Appendix A

Introduction to MATLAB

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.

Matlab is a command-driven, interactive language, aimed at solving mathematical 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
```

Equality "=" implies the assignment of the expression to the variable. The command prompt is represented by 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 suppressed.

```
>> 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 row (line) vector can be created by entering each element (separated by space or comma) between brackets.

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

Matlab returns:

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

A vector with elements evenly spaced between 0 and 20 in increments of 2 (this method is frequently used to create a time or index vector) can be created as follows:

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

Individual items within the vector can be referenced. To change the fifth element in the t vector:

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

Suppose that we want to add 2 to each of the elements in vector a:

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

Adding two vectors of the same length:

```
>> 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 done by entering each row 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. The transpose of a matrix is obtained by 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 is complex, the apostrophe results in the complex conjugate transpose. To obtain the transpose, use "`.'`" (the two commands are the same if the matrix is not complex). Now we might multiply the two matrices B and C. 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
```

Element-wise multiplication is obtained using the "`.*`" operator (for matrices of the same sizes).

```
>> 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
```

Square matrices can also be raised to a given integer power.

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

Element-wise power is obtained by using "`.^`"

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

A.4 Matlab functions

Matlab includes many standard functions. Each function is a block of code that accomplishes a specific task. Commonly used constants such as 'pi' (π), and 'i' or 'j' for the square root of -1, are also incorporated into Matlab. e (the base of natural logarithm) is not included, to obtain e one should use $\exp(1)$ (exponent of 1).

Matlab has available most trigonometric and elementary math functions as shown in Table A.1.

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
sign(x)	Sign of x

Some functions for matrix properties and manipulation are given in Table A.2. To see how a function should be used, type *help function name* at the Matlab command window.

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, the coefficients of the polynomial should be entered 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 nth order polynomial. Thus, also zero coefficients must be entered in the proper places. For instance

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

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 = expression, statement, ..., statement *end*;
while variable, statement, ..., statement, *end*
if variable, statements, *end*

Table A.4: Plotting

plot(x,y)	Plots vector y versus vector x.
semilogx(x,y)	Plots vector y versus vector x. The y-axis is log10, the x-axis is linear.
semilogy(x,y)	Plots vector y versus vector x. The x-axis is log10, the y-axis is linear.
loglogx,y)	Plots vector x versus vector y. Both axes are logarithmic.
mesh(x,y,z)	Creates a 3-D mesh surface.
contour(x,y,z)	Plots the countour of a function (2D).

A.7 Plotting

The simple *plot(x,y)* function will plot the vector y versus the vector x.

Let us plot a sine wave as a function of time. First define the time vector and then compute the $\sin$ value at each time index.

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

The result is shown in Figure A.1.

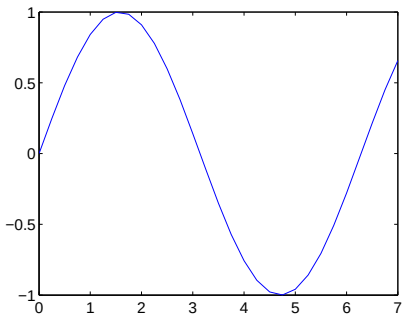

Figure A.1: Plot of $\sin(t)$.

Basic plotting is very easy in Matlab, and the plot command has extensive add-on capabilities. These capabilities include many functions such as those presented in Table A.5.

Table A.5: Plotting accessories

grid	Toggles a grid on and off in the current figure.
axis	Controls axis scaling and appearance
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

Let us now see 3D plotting. Consider the function $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x, y) = x^2 + y^2$. To graphically represent this function, first the values of x and y have to be defined using *meshgrid*.

```
>> [x,y]=meshgrid(-1:.01:1);
```

Different grids can also be defined for x and y, as follows:

```
>> [x,y]=meshgrid(-1:.01:1,-2:.01:2);
```

Then, the graphic (see Figure A.2) is obtained by using *mesh*:

```
>> mesh(x,y,x.^2+y.^2)
```

This in effect means that the value of the function is computed for each point on the grid generated by 'meshgrid'. In many cases more conclusions can be drawn if, instead of the 3D

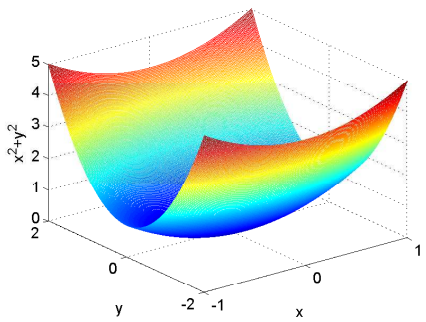

Figure A.2: Graphical representation of $x^2 + y^2$.

representation, one inspects the projection of the representation on the x-y plane, specifically the *contour* plot, see Figure A.3:

```
>> contour(x,y,x.^2+y.^2)
```

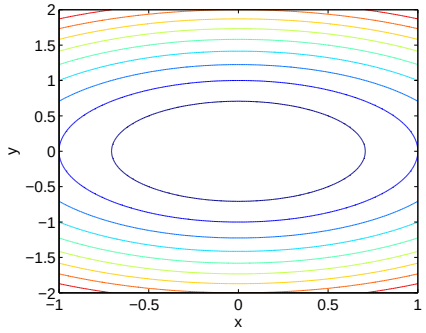

Figure A.3: Contour plot of $x^2 + y^2$.

A.8 Toolboxes and m-files

The functions in Matlab are in general grouped in toolboxes. For instance, the Control systems Toolbox contains functions of direct use in control engineering. It provides commands for Bode plots, time responses, control-design and so on. There are many other toolboxes available for MATLAB, e.g: Optimization, Symbolic Math, Identification, Image Processing, Neural Networks, Spline Functions, Robust Control, Adaptive Control, etc.

The toolboxes 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). An m-file is an ASCII file created using any text editor, and containing a sequence of MATLAB commands, typed exactly as they would be from the keyboard when using MATLAB.

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. Such files are called script files.

Another type of m-file is a **function** file. In contrast with the script files the function files have a name following the word "function" at the beginning of the file. The filename has to be the same as the function name, and it must not start with numbers or contain mathematical operations. 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.

A.9 Symbolic math

The Symbolic Math toolbox allows one to work with symbolic variables. A symbolic variable x is defined as

```
>> syms x
```

Standard Matlab operations and functions can be used on symbolic variables, and the returned result will be symbolic, i.e., generic variables that can be used without values.

```
>> syms x y z
>> f=x+y
f =
x+y
>> g=2*y+z
g =
2*y+z
>> h=f*g
h =
(x+y)*(2*y+z)
>> f=[x y^2 z]
f =
[   x, y^2,   z]
```

To evaluate a symbolic variable at a given value, one can either use *subs* (for one variable)

```
subs(f,x,1)
ans =
[   1, y^2,   z]
```

or *eval*, after specifying the values

```
>> x=1; y=2; z=3;
>> eval(f)
ans =
     1     4     3
```

Symbolic functions can be differentiated or integrated, using *diff* or *int*. In general, symbolic operations must be performed **before** one evaluates the variables.

The symbolic function h defined above can be differentiated wrt. one variable as

```
diff(h,x)
ans =
2*y+z
diff(diff(h,x),y)
ans =
2
```

or wrt. all the variables as

```
g=jacobian(h)
g =
[     2*y+z, 4*y+z+2*x,        x+y]
```

the result being the vector of partial derivatives. Differentiation of a vector function wrt. all the variables results in a symbolic matrix (function):

```
jacobian(g)
ans =
[ 0, 2, 1]
[ 2, 4, 1]
[ 1, 1, 0]
```

One can also *solve* symbolic equations

```
>> syms x
>> sol=solve('x^2+3*x=2')
sol =
 -3/2+1/2*17^(1/2)
 -3/2-1/2*17^(1/2)
```

or systems of equations

```
>> syms x y
>> sol=solve('x^2+3*x=2','x+y=3')
sol =
    x: [2x1 sym]
    y: [2x1 sym]
>> sol.x
ans =
 -3/2+1/2*17^(1/2)
 -3/2-1/2*17^(1/2)
>> sol.y
ans =
 9/2-1/2*17^(1/2)
 9/2+1/2*17^(1/2)
```

The results are treated as symbolic variables. To obtain the exact value, they must be evaluated:

```
>> eval(sol.x)
ans =
    0.5616
   -3.5616
>> eval(sol.y)
ans =
    2.4384
    6.5616
```