EL2310 Scientific Programming
LAB1: MATLAB lab session

Patric Jensfelt

Chapter 1

Introduction

1.1 Goals for this lab

The goals for this lab is

- handle the computers in the computer rooms
- create and edit files
- get started using MATLAB
- basic MATLAB programming

1.2 Reporting errors

As any document, this document is likely to include errors and typos. Please report these to `patric@kth.se`.

1.3 Acknowledgements

Valuable feedback and error reports on this document have been provided by:

- K. Huber, 2009

1.4 Prerequisite

You need to have a computer on which you can run matlab. Matlab runs everywhere and you can download it from http://progdist.ug.kth.se/public/ assuming you have a working kth.se account. If you want to use the CSC Unix computers and need some help to get started you can read the next section. If you want to use some other computer or alread know what is needed to know about the CSC computers you can skip to Chapter **??**.

1.5 Before getting started with CSC Unix computers

If you want to use the CSC computers you need a working CSC computer account. If you are attending the Master Program Systems, Control and Robotics you should have been given a kth.se-account. If you attended the first EE-registration this account should have been transfered to the CSC-computers and you should be able to login with the same name and password as you were given for the kth.se. Note that if you change the kth.se passwd this will not have affected the csc.kth.se account and you should thus use the original login/passwd. If you have any problems doing so please contact the course leader or if you know that you should have a working account go directly to the systems group "Delfi" located at the ground floor at Oscars backe 2.

You can skip this section if you are already familiar with a Unix/Linux system and emacs.

1.5.1 The terminal window

Many of the tasks you want to do can be performed by using the mouse. However for some tasks you may want to be be able to instruct the computer what to do by typing in commands. You do this in a terminal window. The terminal window on a Unix/Linux system looks and works much the same as the command prompt on a Windows machine. To start the terminal window you can right-click on the mouse to get a popup-menu from which you can choose to start a terminal window.

1.5.2 Files and folders

Each of you will be assigned a directory in the file system where you can store your own files. Each directory in the file system contains files and directories. A directory is also known as a folder as it stores things. It is good practice to organize your files into folders so that it is easy to find them afterwards.

If you want to see where you are in the file system you can issue the command

```
pwd
```

at the command prompt in a terminal window. This results in an output similar to

```
/afs/nada.kth.se/home/cvap/patric
```

depending on who you are.

Listing a folder

To list files in the current directory you issue the command

```
ls
```

This will give you a list of files and folders. If you want a bit more information than the name of the files and folders you can do

```
ls -l
```

Just like in Windows there are hidden files/folders in Unix/Linux. These files are prefixed with a “.”. Most of these files are system setting files so be careful with these. To see these files do

```
ls -a
```

or if you want more information at the same time

```
ls -la
```

If you want to see all files of a certain kind the wild card “*” is very useful. The wild card can be used more than once in a filename. For example

```
ls *scr*
```

lists all files that contains the letters “scr” in that order and in lower case. To list all matlab m-files that you have written in a certain directory you would do

```
ls *.m
```

Notice that each directory contains a file “.” and one “..”. These are special files that allows you to move up and down the file hierarchy.

Also notice that the filenames in Unix/Linux are case sensitive. That means that “file.txt” and “File.txt” are not the same.

Moving in the file system

To change the current directory you use the command

```
cd <folder>
```

where `<folder>` is the name of a folder you want to move into.

Using the special file mentioned above you can move to the directory above in the tree (the one containing the current directory) with

```
cd ..
```

To move to you home directory simply do

```
cd
```

i.e. `cd` without argument. You can also use the special symbol “ “ which stands for you home directory, that is

```
cd ~
```

or equivalently

```
cd ~/
```

The slash (“/”) at the end of a folder name is optional but is needed when you want to specify a path that contains more than one folder name, for example

```
cd foo/bar
```

(assuming that you have a directory foo with another directory bar inside).

To move to the previous directory, i.e. the directory that you were in before the current one do

```
cd -
```

Creating a folder

To create a folder you use the command

```
mkdir <folder>
```

Deleting a folder

To delete a folder use the command

```
rmdir <folder>
```

This is not possible unless the directory is empty. If the system claims that the directory is not empty but you cannot see any files/folders it could be that they are hidden (use `ls -a` to see them).

Deleting files

When you delete a file always think twice. There is no easy way to get a file back. You delete a file with

```
rm <file>
```

1.5.3 Using emacs

When editing files under Unix emacs is one of the most common editors. You start emacs from the command line in a terminal window with

```
emacs&
```

the “&” at the end tell the system to start emacs as a separate process and allow you to continue the terminal window for other things.

Notice that during this, the matlab lab session, you do not need to use emacs. You can use the built-in matlab editor instead.

Creating a new file or opening an existing one

To create a new file you select the “Open file” menu option or press `ctrl-x` followed by `ctrl-f`. This will give you a line at the bottom of the emacs window where you can enter the filename. If the file already exists you open that file and if it does not exist you create a new empty file.

Editing the file

You can input text by simply typing in the window.

Saving the file

To save you changes either use the menu option save or press `ctrl-x` followed by `ctrl-s`. If you want to save the file under another name either use menu option “Save buffer as” or press `ctrl-x` followed by `ctrl-w`.

More on emacs

A manual for emacs is avalible at the GNU emacs home page http://www.gnu.org/software/emacs/manual/emacs.html in for example pdf and html format.

Chapter 2

Lab instructions

2.1 Starting Matlab

To start Matlab, open a terminal window and type

```
matlab&
```

Depending on your configuration you might have to issue the command

```
module add matlab
```

before being able to start Matlab.

2.1.1 Getting help on matlab

When using matlab there are several sources of information besides the web.

Tasks

Make yourself familiar with the following ways to get help

`help` If you know what a function is called but you are uncertainty about the syntax you can use the help function.
Syntax: `help <function>`

`doc` The function `doc` is similar to `help` but brings up a separate window with the information.

`lookfor` If you do not know what a function is called and if such a function exist but you know what you want it to do you can search for this with `lookfor`
Syntax: `lokfor <keyword>`

helpdesk If you want to get more general information you can use the `helpdesk` function by simply typing `helpdesk`. This will bring up the manual with documentation starting from the very basics of matlab to specific information about functions. You can accomplish this also by going into the help menu in matlab and selecting “Full product family help”.

2.2 Command line input

Matlab is interactive. When you type something on the command line it is interpreted and text feedback, return values etc are given back. You can supress Matlab's output by ending the line with a ";".
`>> a = 5` will print out something on the screen whereas `>> a = 5;` will not. In this example it does not matter much but when working with large matrices or functions you typically do not want to have a lot of output on the screen unless you are debugging.

2.3 Vectors

There are a number of ways to create sequences/arrays of numbers in matlab

2.3.1 Creating a vector

Enumeration

The simplest way to create vector is to enumerate the elemets.
Ex: `v = [2 4 6 9]`

`colon`

If you want to create a sequence of number with a fixed step inbetween you can use the `colon` function or the colon-notation.
Ex: `v = 1:3:10`
which will create a vector with number between 1 and 9 separated by 3, i.e. the sequence 1,4,7 and 10. You can use any float point value for the start, step and end value and you can use a negative step value if you want to have an ascending sequence of numbers.

`linspace`

The colon-notation above is very convenient when you work with integer values and when it does not matter so much if you actually include the start and the end. The function `linspace` allows you to easily create a sequence of equally spaced number between a start and end value, including both ends.
`linspace(x1, x2, n)` where `x1, x2, n` and start values, end value and number of valus respectiviely.

`logspace`

Similar to `linspace`, `logspace` creates a vector with numbers that are logarithimcally spaced.
`logspace(x1,x2,n)`
will create n numbers between 10^{x_1} and 10^{x_2}.

2.3.2 Accessing elements in a vector

You can access an element of a vector like `v(2) = 3`

NOTE: Indices starts from 1

2.3.3 Length of a vector

You can get the length of a vector with
`length(v)`

2.3.4 Tasks

- Test the function described in this section
- What is the last value of `v = 1:2:10` and why?
- Verifify that logspace(x_1, x_2) $= 10^{linspace(x_1,x_2)}$

2.4 Matrices

2.4.1 Creating a matrix

Matrices can be creates just like vectors by enumerating all the elements. To indicate the next row use ";".
Ex: `A = [1 2 3; 4 5 6]`
which results in the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

You can also use any of the method above to define the elements of the matrix
Ex: `A = [1:3;linspace(4,6,3)]`
will give the same result as above.

eye

The identify matrix is used frequently. Can easily be created with
`eye(n)`

zeros

Often want a matrix with all zeros
`A = zeros(n,m)`
where n,m is number of rows and columns respectively

ones

Often want a matrix with all ones
`A = ones(n,m)`
where n,m is number of rows and columns respectively. To create a matrix with all elements equal to some other number k simple do
`A = k*ones(n,m)`

diag

You can create a diagonal matrix by specifying the diagonal vector
`A = diag(v)`

You can shift the provided vector up and down from the diagonal by adding one more argument
`A = diag(v,k)`
If you instead provide a matrix as argument the function will extract the diagonal.
`v = diag(A)`

blkdiag

You can also create block matrices easily with
`blkdiag(M1, M2, M3, ...)`
where `M1, M2, M3` are matrices that will end up on the diagonal.

rand

You can create a matrix with elements uniformly distributed bwteen 0 and 1 with
`rand(n,m)`
Note that you might want to set the seed for the random generator (do help rand)

randn

You can create a matrix with elements from a normal distribution with mean 0 and standard deviation 1 with
`randn(n,m)`

2.4.2 Accessing elements in a matrix

Accessing elements in a matrix can be done in two way, either using doubld indices or single indices.

Double index

Specify the row and column, like in
`A(2,3) = 4`
where 2,3 is row and column respectively
You often want every row but only some columns or vice versa. In this case you can use the ":" operator. To get 3 column and all rows do

`A(:,3)`
To get the 1st and 3rd row and all columns do
`A([1 3],:)`

Single index

You can also access the elements of a matrix with a single index. The elements in a matrix are numbered column wise. For a 2x3 matrix the numbering would be

$$A = \begin{bmatrix} a_1 & a_3 & a_5 \\ a_2 & a_4 & a_6 \end{bmatrix}$$

In a 2x3 matrix the 3rd element on the second row would have index 6 and would thus be accessed with
`A(6)`
and would in this case (with 2 rows) be equal to
`A(2,3)`

2.4.3 Submatrices

You can manipulate whole submatrices by specifying the range index for rows and columns such as
`A(1:3,1:3) = diag([1 2 3])`
which will make the upper leftmost 3x3 submatrix equal to a diagonal matrix with 1,2,3 on the diagonal.

The range of indices for rows and columns do not have to be "complete". That is something like
`A([1 3],[2 3])`
is allowed.

2.4.4 Size of matrices

You get the size of a matrix with
`size(A)` which will return number of rows and columns. To get the number of rows do
`size(A,1)`
and to get the number of columns do `size(A,2)`

2.4.5 Tasks

- Make sure you can create matrices and access their elements
- How would you create a matrix of the form

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 5 \\ 5 & 0 & 0 & 2 & 2 & 2 \\ 0 & 3 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 2 & 2 \end{bmatrix}$$

- What happens if you (without having defined A before) do
`A(10,3) = 42`?
- Make sure you under stand how the `end` operator does as in `v(4:end)`.
- What will `diag(diag(A))` do?
- Create a matrix with random values between -5 and 5 uniformly distributed
- Create a vector wit values drawn from a normal distrubution with mean 3 and standard deviation 2
- Investigate how the ":"-operator behaves when used as single index for a matrix, i.e. e.g. `A(:)`

2.5 Finding elements

Finding elements in a matrix fulfilling some criteria is done with the function `find`. It can return double or single index reference to the elements.

double indices

`[di, dj] = find(A>5)`
will return two vectors `di,dj` with double indices to all elements in A which are greater than 5.

single indices

`si = find(A==42)`
which will return a vector with single indices of all elements equal to 42.

2.5.1 Tasks

- Create a random 5x5 matrix with integer values between 0 and 9 with
`floor(10*rand(5,5))`
and then replace all elements ≥ 5 with their negative value

2.6 Operators

Most of the standard operators $(\times, -, +-, , \sin, \cos, \ldots)$ are built-in in Matlab. Many of these will operate element wise on a matrix. To force and operator like `*` to operate element wise you prefix it with ".".

Tasks

- Invetsigate the different between `sum, cumsum and cumprod`. When are they useful?
- Given two vectors v1 and v2 with some signal data, what will `cov(v1,v2)` give you? What if you say cov(v1,v1)? What does this mean?

- Make sure you understand the difference between for example
  - Multiplication * and .*
  - To the power of with and without "."
- If you have to vector v1 and v2, how do you calculate the Euclidean distance between them?

2.7 Graphics

2.7.1 2D graphics

plot

You can plot data using the `plot` command. If used with one argument as in

```
plot(x)
```

it will plot the values in x against their indices in the vector. That is if the vector contains 50 elements the plot will have an x-axis from 1 to 50.

By adding a second argument you can plot y against x, as in

```
plot(x,y)
```

If you want to specify that the line should be dashed instead of solid you can say

```
plot(x,y,'--')
```

If you want to change the color to red you do

```
plot(x,y,'r')
```

and dashed red would be `plot(x,y,'r--')`

You can add more than one set of x,y values after each other to the plot command and at the same time specifying color nd line types `plot(x1,y1,'r-.',x2,y2,'g--')`

See `help plot` for information. Also check functions `semilogx, semilogy, loglog, hold`

2.7.2 3D graphics

Plotting in 3D is not more difficult than in 2D with

```
plot3(x,y,z)
```

You can make nice 3D shapes with `mesh and surf` and contour plots are made simple with `contour`. Also take a look at `quiver`.

For the 3D functions the method

```
[X,Y] = meshgrid(x,y)
```

is very useful to get the X and Y arguments for e.g. `surf(X,Y,Z)`.

2.7.3 Axis, Labels, etc

You can specify the axis to use for the plot with

```
axis(x_min x_max y_min y_max)
```

If you plot in 3D then you add min and max values for z as well.

You can easily label the axes and add titles with

```
xlabel('This is the label for the x-axis')
ylabel('This is the label for the y-axis')
title('This is the title')
```

If you want to change the font size on the titles do for example

```
xlabel('Label on x-axis','FontSize',20)
```

If you want to set what extra properties you can specify for different graphical command you can get the handle

```
h = plot(x,y)
```

and then list the avalable properties with

```
get(h)
```

and change it with

```
set(h,'SomeProperty',SomeValue)
```

Also check `grid, zoom, rotate3d, view, clf, figure, close, legend, ginput`

2.7.4 Tasks

- Make sure that you can plot and format data including labels, legends, titlels etc and that you can produce a file with the figure in for example .eps format.
- What happens if you plot(X,Y) if X and/or Y are matrices?
- Plot a 2D Gaussian distribution and sum the probability. Does it come close to what you expect?

2.8 Scripts and Functions

You can extend the functionality of Matlab by writing your own so called m-files. They have the file-ending .m, hence m-files.

You can create/edit and m-file with the built in text editor in Matlab with `edit <filename>`. It will create non-existing files and open and existing one.

These m-files come in two flavors, scripts and functions. The commands written in a script are executed one by one as if they were typed on the command line. A function has input and output arguments and the first line in the file should be something like

```
function[out1,out2] = myfunction(in1,in2,in3)
```

2.8.1 Comments

Everything after a % on a line is interpreted as a comment. Use comments to make it easier for people to read you scripts/functions.

2.8.2 Input and output

To output something on the screen you can use the `disp` function. To get input from the user have a look at the `input` function. For more fancy formatting look at `sprintf`.

2.8.3 Tasks

- Write a function and make sure that the `help` and `lookfor` function works for it
- Test the `which` function
- Investigate the different between scripts and function
  - How are variables in the workspace affected by what happens in scripts and functions?
  - Can you get access to the variables from a function from outside a function
- How can you handle the user just pressing ENTER when he was supposed to enter a value?
- What happens if you do not provide enough input or output arguments?
- What does the keyword `global` do?
- Investigate how the `path` variable and function works
- Investigate the `load/save` functions.

2.9 Timing and profiling

- Try `tic` and `toc`
- Test `cputime`
- Test the profiling tool (`profile`
- Investigate how you can use breakpoints to debug your code

2.10 Symbol manipulations

Matlab (given the right toolboxes) allows you to do symbolic operations as well as numerical ones. This can be quite handy as you can imagine.

The test following

```
clear
syms t
x = 4*cos(2*t)
y = 2*sin(2*t)
f = [x;y]
dfdt=diff(f,'t')
```

```
t = 2
eval(f)
eval(dfdt)
```

What does it do?

2.10.1 Tasks

- Define $f(x, y) = \begin{pmatrix} x + x * y \\ x^2 + y^2 \end{pmatrix}$ and calculate the Jacobian of f and evaluate it for $x = 1, y = 2$