

A summary of basic operations in Matlab

Electrical and Computer Engineering Department

1 Introduction

This document provides an introduction and some exercises to familiarize you with **Matlab**. **Matlab** is widely available and priced so as to be accessible to students. **Matlab** provides excellent numeric capabilities and has recently incorporated access to symbolic capabilities as well. It is also a full-blown computer language which, while it is somewhat slow because it is interpreted, provides substantial capabilities and has become a standard tool for algorithm development in signal processing, controls, and many other areas of engineering.

You are encouraged to peruse the book *The Student Edition of Matlab*, which comes with the student edition of **Matlab**. It is also available at the Bookstore. A single copy of it is also on reserve at the library.

The most important thing is to *play around* with the tool to learn how it works. Ask questions of the computer and your fellow students. While directions have been given on some of this, it is impossible to give you all the information you need. You will benefit from learning how to use this — take some time to become familiar with it. (And, by the way, exercise some independence on this — the author will not babysit you through your learning curve.)

You should turn in (as your homework) the printout of all numbered exercises as well as your manual calculations.

2 Background

Matlab is primarily centered around matrices and vectors. These are entered delimited with square brackets. For example, to enter the vector

$$v = [1, 2, 3, 4]$$

you would type

```
v = [1,2,3,4]
```

or

```
v = [1 2 3 4]
```

That is, either commas or spaces can separate the elements. Entering the matrix

$$m = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 2 \\ 7 & 5 & 3 \end{bmatrix}$$

may be typed by

```
m = [1 2 3; 6 5 2; 7 5 3]
```

or

```
m = [1 2 3  
6 5 2  
7 5 3]
```

That is, either semicolons or returns may separate rows of the matrix. The results of the most recent operation are displayed after you press Enter. If you do not want the results displayed, you can terminate the line with a semicolon. For example,

```
j2 = 1:500;
```

assigns the vector of numbers to `j2`, but does not display the result. For long computations, this can save a lot of screen space. You should also note that when the results of a computation are not assigned to any variable, the result is automatically assigned to the variable `ans`, which you can then use in the next computation. (Note that `ans` may be overwritten after any computation, so you should be careful in its use.)

Arithmetic operations are typed as is typical. The transpose operator is the single quote `'`.

You can enter a range of numbers using a colon. For example,

```
j = 1:4
```

produces the vector

$$j = [1, 2, 3, 4].$$

The `:` operator is also used to indicate an entire range in either the row or column direction. For example, if `m` is the matrix entered before, `m(:,1)` picks off the entire first column of `m` and `m(1,:)` picks off the entire first row of `m`.

Note that there is no graphical interface, as there is for `Mathcad`. If you have any agility with typing, you will find that this saves time over pointing and clicking. It is possible to run `Matlab` inside a Microsoft Word document, so that you can achieve a notebook-like environment. This is not described in this document, however.

Complex numbers may be entered as `1-2i` or `1-2j` — either i or j may be used as $\sqrt{-1}$ in `Matlab`. (This means that these variables should *not* be used for other things, as this might be confusing to you and the program.)

For any of the commands, you may simply type `help commandname`. For example, type `help eig` to learn about how to compute eigenvectors and eigenvalues in `Matlab`.

In the exercises below, some commands that may be useful to you in your computations are given in parentheses. To save your results into a file, use the command

```
diary filename
```

This saves the keyboard input and text output into a file named `filename`.

Perform the following basic operations:

1. Basic arithmetic:

```
r=3  
vol = 4/3*pi*r^3
```

Note that `pi` is a built-in variable.

2. Computation on a whole array:

```
j=1:5  
(3/2).^j
```

This forms the array `[1, 2, 3, 4, 5]`, then raises $(3/2)$ to each element in the array. The dot in the operator `.^` says “apply the operator `^` to each element.”

3. `Matlab` has excellent plotting capability — perhaps the best of any computational tool around. In this exercise you will meet only the barest essentials of the plotting capability. For more information, see the help on the computer or the software manual.

Plot $\sin(2\pi j/50)$ (on the x-axis) versus $\cos(6\pi j/50)$ (on the y-axis) for $j = 1..51$:

```
j=(1:51)'  
x = sin(2*pi*j/50);  
y = cos(6*pi*j/50);  
plot(x,y)
```

Note that the statement `x = sin(2*pi*j/50)` produces an array at the output, with each component in the output array coming from a component in the input array.

4. Plot $j = 1..51$ (on the x-axis) versus $\sin(2\pi j/50)$ (on the y-axis):

```
plot(j,x)
```

5. Compute

$$\frac{-\sqrt{16} - 4}{2e^4 + 1}$$

by entering

```
e = exp(1)
(-sqrt(16)-4)/(2*e^4 + 1)
```

Note that e (the base of natural logarithms) is not a built-in number.

Operations on matrices are very easy to do, since this forms the heart of **Matlab**.

1. Enter the following matrix (see page 1):

$$m = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 2 \\ 7 & 5 & 3 \end{bmatrix}$$

2. Find the eigenvalues and eigenvectors of m . (`eig`). Make sure you know which are eigenvalues and which are eigenvectors.
3. Find the determinant and inverse of m . (`det`, `inv`)
4. Solve the set of equations

$$\begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 2 \\ 7 & 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$$

by entering

```
b = [4;2;1]
x = m\b
```

(See the operations `\` and `/` under `help slash`.)

Matlab can also solve for roots of polynomials. Enter the polynomial coefficients into a vector, then use the `roots` function. For example, to find the roots of

$$x^3 - 10x + 2,$$

form the vector

$$v = [1, 0, -10, 2]$$

(note that the coefficients go in *decreasing* powers of x , and that all coefficients, even if zero, must be included). Then use the function `roots(v)`.

Matlab also has a function that goes from the roots of a polynomial back to the polynomial. The function is called `poly`.

1. Find the roots of the polynomial $x^3 - 10x + 2$.

```
v = [1,0,-10,2]
rts = roots(v)
```

2. Now find the polynomial that has the roots just found. Verify that it is the same as the polynomial you just started with.

```
v1 = poly(rts)
```

3. Find the roots of the polynomial $x^2 - 5x + 10$. Check the results using the quadratic formula (using `Matlab` for your computations).

Try to following calculus type stuff.

1. Compute $\sum_{k=1}^{20} k^2$. Enter

```
k=(1:20)'  
k = k.^2  
sum(k)
```

The first statement makes a column vector of all of the components. The second statement squares each component. (The `.` means component-wise operations). The third statement produces the sum of the vector.

2. Compute $\int_0^5 \sin(t) dt$. Enter

```
quad('sin',0,5)
```

The command `quad` is short for quadrature, which is the numerical analyst's word for integration. Due to the way the `quad` function is set up, the integrand must be a function that returns a vector of values for a vector of inputs. Practically what this means is that for most interesting problems you would have to create your own function. Thus, to integrate

$$\int_0^t t \sin(2\pi t) dt$$

you would have to create your own function to compute $t \sin(2\pi t)$. This is not hard, but would take us too far afield on this introduction.

You may have noticed by now that all the computations return *numbers* as their answers. Numbers are good, but they are not all there is to mathematics. You calculators could have done as well.

Symbolic operations may also be done. Because `Matlab` was originally a tool for numeric computations, some of the syntax for symbolic operations are somewhat more awkward than might be desired, but it works. (`Matlab` 5.0 has a better interface; the instructions here refer to `Matlab` 4.2).

1. To compute

$$\sum_{k=1}^n k^2$$

enter

```
s1 = symsum('k^2','k',0,'n')
```

Now factor the result to put it in more conventional form

```
s2 = factor(s1)
```

Now to find a particular numeric value, substitute a number in place of `n`.

```
subs(s2,20,'n')
```

Compare this result with the one obtained numerically previously.

Others symbolic operations that may be of interest are `sym`, `sym2poly`, `symadd`, `syndiv`, `symmul`, `symop`, `sympow`, `numden`, `compose`, `sub`, `factor`, `simplify`, `simple`. You can use the `help` function to learn more about these.

2. To compute the integral

$$\int_0^T t \cos(2\pi nt) dt$$

enter

```
i1 = int('t*cos(2*pi*n*t)', 't', 0, 'T')
```

This may be simplified by using

```
simple(i1)
```

Verify by integrating by hand the `Matlab` has done its job correctly.

3. Try a derivative. To take the derivative of

$$e = \frac{2x^2 - 3x + 1}{x^3 + 2x^2 - 9x - 18}$$

enter

```
e = '(2*x^2 -3*x +1)/(x^3 + 2*x^2 - 9*x -18)'  
de = diff(e, 'x')
```

You might also want to simplify the result by entering

```
simplify(de)
```

Verify by doing the derivative by hand that `Matlab` is doing it right.

4. To enter the matrix

$$m = \begin{bmatrix} d^2 & 2 & 7 \\ d & d^3 & 9 \\ 1 & 5 & \frac{1}{d} \end{bmatrix}$$

you enter

```
m = sym('[d^2 2 7; d d^3 9; 1 5 1/5]')
```

5. To take the determinant, enter

```
determ(m)
```

Note that this is different from the command which does numeric determinants, which is `det`.

6. To take the inverse of a symbolic matrix, enter

```
inverse(m)
```

Note that this is different from the command which does numeric inverses, which is `inv`.

7. Other symbolic matrix commands that may be of interest are `eigensys` and `transpose`. (There are many more, but these are the ones you will probably use most commonly.)

Another operation that will become important to us when we study Laplace transforms is partial fraction expansions (PFE), which you should have studied in Calculus. There are two ways of accomplishing this in **Matlab**. The first is by using the `residue` command.

1. Let

$$f(x) = \frac{2x^2 - 3x + 1}{x^3 + 2x^2 - 9x - 18}$$

This may be represented by creating two vectors in **Matlab**, with one vector representing the numerator polynomial and the other vector representing the denominator polynomial:

```
a = [1 2 -9 -18]    % denominator polynomial
b = [2 -3 1]        % numerator polynomial
```

Then the partial fraction expansion

$$\frac{b(s)}{a(s)} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + \dots + \frac{r_n}{s - p_n}$$

may be obtained using

```
[r,p] = residue(b,a)
```

Try this using the expression above. Verify by hand computation that the partial fraction works as expected.

2. The other way to do partial fraction expansion is using the symbolic toolbox. There is no single function to do this (but you could write one!), but you could obtain the same effect by first integrating, then differentiating the function. (This works because **Matlab** integrates the expression the way you would if you did it by hand: it first forms a partial fraction expansion, then integrates term-by-term. If you turn around and take the derivative, then you end up with the partial fraction expansion.)

Enter the following:

```
f = '(2*x^2 - 3*x + 1)/(x^3 + 2*x^2 - 9*x - 18)';
pfe = diff(int(f))
```

Verify that this gives the partial fraction expansion correctly. You can turn around and undo the partial fraction expansion by

```
simplify(pfe)
```

3. Unlike **Mathcad**, **Matlab** is able to handle partial fraction expansions even with non-integer coefficients. Do a partial fraction expansion on the following expression using both numeric and symbolic techniques.

$$\frac{2x^2 - 3x + 1}{x^3 + 2x^2 - 9.4x - 18}$$

Even though **Matlab** has symbolic capability, you should be aware that many of the kinds of operations you need to do on polynomials may be done using some of **Matlab's** numeric commands. Explore the following commands:

1. `conv` (multiply two polynomials).
2. `deconv` (divide two polynomials).

You should also explore the use of the `solve` function.

As mentioned above, **Matlab** is also a programming language, with (almost) all of the programming constructs your heart could desire, such as `if-else`, `for`, `while`. The following is a simple recursive function that computes the factorial of a function

```
function f = fact(n)
% compute the factorial of a number recursively

if n==1
    f = 1;
else
    f = n*fact(n-1);
end
```

1. Enter the program into a file using your favorite text editor. Save the file into a file called `fact.m`. The name of the file *is* important.
2. Make sure the file is in your Matlab path (see `help path`).
3. Compute the factorial of various numbers such as 4,5, and 10.

This program could also be written using a `for` loop.

```
function f = fact2(n)
% return the factorial of a number using a for loop

f = 1;
for(i=1:n)
    f = i*f;
end;
```

1. Type this program using a text editor and save it in a file called `fact2.m`.
2. Test its operation on some numbers.

As a final exercise, you are to write your own program. The following Matlab operations may be useful to you: `rem`, `floor`, `for`.

1. Write a function to convert a base-10 number to binary. Return the binary number in an array. Call your function `dec2bin`. Thus, if I were to invoke

```
bina = dec2bin(37)
```

Then `bina` should be `bina = [1 0 0 1 0 1]` (the LSB is on the right).

2. Test your function on a variety of numbers.