Appendix A

Introduction to MATLAB and Control Systems Toolbox

A.1 Introduction

MatLab (The Mathworks Inc.) is a commercial "**Mat**rix **Lab**oratory" package which operates as an interactive programming environment for scientific and engineering calculations.

There are many different toolboxes available which extend the basic functions of Matlab into different application areas; in these exercises, we will make extensive use of the Control Systems Toolbox. Matlab is supported on Unix, Macintosh, and Windows environments. Most of the statements, functions and commands are computer platform independent. Regardless of what particular computer is used, the interaction with Matlab is basically the same.

Matlab is a command-driven, interactive language, aimed at the solution of problems involving vectors and matrices. The only data structure which Matlab uses is a non-dimensional matrix (or array), the dimensions being adjusted automatically by Matlab as required.

A.2 Statements and variables

Statements have the form:

```
>> variable = expression
```

Equals "=" implies the assignment of the expression to the variable. The command prompt is two right arrows "≫".

The assignment of value 1 to the variable a is executed after the enter key is pressed.

```
>> a = 1
a =
     1
```

The value of the variable is automatically displayed after the statement is executed. If the statement is followed by a semicolon (;) the output is supressed.

```
>> a = 1;
```

The usual mathematical operators can be used in expressions. The common operators are "+"(addition), "-"(subtraction), "\" (division), "*" (multiplication), "^" (power). The order of arithmetic operations can be altered by using parentheses.

Matlab can be used in "calculator mode". when the variable name and "=" are omitted from an expression, the result is assigned to the generic variable *ans*.

```
>> 3.7*3
ans =
   11.1000
```

A.3 Entering vectors and matrices

A vector can be created by entering each element (separated by space) between brackets.

```
>> a = [1 2 3 4 5 6 9 8 7]

Matlab should return:

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

If you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2 (this method is frequently used to create a time vector):

```
>> t = 0:2:20
   t =
          0   2   4   6   8   10   12   14   16   18   20
```

You can reference individual items within the vector. To change the fifth element in the t vector:

```
>> t(5) = 23
   t =
          0   2   4   6   23   10   12   14   16   18   20
```

Manipulating vectors is almost as easy as creating them. First, suppose you would like to add 2 to each of the elements in vector 'a':

```
>> b = a + 2
     b =
          3   4   5   6   7   8   11   10   9
```

Now suppose, you would like to add two vectors together. If the two vectors are the same length simply add the two as shown below:

```
>> c = a + b
     c =
          4   6   8   10   12   14   20   18   16
```

Subtraction of vectors of the same length works exactly the same way.

Entering matrices into Matlab is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return:

```
>> B = [1 2 3 4;5 6 7 8;9 10 11 12]
   B =
          1     2     3     4
          5     6     7     8
          9    10    11    12

>> B = [ 1  2  3  4
       5  6  7  8
       9 10 11 12]

     B =
          1     2     3     4
          5     6     7     8
          9    10    11    12
```

Matrices in Matlab can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key:

```
>> C = B'
   C =
          1    5     9
          2    6    10
          3    7    11
          4    8    12
```

It should be noted that if C had been complex, the apostrophe would have actually given the complex conjugate transpose. To get the transpose, use .' (the two commands are the same if the matix is not complex). Now you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices.

```
>> D = B * C
   D =
         30     70    110
         70    174    278
        110    278    446
>> D  = C * B
   D =
        107    122    137    152
```

```
    122   140   158   176
    137   158   179   200
    152   176   200   224
```

Another option for matrix manipulation is that you can multiply the corresponding elements of two matrices using the ".*" operator (the matrices must be the same size to do this).

```
>> E = [1 2;3 4], F = [2 3;4 5], G = E .* F
     E =
          1   2
          3   4
     F =
          2   3
          4   5
     G =
          2   6
         12  20
```

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

```
>> E^3
     ans =
         37    54
         81   118
```

If wanted to cube each element in the matrix, just use the element-by-element cubing.

```
>> E.^3
     ans =
          1     8
         27    64
```

A.4 Functions

Matlab includes many standard functions. Each function is a block of code that accomplishes a specific task. Commonly used constants such as pi, and i or j for the square root of -1, are also incorporated into Matlab.

Matlab has available most trigonometric and elementary math functions of a common scientific calculator as shown in Table A.1.

Some functions for matrix properties and manipulation are given in Table A.2. To determine the usage of any function, type *help function name* at the Matlab command window.

Table A.1: Trigonometric and elementary math functions

sin(x)	Sine of the elements of x
cos(x)	Cosine of the elements of x
asin(x)	Arcsine of the elements of x
acos(x)	Arccosine of the elements of x
tan(x)	Tangent of the elements of x
atan(x)	Arctangent of the elements of x
abs(x)	Absolute value of the elements of x
sqrt(x)	Square root of x
imag(x)	Imaginary part of x
real(x)	Real part of x
conj(x)	Complex conjugate of x
log(x)	Natural logarithm of the elements of x
log10(x)	Logarithm base 10 of the elements of x
exp(x)	Exponential of the elements of x

Table A.2: Matrix manipulation

inv(x)	Inverse of a matrix x
eig(x)	Eigenvalues of the matrix x
det(x)	Determinant of matrix x
rank(x)	Rank of matrix x
eye, ones, zeros, diag	Array building functions

A.5 Polynomials

A polynomial is represented by a vector. To create a polynomial in Matlab, enter each coefficient of the polynomial into the vector in descending order. For instance, to enter the following polynomial:

$$p(s) = s^4 + 3s^3 - 15s^2 - 2s + 9$$

enter it as a vector in the following manner:

```
>> p = [1 3 -15 -2 9]
```

Matlab can interpret a vector of length n+1 as an n-th order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example,

$$p(s) = s^4 + 1$$

would be represented in Matlab as:

```
>> p = [1 0 0 0 1];
```

Some functions to be used for polynomials are given in Table A.3:

Table A.3: Functions for polynomials

roots(p)	Roots of polynomial p
polyval(p,value)	Value of polynomial p at value
conv(p,q)	Polynomial multiplication
deconv(p,q)	Divide two polynomials

A.6 Loops and logical statements

Matlab provides loops and logical statements for programming, like *for*, *while*, and *if* statements. The general forms are:
for variable = expr, statement, ..., statement *end*;
while variable, statement, ..., statement, *end*
if variable, statements, *end*

A.7 Plotting

The simple *plot(x,y)* function will plot the vector x versus vector y. In Table A.4 there are several plot formats available in Matlab:

Table A.4: Plotting

plot(x,y)	Plots vector x versus vector y
semilogx(x,y)	Plots vector x versus vector y. The x-axis is log10, the y-axis is linear
semilogy(x,y)	Plots vector x versus vector y. The y-axis is log10, the x-axis is linear
loglogx,y)	Plots vector x versus vector y. The x-axis is log10, the y-axis is log10
mesh(x,y,z,c)	Creates a 3-D mesh surface

Suppose you wanted to plot a sine wave as a function of time. First make a time vector and then compute the sin value at each time.

```
>> t=0:0.25:7; y = sin(t); plot(t,y)
```

Basic plotting is very easy in Matlab, and the plot command has extensive add-on capabilities. These capabilities include many functions such as

A.8 Toolboxes and m-files

There are many functions in Matlab of direct use in control engineering. The Control Systems Toolbox was written to make use of these commands which extend Matlab's basic facilities. This provides single commands for such things as Bode plots, time responses, state-variable feedback and so on. There are many other toolboxes available for MATLAB,

Table A.5: Functions for plotting

grid	Toggles a grid on and off in the current figure.
axis	Controls axis scaling and appearance
line	Creates a line in the current figure
title('text')	Adds 'text' at the top of the current axis
xlabel('text')	Labels the x-axis with 'text'
ylabel('text')	Labels the y-axis with 'text'
subplot	Creates axes in tiled positions

e.g: Optimization, Identification, Image Processing, Neural Networks, Spline Functions, Robust Control, Adaptive Control, Symbolic Math etc.

The way these toolboxes work is of some relevance. They are actually written in MATLAB (that is, they use the statements and commands of the MATLAB language). They consist of collections of files, called m-files (because they have the filename extension .m). Anyone can write a m-file. It is just an ASCII file created using any ASCII text editor, and containing a sequence of MATLAB commands, typed exactly as they would be from the keyboard when using MATLAB interactively with nothing else added.

Example. Create an m-file called *garbage.m* containing nothing but the following lines:

```
a=[1 2 3; 2 84; 1 7 9];
inv(a)
eig(a)
```

and simply enter the filename (without the .m extension) in response to the MATLAB prompt. This will execute the commands in the file.

```
>> garbage
```

would have exactly the same result as entering the original commands. The file garbage.m has effectively become a new MATLAB command. It is often called a script file.

Another type of m-file is a **function** file. In contrast with the script files the function files has a name following the word "function" at the beginning of the file. The filename has to be the same as the function name The function statement syntax is:

$function[output\_arguments] = function\_name(input\_arguments)$

The input arguments are variables passed to the function. The output arguments are returned.

Example. Create a m-file called *myfunc.m* which contains the following lines:

```
function [sum, product] = myfunc(x,y)
    sum = x+y;
    product = x*y;
```

The function will return the sum and product of two numbers and it can be called in the following way:

```
>> a=10;
>> b=25.9;
>> [alpha,beta]=myfunc(a,b)
or simply
>> [alpha,beta]=myfunc(10, 25.9)
```

The variable alpha will have the value of the sum of a and b and beta the value of the product.

Control System Toolbox

The Control System Toolbox builds on the foundation of MATLAB to provide specialized tools for control system engineering. The toolbox is a collection of algorithms, written primarily as M-files, that implement common control system design, analysis, and modeling techniques.

The Control System Toolbox is a core toolbox for the analysis, design, and tuning of feedback control systems. Its broad range of capabilities encompasses both classical and modern control design methods, including root locus, pole placement, and LQG regulator design. Convenient graphical user interfaces (GUIs) simplify typical control engineering tasks.

With the Control System Toolbox, you can model linear-time-invariant (LTI) systems in transfer function, zero/pole/gain, or state-space form. You can manipulate both continuous-time and discrete-time systems and convert between various model representations. You can compute and graph time responses, frequency responses, and root loci. Other functions let you perform pole placement, optimal control, and estimation. The Control System Toolbox is open and extensible, allowing you to create custom M-files to suit your particular application.

The most relevant functions are given below. For a complete description type *help function name*.

Model dynamics	
pole, eig	System poles.
zero	System (transmission) zeros.
pzmap	Pole-zero map.
damp	Natural frequency and damping of system poles.
Time response	
step	Step response.
impulse	Impulse response.
initial	Response of state-space system with given initial state.
lsim	Response to arbitrary inputs.
Frequency response	
bode	Bode plot of the frequency response.
nyquist	Nyquist plot.
margin	Gain and phase margins.

System interconnections	
parallel	Generalized parallel connection
series	Generalized series connection
feedback	Feedback connection of two systems
Classical design tools	
rltool	Root locus design GUI
rlocus	Evans root locus
rlocfind	Interactive root locus gain determination
Demonstrations	
ctrldemo	Introduction to the Control System Toolbox.