Introduction to Linux and MATLAB

Lecture notes

Contents

1 Linux

1.1 Introduction

Linux is a family of operating systems (OS) based on UNIX, an operating system which was first developed in the 1960s. Several operating systems have been developed following the Unix specification, for example "Mac OS X" and "Linux". Actually Linux is considered Unix-like, because even if it follows the Unix standard, it is not officially registered and then it cannot use the trademark UNIX.
The family Linux includes several *distributions*, for example Ubuntu, Red Hat, Debian, Fedora, Gentoo, Android (yes, the OS of your smartphone) and many others (more than 600!). These distributions can be very different, for example Ubuntu is considered more user-friendly for its easiness of installation, while Gentoo is considered more difficult to use, because it is necessary to recompile all the programs during the installation. However, at the price of a great difficulty of installation, you will get a faster and more efficient operating system. What is in common between all these distributions is the *kernel*, the Linux kernel. The kernel is the core of the operating system, the bridge between the software and the hardware.
It would be more correct to talk about GNU/Linux distributions, because the Linux kernel is only a part of the entire operating system. The other software of these operating systems (shells, compilers, libraries,...) have been developed inside the GNU project. GNU is the recursive acronym for "GNU's Not Unix!", chosen because the founder wanted to create an OS based on UNIX, but free and open to the developers.

1.2 The terminal

The UNIX operating systems are born without a graphical user interface (GUI) and the only way to interact with the computer was to use a command line interface (CLI). The program that interprets the input commands (keyboard keys) and shows the answers of the computer is the *terminal*. The program which actually processes commands and returns output is called *shell*.
Nowadays all the UNIX operating systems have a GUI (a desktop environment), but there still exist a CLI that is possible to use by a *terminal emulator* like "xterm".
To start the terminal emulator, just go in the "Applications Menu", then "Accessories" and finally click "Terminal".

1.3 The Linux file system

In Linux, the files and the folders are organized in a ordered tree-structure. The *root directory* is called simply "/" (only the slash). To go directly in the root directory, use the command:
cd /
To see the content of the directory where you are, use the command:
ls
If you are in the root directory, the list of the main directories of the OS will appear. Let's try to change directory with the command:
cd user/
Now you are in the directory that contains the personal files of each user using the system. Try to list again the folders and the files with the command **ls**, find your username and enter in your directory with command:
cd <username>/
This is your personal directory and you can find your documents, pictures, videos and so on. By the way, when you open the terminal as described in the previous section, you will be always in this directory.

If you want to move up one directory, use the command:
cd ..
If you get lost in the *tree* of the file system, you can see where you are with the command *print working directory*:
pwd

and to go back home, just use:
cd
Finally, to create a new folder use the command **mkdir** followed by the name of the folder, for example:
mkdir first-directory
and enter in the folder with the command **cd**.

1.4 Text editors

There are many text editors in Linux, some of them have a nice graphical interface like "Gedit", others work directly in the terminal, for example "Vi".

1.4.1 Gedit

To create a text file you have just to write in the terminal emulator, the name of the editor followed by the name of the new file (this is a general rule that works with all the text editors). If the name is already used by a file (inside the directory), you will open that file. So let's try
gedit first-file.txt
A graphical interface will open and you can use the editor as you are used in other OS. You can notice that now, you are not able to use anymore the terminal, in the sense that you cannot give any other commands. Then, close the editor and add an ampersand at the end:
gedit first-file.txt &
Now you can use both the editor and the terminal. Just write something, save the file and close it. Try to give the same command, this time you will not create a new file, but you will open the previous one.

1.4.2 Vi

Sometimes, it can happen that you do not have a GUI and you need to read/write a text file from the terminal. The program Vi is a valid solution, even if it can be a bit difficult to use for a beginner. To create/open a file with Vi, you proceed like with gedit, for example:
vi second-file.txt (This time the ampersand is not necessary because the editor works inside the terminal)
Vi is divided in two modes. The *command mode* and the *insert mode*. The first one allows you to give commands, for example, to save your work and to quit the program; the second one is the mode that permits to write a text. When you are in the insert mode, you can go in the command mode by pressing Esc, when you want to move from the command to the insert move, just press I. When you finish to write the text, move in the command mode and to "save and quit" use:
:wq
If you want to quit without saving, use:
:q!
For a list of the commands, I suggest you the webpage `http://www.cs.colostate.edu/helpdocs/vi.html` .

1.5 Manipulating files

1.5.1 Access permissions

Now that in our folder there a couple of files, let's try to manipulate them. First of all, list them with the command **ls**, but this time add the option **-l**:
ls -l
The answer of the terminal will be similar to the following:
-rw-r–r– 1 <user> zedat 253 Oct 24 11:18 first-file.txt
-rw-r–r– 1 <user> zedat 325 Oct 24 11:18 second-file.txt
The first character (dash - in this case) indicates the type of file: **d** for directory, **s** for special file, and - for a regular file. The following nine characters show the access permissions (**r**ead, **w**rite and e**x**ecute). The first three characters **rw-** define the owners permission. In this example, the file owner has read and write permissions only. The next three characters **r**– are the permissions for

the members of the same group of the file owner (which in this example is read only). The last three characters **r–** show the permissions for all other users and in this example it is read only. To change the permissions you can use the command "chmod", for example:
chmod "u=rwx","g=rx","o=x" first-file.txt
will change the permissions of user, group and other users.

chmod "u=rwx","g=rx" first-file.txt
will change the permissions of user and group only.

chmod "ugo=rwx" first-file.txt
will give the same permissions to owner, user and group.

The other information gave by **ls -l** are the number of links to the files, the name of the owner, of the group, the size (in bytes), modified date and time, actual name of the file/directory.

1.5.2 Rename, move, copy and delete a file

Let's create a new directory "second-directory/" in your personal directory.

The command **mv** permits to do two actions, to *rename* a file:
mv first-file.txt third-file.txt

and to *move* it in another directory:
mv third-file.txt second-directory/

If you want to move the file in another "branch of the tree", you will need to write the full path starting from the root:
mv third-file.txt /home/<user>/Documents/

To *copy* a file in another directory, use the command:
cp third-file.txt second-directory/
cp third-file.txt /home/<user>/Documents/

To *copy a directory* and its content:
cp -r second-directory/

To *delete* a file, use the command:
rm third-file.txt

To *delete a directory* and its content:
rm -r second-directory/

1.6 Tar and gzip

Tar is a program used to archive multiple files in packages. **Gzip** is a compression tool used to reduce the size of a file. They can be used together to create compressed packages, like the more popular *zip* or *rar* files.

To archive and compress a folder, use the command:
tar -czvf first-tar.tar.gz first-directory/

To extract the files of a tar archive, use the command:
tar -xzvf first-tar.tar.gz

The meaning of the letters after "-" is the following:
c = to create a new file tar

x = to extract a tar file
v = verbose, to visualize in the terminal the files to compress or extract
f = to create the tar file with the name of the argument
z = to use gzip for the compression/ectraction

1.7 Working from a remote host

If you want to work with the computer of the library remotely (i.e. from a computer connected to internet), you can use the commmand **ssh**:
ssh <user>@pool01.chemie.fu-berlin.de
and after you have inserted the password, you can use the terminal like if you were in the library. You can also add **-X** after **ssh** if you want to use a program installed on the remote host that has a GUI, for example MATLAB or Gedit. But if the connection is not good, the program will be very slow and it could crash. In this case you could prefer to work directly with the terminal, for example with Vi.
If you want to transfer files and directories from your personal computer to the library computer and viceversa, you can with the command **scp**.
To transfer a file from a remote host to a local host:
scp <user>@pool01.chemie.fu-berlin.de:/remote-path/file.ext /local-path
To transfer a file from a local host to a remote host:
scp /local-path/file.ext <user>@pool01.chemie.fu-berlin.de:/remote-path/
If you want to move a directory (and its content), just add **-r** after **scp**.

Of course, you can use these commands if you have *Linux* on your personal computer, but they work also with *Mac OS X*, because it is a UNIX OS. If you want to work with *Windows*, you need to install before an emulator like PuTTY or Cygwin:
`http://www.putty.org/`
`https://www.cygwin.com/`

2 MATLAB

MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. It enables to analyze data, develop algorithms, create models and applications. MATLAB exists for Windows, Mac OS X and Linux. If you are using Linux (ot MAC) you can run it from the terminal with the command **matlab**. The basic MATLAB interface is made by 4 parts:

1. *Current folder* shows the content of the working directory (the same directory of the terminal)
2. *Command window* is the console where you write the MATLAB commands and you run your scripts
3. *Workspace* shows the used variables (in the beginning it is empty)
4. *Command history* lists the commands already sent.

The MATLAB environment is very customizable, you can move, add or remove sections as you prefer.
MATLAB can work in two ways, by command line, i.e. writing the commands directly in the console (just try to write a simple operation like 3+5) or with the scripts.
A script is a list of commands, saved in files ".m". You can edit a script with a text editor like Gedit or you can use the MATLAB editor. If you click on the little white square on the toolbar, you will open the MATLAB editor and you can start to write a new script.

2.1 Let's start...

One of the most important component of MATLAB (and in general of all programming languages), are the *variables*. A variable is a part of computer memory containing some data. In MATLAB, to

declare a variable, you just need to choose a name and to assign some data, for example:

To assign a number, just write:
a=5

To assign a row vector:
b=[3, 4, 5]
NB: when you want to assign a row vector, you can omit the commas

To assign a column vector:
c=[3; 4; 5]

To assign a matrix:
d=[1, 2, 3; 4, 5, 6; 7, 8, 9]

To assign a string:
e='hello'

If you tried these command, you have noticed that matlab replies to you repeating the instruction. When you write a long script, this can be annoying (and it slows the execution), so just add a semicolon at the end of the instruction, e.g:
a=5;

If you want to see the content of a variable, just write the name in the console:
a
a= 5

If you want to access to some particular elements of a vector, you have to use the round brackets:
b(2)
ans= 4
NB: When you do not assign to any variable the result of a command, MATLAB will store the result in the variable ***ans****. This is a temporary variable, because it changes its content every instruction.*

If you want to access to more than one element of a vector, you need to use the double dot:
b(2:3)
ans= 4 5

If you want to access to some elements of a matrix, you have to separate the coordinates with a comma. The left coordinate indicates the row, the right coordinate is the column:
d(2,3)
ans= 6

d(2,:)
ans= 5 6

d(:,2:3)
ans= 4 5 6

2.1.1 Initialize "special" variables

MATLAB permits to initialize automatically some "special" variables. For example if you want a matrix $N \times M$ made by zeros, use the function:
a=zeros(N,M);

To initialize a matrix of ones, use the function:
a=ones(N,M);

To initialize a matrix of uniform random numbers between $[0, 1]$, use the function:
a=rand(N,M);

To initialize a matrix of uniform random integer numbers between $[1, K]$, use the function:
a=randi(K,[N,M]);

To initialize a matrix of normal random numbers with $\mu = 0$ and $\sigma = 1$, use the function:
a=randn(N,M);

Finally, to initialize a vector with a regular sequence of numbers, use the sintax:
a=1:5
a= 1 2 3 4 5

a=1:2:5
a= 1 3 5

2.2 Operations

The basic operations of MATLAB are + - / * ^. They act on the single numbers, on the vectors and on the matrices. Furthermore, you can use them to define new variables. Here there are some examples that you can try:
a=5;
b=4;
c=a*b
c= 20

If you want to add a number to a vector, you can write:
a=[5, 6, 7];
b=4;
a=a+b;
a= 9 10 11

If you want to add a number only to some elements, you have to access them with the procedure previously described.
a=[5, 6, 7];
b=4;
a(1:2)=a(1:2)+b;
a= 9 10 7

You can also do operations involging two (or more) vectors, or matrices. Of course, it is important that the variables are coherent, you cannot sum two vectors/matrices of different size or a row vector with a column vector.
a=[5, 6, 7];
b=[2, 3, 4];
c=a+b
a= 7 9 11

The operator "*", applied to vectors or matrices, defines the *dot product (or scalar product)*. The number of columns of the first factor must equal the number of rows of the second factor.
a=[5, 6, 7];
b=[2; 3; 4];
c=a*b
c= 56

d=b*a

d=
10 12 14
15 18 21
20 24 28

It is possible to define also a multiplication *element by element*, placing a "." before "*". This works also with "/" and "^".
a=[5, 6, 7];
b=[2, 3, 4];
c=a.*b
c= 10 18 28

2.3 Functions

One important characteristic of MATLAB is the presence of functions that will save us a lot of time. For example, if you want to compute the summation of a vector, you do not need to write an algorithm (like in other programming languages), but you just need to use the function sum().
a=1:10;
b=sum(a)
b= 55

There are many functions, depending on the MATLAB version and on the toolkits installed. Here I will list some functions that you will need to solve the exercises, but the full list is much more long. Remember two rules, the first is that the name of a function is similar to the name of the operation you want to use (for example summation → sum()), the second one is that on internet there is a big documentation about MATLAB and often you just need to do a quick research to find the function that you are looking for. Moreover, with MATLAB, you can create new functions or to find on internet functions, useful for you, created by other users. Finally, the last advice, when you do not know how to use a function, use **help** followed by the name of the function, for example:
help sum
S = SUM(X) is the sum of the elements of the vector X. If X is a matrix, S is a row vector with the sum over each column. For N-D arrays, SUM(X) operates along the first non-singleton dimension.
...

- Transpose of a matrix.
  a=[1, 2; 3, 4]
  b=a'
  NB: if it acts on a vector, convert a row vector in a column vector and viceversa

- Inverse of a matrix.
  a=[1, 2; 3, 4]
  b=inv(a)

- Trace of a matrix.
  a=[1, 2; 3, 4]
  b=trace(a)

- Determinant of a matrix.
  a=[1, 2; 3, 4]
  b=det(a)

- Eigenvalues and eigenvector of a matrix.
  a=[1, 2; 3, 4]
  [v lambda]=eig(a)
  In this case, the result is a pair of matrices. The matrix "v" is a matrix where each column is an eigenvector. The matrix "lambda" is a diagonal matrix, where each element of the diagonal is an eigenvalue.

- Mean of a vector
  a=[4, 5, 6]
  b=mean(a)
  If you apply the function to a matrix, you will get, the mean of the columns. If you need the mean of the rows, you can do either the mean of the transpose or the mean of the matrix, using "2" as second argument. If you need the mean of all the elements, you have to repeat the command mean():
  a=[1, 2, 3; 4, 5, 6; 7, 8, 9];
  b=mean(a)
  b=4 5 6

  b=mean(a')
  b=2 5 8

  b=mean(a,2)
  b=2 5 8

  b=mean(mean(a))
  b=5

- This sintax works in general for all the function acting on a matrix. For example **sum(), max(), min(), std()** (standard deviation), and others.

- Other functions that you need for the exercises are **sin(), cos(), exp(), log()**.

2.4 Control Statements: if, for, while loop

The control statements are blocks of instructions executed by MATLAB (or in general by a programming language) only if particular conditions are satisfied or that can be repeated automatically many times. It is important to understand from the beginning that this kind of instructions slow the execution of a program, then, when it is possible, it is important to look for other solutions. In MATLAB for example, there is the possibility to *vectorize* an algorithm, but we will talk about this argument later. To train on the control statements, it is better to use the MATLAB editor, write a script and run it. To run a script you can press F5, or you can just select in the the MATLAB editor the part of algorithm you are interested and press F9.

2.4.1 if-elseif

The "if statements" are a sequence of instructions executed only if a particular condition is satisfied. For example:

```
a=5;
if a>0
  b=a*2;
end
```

The operation a*2 runs only if the variable "a" is positive.

Using the logic operators conjunction "&&", disjunction "||" and negation " " is possible to construct more complex statements. For example:

```
a=5;
if a ~= 0
  b=a*2;
end
```

The operation a*2 runs only if the variable "a" is NOT zero.

```
a=5;
b=7;
if a>0 && b==7
  b=a*2;
end
```

The operation a*2 runs only if the variable "a" is positive and "b" is equal to 7.

```
a=5;
b=8;
if a>0 || b==7
  b=a*2;
end
```

The operation a*2 runs only if the variable "a" either is positive or "b" is equal to 7.

Finally the operators "elseif" and "else" allow to add more conditions.

```
a=5;
if a<0
  b=a*2;
elseif a==0
  b=a+2;
else
  b=a-2;
end
```

if-statement reduction The previous examples work perfectly, but they are not optimized. It is possible to get the same results using the logical operations. The following examples reproduce exactly the same results of the previous ones, but without using the if statements.

```
a=5;
b=a*2*(a>0);
```

The result of "a>0" is "1" if the statement is true, "0" if it is false

```
a=5;
b=a*2*(a~=0);
```

```
a=5;
b=7;
b=a*2*(a>0 & b==7 );
```

Note that in this case you do not have to repeat "&".

```
a=5;
b=8;
b=a*2*(a>0 | b==7 );
```

```
a=5;
b=a*2*(a<0) + (a+2)*(a==0) + (a-2)*(a>0);
```

2.4.2 for

The "for statements" are a sequence of instructions repeated many times. For example:

```
a=[1 2 3 4 5];
b=[6 7 8 9 10];
for n=1:5
  c(n)=a(n)+b(n);
end
```

It creates a new vector "c" where each element is the sum of the elements of "a" and "b".

```
a=[1 2 3 4 5 6];
for n=1:6
   if a(n)>3
      a(n)=a(n)+1;
  end
end
```

It edits the vector "a" adding +1 to each element greater than 3.

2.4.3 while

The "while statements" are similar to the for loops, but they repeat the content of a block until a certain condition is satisfied. For example:

```
a=[1 2 3 4 5];
b=[6 7 8 9 10];
n=0;
while n<=5
  n=n+1;
  c(n)=a(n)+b(n);
end
```

Vectorization The previous algorithms are examples to explain the control-statements, but it was possible to get the same results in a smarter and faster way. MATLAB is optimized for operations involving matrices and vectors. The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations is called vectorization. The previous examples can be vectorized as follows:

```
a=1:5;
b=6:10;
c=a+b;
```

```
a=1:6;
a(a>3)=a(a>3)+1;
```

2.5 Plots

MATLAB offers a variety of data plotting functions plus a set of GUI tools to create and modify graphic displays. This table shows MATLAB 2-D plotting functions.

Line Graphs	Bar Graphs	Area Graphs	Direction Graphs	Radial Graphs	Scatter Graphs
plot 	bar (grouped) 	area 	feather 	polar 	scatter 
plotyy 	barh (grouped) 	pie 	quiver 	rose 	spy 
loglog 	bar (stacked) 	fill 	comet 	compass 	plotmatrix 
semilogx 	barh (stacked) 	contourf 		ezpolar 	
semilogy 	hist 	image 			
stairs 	pareto 	pcolor 			
contour 	errorbar 	ezcontourf 			

This table shows MATLAB 3-D plotting functions.

Line Graphs	Bar Graphs	Area Graphs	Direction Graphs	Radial Graphs	Scatter Graphs
plot 	bar (grouped) 	area 	feather 	polar 	scatter 
plotyy 	barh (grouped) 	pie 	quiver	rose	spy
loglog	bar (stacked)	fill	comet	compass	plotmatrix
semilogx	barh (stacked)	contourf		ezpolar	
semilogy	hist	image			
stairs	pareto	pcolor			
contour	errorbar	ezcontourf			

There are many commands about graphs and their customization and it becomes difficult to summarize them. Here I show an example that allows to plot two graphs in the same window using different colors and line styles. Furthemore this script changes the axis of the graph and add title, labels and legend. Of course on internet it is possible to find much more commands.

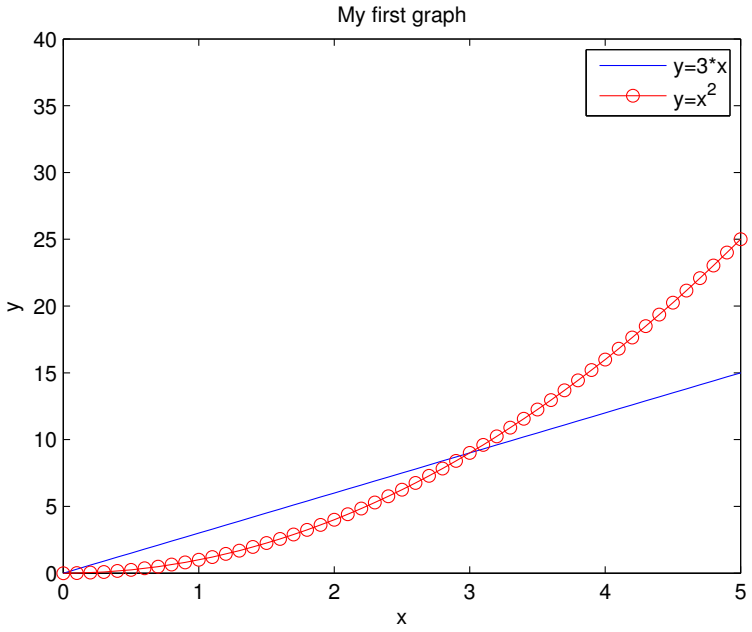
```
x=[0:0.1:10];
y1=[3*x];
y2=[x.^2];

plot(x,y1,'b-'); % Plot a blue 'b' dashed line '-'
hold on %Keep to use the same figure (window).  Whitout this command MATLAB would clean
the previous graph before to plot the new one

plot(x,y2,'ro-'); %Plot the second graph with red color 'r', circle markers 'o' and
solid line '-'

title('My first graph')
xlabel('x')
ylabel('y')
legend('y=3*x','y=x^2')

axis([0 5 0 40]) % Reduce the axis
set(gca, 'XTick', [0:1:5]) % Set the position of the ticks
set(gca, 'XTickLabel', [0:1:5]) %Set the labels of the ticks
```



2.6 Other commands

- **clc** to clears all input and output from in the console
- **clear all** to clear all the variables from the workspace
- **close all** to close all the figures opened

- **figure()** to open a new window where to plot a graph

- **a=importdata('data.txt')** to import in the variable "a" the data saved in the file "data.txt"

- **csvwrite('data.txt',a)** writes matrix "a" into "data.txt" as comma-separated values.

- **tic-toc** "tic" and "toc" functions work together to measure elapsed time. Example:
  ```
  a=[1 2 3 4 5];
  b=[6 7 8 9 10];
  tic
  for n=1:5
     (n)=a(n)+b(n);
  end
  time=toc;
  ```
  This script saves in the variable "time" the number of seconds necessary to complete the for-loop.

- The symbol "%" is used to comment the codes. A "comment" is a sentence that is not executed by MATLAB, it appears in green in the MATLAB text editor.