Université de Bourgogne Franche-Comté

Introduction to COMSOL: Laser heating of a silicon wafer

Authors:
Christophe RAMSEYER
Noah PERREAU

October 26, 2021

Contents

1 Introduction

Most laws in physics are formulated as differential equations which cannot be solved analytically. Numerical methods have been developed to approximate solutions to many of these equations. The *finite element method* is one such an approximation scheme, which is very general and flexible; it especially handles complex and curved geometries well.

COMSOL Multiphysics is a software package implementing the FEM to simulate various phenomena. Its main strength, as the name suggests, is to be able to mix the computation of different phenomena to provide physically accurate models of real systems. For instance, to design a microwave one might be interested in simulating the propagation of radiation as well as the heat pattern it generates; this requires *coupling* between Maxwell's equations and the heat equation.

In this first practical work, the basic features of an FEM program will be presented, as it is a prerequisite for more advanced multiphysics simulations.

1.1 Short description of the finite element method

To understand the main characteristics of the finite *element* (FEM) method, it is useful to compare it to its simpler cousin, the finite *difference* method (FDM). In the FDM, the operators in a differential equation are approximated by finite differences, *i.e.* are approximated on a regular grid. This leads to a linear system of equations, which can be solved using any standard method from linear algebra.

In the FEM, the operators are approximated in a more flexible way, in the following two aspects in particular:

1. The discretization grid is replaced by a *mesh*. It can be of several different types, but by far the most commonly used kind is a *triangulation*: the domain is decomposed in a set of non-overlapping triangles. In addition, the triangulation is usually required to be *conforming*: no triangle edge is split by the vertex of another triangle (stated differently, the only vertices in the triangulation are triangle vertices, which are also called *nodes* in the FEM).

2. The operators are discretized on this mesh using various *basis functions* (also called *shape functions*). In the FEM those basis functions are always localized on neighbouring triangles. The choice of basis functions is a recurring matter in numerical analysis. For instance, *spectral element methods* use periodic functions instead of localized ones. Another example, in quantum chemistry two kinds of basis functions are commonly used: either plane waves, or localized functions resembling atomic orbitals, usually built from gaussian functions for performance reasons.

A commonly used set of basis function is based on *Lagrange polynomials*, which arise in numerical analysis to compute the interpolating polynomial of a set of points. First order Lagrange polynomials correspond to linear interpolation, which when used on a triangular mesh, lead to basis functions of the shape shown in figure 1.

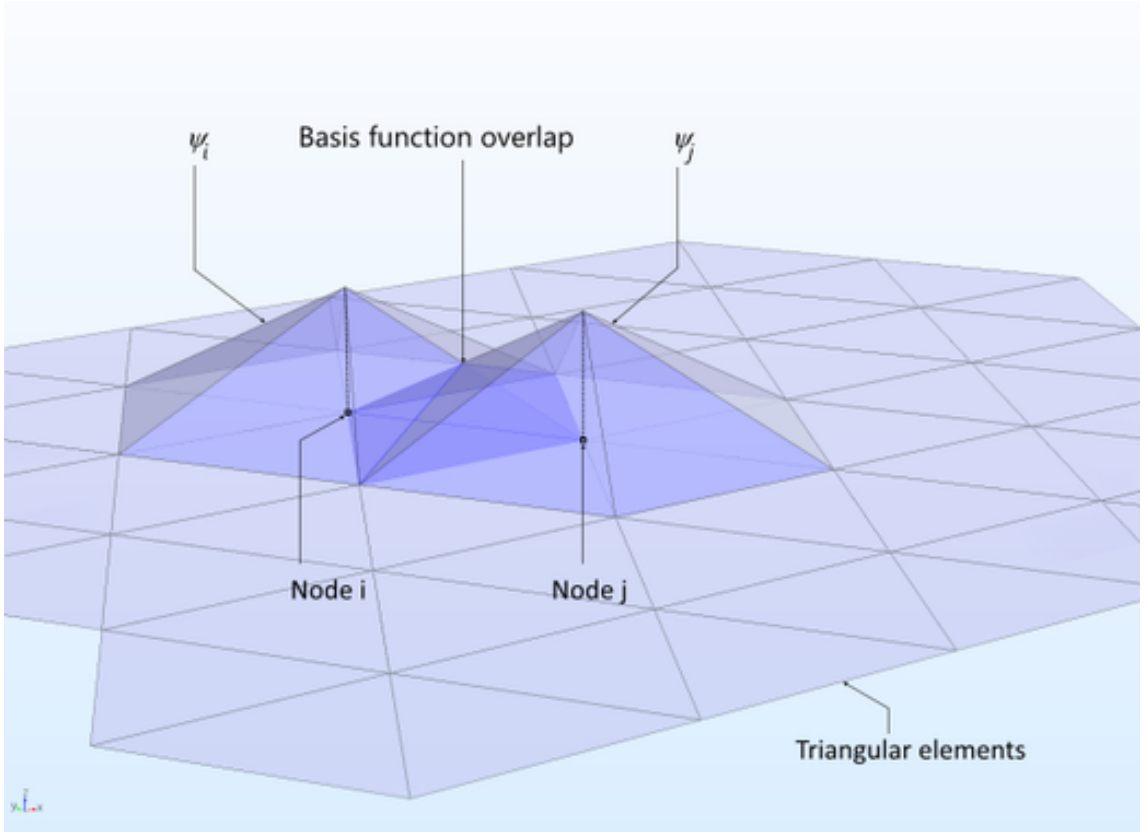

Figure 1: *Linear interpolation on neighbouring triangles leads to pyramid-like localized functions.*

The efficiency of the method can be improved by increasing the number of basis functions (or equivalently, reducing the average size of triangles), or by increasing the order of the Lagrange basis.

Once the basis functions have been chosen, the problem we are trying to solve is transposed to the framework of the FEM. This is rather technical, and many variations exists, but nonetheless this can be summarized by the following steps:

1. Write the differential equation in *weak form*; intuitively, it is called weak because the constraints of the differential operators (continuity, ...) are relaxed by formulating the equation using integrals. This in fact corresponds to writing the problem using generalized functions, in which derivatives can be applied to a much wider variety of objects than traditional, continuous functions.

2. Decompose the sought solution on the set of basis functions; using variational calculus, each of the coefficients associated to the basis functions are to be varied to minimize some measure of error.

3. Formulate the obtained equations, accounting for boundary conditions, group them into a linear system of equations ;

4. And finally, solve this linear system.

Most of the computation is spent in the last step, solving the obtained linear system. Solving linear systems is probably the most commonly carried task in numerical analysis, and therefore, a tremendous amount of research has been carried on the subject. In general, methods for linear systems can be grouped in two categories:

- *Direct methods*, such as Gaussian elimination or various matrix decompositions, compute an exact solution in a fixed number of steps (within numerical error due to approximate arithmetic). These methods work well for small and dense systems, in which not many coefficients are zero. Most of them scale as $O\left(N^3\right)$, and become prohibitively expensive as N grows.

- *Iterative methods* are to be used when direct methods are too expensive, that is when N is large. As the name implies, they compute an approximate solution in a number of steps. The more time is spent on computing iterations, the more precise the result can be *expected* to be. Contrary to direct methods, iterative methods are more diverse as they depend on the type of the matrix (wether it is symmetric, ...), and only a few can handle general matrices.

  Methods based on *Krylov subspaces* are the most commonly used, among which the *conjugate gradients* (CG) for symmetric positive-definite matrices and the *generalized minimal residual* (GMRES) for arbitrary matrices. The theory behind these methods is rather involved, and in most cases, definite theorems demonstrating convergence are not available. Still, these methods are pefectly usable in practice, and in the rare cases where one method fails, another method is usually available.

  A final practical point is the use of *preconditioning* to increase convergence speed. Preconditioning can be seen as a method to change the problem in a reversible way to make it simpler to solve. Once it is solved using the preconditioner, its effect can be reversed an one then obtains an approximate solution to the initial problem.

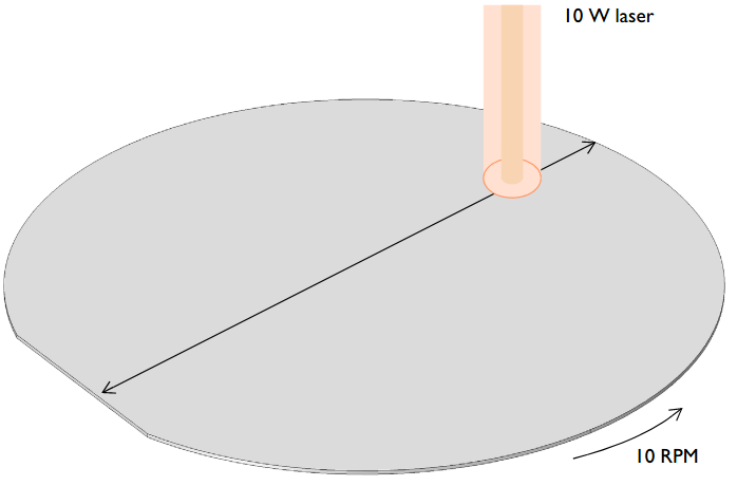

Figure 2: System to simulate: a laser beam heating a silicon wafer.

1.2 Studied system

A simple system will be used to illustrate an FEM computation: a moving laser spot heating a rotating silicon wafer, as shown in figure 2. In microfabrication, many processes require a precisely controlled temperature to work correctly; on the other hand, these processes are carried in controlled environments. A laser constitutes a simple way to heat the wafer from outside of a vacuum chamber, for instance.

1.3 Accessing COMSOL through SSH

The COMSOL program will be launched on a remote computing server, which is more powerful than the local computers. This requires the use of the *terminal* ; if you're not too familiar with it, you can find a small reference in the appendices. To still access it through its normal interface, we will use *X11 forwarding*: graphical commands of the X11 protocol, available on UNIX machines, will be redirected to our local graphics system instead of that of the server. This type of SSH channel is established with the following command (to be typed in a terminal):

```
$ ssh -X user@172.20.134.7
```

where `user` is your user name for the Compuphys server, and `172.20.134.7` is the server's IP address (note that this address may be different ; it will be given during the practical works if required). Then, you can simply launch COMSOL:

```
$ comsol
```

and this should open a window on your local computer.

At times you will be asked to let COMSOL generate some files for you. Of course, because it executes remotely, it will store them remotely as well, and you need to copy them back to your local machine. This can be done with the `scp` command, which is a mix between `cp` and `ssh`, and has to be executed on a terminal on you *local* machine:

```
$ scp user@172.20.134.7:/remote/path /local/path
```

which will prompt you for your password before starting the transfer. In this command, `/remote/path` is the path to a file or directory on the *remote* server, from which we want to download, and `/local/path` is the path on the *local* machine where we want to store the file or directory.

When downloading a single file, this command can be used directly, but you may want to download a full directory with all its subdirectories, or a set of files all at once. In the first case, you can use the `-r` switch (`r` for *recursive*):

```
$ scp -r user@172.20.134.7:/some/directory /local/directory
```

while in the second case, you can specify a list of files enclosed with *escaped* curly braces. This means you type `\{` instead of just `{`, because otherwise the terminal will interpret the curly braces differently. So, to transfer several files at once, the command will typically look like:

```
$ scp user@172.20.134.7:\{file1,file2,file3\} /local/directory
```

You can also use the `-r` switch in this command, so that you can also download several directory hierarchies at once. A useful command is `readlink`, which allows you to display the absolute path of a remote file on the server:

```
$ readlink -f ./some_file
```

You can copy-paste the resulting path and substitute it for `remote_path`; this way, you won't have to care about relative paths and working directories.

2 Main functionalities and user interface

To begin with, launch COMSOL Multiphysics and open the *laser heating* example by searching in the Application Library, available through **File > Application Library**. This loads a predefined computation example which we will reproduce step-by-step.

The most important part of the user interface is the *model builder*, shown in figure 3. It contains all the information concerning the computation, arranged under different tabs. The following subsections traverse these tabs and ask you to identify their function and main parameters.

2.1 *Global definitions* tab

> **Question 1:** Briefly explain the role of this tab. In particular, what are the default parameters for the wafer geometry and the laser beam? What is the advantage of centralizing parameters like this?

2.2 *Components* tab

The *components* tab is central to the definition of the problem we are trying to simulate. It lists all the components in the system, and for each of them, all the associated parameters: its geometry, its materials, the physical equations it is subjected to, the interactions with

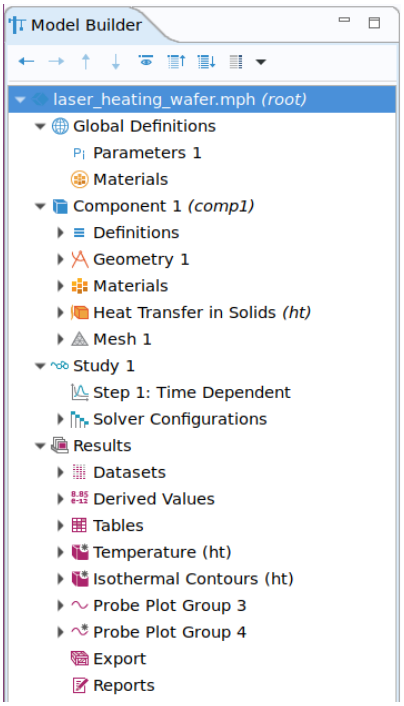

Figure 3: *COMSOL's model builder.*

other components, and so on. It is shown in figure 4.

2.2.1 *Geometry* subtab

> **Question 2:** Quickly explain how the geometry of the wafer was built (*Hint: set theory, union, intersection, ...*). If you are clueless, you can look at the Wikipedia page for *constructive solid geometry*.

2.2.2 *Materials* subtab

> **Question 3:** What is the role of the *materials* subtab? How do you assign materials to your geometry? What is the point of having a (very expensive) builtin materials library? If a material is not in the library, how would you use it in your computation?

2.2.3 *Heat transfer* subtab

This defines the physics to be used in the problem; therefore, it will require you to answer many questions. But then the most difficult part will be done!

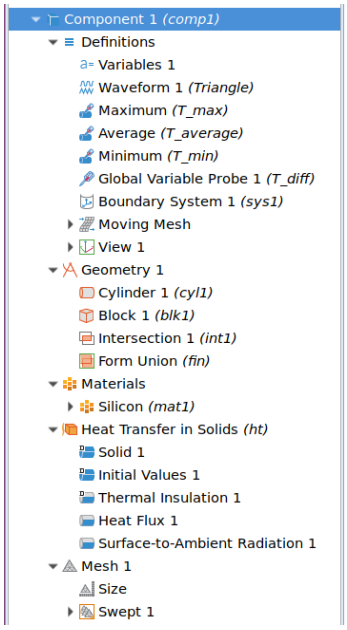

Figure 4: *The component tab*

Question 4: COMSOL lists all the equations involved in your computation. Find the heat equation, which describes what happens in the bulk of the wafer, and describe each term. (*Hint: you can find the meaning of each symbol in COMSOL's help, at the following location:* ***Help > Heat transfer module > Notation > Symbols***)

Question 5: What boundary conditions are applied? For each of them, identify which region is concerned (for instance: top face, bottom face, ...). Describe the associated equations and link them to the phenomena we are trying to model.

Question 6: How is the interaction between the laser beam and the wafer accounted for? Again, describe the equation, and identify the parameters it uses from the *Global definitions* tab.

2.2.4 *Meshing* subtab

The mesh is central to any finite element computation. Various algorithms exist to create such a mesh, but the most common generate triangles (in two dimensions) or tetrahedra (in three dimensions). In this project, meshing is not critical since the geometry of the problem is very simple, but it becomes important with curved surfaces and angles, and when the quantities you compute vary wildly. Still, it is presented here as it will be useful for other pratical works.

Question 7: What parameters would you change to refine the mesh? What other element shapes are available?

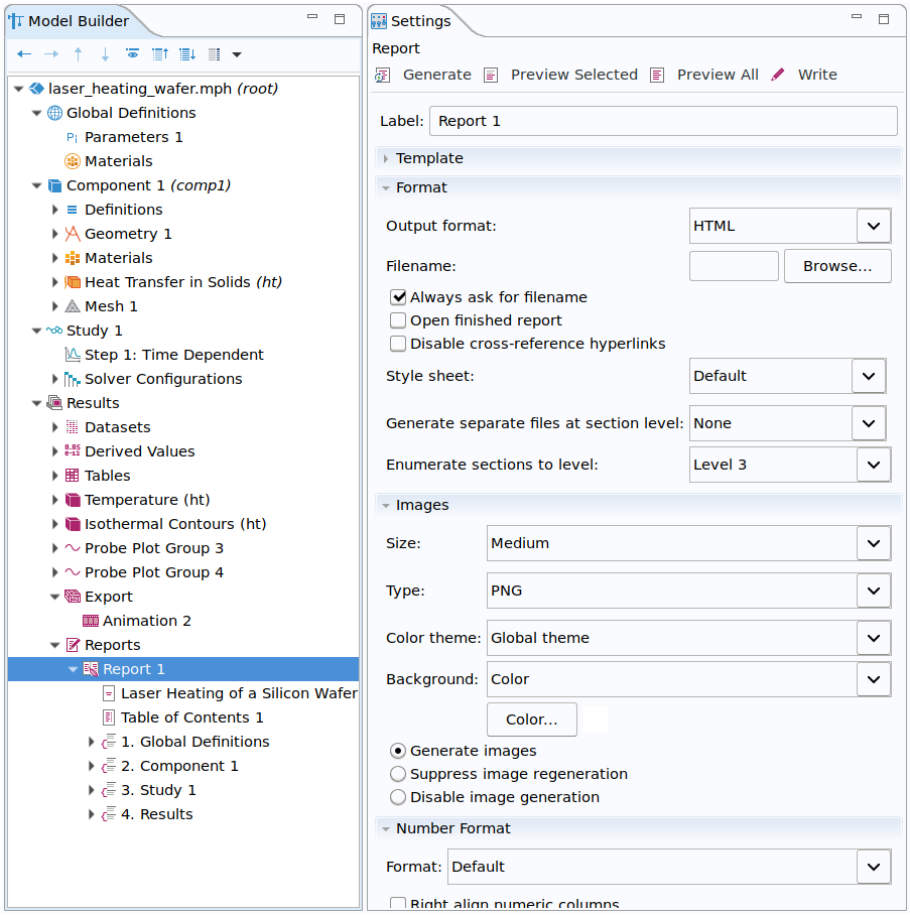

Figure 5: *Parameter tab for generating reports.*

2.3 Report generation

A nice feature of COMSOL is that you can generate reports at the end of a computation, to summarize the input parameters as well as the results. You will be asked to generate such a report regularly, so as to supplement the answers to the questions. Figure 5 shows the menu used to generate reports; to access it, right-click on the **Reports** subsection in the **Results** tab, and select *Brief report*.

You should generate the file in HTML format, but all other parameters can be left as default. Make sure to uncheck the "*Open finished report*" button, since we are accessing COMSOL through SSH. If you don't, COMSOL will crash and all your progress may be lost (you can recover it sometimes on the next COMSOL run).

Of course, the report will be generated on the server; you need to transfer it back to your local machine using the `scp` command. Also, make sure to name them appropriately (for instance, add the question number in the title).

2.4 *Study* tab

The *Study* tab allows you to control the general parameters of the simulation (duration, ...), as well as to launch it. When modifying any parameter, in this tab or any of the

previous tabs, you must recompute the solution by launching it again in this tab.

> **Question 8:** Do launch the computation for the laser heating example, without modifying any parameter. How long does it take? Generate your first report using the method previously described.

2.5 *Results* tab

The simulation would be useless if you couldn't access its results. When designing a system, you would spend most of your time in this tab trying to interpret the results.

> **Question 9:** Familiarize yourself with the results section. This tutorial was preset to output a graph of various quantities of interest; take a screenshot of this graph, and describe all the quantities which appear on it. Does it make sense ? (*Hint: periodicity, amplitude, ...*)

For time-dependent simulations such as this, it is useful (and fun) to see an animation of the results. Find how to play such an animation.

> **Question 10:** It is also useful to keep a record of the animation, and so you can export it to a video. Find how to generate a video, and don't forget copy it back from the server and join it with your report. You should use the same parameters as those shown in figure 6.

> **Question 11:** By default the heat is displayed as a heatmap (makes sense). To try another plotting option, find how to use level lines instead and generate a new video.

3 Changing simulation parameters

3.1 Beam period and wafer rotation

> **Question 12:** Increase the beam period to 1 second and decrease the rotation speed of the wafer to 1 RPM. Run the computation (and generate a report), and describe how this influences the results. Regarding the time-plots related to temperature (especially the temperature difference), is it advisable to use these parameters? (*Hint: thermal stress and cracking*)

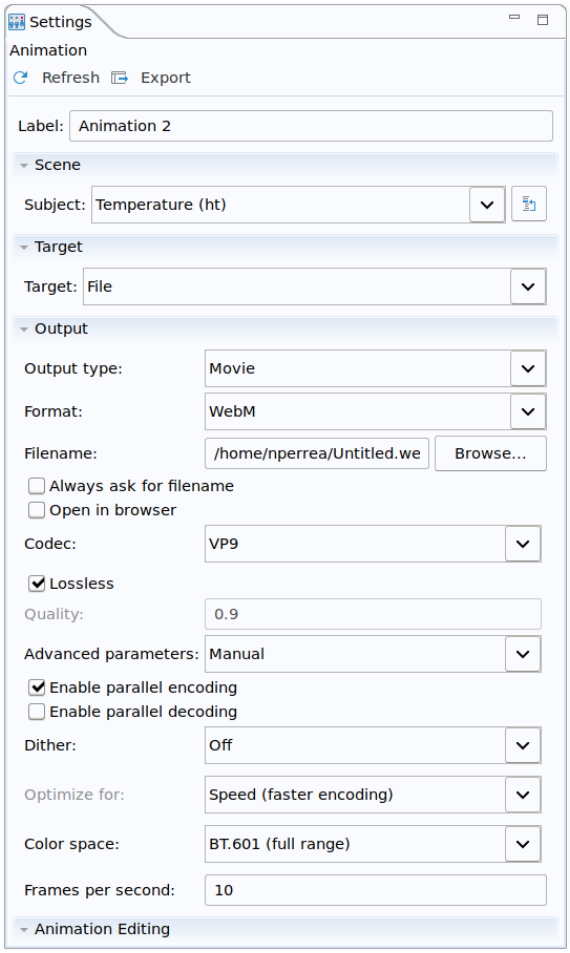

Figure 6: *Parameters to use for video output.*

3.2 Beam size

Question 13: Increase the diameter of the laser beam to 50 mm and recompute the solution (and generate a report). Is this diameter a reasonable value for a commercial laser, considering the beam power? Discuss the advantages and disadvantages of the approach used in this project (speed, heat homogeinety, ...).

3.3 Different wafer material

Silicon is by far the most widely used semiconductor, and hence also the most common wafer material. Alternatives have been explored for a while, but more recently a few candidates have received much attention for their predicted higher efficiency. One of them is *silicon carbide* (SiC), which crystallizes in a wide variety of lattices. It is also a semiconductor, but with a much wider band gap between 2.3 eV and around 3 eV, compared to silicon (around 1.2 eV).

Question 14: Re-establish the previous parameters for the beam size and wafer speed, and change the material of our wafer so that it is now silicon carbide. What is its thermal conductivity (according to COMSOL's material library)? Accordingly, how would you expect this to change the results (min/max temperature, heat diffusion and homogeneity, maximum difference, ...)? Verify you hypotheses by analyzing the results of the simulation (generate a report, also). If it contradicts you earlier suggestions, PLEASE don't change them! The point of having software like COMSOL is *precisely* to be able to verify your hypotheses: it's more useful if it contradicts you.

3.4 Mesh size

Question 15: Decrease the maximum size of a triangle, regenerate the mesh and relaunch the computation (generate a report). Discuss how this influences the results? The computation time? The required memory (if you can find it)? Is it useful to refine the mesh for this geometry?

4 Conclusion

This practical work showed you the basics of carrying an FEM computation using COMSOL. Simulations such as these are performed all the time, by engineers in particular, to understand the influence of even the simplest variation in problem parameters. Arguably, the FEM doesn't show all its strength in such a simple example, but now you are prepared to explore it further.

A The terminal and some useful commands

It is very common for computing environments to involve the use of a *terminal*, which is a textual but direct interface to your machine. The terminal expects you to type in commands by displaying a *prompt*, commonly represented by a dollar sign:

```
$
```

You should not type in the `$` symbol when you want to execute a command. Also, be aware that the prompt can be different on different systems, so do not expect to see a `$` all the time.

There is a large variety of commands available, depending on the system. Some commands are built-in, while some correspond to programs installed by the user (like COMSOL). The following sections describe a few of the most important commands, mostly built-in.

A.1 `pwd`: *program working directory* and filesystem paths

The *current directory* (or *program working directory*, PWD) is your current location in your file system. By default, when loggin into a system or opening a terminal, you will usually end up in your *home* directory. Each user has its home directory, which can only be accessed by its owner.

Using this command is simple, since it does not take any argument:

```
$ pwd
```

It will return you the path of the current directory, such as:

```
-> /home/nperrea
```

Here again, the arrow symbol `->` just indicates this is something output by the terminal. It is just a convention, since in practice most terminals don't print anything at the beginning of an output line.

The path which was returned by `pwd` is actually an *absolute path*, which describes the path of a file or directory from the root of the filesystem. There are also *relative paths*, which reference a file or directory from the current directory. Absolute paths always start with a `/` symbol, while relative paths never do. Therefore, if the `/home/nperrea` folder contains a file named `hello_world.c`, it can be referenced by the following absolute path:

```
/home/nperrea/hello_world.c
```

Or, if the current directory (output by the `pwd` command) is `/home/nperrea`, simply by:

```
hello_world.c
```

A.2 `cd`: *change directory*

The `cd` command allows you to change the current directory, by providing a path as argument:

```
$ cd new_directory_path
```

where `new_directory_path` can be absolute or relative. The following example changes the current directory from `/home/nperrea` to the `comsol_praticals` subdirectory:

```
$ pwd
-> /home/nperrea
$ cd comsol_practicals
$ pwd
-> /home/nperreau/comsol_practicals
```

Notice that `cd` doesn't return any output.

In addition to absolute and relative paths, there are two special paths which are extremely useful. A single dot (`'.'`) refers to the current directory while two dots (`'..'`) refers to the parent directory. Referring to the current directory is sometimes useful, for example when executing programs located in the current directory:

```
$ ./some_program
```

Referring to the parent directory allows you to revert the effects of a previous `cd` command, which otherwise would only allow you going down the filesystem hierarchy. Thus, applied to the previous example, we would have:

```
$ pwd
-> /home/nperrea/comsol_practicals
$ cd ..
$ pwd
-> /home/nperreau
```

A.3 `ls`: listing the contents of a directory

Of course, you do not need to remember the whole file hierarchy of your computer in order to use `cd` and other commands. Without any argument, the `ls` command lists the files and directories contained in the current folder.

It is however more useful when using some options, also called *switches*. By convention, an option to a command can be specified by starting it with a hyphen (`'-'`) or two (`'--'`). Being a convention, there is some variation among systems, but usually a single hyphen introduces so-called *short-options*, which are specified with a single letter, and two hyphens introduce long options which usually describe the purpose of the option.

For instance, `ls` has a short `'l'` option displaying the output as a list instead of a table:

```
$ ls -l
```

It also has long options. The `help` long is commonly implemented in many commands, and is used to print a documentation of what the command can do, among which all the options it can take:

```
$ ls --help
```

It is a matter of taste, but the most useful set of options for `ls` seem to be:

- '`l`', to print a list instead of a table ;
- '`h`', to print files sizes in easily readable units instead of the size in bytes ;
- '`a`', to also print the hidden folders and files.

Short options can be combined together in a single switch; that is, the following two commands display the same output:

```
$ ls -l -h -a
$ ls -lha
```

Since the '`lha`' switches are so commonly used, many systems have an alias command named `ll` for `ls -lha`.

A.4 `cp`: copy files and directories

Another very handy command is `cp`, allowing you to copy files from one location to another. It is used in the following way:

```
$ cp source_path destination_path
```

where you should substitute `source_path` and `destination_path` with the desired locations, either as absolute or relative paths. Notice a command convention amond terminal commands, that source operands precede destination operands.

You can also copy several files at once, in which case the destination is necessarily a directory:

```
$ cp source_A source_B source_C destination_path
```

When you want to copy folders, you have to use the `-r` flag (as in *recursive*):

```
$ cp -r source_directory destination_directory
```

This will copy all the files and subfiles contained in `source_directory` to `destination_directory`.

A.5 `mv`: move files and directories

To move files and directories, you can use the `mv` command, which works lie `cp`, except the source files will not remain afterwards. It is easy to inadvertently erase files without knowing using `mv`, so it is always a good idea to use it with the `v` (verbose) and `n` (don't overwrite) switches:

```
$ mv -vn source_path destination_path
```

A.6 `mkdir`: creating directories

You should of course organise your files in a clever hierarchy. In addition, commands like `cp` will not create directories if they do not exist. To create a directory, you can use the `mkdir` command:

```
$ mkdir new_directory
```

A.7 `man`: print documentation about a command

Many commands have the `help` option (or similarly, a '`h`' short option) to print some documentation when in doubt. More extensive documentation can be found in the venerable *man pages*, short for *manual pages*, through the `man`[1] command. To display documentation about the `ls` command, simply type:

```
$ man ls
```

This will launch a text interface in which you can navigate using the arrow keys, and which you exit by typing '`q`'. It is always a good idea to check the man pages when you are introduced to a new command. As a suggestion, it would be wise for this practical work to see what it can teach you about `ssh` and `scp`:

```
$ man ssh
$ man scp
```

A.8 Command history and autocompletion

Two nice features implemented in most terminals is the *command history* and *autocompletion*. The first allows you to recall previously-typed commands using the up and down arrow keys. It is especially useful when typing long commands involving long paths or addresses, such as `ssh` and `scp` commands.

Autocompletion is useful for lazy people (that is, essentially every single programmer), since it can complete command names and even paths while typing commands. It is used through the tab key. If you press it once, it can either complete the current part of the command you are typing if it can choose unambiguously, or display nothing. In the

[1] No sexism intended.

second case, you can hit tab twice in a row, and this will list all the possibilities it found. This allows you to complete the command a bit until it can choose among the possibilities unambiguously, at which point you can press tab again to let the terminal complete it for you.

It is much more intuitive than it sounds.

B Working with remote computers: SSH and X forwarding

The COMSOL program will be launched on a remote computing server, which is more powerful than the local computers. The terminal introduced in the previous section executes commands locally, that is, on the computer on which they are typed in. When working remotely, you need a way to execute commands *as if* you were typing them on that remote computer.

B.1 The *Secure Shell* (SSH)

This is exactly the purpose of the *Secure Shell* (SSH): it provides you with a terminal executing on a remote machine, in addition to encrypting the channel. Apart from this, the terminal is similar to any terminal. However, you should always remember that SSH executes on the remote computer, and therefore provides you with the environment of the remote system. For instance, a command might be available on you local computer but not of the remote computer, and vice versa.

To use SSH, you need to provide the name of an existing user on the remote machine as well the IP address of the machine. If we use the fake user `my_user` and the IP address of the Compuphys server, the command is simply:

```
$ ssh my_user@172.20.134.7
```

You will then be asked to type your password, and if successful, the remote shell will be launched. You can now use this terminal as if it was a regular terminal.

B.2 `scp`: secure copy

At times you will be asked to let COMSOL generate some files for you. Of course, because it executes remotely, it will store them remotely as well, and you need to copy them back to your local machine. This can be done with the `scp` command, which is a mix between `cp` and `ssh`, and has to be executed on a terminal on you *local* machine:

```
$ scp user@172.20.134.7:/remote/path /local/path
```

which will prompt you for your password before starting the transfer. In this command, `/remote/path` is the path to a file or directory on the *remote* server, from which we want to download, and `/local/path` is the path on the *local* machine where we want to store the file or directory.

When downloading a single file, this command can be used directly, but you may want to download a full directory with all its subdirectories, or a set of files all at once. In the first case, you can use the `-r` switch (`r` for *recursive*):

```
$ scp -r user@172.20.134.7:/some/directory /local/directory
```

while in the second case, you can specify a list of files enclosed with *escaped* curly braces. This means you type `\{` instead of just `{`, because otherwise the terminal will interpret the curly braces differently. So, to transfer several files at once, the command will typically look like:

```
$ scp user@172.20.134.7:\{file1,file2,file3\} /local/directory
```

You can also use the `-r` switch in this command, so that you can also download several directory hierarchies at once. A useful command is `readlink`, which allows you to display the absolute path of a remote file on the server:

```
$ readlink -f ./some_file
```

You can copy-paste the resulting path and substitute it for `remote_path`; this way, you won't have to care about relative paths and working directories.

B.3 Graphical forwarding

SSH only allows you to access a remote terminal, i.e. a text-based interface. On computers using the *X11* graphics protocol such as most UNIX-based systems, it is possible to use *X11 forwarding*: graphical commands from the program are redirected to our local graphics system instead of that of the server. This type of SSH channel is established with the following command (to be typed in a terminal):

```
$ ssh -X user@172.20.134.7
```

where `user` is your user name for the Compuphys server, and `172.20.134.7` is the server's IP address. Then, you can simply launch COMSOL:

```
$ comsol
```

and this should open a window on your local computer.