CD-ROM Appendix E: Matlab

Susan A. Fugett

Matlab® version 7 or 6.5 is a very powerful tool useful for many kinds of mathematical tasks. For the purposes of this text, however, Matlab 7 or 6.5 will be used only to solve four types of problems: polynomial curve fitting, system of algebraic equations, system of ordinary differential equations, and nonlinear regression. This appendix serves as a quick guide to solving such problems. The solutions were all prepared using the Student Edition of Matlab 7 or 6.5. Please note that the Matlab 7 or 6.5 software must be purchased independently of the CD-ROM accompanying this book.

CDE.1 A Quick Tour

When Matlab is opened, the command window of the Matlab Student Edition appears:

> To get started, type one of these commands: helpwin, helpdesk, or demo.

You may then type commands at the prompt.

Throughout this appendix, different fonts are used to represent Matlab **`input`** and `output`, and *italics* are used to explain function arguments.

CDE.1.1 MATLAB's Method

Matlab can range from acting as a calculator to performing complex matrix operations and more. All of Matlab's operations are performed as matrix operations, and every variable is stored in Matlab's memory as a matrix even if it is only 1x1 in size. Therefore, all Matlab input and output will be in matrix form.

CDE.1.2 Punctuation

Matlab is **case sensitive** and recognizes the difference between capital and lowercase letters. Therefore, it is possible to work with the variables "X" and "x" at the same time.

The **semicolon** and **period** play very important roles in performing calculations using Matlab. A semicolon placed at the end of a command line will suppress restatement of that output. Matlab will still perform the command, but it will not display the answer. For example, type **`beta=1+4,`** and Matlab will display the answer `beta =5`; if you then type **`alpha = 30/2;`**, Matlab will not tell you the answer. To see the value of a variable, simply type the name of the variable, **`alpha`**, and Matlab will display its value, `alpha =15`. The command **`who`** can also be used to view a list of current variables: `Your variables are: alpha beta`

The period is used **when element-by-element matrix multiplication** is performed. To perform standard matrix multiplication of two matrices, say A and B, type **`A*B`**. To multiply every element of matrix A by 2, type **`A*2`**. However, to multiply every element of "A" with the corresponding element of B, type **`A.*B`**. This element-by-element matrix multiplication will be used for the purposes of this text.

To learn more about Matlab, type `demo` at the command prompt; to see a demo about matrix manipulations, type `matmanip`.

CDE.1.3 Help

Matlab has an extensive on-line help program that can be accessed through the **`help`** command, the **`lookfor`** command, and by the **`helpwin`** command (or by choosing help from the menu bar). By typing **help** *topic*, for example **`help log`**, Matlab will give an explanation of the topic.

```
LOG Natural logarithm.
LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also LOG2, LOG10, EXP, LOGM.
```

It is likely that in many instances you will not know the exact name of the topic for which you need help. (By typing **`helpwin`**, you will open the help window, which houses a list of help topics.)

The **`lookfor`** command can be used to search through the help topics for a key word. For example, you could type **`lookfor logarithm`** and receive the following list:

```
LOGSPACE  Logarithmically spaced vector.
LOG       Natural logarithm.
LOG10     common (base 10) logarithm.
LOG2      Base 2 logarithm and dissect floating point number.
REALLOG   Real logarithm.
BETALN    Logarithm of beta function.
GAMMALN   Logarithm of gamma function.
LOGM      Matrix logarithm.
```

```
LOGSIG    Logarithmic sigmoid transfer function.
  LOG     Logarithm in a Galois field.
log.m:    %@FINTS/LOG Overloaded for FINTS object: natural
          logarithm (log base e).
log10.m:  %@FINTS/LOG10 Overloaded for FINTS object: common
          logarithm (log base 10).
log2.m:   %@FINTS/LOG2 Overloaded for FINTS object: Base 2
          logarithm and dissect floating point number.
LOG       Symbolic matrix element-wise natural logarithm.
BLKLOG    Defines a function that returns the natural loga-
          rithm of the input.
```

from which the search can be narrowed. Please note that all built-in Matlab commands are lowercase, although in help they are displayed in uppercase letters.

It is strongly recommended that students take time to explore the demo before attempting to solve problems using Matlab.

CDE.1.4 M-files

Many of the commands in Matlab are really a combination of commands and manipulations that are stored in an m-file. Users can also write their own m-files with their own commands and data. The m-files are simply text files that have an "m" extension (e.g., example1.m). The name of the file can then be called upon later to execute the commands in the m-file as though they were being entered line by line by the user at the prompt. The m-file saves time by relieving the user of the need to type lines of commands over and over and by enabling him or her to change values of one or more variables easily and repeatedly.

CDE.2 Examples

Examples of each of the four types of problems listed earlier will now be explained. Please refer to the examples in the book that were solved using Polymath.

It may be wise to type the command `clear` before starting any new problems to clear the values from all variables in Matlab's memory.

CDE.2.1 Polynomial Curve Fitting: Example 2-3

In this example, a third-order polynomial is fit to conversion-rate data.
Step 1: First, the data have to be entered as matrices by listing them between brackets, leaving a space between each entry.

```
X=[0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.85];
ra=[0.0053 0.0052 0.005 0.0045 0.004 0.0033 0.0025
0.0018 0.00125 0.001];
```

Step 2: Next, the function polyfit is used to fit the data to a third-order polynomial. **To learn more about the function polyfit, type `help polyfit` at the command prompt.**

```
p=polyfit(X,ra,3)
```
(matrix of coefficients=polyfit(ind. variable, dep. variable, order of polynomial)

```
p  =0.0092    -0.0153    0.0013    0.0053
```

The coefficients are arranged in decreasing order of the independent variable of the polynomial; therefore,

$$r_a = 0.0092X^3 - 0.0153X^2 + 0.0013X = 0.0053$$

Please note that the typical mathematical convention for ordering coefficients is

$$y = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

whereas, Matlab returns the solution ordered from a_n to a_0.

Step 3: Next, a variable *f* is assigned to evaluate the polynomial at the data points (i.e., *f* holds the r_a values calculated from the equation of the polynomial fit.) Because *X* is a 1×10 matrix, *f* will also be 1×10 in size.

`f=polyval(p,X);` *f=polyval(matrix of coefficients, ind. variable)*

To learn more about the function polyval, type `help polyval` at the command prompt.

Step 4: Finally, a plot is prepared to show how well the polynomial fits the data.

`plot(X,ra,'o',X,f,'-')` *plot(ind. var., dep. var., 'symbol', ind. var., f, 'symbol')*

where 'symbol' denotes how the data are to be plotted. In this case, the data set is plotted as circles and the fitted polynomial is plotted as a line.

The following commands label and define the scale of the axes.

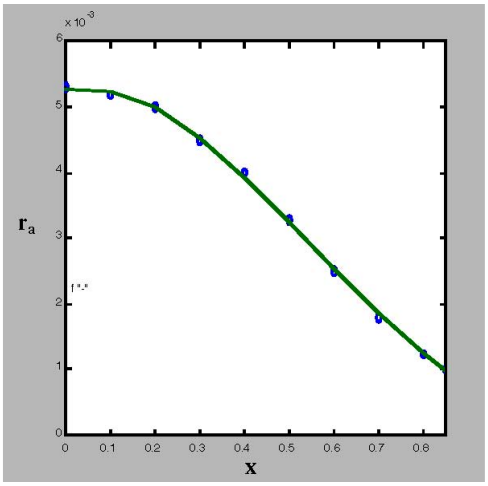
```
xlabel('X'); ylabel('ra "o", f "-"');
axis([0 0.85 0 0.006]);
```
xlabel('text'); ylabel('text'); axis([xmin xmax ymin ymax])

Please refer to `help plot` for more information on preparing plots.

The variance, or the sum of the squares of the difference between the actual data and the values calculated from the polynomial, can also be calculated.

```
variance=sum((ra-f).^2)
variance =1.6014e-008
```

The command `(ra-f)` creates a matrix the same size as r_a and *f* and contains the element-by-element subtraction of *f* from r_a. Every element of this new matrix is then squared to create a third matrix. Then, by summing all the elements of this third matrix, the result is a 1x1 matrix, a scalar, equal to the variance.



CDE.2.2 Solving a System of Algebraic Equations: Example 6-8

In Example 6-8, a system of three algebraic equations

$$0 = C_H - C_{H_0} + (k_1 C_H^{1/2} C_M + k_2 C_H^{1/2} C_X)\tau$$

$$0 = C_M - C_{M_0} + k_1 C_H^{1/2} C_M \tau$$

$$0 = (k_1 C_H^{1/2} C_M + k_2 C_H^{1/2} C_X)\tau - C_X$$

is solved for three variables, C_H, C_M, and C_X.

Step 1: To solve these equations using Matlab, the constants are declared to be symbolic, the values for the constants are entered in the equations and the equations are entered as eq1, eq2, and eq3 in the following form: *(eq1=symbolic equation)*.

It is very important that the three variables, C_H, C_M, and C_X, be represented by variables of only one character in length (e.g., h, m, and x).

$$h = C_H \qquad C_{H_0} = .021 \qquad k_1 = 55.2$$

$$m = C_M \qquad C_{M_0} = .0105 \qquad k_2 = 30.2$$

$$x = C_X \qquad t = 0.5$$

```
syms h m x;
eq1=h-.021+(55.2*m*h^0.5+30.2*x*h^0.5)*0.5;
eq2=m-0.0105+(55.2*m*h^0.5)*0.5;
eq3=(55.2*m*h^0.5-30.2*x*h^0.5)*0.5-x;
```

Step 2: Next, to solve this system of equations, type

```
S=solve(eq1,eq2,eq3);
```

The answers can be displayed by typing the following commands:

```
S.h
ans = .894358044991691397750649762302 42e-2

S.m
ans = .290846967571707015075384932598 10e-2

S.x
ans = .312664109848277367599879897106 24e-2
```

Matlab may output an algebraic expression instead of a value. If that happens, type

```
eval(S.h)
eval(S.m)
eval(S.x)
```

and the numeric values should appear. Therefore, $C_H = 0.00894$, $C_M = 0.00291$, and $C_X = 0.00313$.

CDE.2.3 Solving a System of Ordinary Differential Equations: Example 4-6

In Example 4-6, a system of two differential equations and one supplementary equation

$$\frac{d(X)}{d(W)} = \frac{\text{rate}}{fa0}; \quad \frac{d(y)}{d(W)} = -\text{alpha}\frac{(1+eps\cdot X)}{2y}; \quad f = \frac{1+eps\cdot X}{y}$$

was solved using Polymath. Using Matlab to solve this problem requires two steps: (1) Create an m-file containing the equations, and (2) use the Matlab ode45 command to integrate numerically the equations stored in the m-file created in step 1.

Part 1: Solving for *X* and *y*

Step 1: To begin, choose **New** from the **File** menu and select **M-file**. A new text editor window will appear; the commands of the m-file are to be written there.

Step 2: Write the m-file. The m-file for this example may be divided into four parts.

Part 1: The first part contains only **comments** and information for the user or future users. Each comment line begins with a percent sign because **Matlab ignores the rest of a line following %**.

Part 2: The second part is the **function** command, which **must be the first line in the m-file that is not a comment line**. This command assigns a new function to the name of the m-file. The new function is composed of any combination of existing commands and functions from Matlab. The information and commands that define the new function must be saved in a file whose name is the same as that of the new function.

Part 3: The third part of the m-file contains all other information and auxiliary equations used to solve the differential equations. It may also include the **`global`** command that allows the value for variables to be passed into or out of the m-file.

Part 4: The final part of the m-file contains the differential equations to be solved. Matlab requires that the variables of the ODEs be the elements of a single column vector. Therefore, a vector x is defined such that, for N variables, x = [var1; var2; var3; ...; varN] or $x(1)$ = var1, $x(2)$ = var2, x(N) = varN. In the case of Example 4-6, var1=X and var2=y.

Step 3: Save the m-file under the name LEP_4_6.m. This file must be saved in a directory in Matlab's path. The path is the list of places Matlab looks to find the files it needs. **To see the current path, to add a directory temporarily to the path, or to change the path permanently, use the `pathtool` command.**

Step 4: To see the m-file, type

```
type LEP_4_6
```

This command tells Matlab to type the m-file named LEP_4_6.

Step 5: Now to solve the problem, the initial conditions need to be entered from the command window. A matrix called "ic" is defined to hold the initial conditions of x(1) and x(2), respectively, and "wspan" is used to define the range of the independent variable.

```
ic=[0;1]; wspan = [0 60];
```

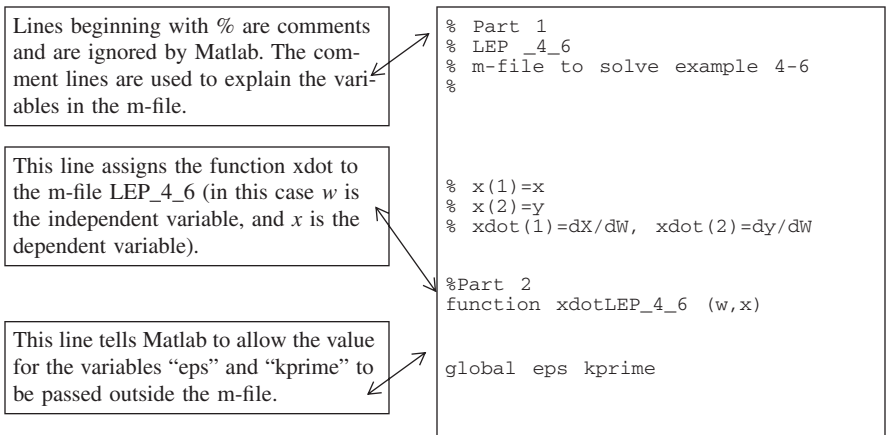
Step 6: The global command is also repeated from the command window.

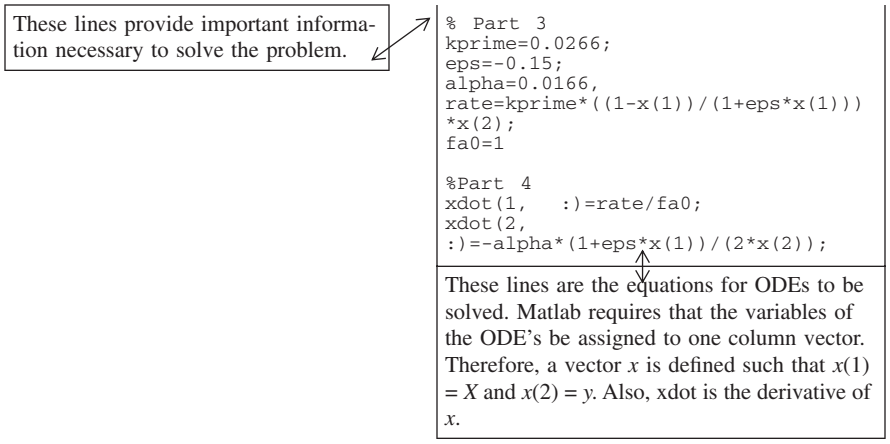
```
global eps kprime
```

Step 7: Finally, we will use the ode45 built-in function. This function numerically integrates the set of differential equations saved in an m-file.

```
[w,x]=ode45('LEP_4_6',wspan,ic);
```

[ind. var., dep. var.] = ode45('m-file', range of ind. variable, initial conditions of dep. variables)





For more information, type `help ode45` at the command prompt.

Part 2: Evaluating Variables not Contained in the Solution Matrix

Step 1: We want to solve for *f*, which is not contained in the solution matrix, *x*, but is a function of part of the solution matrix. To see the size of the matrix *x*, we type **`size(x)`**. This returns the following: `ans = 57 2`.

Therefore, *x* is a 57 by 2 matrix of the form:

$$x = \begin{bmatrix} x1(1) & x2(2) \\ x1(2) & x2(2) \\ x1(3) & x2(3) \\ \vdots & \vdots \\ x1(57) & x2(57) \end{bmatrix}$$

Step 2: Next we need to write the equation for *f* in terms of the *x* matrix.

Using Matlab notation, *x*(1:*z*,1:*y*) represents rows 1 through *z* and columns 1 through y of the matrix *x*. Similarly, *x*(1:57,1) represents all the rows in the first column of the *x* matrix, which in our case is *X*. Similarly *x*(1:57,2) defines the second column, *y*.

x(row 1:row *n*, column 1:column *n*)
X = *x*(1:57,1:1) = *x*(1:57,1)
y = *x*(1:57,2:2) = *x*(1:57,2)

The notation *x*(: , 1) also defines all the rows in the first column of the *x* matrix. This is usually more convenient than sizing the matrix, but at times, only part of the solution matrix may be needed. For example, you may want to plot only part of the solution.

So, we can write the formula *(f=(1+eps*X/y)* in the following way:

```
f=(1+eps.*x(:,1))./x(:,2);
```

Multiplication and division signs are preceded by a period to denote **element-by-element** operations as described in the Quick Tour. (The operation is performed on every element in the matrix.)

And we can write the formula for rate as follows:

```
rate=kprime.*((1-x(:,1))./(1+eps.*x(:,1))) .*x(:,2);
```

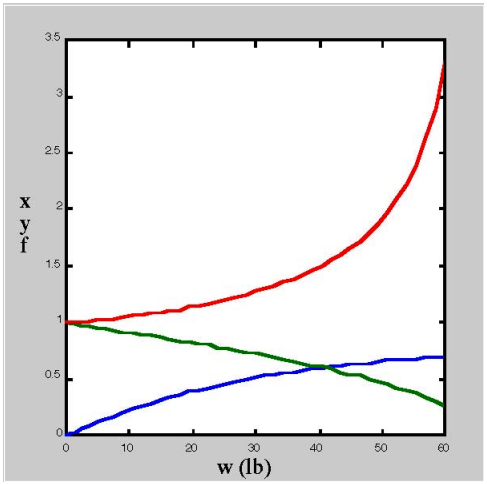
Note: This is why we used the global command. We needed the values for "eps" and "kprime" to solve for rate and *f*.

Step 3: A plot can then be made displaying the results of the computation. To plot "X", "y," and "f" as a function of "w":

`plot(w,x,w,f);` *plot(ind. var., dep. var., ind. var., dep. var.);*
`title('Example 4.6');xlabel('w (lb)');ylabel('X,y,f')`

Since the solution matrix *x* contains two sets of data (two columns) and *f* contains one column, the plot should display three lines.

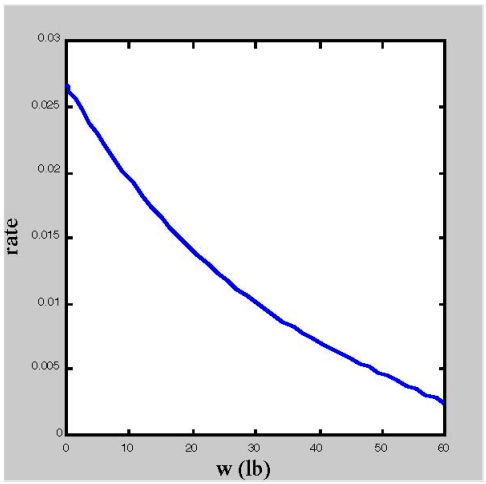
Example 4-6



To plot the rate, type

```
plot(w,rate);title('Example 4.6;xlabel('w
(lb)');ylabel('rate');
```

Example 4-6



CDE.2.4 Solving a System of Ordinary Differential Equations: Example CDR4-1

To review what you learned about Example 4-6, please examine Example CDR4-1.

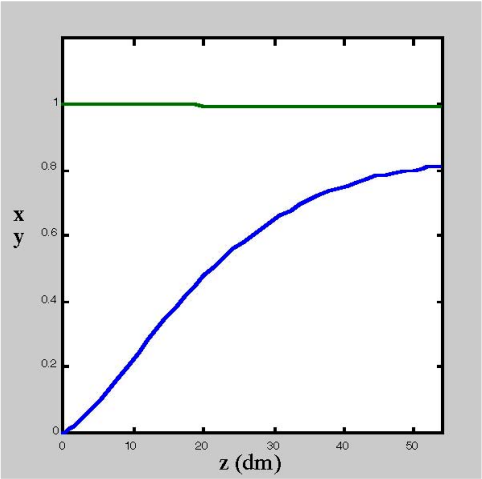
```
type CDR4-1
```

```
% "CDR4-1"
% m-file to solve CDR4-1
%
% x(1)=X
% x(2)=y
% xdot(1)=dX/dz, xdot(2)=dy/dz

function xdot=ex4_8(z,x)

Fa0=440;
P0=2000;
Ca0=0.32;
R=30;
phi=0.4;
kprime=0.02;
L=27;
rhocat=2.6;
m=44;
Ca=Ca0*(1-x(1))*x(2)/(1+x(1));
Ac=pi*(R^2-(z-L)^2);
V=pi*(z*R^2-1/3*(z-L)^3-1/3*L^3);

G=m/Ac;
ra=-kprime*Ca*rhocat*(1-phi);
beta=(98.87*G+25630*G^2)*0.01;
W=rhocat*(1-phi)*V;

xdot(1,:)=-ra*Ac/Fa0;
xdot(2,:)=-beta/P0/x(2)*(1+x(1));
```

Now, from the command window enter

```
ic=[0;1]; zspan = [0 54];
[z,x]=ode45('ex4_8',zspan,ic);
plot(z,x);title('Example4.8');xlabel('z(dm)')
;ylabel('X,y'); axis([0 54 0 1.2])
```

Example CDR4-1



CDE.2.5 Nonlinear Regression: Example 10-4

In this example, rate-pressure data are fit to four rate equations to evaluate the rate constants. These fits are then compared to determine the best rate equation for the data.

To accomplish this, an m-file is required to compute the least-squares regression for the data. Only part (a) of Example 10-4 will be demonstrated here. The rate equation for part (a) is

$$r_a = \frac{kP_E P_H}{1 + K_A P_{EA} + K_E P_E}$$

Step 1: Write the m-file.

The structure of this m-file is very similar to the m-file for Example 4-6. Please refer to Example 4-6 for comparison and further explanation.

The function *f* is assigned to the m-file

f must have an initial value. Therefore, it is given a value of 0 before the "for" loop to initiate the sum at zero.

```
% "LEP_10_4a"
% m-file to perform least-squares regression Exam-
ple 10-4 rate a
% x(1)=k; x(2)=Ke; x(3)=Ka

function f=LEP_10_4a(x)

global ra pe pea ph2 n

f=0


for i=l:n
    f=f+(ra(i)-(x(1)*pe(i)*ph2(i))/(1+x(3)*pea(i)+x
(2)*pe(i)))^2;
end

ra=[1.04 3.13 5.21 3.82 4.19 2.391 3.867 2.199 .75]
pe=[1 1 1 3 5 .5 .5 .5 .5]
pea-[1 1 1 1 1 1 .5 3 5]
ph2=[1 3 5 3 3 3 5 3 1];

n=9;
```

This "for" loop calculates the square of the difference between the actual rate and the proposed rate equation. The result of the loop is the sum of the squares we are trying to minimize and is saved in the variable *f*. This equation will be different for each rate law.

$$f = \sum_{1}^{n} (r_{a_{actual}} - r_{a_{calculated}})^2$$

Step 2: To perform the least-squares regression for the data, the `fmins` command is used to find the values of the constants that minimize the value of *f*. **Type `help fminsearch` for more information.**

```
xo=[1 1 1];
x=fminsearch('LEP_10_4',xo)
```

dep. variable = fminsearch('m-file',[matrix of initial guesses])

The solution is

```
x =3.3479 2.2111 0.0428
```

Therefore, k = 3.35; K_E = 2.21; and K_A = 0.043.

Step 3: To see how closely the solution fits the data, look at the sum of the squares to see the final value of *f* for the solution in Step 2. Assign to the variable "residual" the final value of *f*:

```
residual=LEP_10_4(x)
residual =0.0296
```

The sum of the squares (σ^2) is 0.0296.