

Module 1

Introduction to MATLAB

The purpose of this module¹ is to review MATLAB for those that have used it before, and to provide a brief introduction to MATLAB for those that have not used it before. This is a "hands-on" tutorial introduction. After using this tutorial, you should be able to:

- enter matrices
- make plots
- write simple m-files
- perform matrix operations
- use MATLAB functions

- 1.1 Background
- 1.2 Using this tutorial
- 1.3 Entering matrices
- 1.4 The MATLAB workspace
- 1.5 Complex variables
- 1.6 Matrix multiplication
- 1.7 Plotting
- 1.8 More matrix stuff
- 1.9 FOR loops and IF-THEN statements
- 1.10 m-files
- 1.11 Diary
- 1.12 Toolboxes
- 1.13 Limitations to Student MATLAB
- 1.14 Contacting MATHWORKS
- 1.15 Summary of Commands

¹ A good reference is "Introduction to MATLAB For Engineers and Scientists" by D.M. Etter (Prentice Hall, 1996).

1.1 Background

MATLAB is an interactive program for numerical computation and data visualization. It was originally developed in FORTRAN as a MATrix LABoratory for solving numerical linear algebra problems. The original application may seem boring (except to linear algebra enthusiasts), but MATLAB has advanced to solve nonlinear problems and provide detailed graphics. It is easy to use, yet very powerful. A few short commands can accomplish the same results that required a major programming effort only a few years ago.

1.2 Using This Tutorial

This tutorial provides a brief overview of essential MATLAB commands. *You will learn this material more quickly if you use MATLAB interactively as you are reviewing this tutorial.* The MATLAB commands will be shown in the following font style:

Monaco font

the prompt for a user input is shown by the double arrow

»

MATLAB has an extensive on-line help facility. For example, type `help pi` at the prompt

» `help pi`

PI `PI = 4*atan(1) = 3.1415926535897....`

so we see that MATLAB has the number π "built-in". As another example

» `help exp`

EXP `EXP(X)` is the exponential of the elements of X, e to the X.

sometimes you do not know the exact command to perform a particular operation. In this case, one can simply type

» `help`

and MATLAB will provide a list of commands (and m-files, to be discussed later) that are available. If you do not know the exact command for the function that you are after, another useful command is `lookfor`. This command works somewhat like an index. If you did not know the command for the exponential function was `exp`, you could type

» `lookfor exponential`

EXP Exponential.
EXPM Matrix exponential.

EXPM1 Matrix exponential via Pade' approximation.
 EXPM2 Matrix exponential via Taylor series approximation.
 EXPM3 Matrix exponential via eigenvalues and eigenvectors.
 EXPME Used by LINSIM to calculate matrix exponentials.

1.3 Entering Matrices

The basic entity in MATLAB is a rectangular matrix; the entries can be real or complex. Commas or spaces are used to delineate the separate values in a matrix. Consider the following vector, \mathbf{x} (recall that a vector is simply a matrix with only one row or column)

```
» x = [1,3,5,7,9,11]
```

```
x =
     1     3     5     7     9    11
```

Notice that a row vector is the default. We could have used spaces as the delimiter between columns

```
» x = [1 3 5 7 9 11]
```

```
x =
     1     3     5     7     9    11
```

There is a faster way to enter matrices or vectors that have a linear pattern. For example, the following command creates the previous vector

```
» x = 1:2:11
```

```
x =
     1     3     5     7     9    11
```

transposing a row vector yields a column vector (' is the transpose command in MATLAB)

```
» y = x'
```

```
y =
     1
     3
     5
     7
     9
    11
```

If we want to make \mathbf{x} a column-vector we use a semicolon as the delimiter between rows

```
» x = [1;3;5;7;9;11]
```

```
x =  
    1  
    3  
    5  
    7  
    9  
   11
```

To make x a row vector gain, we use the transpose

```
» x = x'
```

Say that we want to create a vector z , which has elements from 5 to 30, by 5's

```
» z = 5:5:30
```

```
z =  
    5    10    15    20    25    30
```

If we wish to suppress printing, we can add a semicolon ($;$) after any MATLAB command

```
» z = 5:5:30;
```

the z vector is generated, but not printed in the command window. We can find the value of the third element in the z vector, $z(3)$, by typing

```
» z(3)
```

```
ans =  
    15
```

Notice that a new variable, `ans`, was defined automatically.

1.4 The MATLAB Workspace

We can view the variables currently in the workspace by typing

```
» who
```

Your variables are:

```
ans      x      y      z
```

leaving 621420 bytes of memory free.

More detail about the size of the matrices can be obtained by typing

```
» whos
```

Name	Size	Total	Complex
ans	1 by 1	1	No
x	1 by 6	6	No
y	6 by 1	6	No
z	1 by 6	6	No

Grand total is (19 * 8) = 152 bytes,

leaving 622256 bytes of memory free.

We can also find the size of a matrix or vector by typing

```
» [m,n]=size(x)
```

```
m =
    1
```

```
n =
    6
```

where m represents the number of rows and n represents the number of columns.

If we do not put place arguments for the rows and columns, we find

```
» size(x)
```

```
ans =
    1    6
```

Since `x` is a vector, we can also use the `length` command

```
» length(x)
```

```
ans =
    6
```

It should be noted that MATLAB is case sensitive with respect to variable names. An X matrix can coexist with an x matrix. MATLAB is not case sensitive with respect to "built-in" MATLAB functions. For example, the length command can be upper or lower case

```
» LENGTH(x)
```

```
ans =  
     6
```

Notice that we have not named an upper case X variable. See what happens when we try to find the length of X

```
» LENGTH(X)  
??? Undefined function or variable.  
Symbol in question ==> X
```

Sometimes it is desirable to clear all of the variables in a workspace. This is done by simply typing

```
» clear
```

more frequently you may wish to clear a particular variable, such as x

```
» clear x
```

this is particularly true if you are performing a new calculation of x and the new vector is shorter than the old vector. If the new vector has length n, then all of the elements of the new x greater than x(n) will contain values of the previous x vector.

You may wish to quit MATLAB but save your variables so you don't have to retype or recalculate them during your next MATLAB session. To save all of your variables, use

```
» save file_name
```

(saving your variables does not remove them from your workspace; only `clear` can do that)

You can also save just a few of your variables

```
» save file_name x y z
```

To load a set of previously saved variables

```
» load file_name
```

1.5 Complex variables

Both `i` and `j` represent the imaginary number, $\sqrt{-1}$, by default

```
» i
```

```
ans =  
      0 + 1.0000i
```

```
» j
```

```
ans =  
      0 + 1.0000i
```

```
» sqrt(-3)
```

```
ans =  
      0 + 1.7321i
```

Note that these variables (`i` and `j`) can be redefined (as the index in a `for` loop, for example), as will be shown later.

Matrices can be created where some of the elements are complex and the others are real

```
» a = [sqrt(4), 1; sqrt(-4), -5]
```

```
a =  
      2.0000          1.0000  
      0 + 2.0000i  -5.0000
```

Recall that the semicolon designates the end of a row.

1.6 Some Matrix Operations

Matrix multiplication is straight-forward

```
» b = [1 2 3; 4 5 6]
```

```
b =  
      1      2      3  
      4      5      6
```

using the `a` matrix that was generated in section 1.5:

```
» c = a*b
```

```
c =
```

```

    6.0000          9.0000          12.0000
-20.0000 + 2.0000i -25.0000 + 4.0000i -30.0000 + 6.0000i

```

Notice again that MATLAB automatically deals with complex numbers.

Sometimes it is desirable to perform an element by element multiplication. For example, $d(i,j) = b(i,j)*c(i,j)$ is performed by using the `.*` command

```
» d = c.*b
```

```

d =
    1.0e+02 *
    0.0600          0.1800          0.3600
-0.8000 + 0.0800i -1.2500 + 0.2000i -1.8000 + 0.3600i

```

(notice the scaling that is performed when the numbers are displayed)

Similarly, element by element division, $b(i,j)/c(i,j)$, can be performed using `./`

```
» e = b./c
```

```

e =
    0.1667          0.2222          0.2500
-0.1980 - 0.0198i -0.1950 - 0.0312i -0.1923 - 0.0385i

```

Other matrix operations include: (i) taking matrix to a power, and (ii) the matrix exponential. These are operations on a square matrix

```
» f = a^2
```

```

f =
    4.0000 + 2.0000i   -3.0000
         0 - 6.0000i   25.0000 + 2.0000i

```

```
» g = expm(a)
```

```

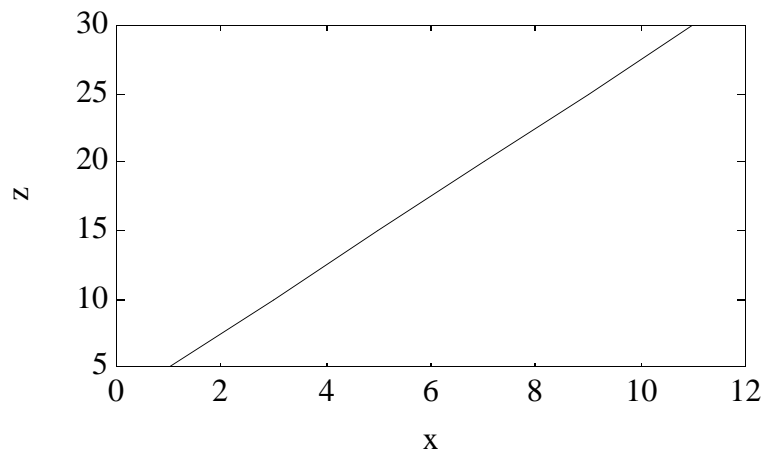
g =
    7.2232 + 1.8019i   1.0380 + 0.2151i
-0.4302 + 2.0760i   -0.0429 + 0.2962i

```

1.7 Plotting

For a standard solid line plot, simply type

```
» plot(x,z)
```



Axis labels are added by using the following commands

```
» xlabel('x')
```

```
» ylabel('z')
```

For more plotting options, type

```
» help plot
```

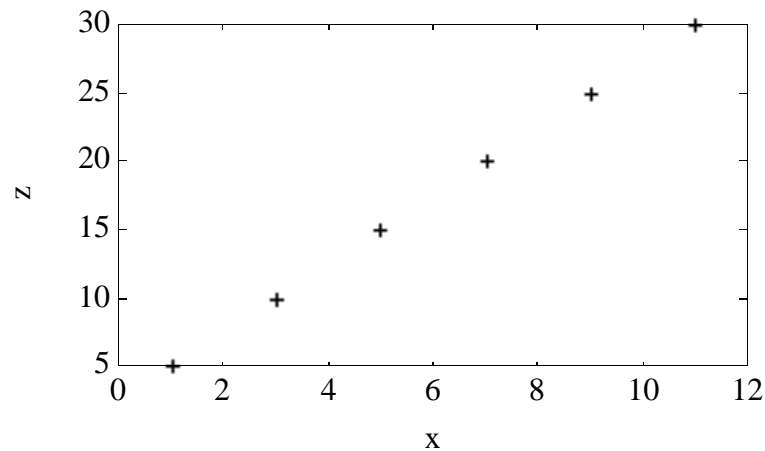
PLOT Plot vectors or matrices. `PLOT(X,Y)` plots vector `X` versus vector `Y`. If `X` or `Y` is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever lines up. `PLOT(X1,Y1,X2,Y2)` is another way of producing multiple lines on the plot. `PLOT(X1,Y1,':',X2,Y2,'+')` uses a dotted line for the first curve and the point symbol `+` for the second curve. Other line and point types are:

solid	-	point	.	red	r
dashed	--	plus	+	green	g
dotted	:	star	*	blue	b
dashdot	-.	circle	o	white	w
		x-mark	x	invisible	i
				arbitrary	c1, c15, etc.

`PLOT(Y)` plots the columns of `Y` versus their index. `PLOT(Y)` is equivalent to `PLOT(real(Y),imag(Y))` if `Y` is complex. In all other uses of `PLOT`, the imaginary part is ignored. See `SEMI`, `LOGLOG`, `POLAR`, `GRID`, `SHG`, `CLC`, `CLG`, `TITLE`, `XLABEL`, `YLABEL`, `AXIS`, `HOLD`, `MESH`, `CONTOUR`, `SUBPLOT`.

If we wish to plot discrete points, using `+` as a symbol, we can use the following

```
» plot(x,z, '+')
```



Text can be added directly to a figure using the `gtext` command.

`gtext('string')` displays the graph window, puts up a cross-hair, and waits for a mouse button or keyboard key to be pressed. The cross-hair can be positioned with the mouse. Pressing a mouse button or any key writes the text string onto the graph at the selected location.

Consider now the following equation

$$y(t) = 4 e^{-0.1 t}$$

we can solve this for a vector of t values by two simple commands

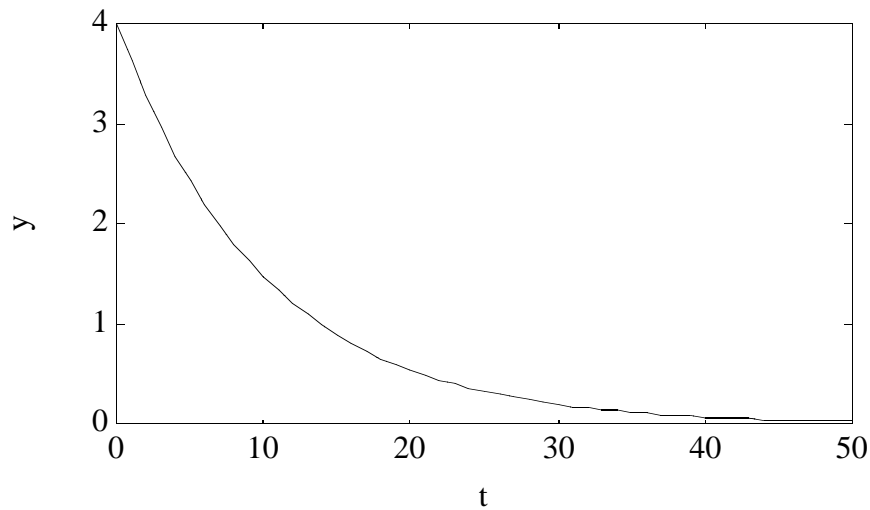
```
» t = 0:1:50;  
» y = 4*exp(-0.1*t);
```

and we can obtain a plot by typing

```
» plot(t,y)
```

Notice that we could shorten the sequence of commands by typing

```
» plot(t,4*exp(-0.1*t))
```

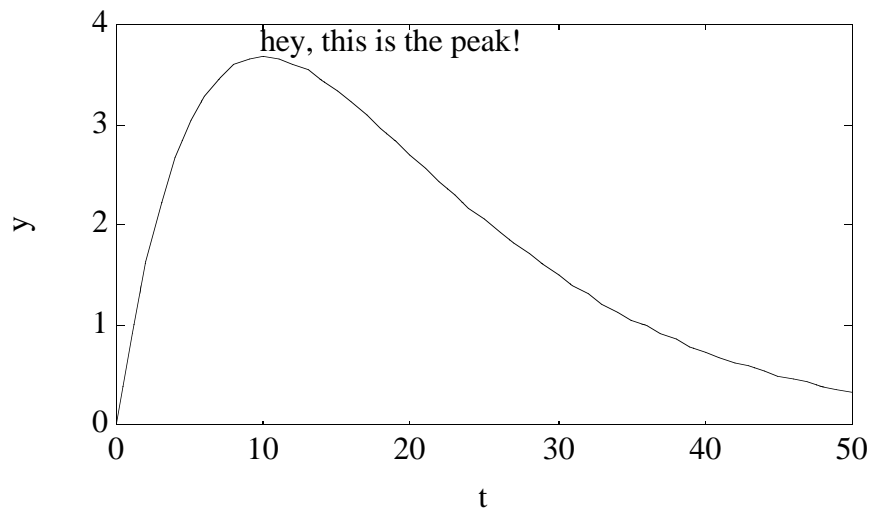


we can plot the function $y(t) = t e^{-0.1 t}$ by using

```

» y = t.*exp(-0.1*t);
» plot(t,y)
» gtext('hey, this is the peak!')
» xlabel('t')
» ylabel('y')

```



`axis('square')` will place the plot in a square box, while `axis('normal')` will change back to a normal aspect ratio.

You can also explicitly set the upper and lower bounds on the plot with

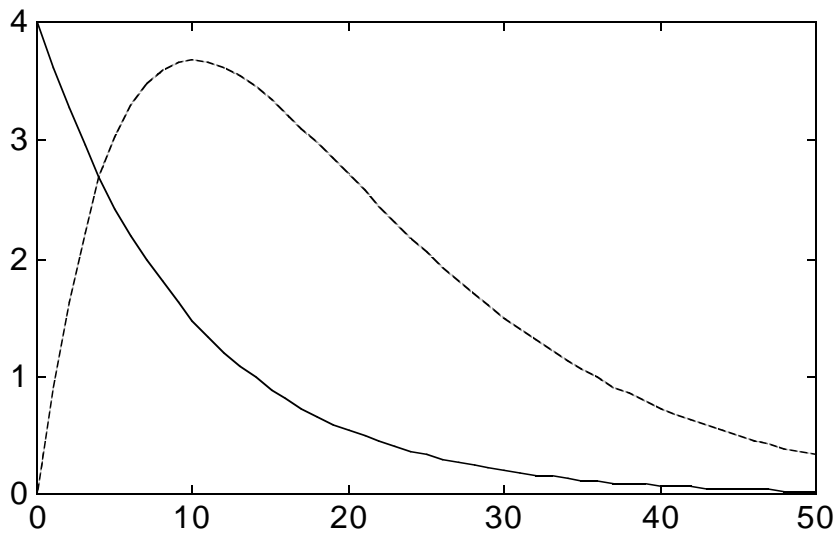
```
axis([xlow xhigh ylow yhigh])
```

for this example we would use

```
» v = [0 50 0 4];  
» axis(v);
```

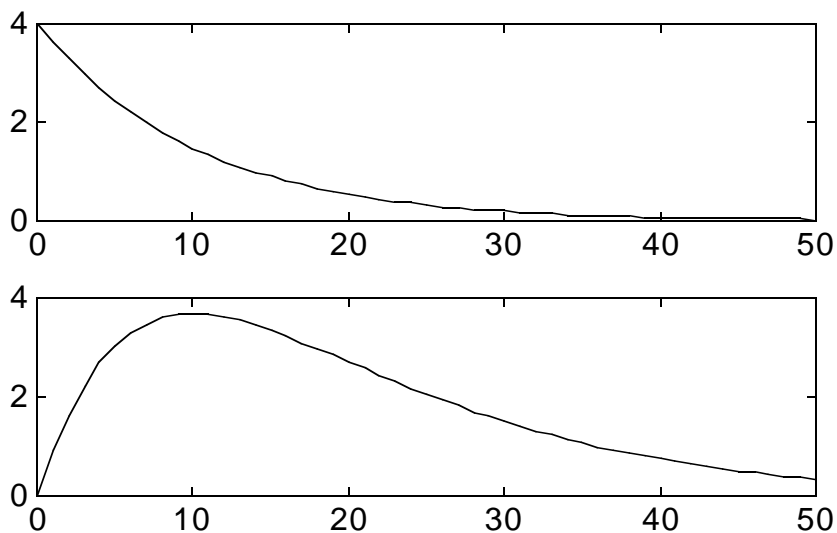
Multiple curves can be placed on the same plot in the following fashion.

```
plot(t,4*exp(-0.1*t),t,t.*exp(-0.1*t),'--')
```



The subplot command can be used to make multiple plots.

```
»subplot(2,1,1), plot(t,4*exp(-0.1*t))  
»subplot(2,1,2), plot(t,t.*exp(-0.1*t))
```



To return to single plots, simply enter `subplot(1,1,1)`.

1.8 More Matrix Stuff

A matrix can be constructed from 2 or more vectors. If we wish to create a matrix v which consists of 2 columns, the first column containing the vector x (in column form) and the second column containing the vector z (in column form) we can use the following

```
» v = [x',z']
```

```
v =  
    1     5  
    3    10  
    5    15  
    7    20  
    9    25  
   11    30
```

If we wished to look at the first column of v , we could use

```
» v(:,1)
```

```
ans =  
    1  
    3  
    5  
    7  
    9  
   11
```

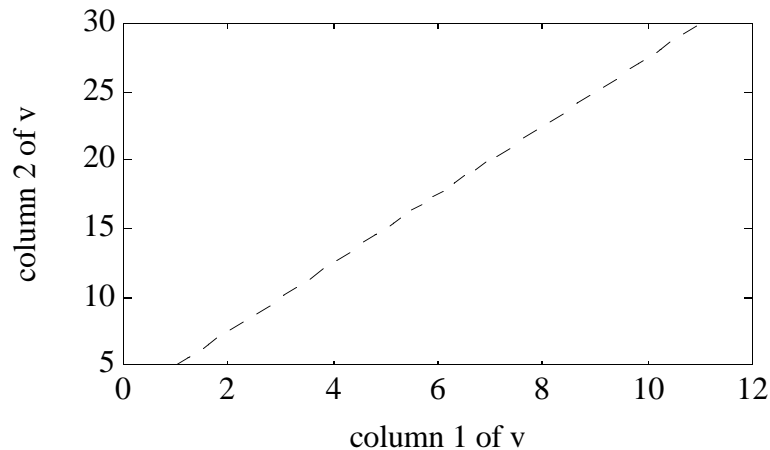
If we wished to look at the second column of v , we could use

```
» v(:,2)
```

```
ans =  
    5  
   10  
   15  
   20  
   25  
   30
```

And we can construct the same plot as before, by using ('--' gives a dotted line)

```
» plot(v(:,1),v(:,2),'--')
```



1.9 For loops and if-then statements

A `for` loop in MATLAB is similar to a `DO` Loop in FORTRAN. The main difference is that the FORTRAN `DO` loop must have an integer index variable; `for` does not have this restriction. An example of a `for` loop which is virtually identical to a `DO` loop is

```
» for k = 1:5001;
    t(k) = (k-1)*0.01;
    y(k) = sin(t(k));
end
```

Another way of implementing the same loop is increment `t` from 0 to 50 in increments of 0.01

```
» k = 0
» for t = 0:0.01:50;
    k = k + 1;
    y(k) = sin(t);
end
```

The developers of MATLAB highly recommend that you use the vectorized version of the above `for` loops

```
t = 0:0.01:50;
y = sin(t);
```

since the computation time for this method is over 200 times faster than the non-vectorized methods.

1.10 m-files

Thus far we have shown the interactive features of MATLAB, by entering one command at a time. One reason that MATLAB is powerful is that it is a language, and programs of MATLAB code can be saved for later use. There are two ways of generating your own MATLAB code: (1) script files and (2) function routines.

1.10.1 Script files

A script file is simply a sequence of commands that could have been entered interactively. When the sequence is long, or must be performed a number of times it is much easier to generate a script file.

The following example is for the quadratic map population growth model

$$x_{k+1} = \alpha x_k (1 - x_k)$$

where x_k represents the value of the population (dimensionless) at the k th time step. We have titled the file `popmod.m` and stored it in our directory. The file is run by simply typing `popmod` in the MATLAB command window.

```
% popmod.m
% population model, script file example
%
clear x,k
n      = input('input final time step ');
alpha  = input('input alpha ');
xinit  = input('input initial population ');
x(1)   = xinit;
time(1)= 0;
for k = 2:n+1;
    time(k) = k-1;
    x(k)    = alpha*x(k-1)*(1-x(k-1));
end
plot(time,x)
% end of script file example
```

Notice that we have used the MATLAB `input` function to prompt the user for data. Also note that a percent sign (%) may be used to put comments in a script or function file. Any text after a % is ignored.

1.10.2 Function routines

A more powerful way of solving problems is to write MATLAB function routines. Function routines are similar to subroutines in FORTRAN. Consider the previous example.

```
function [time,x] = pmod(alpha,xinit,n)
% population model example, pmod.m
```

```

% 29 August 1993 by Professor Bequette
clear time; clear x; clear k;
x(1) = xinit;
time(1)= 0;
for k = 2:n+1;
    time(k) = k-1;
    x(k) = alpha*x(k-1)*(1-x(k-1));
end
% end of function file example

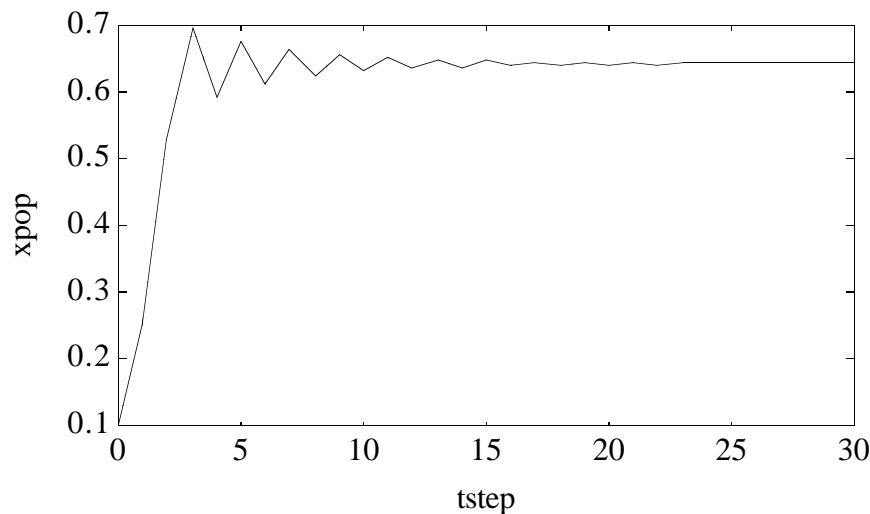
```

we can now "run" this function routine (using $\alpha = 2.8$, $x_{init} = 0.1$, $n = 30$) by typing

```

» [tstep,xpop]=pmod(2.8,0.1,30);
» plot(tstep,xpop)

```



This function routine can also be called by other function routines. This feature leads to "structured programming"; structured programs are easy to follow and debug.

MATLAB has many built-in function routines that you will use throughout this course.

1.11 Diary

When preparing homework solutions it is often necessary to save the sequence of commands and output results in a file to be turned in with the homework. The `diary` command allows this.

`diary file_name` causes a copy of all subsequent terminal input and most of the resulting output to be written on the file named `file_name`. `diary off` suspends it. `diary on` turns it back on. `diary`, by itself, toggles the diary state. Diary files may be edited later with a text editor to add comments or remove mistaken entries.

Often the consultants (or TA) wish to see a diary file of your session to assist them in troubleshooting your MATLAB problems.

1.12 Toolboxes

MATLAB Toolboxes are a collection of function routines written to solve specialized problems. The Signals and Systems Toolbox is distributed with the Student Edition of MATLAB. The newest edition of Student MATLAB also contains the Symbolic Algebra toolbox, which is a collection of routines that allows one to obtain analytical solutions to algebraic and differential calculus problems, similar to MAPLE. One toolbox used frequently in the chemical process control course is the Control Systems Toolbox.

1.13 Limitations to Student MATLAB

The Student Edition of MATLAB has a few limitations compared with the Professional version. Version 3.5 of Student MATLAB is limited to arrays with 1024 elements. The most recent version of Student MATLAB (4.0) is limited to 8192-element arrays.

1.14 Contacting MATHWORKS

MATHWORKS has a homepage on the World Wide Web. The URL for this homepage is:

<http://www.mathworks.com/>

You can find answers to frequently asked questions (FAQ) on this homepage. Also, there are a number of technical notes which give more information on using MATLAB function routines. Suggestions are made for modifying these routines for specific problems.

There is also a MATLAB newsgroup (bulletin board) which has up-to-date questions (usually supplied by MATLAB users) and answers (supplied by other users as well as MATHWORKS personnel). The newsgroup is:

`comp.soft-sys.matlab`

Before posting questions on this newsgroup it is a good idea to read the FAQ from the MATHWORKS.

1.15 Summary of Commonly Used Commands

<code>clear</code>	removes all variables from workspace
<code>clc</code>	clears command window
<code>diary</code>	save the text of a MATLAB session
<code>end</code>	end of loop
<code>exp</code>	exponential function
<code>for</code>	generates loop structure
<code>format</code>	output display format
<code>function</code>	user generated function
<code>gtext</code>	place text on a plot
<code>help</code>	
<code>hold</code>	holds current plot and allows new plot to be placed on current plot
<code>if</code>	conditional test
<code>length</code>	length of a vector
<code>lookfor</code>	keyword search on help variables
<code>plot</code>	plots vectors
<code>size</code>	size of the array
<code>subplot</code>	multiple plots in a figure window
<code>who</code>	view variables in workspace
<code>whos</code>	view variables in workspace, with more detail (size, etc.)
<code>*</code>	matrix multiplication
<code>'</code>	transpose
<code>;</code>	suppress printing (also - end of row, when used in matrices)
<code>.*</code>	element by element multiplication
<code>./</code>	element by element division
<code>:</code>	denotes a column in a matrix or creates a vector

Practice Exercises

- Plot the following three curves on (i) a single plot and (ii) multiple plots (using the `subplot` command): $2 \cos(t)$, $\sin(t)$ and $\cos(t)+\sin(t)$. Use a time period such that 2 or 3 peaks occur for each curve. Use solid, dashed, and '+' symbols for the different curves. Use roughly 25-50 points for each curve.
- Calculate the rank, determinant and matrix inverse of the following matrices (use `help rank`, `help det` and `help inv`)

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 1 \\ -1 & -2 & -1 \\ 2 & 4 & 2 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 2 & 1 \\ -1 & 4 & -1 \\ 2 & 4 & 2 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 2 & 1 \\ -1 & 4 & -1 \\ 2 & 4 & 5 \end{bmatrix}$$

3. Find $\mathbf{C}\mathbf{C}^{-1}$, where

$$\mathbf{C} = \begin{bmatrix} 1 & 2 & 1 \\ -1 & 4 & -1 \\ 2 & 4 & 5 \end{bmatrix}$$

4. a. Calculate $\mathbf{x}^T\mathbf{x}$, and

b. Calculate $\mathbf{x}\mathbf{x}^T$, where

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

5. Find the eigenvalues of the matrix

$$\mathbf{D} = \begin{bmatrix} -1 & 0 & 0 & 2 \\ 1 & -2 & 0 & 6 \\ 1 & 3 & -1 & 8 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

6. Consider the expression

$$-\mathbf{K}\mathbf{A} - \mathbf{A}^T\mathbf{K} - \mathbf{Q} - \mathbf{K}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{K} = 0$$

with

$$\mathbf{A} = \begin{bmatrix} 0 & 3 \\ 2 & -1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} -11.896 & -20.328 \\ -17.192 & -18.856 \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} 7 & 3 \\ 5 & 2 \end{bmatrix}$$

solve for \mathbf{R} .

7. Find the solutions to the equation $f(x) = 3x^3 + x^2 + 5x - 6 = 0$. Use `roots` and `fzero`.

8. Integrate the following equations from $t = 0$ to $t = 5$

$$\frac{dx_1}{dt} = -x_1 + x_2$$

$$\frac{dx_2}{dt} = -x_2$$

with the initial condition $x_1(0) = x_2(0) = 1$. Use `ode5` and plot your results.

9. Write your own function file for the following equation

$$k(T) = a \exp\left(b - \frac{c}{T} - \ln dT - eT + fT^2\right)$$

for $a = 3.33$, $b = 13.148$, $c = 5639.5$, $d = 1.077$, $e = 5.44 \times 10^{-4}$, $f = 1.125 \times 10^{-7}$

T is in units of Kelvin

Plot k as a function of T for temperatures from 373 to 600 K. (we suggest 50 points)

10. Find \hat{V} ($\frac{\text{cm}^3}{\text{gmol}}$) for the following equation of state

$$P = \frac{RT}{\hat{V} - b} - \frac{a}{T^{0.5} \hat{V}(\hat{V} + b)}$$

for $P = 13.76$ bar, $b = 44.891 \frac{\text{cm}^3}{\text{gmol}}$, $T = 333$ K, $a = 1.56414 \times 10^8 \frac{\text{cm}^6 \text{ bar}}{\text{gmol}^2 \text{ K}^{0.5}}$, $R =$ ideal gas constant in appropriate units.

Appendix Using MATLAB on the RCS

To use MATLAB on the Rensselaer Computing System

Using a workstation:

- login using your Rensselaer computing ID and password.
- select MATLAB from the applications menu
- when the MATLAB command window pops open, you are ready to begin