1 Using MatLab Help

- Command `help` provides a short description of all functions. For example, to get help on the `diary` command, type `help diary`. For more information type `help -i diary`.
- Command `lookfor` followed by a key word (which can be anything and not only a command) gives help related to the key word.
- One can also use the help in the Menu bar.
- Or one can visit the MatLab web site at `www.mathworks.com`

2 Getting started with MatLab

2.1 Saving your work

It is useful to save all work in a separate text file. One can do it using the command

```
>> diary name.txt
```

at the beginning. Here "`name.txt`" is an arbitrary filename. When you type

```
>> diary off
```

the commands will be saved in `name.txt`.

2.2 Simple computations

You can use MatLab as a calculator: if you type

```
>> 2+2
```

then MatLab will answer

```
ans = 4
```

Here `ans` is a variable, where the answer of the last computation is stored. It can be used to perform subsequent computation.

For example, try the following

```
>> sqrt(5)
```

and then

```
>> ans^2+ans^4
```

You can also save the result of your computation in another variable. Try

```
>> a=2+2
```

2.3 The dialogue between the user and the system

Generally, every line of input is one command. If your command is too long, you can split it into two rows using `...`

```
>> 1+2+3+4+5+...
6+7+8+9+10
ans = 55
```

You can also write several commands in one line, separating them with commas.

If you want to suppress answers of MatLab after some commands, you can type `;` at their end. For example, the following sequence of commands computes $\frac{\frac{\sin 1.3\pi}{\log 3.4}+\sqrt{\frac{\tan 2.75}{\tanh 2.75}}}{\frac{\sin 1.3\pi}{\log 3.4}+\sqrt{\frac{\tan 2.75}{\tanh 2.75}}}$

```
>> x=sin(1.3*pi)/log(3.4);
>> y=sqrt(tan(2.75)/tanh(2.75));
>> z=(x+y)/(x-y)
```

We did not put a semicolon at the end of the last command in order to see the result.

You can use semicolon to separate commands which appear in one line. Then also the answers to the commands followed by a semicolon will be suppressed.

You can browse through the list of commands which you entered before, by pressing the up and down arrow keys.

Another useful feature is the incremental search: if you want to recall the previous command calculating some square root, press `Control-R` and type `sqrt`, and the computer will display the previous command which contains the text `sqrt`. If you press `Control-R` again, the computer will look for earlier commands with the text `sqrt`. Some other standard Unix idiom are also supported.

The command quit ends the current session.

3 Basics

3.1 Numbers and their formats

You can use exponential notation to write numbers.

Try

```
>>13e3
```

Try the following sequence of commands:

```
>> 1/3
>> format long
>> ans
>> format short
>> ans
```

```
>> format short e
>> ans
>> help format
>> format
>> ans
```

The command `format compact` produces a more compact output, and `format loose` is used to go back to the default.

Complex numbers:

```
>> 0.5+2i
```

You can also use the exponential notation:

```
>> 2.5e3 + 1.25e-3i
```

You can also use `j` instead of `i`. You must not type a space before the `i`.

`abs(`x`)` is

Try the following commands:

```
>> 1/0
>> 0/0
```

`Inf` is infinity and and `NaN` is "not a number".

3.2 Elementary functions

Here is a short list of available functions in MatLab

a`+`b, a`-`b	addition, subtraction
a`*`b, a`/`b, a`^`b	multiplication, division, exponentiation
`pi`, `i`	the constants $\pi \approx 3.1415$ and $i = \sqrt{-1}$
`exp(`x`)`, `log(`x`)`	e^x and the (natural) logarithm of x
`pow2(`x`)`	2^x
`log2(`x`)`	$\log_2 x$
`log10(`x`)`	$\log_{10} x$
`sin(`x`)`, `cos(`x`)`, `tan(`x`)`	sine, cosine, tangent (x should be in radians)
`asin(`x`)`, `acos(`x`)`, `atan(`x`)`	inverse trigonometric functions
`sqrt(`x`)`	square root
`real(`z`)`	real part of a complex number
`imag(`z`)`	its imaginary part
`abs(`z`)`	$\lvert z \rvert$
`angle(`z`)`	argument of z (in radians from $-\pi$ to π)
`conj(`z`)`	conjugate $\overline{z}$

Try

```
>> abs(3+4i)
>> angle(1+i)
>> 4*ans
```

Other useful functions:

- `floor(`x`)` computes the largest integer smaller than or equal to x.

```
>> floor(pi)
ans = 3
```

- `ceil(`x`)` computes the smallest integer larger than or equal to x.

```
>> ceil(pi)
ans = 4
```

- `round(`x`)` is the nearest integer to x:

```
>> round(3.5)
ans = 4
>> round(3.3)
ans = 3
```

- `lcm` and `gcd` compute the least common multiple and the greatest common divisor of their arguments, respectively.

```
>> gcd(24, 36)
ans = 12
>> lcm(24, 36)
ans = 72
```

3.3 Variables

Any sequence of letters, digits, and underscores can be used as the name of a variable, as long as the first character is a letter. Variable names are case sensitive, so `variable_two` and `VARIABLE_TWO` and `Variable_Two` are three di.erent variables.

It is not possible to use variables which have not been assigned a value.

If you want to know what variables are already used, type `who`. Command `whos` gives more information.

You can clear a variable using command `clear`

Try the following sequence of commands:

```
>> a=2*2
>> b=1+1
>> c=2^2
>> who
>> whos
>> clear a
>> who
>> clear
>> who
```

You can also save values of all your current variables by the command

```
>> save work
```

in a binary file `work.mat`. Here "`work`" is also an arbitrary name.

Next time you can use

```
>> load work
```

to load the saved variables.

4 Vectors and Matrices

4.1 Entering vectors and matrices

```
>> v = [ 1, 2, 3, 5, 8 ]
```

You get the same result entering

```
>> v = [ 1 2 3 5 8 ]
```

A convenient way to enter some vectors is provided with the range notation. Try to guess the meaning of the following commands

```
>> 1:10
```

```
>> 1:2:10
```

```
>> linspace(1,10,5)
```

You can also enter column vectors. Compare:

```
>> a=[1 2 3 4 5]
>> b=[1; 2; 3; 4; 5]
>> whos
```

Try entering matrices.

```
>> A = [ 1, 2, 3; 4, 5, 6; 7, 8, 9 ]
```

You can also use a more intuitive way:

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

You can also see the size of the matrices:

```
>> size(a)
>> size(b)
>> size(A)
>> c=1
>> size(c)
```

You see that vectors and even numbers are considered to be matrices of size $1 \times n$, $n \times 1$ or 1×1.

Another important function is

```
>> length(a)
```

We can build up vectors and matrices from smaller vectors and matrices.

```
>> v=[1 2 3 5 8]
>> [v; v; v; v]
```

Try now

```
>> [ b [ v; v; v; v ] b ]
```

4.2 Index expressions

Take

```
>> p = [ 2 3 5 7 11 13 17 19 23 29 ]
```

You can get particular elements of a vector

```
>> p(5)
```

We can retrieve the last entry as follows:

```
>> p(end)
```

instead. Similarly, `p(end-1)` denotes the penultimate entry of `p`.

You can also use vectors of indices:

```
>> p([1 5 3 8])
```

Try also

```
>> p(1:5)
>> p(1:2:end)
>> p(end:-1:1)
```

Let us proceed to matrices, like the following

```
>> A = [ 1 2 3 4; 9 8 7 6; 10 20 40 80 ]
```

Matrices require two indices, so to get the entry in the second row an third element, we use

```
>> A(2,3)
```

We get a whole row by using a colon as column index.

```
>> A(2,:)
```

Similarly, we can get the third column of `A` with the expression `A(:,3)`.

Try the next commands and explain its effect

```
>> A(:, 4:-1:1)
>> A([1 3], 2:4)
```

You can change some entries of vectors and matrices:

```
>> p(5) = 100
```

We can even remove the fifth entry, by assigning the empty matrix to it.

```
>> p(5) = []
```

Type

```
>> p(10) = 99
>> p(20) = 99
```

The same game can be played with matrices:

```
>> A = [ 1 2 3 4; 9 8 7 6; 10 20 40 80 ]
>> A(2,3) = 66
>> A(1:2,2:3) = [ -1, -2; -3, -4 ]
>> A([1 3],:) = A([3 1],:)
```

4.3 Basic operations

Recall that `size` returns a vector with two elements: the number of rows, and the number of columns.

```
>> A = [ 1 2 3 4; 11 12 13 14; 21 22 23 24 ]
>> size(A)
```

Using an extra argument, we can get the number of rows with `size(A,1)` and the number of columns with `size(A,2)`.

Complex conjugate transpose of a matrix is formed by putting a `'` after the matrix.

```
>> A'
```

If you want just to transpose the matrix, you have to use `A.'`. Of course, this does not make a difference for real matrix.

Addition of matrices:

```
>> A = [ 1 2
       3 4 ];
>> B = [ 0 10
      10  0 ];
>> A + B
```

If we add a scalar to a matrix (or a vector), it is added to every element in turn.

```
>> A + 2
```

If we add two matrices whose size does not match, we get an error. Substraction works exactly the same.

The operator `*` denotes matrix multiplication

```
>> A * B
```

If you want to do element-by-element multiplication, you have to use the `.*` operator.

```
>> A .* B
```

The same goes for division and exponentiation: if you want to perform these operations element-by-element, you have to prefix the operator with a dot. Hence, we can make the row vector $(1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{8})$ like this

```
>> 1 ./ (1:7)
```

The exception to the rule that you need a dot if you want to calculate on an element-by-element basis, is when you combine scalars and matrices. On the one hand, `2*A` doubles all entries in `A`, and `A/2` halves them. But, on the other hand, `1/A` denotes the matrix inverse, and `A^2` and `2^A` are matrix exponentials.

Most functions which acts on scalars, are *mappable*. This means that if you apply them to a matrix or a vector, they are applied to every element in turn. So if we have a complex-valued matrix, we can get the imaginary part of every element by apply `imag` to it:

```
>> imag([ 2 5+i; -i 7+2i ])
```

And we can get the square root of the numbers 1, 2, ..., 7 as follows

```
>> sqrt(1:7)
```

The function `reshape(A,m,n)` takes the entries of the matrix (or vector) `A`, and puts them into an `m` $\times$ `n` matrix. The entries are retrieved, and stored, from top to bottom and from the left to the right, in that order.

```
>> reshape([1 1 2 3 5 8], 2, 3)
```

If `A` is an $m \times n$ matrix, then the matrix returned by `repmat(A, p, q)` has dimensions $(mp) \times (nq)$ and is constructed out of copies of `A`. For instance,

```
>> repmat([1 2; 3 4], 2, 3) ans =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

Some more commands to construct matrices are `ones(m,n)` for constructing an $m \times n$ matrix with all ones, `zeros(m,n)` for an all-zero matrix, and `eye(m,n)` for the matrix with ones on the diagonal and zeroes elsewhere. Furthermore, `rand(m,n)` produces a random matrix with elements uniformly distributed between 0 and 1, and `randn(m,n)` does the same but with the standard normal distribution. All these functions can also be called with one argument, to get a square matrix with the specified dimension.

The final command in this section is `diag`. In it simplest form, it extracts the (main) diagonal of a matrix:

```
>> diag([1 2 3; 4 5 6; 7 8 9])
```

It is possible to specify which diagonal to use by adding a second argument. To get the diagonal immediately above the main diagonal, ask for `diag(A,1)`. The diagonal above that is called `diag(A,2)` etc. To get subdiagonals, use a negative index.

```
>> diag([1 2 3; 4 5 6; 7 8 9], -1)
```

But we can also go the other way around. Given a vector `v`, the command `diag(v)` returns a matrix whose (main) diagonal is given by `v`, while the off-diagonal elements are zero. Again we can add a second argument, to specify another diagonal.

```
>> diag([1 2 3 4]) + diag([10 20], 2)
```

5 Plotting

Try the following commands:

```
>> ezplot('sin(x)')
>> ezplot('sin(x)',[0,8*pi])
```

A convenient way to plot data is using `plot` command, which plots vectors against vectors.

```
>> t = 0:0.2:1
>> y = t.^3
>> plot(t,y)
```

You can put command `shg` (meaning "show the graph") after the plot command to bring the current figure window to the front.

Try to understand what the following sequence of command will produce and then try to run them.

```
>> t = linspace(0,8*pi,200);
>> x = t.*cos(t);
>> y = t.*sin(t);
>> plot(x, y), shg
```

MATLAB will erase the first plot when the second plot command is executed. If you want to put two plots on one picture, use command `hold on` to hold the plots. Use `hold off` to alow MATLAB erase previous plots.