 1 0]**
  » **b = [0.369*275;0.821*275;0];**

  Observe the two different ways semicolon has been used here. What are they?

Why was "A" echoed on the screen, while "b" was not? Is "b" a row or a column vector?

- Solve the equation $Ax = b$ using,

  » $x = A\backslash b$

- Verify that x satisfies the equation by calculating,

  » **A*x - b**

Function name	Action
	Matrix analysis
cond	Matrix condition number
norm	Matrix or vector norm
rcond	LINPACK reciprocal condition estimator
rank	Number of linearly independent rows or columns
det	Determinant
trace	Sum of diagonal elements
null	Null space
orth	Orthogonalization
rref	Reduced row echelon form
	Linear equations
\ and /	Linear equation solution; use "help slash"
chol	Cholesky factorization
lu	Factors from Gaussian elimination
inv	Matrix inverse
qr	Orthogonal-triangular decomposition
qrdelete	Delete a column from the QR factorization
qrinsert	Insert a column in the QR factorization
nnls	Non-negative least-squares
pinv	Pseudoinverse
lscov	Least squares in the presence of known covariance
	Eigenvalues and singular values
eig	Eigenvalues and eigenvectors
poly	Characteristic polynomial
hess	Hessenberg form
qz	Generalized eigenvalues
rsf2csf	Real block diagonal form to complex diagonal form
cdf2rdf	Complex diagonal form to real block diagonal form
schur	Schur decomposition
balance	Diagonal scaling to improve eigenvalue accuracy
svd	Singular value decomposition
	Matrix functions
expm	Matrix exponential
expm1	M-file implementation of expm
expm2	Matrix exponential via Taylor series
expm3	Matrix exponential via eigenvalues and eigenvectors
logm	Matrix logarithm
sqrtm	Matrix square root
funm	Evaluate general matrix function

Table B.3: Linear algebra related functions in MATLAB

- Next calculate the norm of the residual,

  » **norm(A*x - b)**

- Determine the rank of A using

  » **rank(A)**

- Carry out the LU decomposition using

  » **[L,U]=lu(A)**

  Why does "L" not appear to be lower triangular?

- Calculate the determinant of A using

  » **det(A)**
  » **det(L)*det(U)**

- Calculate the eigenvalues and eigenvectors of A.

  » **[v,d]=eig(A)**
  » **prod(diag(d))**

  Can you explain this result?

- Find out the characteristic polynomial of A and its roots.

  » **c1=poly(A)**
  » **roots(c1)**
  » **prod(ans)**

  Can you explain these results?

B.3.4 Root finding

x=fsolve('fun',x0) solution to a system of nonlinear equations (or zeros of a multivariable function).
fun(x) is a function that you should write to evaluate f(x) - *i.e.*, you define your problem in an m-file.
x0 is the initial guess for the root. [There is obviously more to it than I can describe here! Read the manual or try `help fsolve`].

fzero('fun',x0,tol) finds the root of a single nonlinear algebraic equation.
fun(x) is the external function describing your problem that you should write in a m-file.
x0 is the initial guess.

poly(V) if V is a vector then it returns the coefficients of the polynomial with roots determined by V. *i.e.*, roots and poly are inverse functions of each other.

roots(c) computes all the roots of a polynomial whose coefficients are in **c**. *i.e.*, $P_n(x) = (c_1 x^n + c_2 x^{n-1} + \cdots + c_{n+1})$.

B.3.5 Curve fitting

c=polyfit(x,y,n) least-squares curve fitting of degree n. The coefficients in descending powers of x are returned in c.

polyval(c,s) evaluates the polynomial whose coefficients are in **c** at locations determined by **s**.

yi=spline(x,y,xi) Generates a cubic spline through the data vectors (x, y) and then computes a vector of interpolated values of yi at xi.

diff(x,n) computes the **n** forward differences from the vector **x**.

The other functions of possible interest are **fmin, fmins, residue, conv , table1**.

B.3.6 Numerical integration, ordinary differential equations

quad('fun',a,b,tol,trace) computes the definite integral over the limit (a,b) using adaptive recursive Simpson's rule.
fun(x) is an external function that you must provide in a m-file.
tol is the acceptable global error. **trace** is an optional flag to monitor the integration process.

[t,y]=ode45('fun',t0,tf,y0,tol,trace) integrates a system of nonstiff differential equations of the form $dy/dt = f(y)$ using 4 and 5 order Runge-Kutta methods.
fun(y)is the external function which defines your problem. You must provide this via a m-file.
(t0,y0) is the initial condition.
tf is the final point at which you want to stop the integration.
tol is the acceptable global error in the solution. **trace** trace is the optional flag to print intermediate results.

The other functions of possible interest are **ode23, quad8**

B.3.7 Basic graphics capabilities

MATLAB ver 4.0 maintains separate graphics windows and a text window. Your interactive dialogue takes place on the text window. When you enter any graphics command, MATLAB plots that graph immediately on the graphics window. It can open several graphics windows. So, clearly commands are needed to select a specific window to be the current one. The list of graphics related commands are given in Table B.4. Work through the following exercise interactively and observe the computer response in order to understand the basic graphic capabilities of MATLAB. Make sure that you are running MATLAB under X-windows. Text following the percent sign (%) are explanatory comments. You need not enter them.

Exercise - *producing a simple graph*

```
»x=0:0.1:2*pi;          % create a vector x in the range (0,2 Pi)
»figure(1)              % open a graphics window labeled Figure 1
»figure(2)              % open a graphics window labeled Figure 2
»plot(x,sin(x))         % plot sin(x)
»hold                   % keep the graph of sin(x)
»plot(x,cos(x),'go')    % add graph of cos(x) with line type 'go'
»title('My first plot') % put some title
»xlabel('x-axis')       % label the x-axis
»ylabel('y-axis')       % label the x-axis
»print -deps fig1.eps   % produce a postscript copy in file fig1.eps
»!ls -al fig1.eps       % verify that the figure is saved in a file
»!xpreview fig1.eps     % use the postscript previewer of AIX (optional)
```

Function name	Action
	Figure window creation and control
figure	Create Figure (graph window)
gcf	Get handle to current figure
clf	Clear current figure
close	Close figure
	Axis creation and control
subplot	Create axes in tiled positions
axes	Create axes in arbitrary positions
gca	Get handle to current axes
cla	Clear current axes
axis	Control axis scaling and appearance
caxis	Control pseudocolor axis scaling
hold	Hold current graph
	Handle Graphics objects
figure	Create figure window
axes	Create axes
line	Create line
text	Create text
patch	Create patch
surface	Create surface
image	Create image
uicontrol	Create user interface control
uimenu	Create user interface menu
	Handle Graphics operations
set	Set object properties
get	Get object properties
reset	Reset object properties
delete	Delete object
drawnow	Flush pending graphics events
newplot	M-file preamble for NextPlot property
	Hardcopy and storage
print	Print graph or save graph to file
printopt	Configure local printer defaults
orient	Set paper orientation
capture	Screen capture of current figure
	Movies and animation
moviein	Initialize movie frame memory
getframe	Get movie frame
movie	Play recorded movie frames
	Miscellaneous
ginput	Graphical input from mouse
ishold	Return hold state
whitebg	Set graphics window defaults for white background
graymon	Set graphics window defaults for gray-scale monitors

Table B.4: Graphics related function in MATLAB

```
»gcf                % get current figure (should be 2)
»figure(1)          % make figure 1 the current figure
»close(1)           % close window 1
»gcf                % get current figure (should be 2)
»close(2)           % close window 1
```

In this exercise you produced the data from within MATLAB. If you have columns of data in a file, you can read them into MATLAB and plot them as above. The postscript file produced in the above example can be merged with other documents or printed on a postscript printer. Use **help print** to find out about support for other type of printers and plotters.

B.3.8 Control System Toolbox

The Control system toolbox, which uses MATLAB matrix functions, was built to provide specialized functions in control engineering. The Control system toolbox is a collection of algorithms, expressed in m-files, that implement common control system design, analysis, and modeling techniques.

Dynamic systems can be modeled as transfer functions or in state-space form. Both continuous-time and discrete-time system are handled. Conversions between various model representations are possible. Time responses, frequency responses, and root-locus measures can be computed and plotted. Other functions allow pole-placement, optimal control, and estimation.

The following example shows the use of some of the control system design and analysis tools available in MATLAB.

Example

The process transfer function, G is defined as:

$$G = \frac{1}{(s+1)(s+2)(s+3)}$$

The transfer function is entered into Matlab by entering the numerator and the denominator coefficients separately as follows:

```
» num = 1;
» den1 = [1 1];
» den2 = [1 2];
```

» **den3 = [1 3];**

The denominator polynomial is the product of the three terms. Convolution, **conv**, is used to obtain the polynomial product:

» **den = conv(den1,conv(den2,den3));**

To get an open-loop process response to a unit step change, the function **step** can be used:

```
» t = 0:0.5:5;
» y = step(num,den,t);
» plot(t,y,'*');
```

Define time in the range 0-5. Generate step response and plot.

The **Bode plot** can be obtained by first defining a vector of frequencies, and then using the function **bode**:

```
» w = logspace(-1,1);
» [mag,phase] = bode(num,den,w);
```

Generate equally spaced data in the range 10^{-1} and 10^{1}

The bode plots for amplitude ratio and phase can be obtained by typing:

```
» loglog(w,mag)
» semilogx(w,phase)
```

The root-locus can be obtained by defining a vector of desired gains, and then using the function **rlocus**:

```
» k = 20:5:70;
» y = rlocus(num,den,k);
» plot(y,'*')
```

Define gains in the range 20-70. Generate and plot the root-locus.

The closed-loop transfer function can be represented by:

$$\frac{Y}{Y_{sp}} = \frac{G_c G_p}{1 + G_c G_p}$$

The closed-loop transfer function is calculated and entered into Matlab for analysis using the same functions used in the open-loop system.
Discretization can only be done through the state-space model representation. Therefore, it is necessary to transform transfer function models to state-space models. The transfer function model can easily be transformed into the state-space model by using the function **tf2ss**:

» **[A,B,C,D] = tf2ss(num,den);**

where A, B, C, D are matrices in the differential equations $\frac{dx}{dt} := Ax + Bu$ and $y = Cx + Du$. To obtain a discretized model, the function **c2d** is used:

» **[ad,bd] = c2d(A,B,Ts); % Ts is the sample time**

This function converts state-space models from continuous time to discrete-time assuming a zero-order hold on the input. To obtain a step response on the discretized model, the function **dstep** can be used:

» **y = dstep(ad,bd,C,D,1,100);**
» **plot(y),title('step response');**

Several additional control functions that are available in Matlab are listed in Table B.5 The online help screen should be referred to for information on how to use these tools. The function **what** can be used to find out what other functions are available.

B.3.9 Producing printed output of a MATLAB session

If you want to produce a hard copy of your interactive MATLAB session, you can log a record of the entire session in a file with the **diary** command. The command

» **diary** *file*

will start recording every keyboard entry and most of the computers textual response (not graphics) in *file*. To suspend the recording, use

» **diary off**

and to resume recording you can use,

» **diary on**

The *file* contains simple text (ASCII) and can be printed on the network printer.

Function name	Purpose
Functions for model conversion	
$[num, den] = ss2tf(a, b, c, d, iu)$	State-space to transfer function
$[z, p, k] = sstzp(a, b, c, d, iu)$	State-space to zero-pole
$[a, b, c, d] = tf2ss(num, den)$	Transfer function to state-space
$[z, p, k] = tf2zp(num, den)$	Transfer function to zero-pole
$[a, b, c, d] = zp2ss(z, p, k)$	Zero-pole to state-space
$[num, den] = zp2tf(z, p, k)$	Zero-pole to transfer function
$[ad, bd] = c2d(a, b, Ts)$	Continuous to discrete
$[a, b] = d2c(ad, bd, Ts)$	Discrete to continuous
Functions for modeling	
append	Append system dynamics
connect	System interconnection
parallel	Parallel system connection
series	Series system connection
ord2	Generate A,B,C,D for a second order system
Continuous time and frequency domain analysis	
impulse	impulse response
step	Step response
lsim	Simulation with arbitrary inputs
bode	Bode and Nichols plots
nyquist	Nyquist plots
Discrete time and frequency domain analysis	
dimpulse	Unit sample response
dstep	Step response
dlsim	Simulation with arbitrary inputs
dbode	Discrete Bode plots

Table B.5: List of functions from control system tool box

B.3.10 What are m-files?

MATLAB derives its strength and wide popularity from being extensible through the m-file facility. Extensibility means that using a core set of built-in functions, users can extend the capabilities of MATLAB by writing their own functions. The functions are stored in files with the extension ".m". Any file with the extension ".m" in the search path of MATLAB is treated as a MATLAB m-file. To find out the current path of MATLAB enter,

» **path**

You can list the contents of a m-file with the **type** command. While the **help** command produces only documentation on the function, the **type** command produces a complete listing of the function. Try,

» **type sin**
» **help sin**
» **type erf**
» **help erf**

Note the "sin" is a built-in function and hence no code is listed. On the other hand "erf" is the error function implemented as a m-file and hence a complete listing is produced.

The m-files can take two forms - viz. (i) a script file and (ii) files that define entirely new functions. Such files should be in the MATLAB search path.

Example of a script file

In a script file, you can put any MATLAB commands that you would normally enter in an interactive session. Simply entering the name of the file would then execute the contents of that file. For example to enter a large matrix, create a file called `"A.m"` in your home directory using your favorite editor. This file should contain the following text.

```
B = [ 1 2 3 4 5 6 7 8 9;
      2 3 4 5 6 7 8 9 0;
      3 4 5 6 7 8 9 0 1;
      4 5 6 7 8 9 0 1 2;
      5 6 7 8 9 0 1 2 3]
b=sin(B)
```

To execute the contents of this file from within MATLAB enter,

» **A**

Note that a matrix variable "B" of size (5×9) has been defined in your workspace and the variable "b" contains the values of `sin(B)`.

In a script file you can include any such sequence of valid MATLAB commands, including program flow control commands like `for, if, while` loops *etc.* However a script file is *not* a function file and hence you cannot pass any arguments to the script. Also, when you execute a script file from the workspace, all of the variables defined in a script file become *global* variables. In contrast any variable defined within a function file is *local* to that function and only the results returned by the function become global in nature.

Example of a function file

Let us take the example of an isothermal, multicomponent flash equation, given by,

$$f(\psi) := \sum_{i=1}^{N} \frac{z_i(1-K_i)}{1+\psi(K_i-1)} = 0$$

In this equation, (z_i, K_i) are known vectors of length N and ψ is the unknown scalar variable. So we should like to write a function, say, flash(psi) that would return the value of $f(\psi)$. This function should be saved in a file named "`flash.m`" in your home directory. Such a function might look as follows:

```
function f=flash(psi)
% Calculates the flash equation f=flash(psi,K,z)
% K is a vector of any length of equilibrium ratios
% z is the feed composition (same length as K)
% K, z are known.
% psi is the vapor fraction
% This is the last line of help. Notice the blank line below!

global K z
f=((1-K).*z) ./ (1+(K-1)*psi);
f=sum(f);
```

Let us understand the anatomy of this function. The first line should always contain the keyword "`function`" in order to identify it as a func-

tion definition and not a script file. Then a list of values computed and returned by the function should appear - in the present case only "`f`" is being returned. If you have more variables being returned you would list them as "`[f1, f2, f3]` *etc.* Next, the equal sign is followed by the name of the *function*. Then the list of input variables are given in parenthesis. The next several lines begin with the percent sign (%) and hence are treated as comments. Here is the place to put the documentation on what the function does and how to use it. This is also the part that is printed out when a user asks for help on this function. A blank line signifies the end of the help message. The actual code follows the blank line. Notice the use of element-by-element multiplication of two vectors which avoids the use of do loops. How elegant!

Note that the file name is constructed by appending ".m" to the function name. In the above example the file name will be `flash.m`

Assuming that you have created a file called "`flash.m`" containing the above lines, work through the following steps.

» **help flash**
» **type flash**
» **global K z**
» **z=[.25 .25 .25 .25]**
» **K=[1 .5 .4 .1]**
» **whos**
» **flash(0.1)**

As a challenge, take up the task of modifying the function `flash` such that it will take in a vector of ψ values and return the corresponding function values!

B.3.11 Programming features

If you know any one high level programming language such a FORTRAN, C or even BASIC, you should have no difficulty in understanding the elementary program flow control features of MATLAB. A list of help topics is given in Table B.6. Let us take the example of "flash.m" and illustrate the use of "if" and "for" constructs. First we check if the length of vectors `K, z` are the same; if not we generate an error message. Note the `length` and `error` are built-in MATLAB functions. In the next section we determine the length of input vector "`x`" and build a loop to calculate the function for each element of "`x`" and store it in the corresponding element of "`f`". Use "**help relop**" and "**help lang**" to find out more about relational operators and programming language features.

Example

```
function f=flash(x)
% K is a vector of any length of equil ratios.
% z is the feed composition (same length as K)
% K, z are defined as global in main
% x is the vapor fraction
% The following is the isothermal flash eqn.

global K z

if ( length(K) ~= length(z) )
 error('Number of K values & compositions do not match')
end

n=length(x);     %Find the length of the input vector
for i = 1:n      %setup a loop for each element of x
 t=((K-1).*z) ./ (1+(K-1)*x(i));
 t=sum(t);
 f(i) = t;
end
```

B.3.12 Debugging tools

Version 4.0 of MATLAB provides for the first time some debugging tools. If you are familiar with the debugging concepts, use of this facility should be straight forward. Basically it is a tool for debugging new functions that a user develops. It provides tools for the following:

- to setup break points, where the execution of a function will be suspended and the control returned to the user, (**dbstop** *function-name*)
- examine values of variable,
- execute one line at a time. (**dbstep**)
- trace the sequence of function calls (**dbstack**),
- resume execution. (**dbcont**)
- list a function with line numbers. (**dbtype** *function-name*)

Function name	Action
MATLAB as a programming language	
script	About MATLAB scripts and M-files
function	Add new function
eval	Execute string with MATLAB expression
feval	Execute function specified by string
global	Define global variable
nargchk	Validate number of input arguments
Control flow	
if	Conditionally execute statements
else	Used with IF
elseif	Used with IF
end	Terminate the scope of FOR, WHILE and IF statements
for	Repeat statements a specific number of times
while	Repeat statements an indefinite number of times
break	Terminate execution of loop
return	Return to invoking function
error	Display message and abort function
Interactive input	
input	Prompt for user input
keyboard	Invoke keyboard as if it were a Script-file
menu	Generate menu of choices for user input
pause	Wait for user response
uimenu	Create user interface menu
uicontrol	Create user interface control
Debugging commands	
dbstop	Set breakpoint
dbclear	Remove breakpoint
dbcont	Resume execution
dbdown	Change local workspace context
dbstack	List who called whom
dbstatus	List all breakpoints
dbstep	Execute one or more lines
dbtype	List M-file with line numbers
dbup	Change local workspace context
dbquit	Quit debug mode

Table B.6: Program control related help topics

- quit debugging session. (**dbquit**)

Since all the variables with in a function are treated as local variables, their values are not available in the workspace. To examine their values within a function, you have to be able to stop the execution at a specified line within a function and examine the values of local variables. The following exercise illustrates the debugging process using the flash function developed earlier. So we assume that a "`flash.m`" file exists in your current directory.

Exercise - *debugging a function*

```
»K=[2 1 .5 .3];                 % define K values
»z=[.25 .25 .25 .25];           % define z
»global K z                     % define K z to be global variables
»whos                           % examine current variables
»dbstop flash                   % set up break point in flash
»psi=[.2:.2:.8]                 % define psi (a vector)
»flash(psi)                     % begin execution of flash, it will stop at
8 global K z                    the first executable line (line 8 here)
K»dbtype flash                  % list the function with line numbers
1 function f=flash(x)
2 % K is a vector of any length of equil ratios.
3 % z is the feed composition (same length as K)
4 % K, z are defnied as global in main
5 % x is the vapor fraction
6 % The following is the isothermal flash eqn.
7
8 global K z
9 if ( length(K)  = length(z) )
10 error('Number of K values & compositions do not match')
11 end
12 n=length(x)
13 for i = 1:n
14 t=((K-1).*z) ./ (1+(K-1)*x(i));
15 t=sum(t);
16 f(i) = t;
17 end
K»dbstop 12                     % set up a new break point at line 12
K»dbcont                        % resume execution from current line 8
12 n=length(x)                  stops before executing line 12
K»x                             % examine the value of x
```

```
K»n                    % n - should be undefined at this stage
K»dbstep               % execute one line and stop at line 13
13 for i = 1:n
K»n                    % now display value of n (should be 4)
K»dbstatus flash       % display the break points.
K»K                    % display the value of K
K»z                    % display the value of z
K»dbcont               % resume execution till the end of function.
K»dbquit               % terminate debugging of flash
```

This simple example illustrates how to set up break points, examine values of variables and step through execution one line at a time.