Theoretical Physics 3 (TP3)
Scientific Computing Laboratory

Dr. Christoph Englert, Dr. David Miller
SUPA, School of Physics and Astronomy, University of Glasgow

September 30, 2014

Contents

Part I
Recap: Using a Scientific Linux Environment

A lot of what we will discuss in this section will be familiar from P2T, but our review should be considered a reference for the later sections to make the course self-contained.

1 A basic introduction to Linux

1.1 Getting Started

An **operating system** (often abbreviated OS or O/S) is the software on a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer. Probably most of the computers you have interacted with have had the operating system Windows (in one of its forms). However, many scientists doing scientific research (and many other computer users too) use a different operating system called **Linux**[1]. Linux is free software and allows open source development – the underlying source code can be freely modified, used, and redistributed by anyone.

There are many different distributions of Linux, and some companies will charge for the distribution, set-up, and user interface of Linux. The version of Linux we will use is Ubuntu, which is entirely free and can be downloaded from `http://www.ubuntu.com`. In order to be as distribution independent as possible, we will avoid using the fancy functionality of its user interface, and work (almost) entirely from the **command line** of a **terminal** window. The terminal is sometimes referred to as the console or the shell. So our first task is to open up the terminal and get to know what it is for.

▶ **Task:** After logging in to Ubuntu, depending on the version, you will see a menu labelled *Applications* on the top left of your screen. Click on this to open the menu, then on *Accessories* and finally on *Terminal*.

Applications ⟶ Accessories ⟶ Terminal

Alternatively, if you have the newest version of Ubuntu installed, which runs the *Unity* interface, you will see a set of icons to the left of the screen. The top one, a grey circle containing the Ubuntu logo, is the "Dash home" (which will identify itself as such if you hover your mouse over it). Click this and a box you can type in should appear. Type "terminal" in this box and hit ENTER, and a terminal window will appear. (Alternatively, before hitting ENTER you can click on the Terminal icon that should appear beneath the box. You can also drag the Terminal icon to the menu of

[1] Linux is a registered trademark of Linus Torvalds.

icons on the left so that in future you can just click on the icon to open a new terminal window. You can do this with many Ubuntu programs.)

Now you should have a terminal window on your screen which you can move about by clicking and dragging on the title-bar, or resize by clicking and dragging on the edges or corners. To manage our files, run programs, compile C++ code and do many other things, we will type commands into this window (at the *command prompt*, here $) followed by pressing ENTER. Which commands we use varies slightly depending on the **shell** we use – here we will be using the **bash** shell, which is now most standard.

1.2 Navigating the File System

Linux uses a **directory** structure for organising files (you may be more familiar with directories being called *folders*). Each directory may contain files or other directories, building up a structure a bit like an inverted tree. You can move about in this directory structure, copy or move files, make new directories or delete old ones and many other organisational tasks simply by typing commands into the terminal.

Your current working directory

To find out which directory you are currently in, use the command `pwd`.

▶ **Task:** Type `pwd` (all lower-case) and press ENTER.

This command stands for *print working directory* and should return

```
/home/username
```

where *username* will be specific to you. This tells you that you are in a directory called *username* which in turn is in a directory called *home*. `/home/username` is known as the **path** for the directory. Notice that Linux commands are case-sensitive. Also, from now on I will stop telling you to press ENTER at the end of commands.

If you ever forget what a command does, or want to find out which options or syntax you can use with a command, you can access its **manual page** by entering the command `man` followed by the command name you are interested in. The resulting manual page can be scrolled through using the up and down arrow keys and you can leave the page by pressing `Q`.

▶ **Task:** Enter `man pwd`, scroll up and down and return to the prompt by pressing `Q`.

All commands that you type into the terminal will by default be applied to your current working directory unless otherwise specified. For example, as you will see later, a command `emacs report.tex` would look for the file `report.tex` in your working directory and open it using the emacs editor.

Changing your current directory

To navigate to another directory, you should use the command

```
cd destination_directory
```

where `destination_directory` is the directory you would like to be your new working directory. This can be either the entire path of the directory (for example `/home/username`), the name of a directory sitting in your current working directory, or "`..`" (two dots) which is a shortcut for the current parent directory (similarly one dot, "." is a shortcut for the current working directory). Incidentally, this command is an abbreviation for *change directory*.

► **Task:** Enter the command `cd ..` and then see where you are using `pwd`. Now navigate back to your home directory, i.e. `/home/username`. Note that you can always return to your home directory, from anywhere, just by using the command `cd` with no argument.

Listing the contents of a directory

To list the files and subdirectories of the directory you are currently in type `ls`. You can also list the content of directories other than your working directory by typing the entire path of the directory after `ls`, e.g. `ls /home`.

► **Task:** Explore the manual page for `ls` and display the contents of the directory `/home` in a "long listing format".

Moving and copying files

To copy a file use the command

```
cp file destination
```

where `file` is the file name (or path to the file) you want to copy and destination is the either the name of the destination file or the path of a directory you want to copy the file to. For example, `cp /data/p3_t_lab/report.tex .` would copy the file `report.tex` (if it exists) from the directory `/data/p3_t_lab` and put the copy in the current working directory (remember the "." is shorthand for the current working directory). The command `cd /data/p3_t_lab/report.tex my_report.tex` would also copy the file but the copy in your working directory would now be called `my_report.tex`.

The command `mv` (short for *move*) has a similar functionality, but destroys the original (i.e. retaining only one file). Obviously you usually cannot move other people's files, but often can copy them.

When using one of these commands, `*` is a wildcard which can be substituted for any string. So, for example, `cp /data/p3_t_lab/*.tex .` would copy all files in `cp /data/p3_t_lab` which end in "`.tex`" to the current working directory.

cd	Change directory
chmod	Change access permissions
clear	Clear terminal window
cp	Copy one or more files to another location
date	Display or change the date & time
diff	Display the differences between two files
echo	Display message on screen
exit	Exit the shell
find	Search for files that meet a desired criteria
grep	Search file(s) for lines that match a given pattern
gzip	Compress or decompress named file(s)
head	Output head lines of a file
less	Examine the contents of a file in the terminal window
ls	List information about file(s)
make	Recompile a group of programs
man	Help manual
mkdir	Create a new directory
mv	Move or rename files or directories
pwd	Print working directory
rm	Remove files
rmdir	Remove folder(s)
tail	Output tail lines of a file
time	Measure program running time
top	List processes running on the system
who	Print all usernames currently logged in
whoami	Print the current user id and name ('id -un')

Table 1: Some commonly used bash commands.

Other commands

Some other useful commands in bash shell are given in the Table 1.

► **Task:** Explore the manual pages for the commands listed in Table 1 (where available), and familiarise yourself with their usage.

It is not necessary to understand every specific of the commands at this time, but you should at least figure out what they do. You will need to use many of these commands later, so remember that you can review their manual pages using the command `man`. Pay special attention to `cd`, `cp`, `mkdir`, `mv`, `rm`, and `rmdir`, and feel free to play about with them (I don't think you can do any major damage at this point but try not to delete anything you may need).

1.3 Your path

All of the commands you have run above are actually little programs (mainly scripts) which have been placed somewhere in the directory structure. You execute these programs just by typing their names at the command prompt, but you need to make sure that Linux knows where they are. You can do this by making sure you include the complete location of the file. For example `./myprogram` will run the program `myprogram` if it is in the current directory (which you indicted with `./`). You can also use the full path of the program, e.g. `/home/username/myprogram`, where of course `username` should be replaced by your actual username.

If you don't add the full location of the program, Linux looks for them by examining your "`path`".

▶ **Task:** Type `echo $PATH` to see your current path.

This should show a list of directory locations, separated by colons, and Linux looks in each of these locations in order to find the file you want to run. You can append directories to your path by using the command

```
PATH="$PATH:$HOME/mydirectory"
```

where `/mydirectory` is the location of the directory you want to add (`$HOME` returns the location of your home directory).

▶ **Task:** Make a new directory in your home directory called *bin*. Make sure it is in your *path* by entering `echo $PATH` and looking to see if it is listed. If it isn't, use the command given above to add it to your path, and check that it is now there. You should now be able to run any executable in `~/bin` just by typing its name at the command prompt.

1.4 Some more useful tips

Also note that you can bypass a lot of the tedium of typing commands in Linux by using the TAB key (the key above CAPS LOCK and to the left of Q). Typing a partial command and pressing TAB will either give you a list of all commands starting with those letters or complete the command for you (if the letters typed already uniquely specify the command). For example, typing `mk` followed by TAB will give you a list of commands starting with `mk`, while typing `mkd` will (probably!) complete the command to `mkdir`. This also works for path names and file names, including executables.

You can browse through your previously typed commands (so you don't have to type them again) by pressing the up and down arrow keys, and you can browse this history with the command `less ~/.bash_history` (the `~/` is shorthand for your home directory, and the key with "~" is typically just to the left of ENTER, though this may vary from keyboard to keyboard). You can reuse commands you previously entered by typing `!` followed by the first letter(s) of the command, e.g. `!ma` will probably reproduce your last `man` command (including argument).

Finally, you can *pipe* the output of one program into another using

```
program_1 | program_2
```

For example, if your program called `my_prog` creates a very large output to the screen, it may be convenient to run it using the command

```
my_prog | less
```

which will display the output one screen at a time. Using the symbol `>` you can also send output directly to a file. For example,

```
my_prog > my_prog.dat
```

would send the terminal output of `my_prog` to the file `my_prog.dat` instead.

2 Editing files with emacs

2.1 Getting started with emacs

The text editor we are going to use is **emacs** (this was originally an abbreviation of *Editor MACroS*). You are free to use any editor you like, and if you are familiar and happy with another editor available with Linux, just skip this section entirely. If you are not familiar with another editor, emacs is a good one to start with, primarily because it can be navigated entirely without using a mouse. Though it is less of an issue with modern computers, you may sometimes find yourself working from a platform where you don't have a mouse to navigate and have to rely on keyboard input only.

You can start emacs by typing `emacs` at the command prompt. This will open a window running the Emacs editor, and will suspend your terminal until you close emacs again. Since it is often useful to type commands in the terminal while emacs is open, it is better to run emacs *in the background* – that is, run it without a terminal "watching" it. To do this type `&` before hitting ENTER. This can be done with any process, but bear in mind that messages the process wants to send to the terminal may be lost.

▶ **Task:** Open Emacs using the command `emacs &`.

This should open a new window for emacs, but still let you enter commands in the terminal. Emacs commands are often entered using the CTRL (short for *control*) and ALT keys, both in the bottom left of your keyboard. The notation `C-x y` would mean "*hold down the* CTRL *key; while it is down, press* X*; release the* CTRL *key and press* Y". Similarly `M-x y` would mean the same procedure but with the ALT key instead of CTRL (the M originally stood for *meta*).

▶ **Task:** Type `M-x tetris` (hold down the ALT key and press X; release the ALT and type "tetris"). Play **one** game only! (If the emacs window splits into 2 when your game finishes, turn it back into one using `C-x 1`.)

▶ **Task:** Type `C-h t` (hold down the CTRL key and press H; release the CTRL key and press T). This starts the emacs tutorial[2]. The rest of this section will borrow rather heavily from the emacs tutorial which you can access via `C-h t` for more detail.

It is sometimes useful to be able to open emacs without a new window. To do this one would type (in the terminal) `emacs -nw` where `-nw` stands for *no window*. This opens emacs in the terminal window itself, which is incredibly useful if you are having difficulty with Linux giving you permission to open a new window, which sometimes occurs when remotely connecting to another computer.

If emacs hangs (stops responding) you can stop it safely using `C-g`. You can exit emacs using `C-x C-c`.

2.2 Windows and buffers

Emacs can run several windows at once, allowing you to edit multiple texts simultaneously.

▶ **Task:** Type `C-x 2` and you will see the emacs window split into 2. Each of these windows can be operated on entirely separately, and you can click on the window to change focus. Type `C-x 1` to return to one window.

You can save the text you type into the editor using the command `C-x C-s` (the `s` standing for *save*). If the editor is currently editing a known file, it will save the changes you have made. If there is no file yet, it will prompt you for the name of a file, and create a file with that name in your current directory.

▶ **Task:** Type something into the emacs window and save it to a file called test1.tex.

You can edit a text file (which already has stuff in it) by typing `C-x C-f` (the `f` stands for *find*), and the name of the file you want to edit. You can then edit this file and save your changes using `C-x C-s` as before. Note that the file itself is not changed until you use `C-x C-s` (that is, emacs does not write to the file as you type) and even then, it will save the old version of the file with the same name but with a tilde (~) on the end.

▶ **Task:** Exit emacs, and restart it again. Find the file test1.tex (that you created earlier), edit it and save your changes. Look in your directory to see the extra file test1.tex~.

You can also open emacs so that it starts by with the contents of a file by typing the file name after the `emacs` command, for example `emacs test1.tex &`.

The text you are editing (which can be different from that in the file until you save it) is called a *buffer*. You can have multiple buffers active at once, even if you only

[2]Copyright (c) 1985 Free Software Foundation, Inc

have one emacs window. When you open a new buffer, using `C-x C-f` emacs creates the new buffer in your current window, but doesn't forget the information in the old buffer. You can switch back to it any time you like using `C-x C-f` again.

▶ **Task:** Type `C-x C-f test2.tex` to open a new buffer, type a few words in, and then switch back to test1.tex using the command `C-x C-f test1.tex`.

You can list all the current buffers with the command `C-x C-b`, and get rid of the buffer list using `C-x 1`. Don't forget that you can view different buffers simultaneously by having more than one window.

2.3 Editing text

You can enter test just by typing and remove text again using either BACKSPACE or DELETE. BACKSPACE removes the character just before the cursor while DELETE removes the character at the cursor. You can also delete the character just after the cursor using `C-d`, delete the next word using `M-d`, and delete to the end of the line using `C-k`, or the end of the sentence using `M-k`.

You can also delete any portion of the buffer by first placing a (non-visible) *mark* at the start of the text you want to remove using `M-space` (ALT together with the space bar), and then moving the cursor to the end of the test you want to remove and typing `C-w` (`w` stands for *wipe*).

Any text removed using `M-d`, `C-k`, `M-k` or `C-w` (that is, anything other than just a single letter) is temporarily remembered by emacs and can be reinserted where ever you like (even in a different buffer) by moving the cursor to the reinsertion point and typing `C-y` (`y` for *yank*).

▶ **Task:** Remove the last two words from test1.tex (using `M-k` or `M-space` and `C-w`), and copy them into test2.tex (using `C-y`).

You can access previously deleted material by typing `M-y` (as many times as you like) after `C-y`. Additionally, you can undo the last edit you made by typing `C-x u` (`u` for *undo*). You can do this multiple times, progressively undoing all your edits.

One more command is very useful: `M-%` (this has a little tricky fingering since you need to use ALT but also SHIFT to get at the % character on most keyboards). At the bottom of the buffer you will be prompted with `Query replace:` and if you type in a word of phrase (followed by ENTER) you will be prompted `with:` and can type another word or phrase. Emacs will then move through the rest of the file looking for the first phrase and replacing it with the second phrase. In this version of the command, it will always ask you whether to replace or not (respond with `y` or `n`).

3 Automation and Coding: Shell Scripts, C++

Shell Scripts

A shell script is a plain-text file that contains shell commands (commands like `mkdir` and `mv` that you saw earlier). It can be executed by typing its name into the terminal, or by placing its name in another shell script.

► **Task:** Before going on, we want to make sure that `~/bin` is in your path. You added it manually to the path earlier, but this resets every time you log off. We want this to be added automatically via your login shell.

Look to see if `~\bin` is in your path (it will be listed with the full path, including your username, rather than the abbreviation `~`). If it isn't, we will want to include the following in `~/.bashrc`:

```
# Set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
  PATH="$HOME/bin:$PATH"
fi
```

This should make sure that `~\bin` is included in your path for every terminal you open.

► **Task:** Open the web browser `Firefox` (can you do this from the terminal?) and go to the following address[3]:

`http://www.linuxcommand.org/writing_shell_scripts.php`

Note: You may have to set up your proxies first. Just follow the instructions in the browser.

Read the tutorial, and follow the examples by creating and running your own shell scripts, up to an including Section 9. Feel free to go further if you have plenty of time and are interested.

► **Task:** Write a shell script called `welcome.sh`, which, when executed prints to the screen either "Good Morning", "Good Afternoon" or "Good Evening", according to system time. Once it is working, move this script into your `bin` directory and execute it in `.bashrc` to make it run whenever you log-on to the system.

► **Task:** Go back and reread the script earlier in this section (after the first task). Do you understand the script? This allows you to put all your scripts in the directory `~/bin` and have them executable from any directory. The variable `PATH` is essentially a list of where the OS is allowed to look for programs that you try to run at the command

[3] Copyright (c) 2000-2012, William E. Shotts, Jr.

line - so if you execute a script program by typing its name, the OS will first look in your current directory, and if it can't find the script (or program) it will explore the directories in your `PATH` until it finds it.

Compiling C++ programs

In subsequent weeks you will be taught how to program in basic C++ (in actuality, you may not get far enough to see the sort of things which distinguish C++ from vanilla C, such as "classes"). You will see that the *source code* is written in a simple text file, which must then be *compiled* to produce an *executable* which can be run to perform the desired task.

You can compile a C++ program by using the command `g++`. For example, if the name of your program file is `prog.c`, to compile it you may type:

```
g++ prog.c
```

(To compile a C program, you can also use the command `gcc`.) If there are errors in your program the compiler will show them on the screen and then return you to the prompt. You should then correct these errors by opening the program in emacs.

After you have corrected your errors, you may use the same `g++` command to compile it again. If there are no errors in your code the compiler will return you to the prompt without any error messages. This means that your code has been compiled into a separate executable file which by default is named as `a.out`. You may run the code by typing `a.out`.

If you wish to give a specific name to your compiled program (suppose you want to call it `my_prog.out`) you may do this by using the option `-o` and specifying the name of the executable. For example,

```
g++ -o my_prog prog.c
```

will produce an executable called `my_prog`.

▶ **Task:** Copy the file `hello_world.c` from the directory `/data/p3_t_lab`. Unfortunately, our computer cluster set-up changes from year to year, depending on class size, room availability and computer availability, so `/data/p3_t_lab` may not be visible from your machine. In case it is not, the contents of `/data/p3_t_lab` have also been placed in `http://www.physics.gla.ac.uk/~dmiller/TP3/` so that you can still access them. Once you have copied the file to your home space, try and compile it. To check how a typical error message looks like, try removing the semicolon in, e.g., the

```
cout << "Hello World!" << endl;
```

Note that in order to run the executable you may need to type `./a.out` if your current directory is not in your path. (Alternatively you could add `PATH="$PATH:."` to your `.bashrc` so that your current directory is always in your path.)

We will come back to compiling C++ codes using *Makefile*s later when we look at C++ more closely.

4 Visualising Data: Gnuplot

Numerical simulations or calculations performed using a programming language like C++ often return data as columns of numbers in output files. In order to present this data, you have to use some package to plot the data in graphs that can be included in your report or paper. We are going to use *gnuplot* for this, though there are many other choices you could make. We will use gnuplot because of its widespread availability (it's free) and its convenient interfacing with the typesetting package LaTeX which we will use for writing reports. People in my own field (Particle Physics) usually use a package called `Root` but this is overly complicated for the sort of things we want to do here (and rather user unfriendly).

▶ **Task:** Open your browser and navigate to

`http://security.riit.tsinghua.edu.cn/~bhyang/ref/gnuplot/index-e.html`

where you will find a tutorial on gnuplot. Follow at least the first two and last two sections – you can skip most of the middle two if you like, but also read "Output in the Postscript format".

▶ **Task:** Copy the file `gaussian.c` from the directory `/data/p3_t_lab`. This is a little C++ program which calculates a the value of the Gaussian function,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right],$$

for equally spaced values of x and writes the x and $f(x)$ into an output file. Compile and run the program, entering your own values for the mean, standard deviation etc, and call the output file `gaussian.dat`.

▶ **Task:** Now write a gnuplot script to plot the data in `gaussian.dat` and output as a *postscript* file `gaussian.eps`.

You can view an `eps` or `ps` file using *ghostview* via `gv filename.eps`, if it is installed. If *ghostview* is not installed you should use *evince* via `evince filename.eps`. It should look something like Figure 1.

▶ **Task:** Now produce a plot as a *postscript* file `gaussian_multi.eps`, which shows three separate Gaussian distributions, each with the same mean but with different standard deviations. Include a key on your plot.

▶ **Task:** Can you similarly produce a 2d plot showing

$$f(\vec{r}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(\vec{r}-\vec{\mu})^2}{2\sigma^2}\right],$$

using x and y ($\vec{r} = (x, y)$) as the axes and different colours to represent the value of $f(\vec{r})$? It should look something like Figure 2

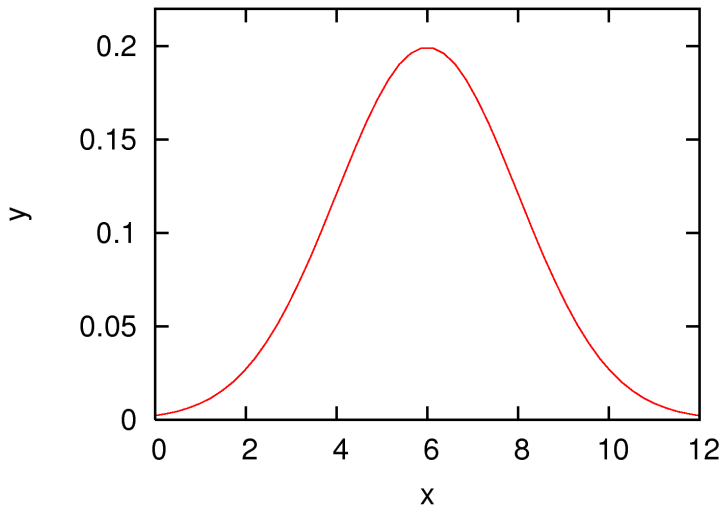

Figure 1: A Gaussian distribution centered around $x = 6$ with standard deviation of 2.

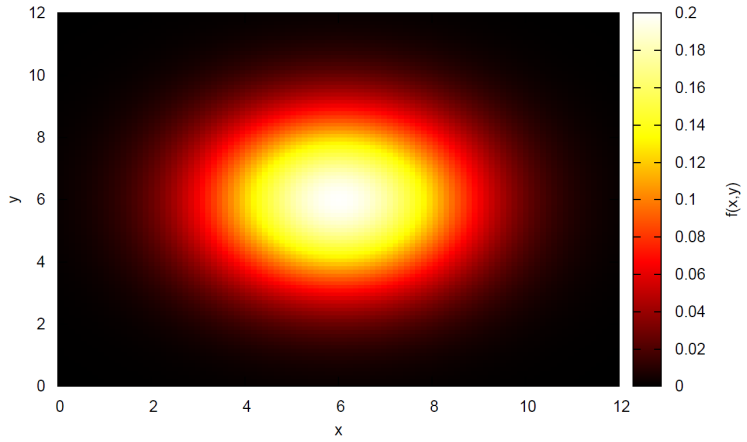

Figure 2: A 2-dimensional Gaussian distribution centered around $(x, y) = (6, 6)$ with standard deviation of 2.

4.1 Putting things together

A common problem that you might encounter in your projects is to repeat the same task multiple times, like plotting the same histogram for different parameter choices or measurement series. A similar problem can arise when you might be hosting a webserver in the future (i.e. totally unrelated to physics).

This can quickly become a tedious exercise and asks for automation! There are obviously many ways to tackle such issues using scripting languages like bash or python. Let's do a task to see how you can do this in a quick and straightforward way.

▶ **Task:** Get the data set `www.christophenglert.co.uk/histograms.tar.gz` (you might want to use the terminal and `wget` to download it. Extract the data set, you will find 12 histogram files in a two column format with gnuplot-specific headers (have a look with Emacs). In fact a single file contains multiple histograms: Use `gnuplot histo_1` to generate a postscript file from the gnuplot source to see for yourself!

We are interested in what particle physicists call an invariant mass distribution: local excesses in such distributions are new particle resonances that get produced abundantly at collider experiments like the CERN LHC. Look in the source file for "mZZ"; the histogram that is listed after the header information is the one we are after.

Now to the actual task: Write a bash script that extracts this particular histogram and parses it to a specifically labelled file, i.e. the invariant mass distribution of `histo_1` should be stored in `file1.dat`. Plot all extracted histograms in a single pdf file generated with gnuplot with a logarithmic y axis.

What you are seeing is in fact a Monte Carlo simulation (we deal with this technique later in the course) of Higgs boson production at the LHC. The Higgs boson is the peak at 125 GeV and the height of the peak as well as the tail of the distribution is sensitive to the presence of new, yet to be discovered physics. The different curves are exactly that: new physics that shifts the curves up and down. By measuring this distribution at the LHC and comparing it to such calculations we can constrain the presence of such new physics or even discover it!

5 Typesetting: LaTeX

Many, if not most, research physicists write their scientific papers using a package called LaTeX(pronounced *lah-tek*). For example, it is the preferred format for submission at arXiv.org (an on-line archive for papers in physics, mathematics, computer science, quantitative biology and statistics). This is because it has by far the best utility in typesetting mathematical formulae of any package available. In fact, this document is written using LaTeX.

Most of the editors you are probably more familiar with are WYSIWYG, that is **w**hat-**y**ou-**s**ee-**i**s-**w**hat-**y**ou-**g**et. The document you will finally print looks exactly like the document you are typing in. This is not so for LaTeX – you need to write your text and commands in a file of the form `filename.tex` and "compile" it using the command

```
latex filename
```

This will produce a `dvi` file called `filename.dvi` which you can turn into *postscript* using

```
dvips filename
```

(Remember that you can view postscript using `gv`.)

▶ **Task:** Using your web browser, navigate to

```
http://www.andy-roberts.net/misc/latex/
```

and follow the tutorial on LaTeX. If you need more detail, take a look at `latex_short.pdf` in the `/data/p3_t_lab` directory. (You can also view pdf files using `gv`.)

▶ **Task:** Write a shell script which will, with one command, LaTeX a file (given in the command line), run `dvips` on the resulting file to produce a *postscript* output and remove all the intermediate and junk files like `.dvi`, `.aux` and `.log`.

▶ **Task:** Write a LaTeX template for a report. By "template" I mean that you should include a title, abstract, an introduction, conclusions, and a bibliography, but the content of these is unimportant (for now). However, also include in the main body the equation described in Section 5, and a figure containing the plot from the last task of section 5 (with a caption). Also refer to both the equation and the figure in the main text using automatic label numbering (i.e. `\label` and `\ref`). This will provide you with a template for a report you will write later. You might want to check out BibTeX, too. This is a typesetting add-on for LaTeX(keep in mind that you will have to execute the `latex` command multiple times before you `bibtex` for all necessary log files to be present).

For more on the integration of LaTeX and *gnuplot* (which is essential if you want axis labels, titles or legends with complex mathematics) also take a look at:

```
http://www.gnuplot.info/docs/tutorial.pdf
```

6 Programming in C++

6.1 Using the C++ tutorials

In the second part of our laboratory on Scientific Computing we will give a quick reminder of how to program with C++. Knowledge of a computing language if essential to scientists because most advanced physics problems do not have easily obtainable analytic solutions. Even when they can be solved analytically, the evaluation of the solutions for particular parameters would be tedious (and error producing) to do by hand, particularly if the solution is required for more than one set of parameters. Computers allow us to do calculations which would otherwise be impossible or impractical to do by hand.

For a long time, FORTRAN77 was the language of choice, and indeed is still used by many researchers. However, the more standard choice now-a-days is C++, so that is the language we will learn here. This has the advantages of allowing *object orientated* code, and of being the most widely used computer language in the "real world" outside of academia (thus improving your job prospects if you can put it on your CV). You should be careful not to confuse C++ and C. C is very like C++ except that it lacks object orientation and a few other features. Incidentally, Linux is written in C.

In fact, the widely used GNU compilers do not make too much of a difference between the supported languages, however, it is clear that object-orientation is a key feature of C++ and its full functionality is, with some exceptions, is reserved to source code written in C++.

It is worth pointing out that coding is a very dynamical skill, which is best acquire through "learning by doing". Programming and scripting languages like C++ and bash are subject to constant improvements and developments. Examples for this are recent developments leading to Objective-C, Python, Swift, C# etc. The logical basics behind coding and its application to physics problems, however, remains the same. The choice of programming language then depends on the type of a problem (e.g. creating a fancy webpage vs simulating Higgs boson production at the Large Hadron Collider), taste, and code elements that are already available.

We will tackle a number of fairly easy tasks to brush up your C++. Again online courses and tutorials will be helpful throughout this part. Some of the best publicly available online tutorials on C++ can be found at the following web addresses:

```
http://www.learncpp.com/
http://www.cplusplus.com/doc/tutorial/
http://www.cprogramming.com/tutorial.html#c++tutorial
```

We will be using the first of these, but they are all very good, so I encourage you to take a look at the others one as well (bookmark them in your browser). If there are any concepts that we cover which are not clear to you from reading the first tutorial, have a look at the others and see if it makes it any clearer.

Remember to document your work. Also remember that the best way to learn programming is to try it for yourself, so feel free to copy and paste code examples from the tutorials and try them for yourself - experiment by altering them a bit and see what happens.

6.2 Tasks

There are no assigned tasks for Chapter 0, while for Chapters 1 and 2, the tutorial's "comprehensive quiz" should be done. With all tasks, remember to properly document your work - your programs should be properly commented and you should keep a record of some sample outputs. You should also be aware that the chapters each task is assigned to are meant as a guide for when you should be ready to tackle them - but if you want to do then out of order, or read ahead further before completing them, that is fine too.

▶ **Task:** A checkout operator wants to minimize the number of bank notes (and pound coins) they give out in change. Write a `C++` program that accepts as input a value for the change in pounds and returns the configuration with the minimum of notes. For example, if the user enters 68, the program should return:

```
Number of 20 pound notes = 3
Number of 10 pound notes = 0
Number of 5 pound notes  = 1
Number of 1 pound coins  = 3
```

▶ **Task:** Write a `C++` program to convert a temperature in Fahrenheit to Celsius and Kelvin. The program should ask for the temperature in Fahrenheit from the user and return the Celsius and Kelvin values.

▶ **Task:** Write a program which will raise any number x to the power of an integer n, where x and n are inputted by the user. (Also try using something else than the standard function pow()!)

▶ **Task:** The Fibonacci sequence $\{a_n\}$ is defined by,

$$\begin{aligned} a_1 &= 1, \\ a_2 &= 1, \\ a_n &= a_{n-1} + a_{n-2} \text{ for } n > 2, \end{aligned}$$

giving

$$1, 1, 2, 3, 5, 8, 13, 21 \ldots$$

Write a `C++` program which takes an integer as input from the user and returns a_n.

▶ **Task:** Write a program that prompts the user to enter an integer between 1 and 1000 and tells them whether or not it is a prime number.

▶ **Task:** Write a program that allows the user to enter an arbitrary (i.e. user defined) number of real numbers, and returns their mean and standard deviation.

▶ **Task:** Write a program which counts (and returns) the number of characters in its own source code file, excluding blanks.

▶ **Task:** Write a program that uses 2-dimensional arrays to multiply together two user defined matrices.

▶ **Task:** Repeat the earlier task where you returned the nth term of the Fibonacci sequence, but this time, implement it in a function (i.e. fib(n) returns the integer a_n). Don't use any loops - do it entirely with a (single) function.

▶ **Task:** Write a program that allows the user to enter an arbitrary sized (i.e. user defined) square matrix, and returns its inverse (nicely formatted please!). Both the input of the matrix and the inversion itself should be done via functions.

▶ **Task:** Make a class for arbitrary sized square matrices. It should contain functions for multiplying matrices together, taking the inverse and calculating the determinant.

Part II
Computer Algebra Systems

7 A general comment on computer algebra systems

In physics we often deal with complicated systems and highly non-linear dynamics. For example in particle physics we are forced to make very complicated numerical simulations to obtain a realistic modeling of collisions as we observe them at the Large Hadron Collider in Geneva.

"Quick-and-dirty" numerical solutions might solve easy problems, but it is almost certain that sooner or later you will encounter a problem that cannot be solved by simply putting it on a computer. This can have multiple reasons: The code becomes too big to compile or it becomes to slow at runtime because you are performing a lot of useless calculations that could be simplified if you had access to the analytical expression. Sometimes you might also want to understand the dynamics of system by directly checking the mathematical formulae, this is obviously impossible if you will just have a numerical implementation.

Quite obviously, we can only scratch the surface of this tool in two weeks and you should think of this section as a guided introduction to the world of Computer Algebra systems. There is enough time during the labs to try out own ideas and maybe even cross check some of your exercises for the supervisions.

8 Mathematica

This is where Computer Algebra Systems come into play. There are a number of these programs, but we'll use Wolfram's MATHEMATICA because it's widely used. MATHEMATICA is very (very!) powerful. You can perform a whole range of analytical but also numerical evaluations, and the idea of this part of the course is to give you an idea about how you can use it to solve your problems in the future. In particular we will see how to read in and output data, manipulate it, visualize it, and fit it to arbitrary functions. We will see how to analytically as well as numerically solve (complicated) differential equations, which would normally want to pull your teeth out.

To access MATHEMATICA, **switch to Windows and start the program there**.

9 Using the help

There is a comprehensive help functionality in Mathematica. If you want to find a specific functionality you can also browse Wolfram's webpage or, as always, try Google.

▶ **Task:** Try

```
?Plot
```

and hit shift+enter (this tells MATHEMATICA to execute the command). Note that Mathematica is case sensitive! You'll get a short description about how to plot simple functions; click on "`>>`" to get more information. Read through the page to plot

$$f(x) = x^2\,, \quad g(x) = \sin x$$

into a single panel. You can export by right-clicking on the canvas and follow the context menu.

10 Basics, Data handling and Visualisation

10.1 Solving (Differential) Equations

As alluded to above, one of MATHEMATICA's strengths lies in the analytical handling of big equations. This functionality can be accessed with `Solve` or `NSolve` if you are interested in a numerical solution in case there is no closed analytical one.

▶ **Task:** We start with a simple example. Type

```
f[x_] := x^4 - x^3 - x^2 + 1
```

This is how you define a function in MATHEMATICA. Try output $f(y)$ or $f(z)$. Plot f on the interval $[-1, 2]$ and $f \in [-1, 1]$. Find the roots of the equation and the maxima and minima of $f(x)$ only using `Solve` and `D` to compute the derivatives.

▶ **Task:** Let's do something related to the Higgs mechanism that underpins electroweak symmetry breaking and which predicted the existence of the Higgs boson discovered in 2012. The Higgs field has a potential energy density

$$V(|\phi|) = -\mu^2|\phi|^2 + \lambda|\phi|^4$$

with $\lambda, \mu^2 > 0$. Find the minimum of the potential energy which we call $v/\sqrt{2}$ (the vacuum, i.e. the state with lowest energy). What is the value of v as a function of μ^2 and λ?

▶ **Task:** Consider the following differential equation

$$\mu\frac{\mathrm{d}g}{\mathrm{d}\mu} = -\frac{7g^3}{16\pi^2}$$

Find the analytical solution (on paper) and compare your result against the MATHEMATICA's solution using the `DSolve` command. What happens if you change the minus sign to a plus sign? Directly access the numerical solution via `NDSolve` and plot the difference of analytical and numerical solution as a function of μ.

In fact the above equation is very famous, and even that famous that David J. Gross, Hugh David Politzer, and Frank Wilczek received the Nobel Prize in 2004 for

writing it down for the first time. It describes how a the strong interaction scales when it is probed at smaller and smaller distances $\sim \mu^{-1}$. Since $g \sim 0$ for large μ, the theory of strong interactions behaves like a theory without interactions at large energies (the minus sign is crucial!). This insight is a basis of the theoretical underpinning of contemporary Particle Physics at Hadron Colliders such as the LHC.

▶ **Task:** Consider te following set of differential equations

$$\begin{aligned}
\mu\frac{\mathrm{d}g_1}{\mathrm{d}\mu} &= \frac{41}{6}\frac{g_1^3}{16\pi^2} \\
\mu\frac{\mathrm{d}g_2}{\mathrm{d}\mu} &= -\frac{19}{6}\frac{g_2^3}{16\pi^2} \\
\mu\frac{\mathrm{d}g_3}{\mathrm{d}\mu} &= -7\frac{g_3^3}{16\pi^2} \\
\mu\frac{\mathrm{d}\lambda}{\mathrm{d}\mu} &= \left(\frac{3g_1^4}{8} + \frac{3g_1^2g_2}{4} + \frac{9g_2^4}{8} - 6y_t^4 - (3g_1^2 + 9g_2^2 - 12y_t^2)\lambda + 24\lambda^2\right)/(16\pi^2) \\
\mu\frac{\mathrm{d}y_t}{\mathrm{d}\mu} &= \left(\frac{9y_t^2}{2} - \frac{17g_1^2}{12} - \frac{9g_2^2}{4} - 8g_3^2\right)y_t/(16\pi^2)
\end{aligned}$$

subject to the initial conditions

$$g_1(80) = 0.35,\ g_2(80) = 0.62,\ g_3(80) = 1.22,\ \lambda(80) = 0.13,\ y_t(80) = 1$$

Plot the solution curves in a single canvas. Can you find an input value for which λ becomes negative for large μ?

What you have solved here is the Renormalization Group Equations of the couplings in the Standard Model. This tells you how the coupling constants scale with the energy at which we probe them. λ is a very particular parameter as we have seen in the previous task! It is a necessary parameter to have $v \neq 0$. If $\lambda < 0$ then we only encounter the trivial vacuum of the previous task $v = 0$. At high enough energy the vacuum would destabilize and we would have a first order phase transition from our vacuum to the trivial vacuum.

As a consequence, the weak interactions would be a lot stronger and the proton would outweigh the neutron, we would have inverse beta decay at a very rapid rate. Hydrogen could not form not speaking of a complex chemistry like us! You can translate the validity range of λ onto a validity range of the Higgs boson mass, and the observed 125 GeV is consistent with a stable Universe (thankfully).

Note that the solutions to the coupling constants g_1, g_2 and g_3 almost overlap at high μ. If they actually meet there is a chance that they can be described in terms of a single coupling g at the energy at which they meet. In particle physics we interpret this as a potential sign of grand unification, where all interactions follow from a single one. Obviously this is not the case for the SM spectrum. However, in supersymmetric extensions of the SM we get corrections to the above equations an in fact the curves meet in a single point. This among other reasons is why Supersymmetry is one of theoretical physicists' most beloved theories.

Let's have another look at a standard problem in Theoretical Mechanics:

▶ **Task:** Solve the harmonic oscillator equation

$$\ddot{\theta} = -k\theta \tag{2}$$

using `DSolve` for initial conditions $\theta(0) = 1,\ \dot{\theta}(0) = 0$.

You all know that the harmonic oscillator is a solution for a pendulum with small amplitude. In this case we expand the equation of motion. But we can do the whole thing numerically!

▶ **Task:** Solve the differential equation

$$l\ddot{\theta} = -g \sin\theta \tag{3}$$

numerically for the same initial conditions, you can assume $g/l = 1$ and forget about units. Compare the two results graphically.

10.2 Elementary Data handling

▶ **Task:** Lets do some data manipulation; access one of the file from a previous task of Sec. 4.1. Read in the data set using `Import`. Define an empty list, you can do this by

```
mylist={};
```

Fill the list with the read-in data from the file. You will need the `Do` and `Append` functions to loop over the initial data set and redefine the empty list. Plot the list copied list using `ListPlot`.

Sometimes data in the tails of distributions can become a bit bumpy if you have used Monte-Carlo driven methods to simulate it and you might need to extrapolate it or determine the parametric dependence of the histogram on some underlying model parameters (like the mass of a particle for instance). To do this it is often convenient to fit a distribution. MATHEMATICA can fit arbitrary functions (in principle) with the `NonlinearModelFit` functionality.

▶ **Task:** Create a new list and fill the list with entries that have an x axis value bigger than 400. Plot the list on a logarithmic scale with `ListLogPlot`. What function would you pick as function to fit the data? Fit that function to the data, overlay data and function in a single plot and also include the 95% confidence level bands. Is the fit reliable?

▶ **Task:** Instead of fitting the data directly, create a list that stores the logarithm of the y values and repeat the previous task.

Plot	Plot functions
ListPlot	Plot lists
ListLogPlot	Plot lists on a logscale
Plot3d	Plot 3d functions
Simplify	Simplify algebraic expression
FullSimplify	Higher level simplification of algebraic expression
Series	Taylor Expansion of functions
Integrate	analytical Integration
NIntegrate	numerical Integration
Diff	Differentiation
DSolve	Solve differential equations analytically
NDSolve	Solve differential equations numerically
NonlinearModelFit	Fit arbitrary functions

Table 2: Some helpful MATHEMATICA commands.

10.3 More on Functions

MATHEMATICA is not only capable of doing (pseudo)numerical operations, but you can also define intrinsic functions.

▶ **Task:** Define a function that Taylor-expands an arbitrary function $f(x)$ around $x = 0$ and returns only the term linear in x, e.g.

$$f(x) = 1 + x + x^2 \quad \overset{\text{returns}}{\Longrightarrow} \quad x\,. \tag{4}$$

Try the function with $f(x) = \sin(x), \cos(x), \exp(32\pi^2/x^2)$. Why is there an issue with the last function? What is the generalization of the above function to monomials of arbitrary degree?

▶ **Task:** Take the function of the previous task and modify it such that you compare the numerical integral value of the returned function to the numerical integral value of the original function for input interval $x \in [a, b]$.

Part III
Numerical Techniques with C++

11 Using computers to solve physics problems

The vast majority of real life physics problems do not have simple exact analytic solutions that can be easily obtained. Even for your lecture courses, your lecturers often go to great lengths to think up special case examples that you can solve analytically. This leaves two options:

- make approximations so that an approximate solution can be found (perturbation theory in quantum mechanics is a good example of this);
- solve the mathematics numerically after completing as much of the calculation analytically as is practical.

In this course, we will focus on the latter. Remember, however, that you can always try to and should cross check results against different calculations using, e.g., MATHEMATICA.

In most cases, numerical problems in physics boil down to one of three things:

Algebra: the manipulation of numbers, vectors, matrices etc., whose manipulation, though conceptually straightforward, is too long and arduous to do by hand. Hopefully, the course on C++ has equipped you to handle this already (for example, inverting a large matrix), so we won't cover this in any more detail here.

Numerical Integration: Most often our analytic calculations are stopped by encountering a nasty integral. Maybe it doesn't have a closed form, or if it does, it is too complicated to reliably write down, or maybe we just want the flexibility to change the integral's boundary conditions without having to recalculate everything. This is probably the most important class of numerical problem for physics, so this is where we will spend most of our time.

Differential Equations: Many of the laws of physics are described in terms of differential equations. For example, wave mechanics, quantum mechanics, fluid mechanics, thermodynamics, electromagnetism, general relativity and even classical dynamics, all involve differential equations.

Root Finding: Often in solving equations in physics one has to find the roots of some complicated function or polynomial. This can be a highly non-trivial task, and must often be done numerically. Unfortunately we won't have time to go into this here.

This categorization is not exclusive - there are many other cases - and, of course, it is rather fluid. Many problems that appear at first sight to require a numerical treatment

of differential equations are more easily done via numerical integration. An obvious example would be solving the differential equation,

$$\frac{dy}{dx} = f(x), \tag{5}$$

which can be expressed instead as,

$$y = \int f(x)\,dx. \tag{6}$$

(One cannot always do this though since the right-hand side may be a function of both x and y, i.e. $f(x, y)$.)

12 Numerical Integration

12.1 Integrals as averages

In essence, all numerical integrations are attempts to find the mean value of a function over a (possibly multi-dimensional) integral,

$$I = \int_{x_1}^{x_2} f(x)\,dx = (x_2 - x_1)\,\langle f(x)\rangle. \tag{7}$$

The most obvious way to calculate this numerically is simply to take some example choices of x, let's call them x_i with $i = 1 \ldots N$, calculate $f(x_i)$ and take the average of value of $f(x_i)$ to approximate the mean,

$$I \approx (x_N - x_1)\frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{8}$$

We could be slightly more sophisticated and put *weights* w_i on each of these to make some choices of x_i more important than others,

$$I \approx (x_N - x_1)\frac{1}{W}\sum_{i=1}^{N} w_i f(x_i), \tag{9}$$

where the extra normalisation factor is $W = \sum_{i=1}^{N} w_i$. Different numerical integration techniques differ in their choice of how to pick a representative sample of x_i and their relative importance (what weights to choose).

12.2 Trapezoidal Rule

The trapezoidal rule, in its simplest form, is simply approximating the area under a curve by that of a trapezoid. Clearly, this is Eq.(9) with $N = 2$ and $w_i = 1/2$. We could go further and split up the integration region into lots of little trapezoids of equal

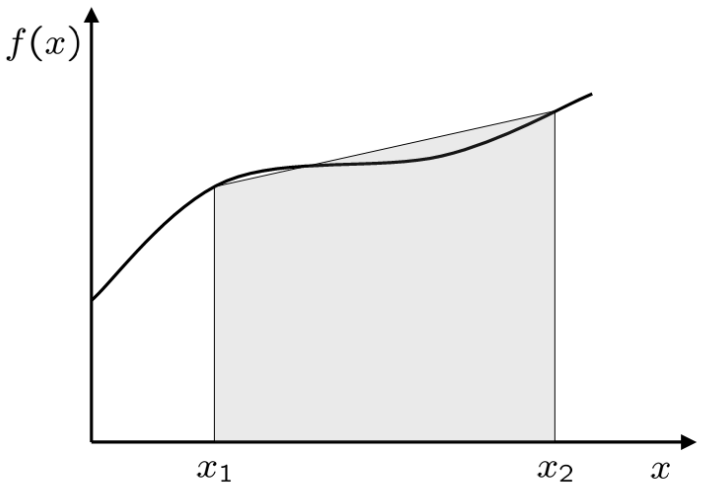

Figure 3: A graphical representation of the simplest ($N = 2$) trapezoidal rule.

spacing in x. The total area will then be the sum of these smaller areas which we can approximate via a trapezium. So,

$$\begin{aligned} I &\approx (x_2 - x_1)\frac{1}{2}(f(x_1) + f(x_2)) + (x_3 - x_2)\frac{1}{2}(f(x_2) + f(x_3)) \\ &\quad + (x_4 - x_3)\frac{1}{2}(f(x_3) + f(x_4)) + \ldots + (x_N - x_{N-1})\frac{1}{2}(f(x_{N-1}) + f(x_N)) \quad (10) \\ &= (x_N - x_1)\frac{1}{N-1}\left[\frac{1}{2}f(x_1) + f(x_2) + f(x_3) + \ldots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right], \quad (11) \end{aligned}$$

where, in the last step, we have assumed that all the intervals of equal size (so for example, $(x_2 - x_1) = (x_N - x_1)/(N - 1)$). Clearly this is Eq.(9) with $w_1 = w_N = 1/2$ and $w_i = 1$ for $1 < i < N$.

► **Task:** Write a `C++` code to calculate the integral

$$I = \int_0^2 \left[2 + \cos(2\sqrt{x})\right]\, dx,$$

using the trapezoidal rule. In this case, the integral is doable analytically, so compare your answer to the correct expression for $N = 2, 4, 8, 16$. Can you see any pattern in the error's dependence on N?

12.3 Simpson's Rule

In the trapezoidal rule we are essentially approximating the curve by a set of straight lines between fixed points on the curve. We could have instead approximated the curve

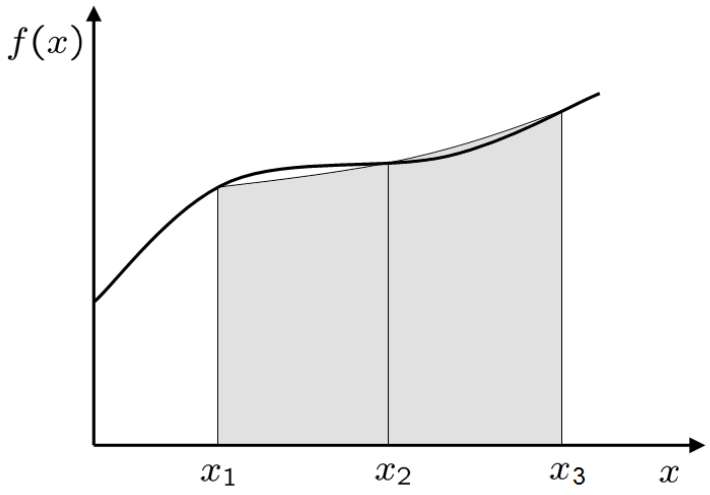

Figure 4: A graphical representation of the simplest ($N = 3$) Simpson's rule.

using a set of parabolas rather than straight lines, and if the curve is quite "wiggly" (that is, not straight) we might expect to do a little better, see Figure 4.

You can calculate the area under a parabola quite easily, and anchoring it at the end and in the middle we find,

$$I \approx (x_3 - x_1)\frac{1}{2}\left[\frac{1}{3}f(x_1) + \frac{4}{3}f(x_2) + \frac{1}{3}f(x_3)\right]. \tag{12}$$

This does much better than the simplest trapezoidal rule. The generalization to more points (in exactly the same way as we did for the trapezoidal rule) is,

$$\begin{aligned} I \;\approx\; & (x_N - x_1)\frac{1}{N-1}\left[\frac{1}{3}f(x_1) + \frac{4}{3}f(x_2) + \frac{2}{3}f(x_3) + \frac{4}{3}f(x_4) + \ldots\right. \\ & \left. + \frac{2}{3}f(x_{N-2}) + \frac{4}{3}f(x_{N-1}) + \frac{1}{3}f(x_N)\right], \end{aligned} \tag{13}$$

or in other words Eq.(9) with $w_1 = w_N = 1/3$, $w_i = 4/3$ for $1 < i < N$ and i even, and $w_i = 2/3$ for $1 < i < N$ and i odd.

► **Task:** Repeat the last exercise using Simpson's Rule. What is the dependence of the error on N this time?

12.4 Gaussian Quadrature

In the previous example, Simpson's Rule, we used a parabola to mimic the behaviour of the function over the range (x_1, x_3) and then extended this by dividing our integration up into lots of "mini-integrations". We could have taken an alternative approach, and instead of dividing up the region, we could have mimicked the function with more

complicated polynomials. Clearly these more complicated polynomials would also require more points in order to specify them, but if we choose a set of polynomials which is *complete* (i.e. any well behaved function can be built out of them) then we should be able to get arbitrarily close to the exact answer just by including more of them. The terminology is that an n-point Gaussian Quadrature yields an exact result for the integration of a $2n-1$ degree polynomial.

There are many different types of Gaussian Quadrature, depending on the polynomials you choose to mimic your function with. Examples of these functions are Legendre polynomials, Jacobi polynomials, Chebyshev polynomials, Laguerre polynomials, Hermite polynomials and many more. Since you have not met most of these functions yet, this integration method is a little too advanced for this course and we won't take it any further.

12.5 Monte Carlo Integration

In principle, it shouldn't matter how we choose our x_i and our weights w_i, we should eventually, for N large enough, reproduce the correct answer for any well behaved function. (Although how we choose our x_i and w_i does effect how fast we converge on the correct result.) Monte Carlo integration takes this principle to the extreme and chooses the x_i *randomly*!

Looking again at Eq.(9) (this time with weights $w_i = 1$ for simplicity),

$$I \approx (x_N - x_1)\frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{14}$$

we see no specification for the values x_i. We just assumed that they should be uniformly spaced, but this didn't need to be the case.

▶ **Task:** Repeat the numerical integration of

$$I = \int_0^2 \left[2 + \cos(2\sqrt{x})\right]\, dx,$$

this time using random values for x_i and unit weights. You can use the `C++` function `rand()` to generate random numbers. How fast does this converge?

If you have done the above tasks correctly, you will see that the Trapezoidal Rule has an error $\propto 1/N^2$, Simpson's rule has an error $\propto 1/N^4$ and the Monte Carlo integration has an error $\propto 1/\sqrt{N}$.

In fact, the error of a Monte Carlo integration is related to the standard deviation, σ, of the integrand over the region. For a one dimensional integration, this is defined by,

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left[f(x_i) - \bar{f}\right]^2 = \frac{1}{N}\left(\sum_{i=1}^{N}\left[f(x_i)\right]^2\right) - \bar{f}^{\,2} \tag{15}$$

where the mean is,

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{16}$$

Then the Monte Carlo error is

$$\sigma_{\mathrm{MC}} = \frac{\sigma}{\sqrt{N}}. \tag{17}$$

12.6 The advantages of Monte Carlo integration

So if the convergence of Monte Carlo integration is slower than the other methods we have already looked at, what is it good for?

- **Fast convergence for many dimensions:** Although the other methods give much smaller errors for one dimensional integrations, their errors grow with the number of dimensions. For a d dimensional integration, the Trapezoidal Rule's error is $\propto N^{-2/d}$ and Simpson's Rule's error is $\propto N^{-4/d}$, but a Monte Carlo integration error is always $\propto 1/\sqrt{N}$.

- **Easy to incorporate awkward boundary conditions:** In the one dimensional cases we have looked at so far, the boundary conditions were rather trivial (just numbers), but for higher dimensional integrations they can become quite complex (surfaces). This make the other rules quite difficult to program because you need to know explicit formulae for the boundaries in terms of the variable you are integrating over. For Monte Carlo integration, you can simply choose to generate the $\vec{x}_i$ over a larger space which has simple boundaries, check whether or not the point lies in the space you are interested in, and throw it away if it isn't.

- **Small feasibility limit:** You can see that the trapezoidal rule and Simpson's rule both need at least two or three x_i before you even start. While this is fine for small numbers of dimensions, for larger dimensional integrals you need to have at least this number of points raised to the power of the number of dimensions. For example, for a 10-dimensional integration, Simpson's rule needs a minimum of $3^{10} = 59049$ points! This is not the case for a Monte Carlo integration where in principle one point (or two if you want an estimate of the error) will do (but of course would give very bad errors).

- **Ability to improve the calculation:** For other integration methods you are "locked in" to the accuracy you choose at the start of your calculation. For example, in Simpson's rule, you choose N and then set your x_i and w_i accordingly. If you then decide that your answer is not accurate enough, you need to increase N and start the calculation again. For a Monte Carlo integration you can just keep adding points until you get the accuracy you want - N is not fixed at the start.

- **Integration as physical simulation:** Many physics systems are stochastic, relying on apparently random effects (e.g. thermodynamics) or possible even truly random effects (e.g. quantum mechanics). Monte Carlo's can then model physical systems where each generated point is analogous to a physical event. This gives a much more intuitive physical picture of what is going on in the Monte Carlo integration/simulation.

► **Task:** To demonstrate the second of these advantages, write a `C++` program to calculate the volume of a sphere embedded in $n \leq 10$ dimensions. Compare your numerical result against an analytical calculation. For 10 dimensions you should get

$$V = \frac{\pi}{120} r^{10},$$

where r is the radius. (This problem is much easier than it sounds, so if you are having trouble figuring it out, please *ask* one of the demonstrators for hints!)

► **Task:** Consider a crescent, as seen in Figure 5, and bounded by the curves,

$$x^2 + y^2 = a^2, \tag{18}$$
$$(x - c)^2 + y^2 = b^2, \tag{19}$$

were a, b and c are constants with $b < a$ and $a - b < c < a + b$. Assuming it is uniform in the z-direction (out of the page) write a `C++` program to calculate its moment of inertia about the axis $x = y = 0$.

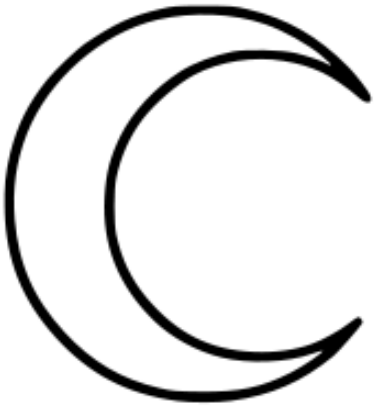

Figure 5: A crescent.

Note that both of the above tasks would be very difficult to do using non-Monte Carlo numerical integrations (though the first one is straightforward to do analytically).

12.7 The importance of flat distributions

It should be fairly obvious that if the distribution we are integrating is perfectly flat, then $\langle f(x) \rangle = f(x)$ for all x and all these integration methods give a perfect result with smallest N. Of course, in this case one would not be integrating it numerically! However, it is generally true that these methods all work better if the integrand is reasonably flat, and work poorly if the integrand has a strong dependence on the parameter you are integrating over. So, if we can manipulate our integration to make

it reasonably flat before numerical integration, we will need far fewer points for the integration to reach the desired accuracy.

So how do we do this? Consider integrating a function $f(x)$, and suppose we know of a function $g(x)$ which *we know how to integrate analytically* which is approximately the same as $f(x)$ over the range of x we are integrating over. Then,

$$\int f(x)dx = \int \frac{f(x)}{g(x)}dy, \tag{20}$$

where

$$dy = g(x)dx \quad \Rightarrow \quad y = \int g(x)dx. \tag{21}$$

Since we know how to integrate $g(x)$ we can work out y analytically. A numerical integration over y will be more efficient than an integration over x since $f(x)/g(x)$ is reasonably flat.

Let's have a concrete example. Consider the integration,

$$I = \int_{-10}^{10} \frac{1 + \frac{1}{4}\cos x}{1 + x^2}\, dx, \tag{22}$$

As shown in figure 6. Unaltered, this would be difficult to integrate numerically since our Monte Carlo will very often choose values of x that contribute little to the mean.

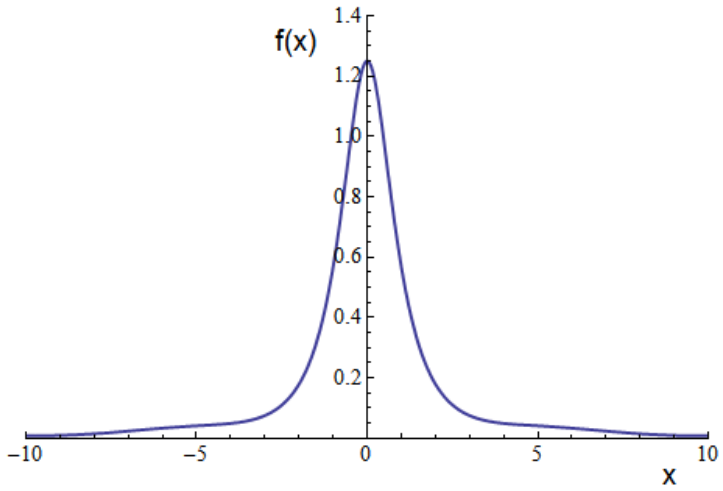

Figure 6: The function $f(x) = \left(1 + \frac{1}{4}\cos x\right)/(1 + x^2)$.

However, the function $g(x) = 1/(1 + x^2)$ is easy to integrate,

$$\int \frac{1}{a^2 + x^2}\, dx = \tan^{-1} x + \text{constant}, \tag{23}$$

so we can change variables to $y = \tan^{-1} x$ and write,

$$I = \int_{-10}^{10} \frac{1 + \frac{1}{4}\cos x}{1 + x^2}\, dx = \int_{-\tan^{-1} 10}^{\tan^{-1} 10} \left[1 + \frac{1}{4}\cos\left(\tan y\right)\right] dy. \tag{24}$$

This is now quite flat, so the error is greatly reduced.

▶ **Task:** Calculate the integral Eq.(22) using a Monte Carlo integration, first using x as your integration variable and then $y = \tan^{-1} x$. In both cases, quantify the Monte Carlo error (as in Eq.(17)) and compare between the two methods.

12.7.1 Interlude: Monte Carlo Methods in Particle Physics

Flat distributions are massively important if we want to have a quick an accurate Monte Carlo integration. In particle physics, we often deal with virtual intermediate particles that only live for a very short time. If we imagine a collection of particles, their number will decay exponentially; in Quantum Mechanics we can then speak (a bit inaccurately) of an exponential decay of a particle's wave function with time: $\sim \exp(t/\tau)$, where τ is the characteristic lifetime of the particle. We prepare particles at colliders with a given momentum rather than at a given space so it's better to write everything in Fourier space, where the exponential decay reads

$$\sim \frac{1}{E_{\rm cm}^2 - m_i^2 + i m_i \Gamma_i}$$

where Γ_i is the inverse lifetime. How does this function look as function of the centre of mass energy $E_{\rm cm}$ and Γ_i/m_i, and why is this distribution complicated to sample?

▶ **Task:** Let's have a look at how particle physicists work with Monte Carlo programs in this context.

Download MadGraph[4] from

`http://www.christophenglert.co.uk/MG5.tar.gz`

Unpack it and run it via `./bin/mg5`. We want to simulate electron-positron production at the Large Hadron Collider. To do this, type the following commands

- Specify the process: `generate p p > e+ e-`
- Set up the numerical code: `output testrun`
- Run the code: `launch testrun`. Type `0` twice to start the program.

The webbrowser should open giving you details about the simulation (alternatively you can navigate to `./MG5_aMC_v2_1_2/testrun/index.html`. When the run is finished click on `LHE` to look up the directory of the results. Alternatively you can `cd` to `MG5_aMC_v2_1_2/testrun/Events/run_01`. The file we are looking for is

[4] MadGraph is widely used in the High Energy Physics Community. The authors of the code provide details here: http://arxiv.org/abs/arXiv:1106.052.

`unweighted_events.lhe.gz`

Let's analyse the data. Get the analysis code from

`http://www.christophenglert.co.uk/Analysis_labs.tar.gz`

`cd` to the unpacked directory and type `make` to build the analysis code and copy the data file into this directory and unzip it with `gunzip`. Launch the analysis with `./plot_events` and type in the name of the unzipped data file. The code produces a file `plots.top` that is ascii. Look for "mij" and plot the histogram (the three columns quoted under the header). This is a histogram that gives you the invariant mass of the electron positron pair. Have a look at the process information in the browser. Where does that sharp peak come from?

12.8 Monte Carlo as a physical simulation

So far, Monte Carlo's have been just a tool for performing integrations. However, as described in section 12.6 they can be used to simulate experiments with some random element.

To understand this, let's again consider an example: Buffon's Needle. In 1777 the *Comte de Buffon* posed the following problem: imagine a board with lines drawn on it at equal spacing t, upon which a needle of length l is randomly thrown; what is the probability of the needle landing across one of the lines. See Figure 7 for a pictorial representation.

Analytically, this is quite straightforward. If x is the distance of the centre of the needle to the closest line, and θ is the angle between the needle and the lines, then it will cross the line if,

$$x \leq \frac{l}{2}\sin\theta. \tag{25}$$

Since x is random between $0 < x < \frac{t}{2}$ and the angle θ is random between $0 < \theta < \pi/2$ then the probability of the needle landing on a line is,

$$\frac{\int_0^{\pi/2} d\theta \int_0^{(l/2)\sin\theta} dx}{\int_0^{\pi/2} d\theta \int_0^{t/2} dx} = \frac{\int_0^{\pi/2} d\theta \, \frac{l}{2}\sin\theta}{\int_0^{\pi/2} d\theta \, \frac{t}{2}} = \frac{l/2}{\pi t/4} = \frac{2l}{\pi t}. \tag{26}$$

So one could, in principle, determine the value of π by throwing lots of needles on such a board: if you throw N needles and n land on a line, then,

$$\pi \approx \frac{2lN}{nt}. \tag{27}$$

This experiment has been done many times, most famously in 1901 by an Italian mathematician called Lazzerini, who found $\pi = 3.1415929$ after 3408 throws .

We can *simulate* this experiment on a computer by generating random numbers and using them give us values of x and θ for each needle thrown. For each x and θ choice

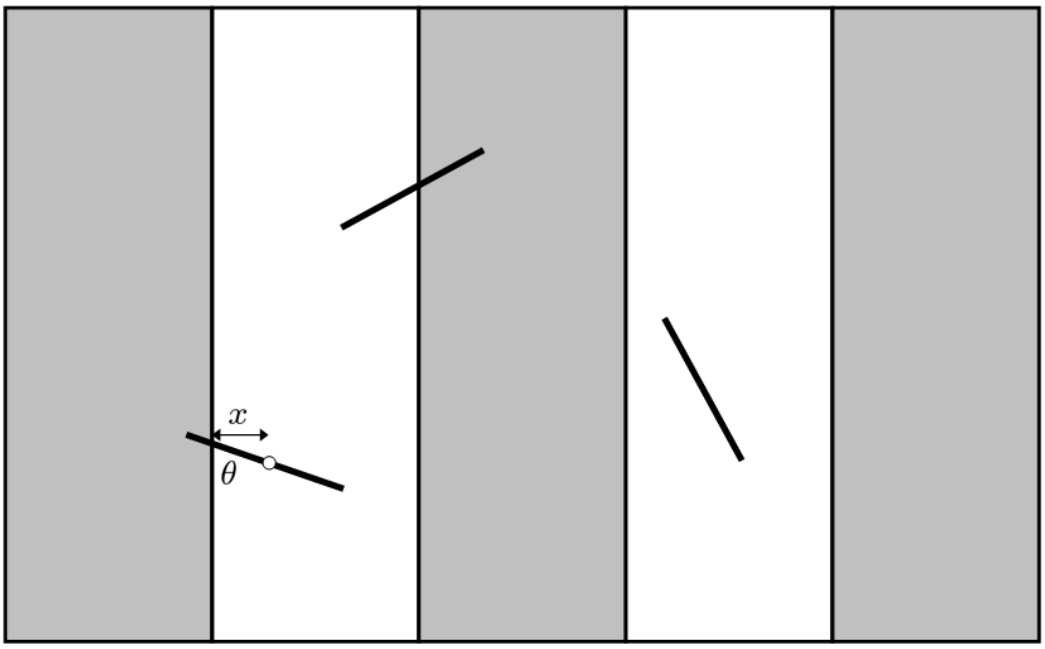

Figure 7: Buffon's Needle problem. A needle of length l is randomly thrown onto a board with lines drawn on it at equal spacing t. What is the probability of the needle landing on one of the lines?

we then decide if the needle crosses a line (i.e. if $x \leq (l/2)\sin\theta$) and add one to n if it does. In essence, we are using a Monte Carlo to do the integration in Eq.(26), but due to the probabilistic nature of the experiment, this is also a simulation.

NB: You should realise that this is not an actual calculation of π since we need to input π into our program in order to generate a random number between 0 and $\pi/2$ (for θ). This is only a *simulation* of an experiment. (Can you think how we might modify it to actually calculate π with no π inputted?)

12.9 Final task

This prompts us to the final task of TP3:

► **Task:** Write a Monte Carlo to simulate Buffon's experiment and quantify your Monte Carlo error on the measurement. Write a report on this task. Your report should be written using LaTeX (remember you made a report template when we first learned about LaTeX). Your report should include an abstract, and an introduction that explains why the task is interesting and provides a summary of the "project". In the main part of your report you should explain how you solved the problem, including the algorithm (without detailed `C++` code) and present your results. Remember to make an estimate of your errors! Finally you should draw some conclusions and summarise your work, making suggestions on how your calculation could be improved (or not!). Your

`C++` code and outputs should be reserved for an appendix. If you use anybody else's algorithm, results, formulae or information, be sure to reference them. Remember that the report should comprehensible to somebody who has not participated in TP3 - just referencing these course notes is *not* enough.

The report should be handed in via Moodle by the end of week 12 for assessment.

13 Feedback

▶ **Task:** An optional task is to provide feedback on the course. Please send an email to `christoph.englert@glasgow.ac.uk` detailing your thoughts. Please split this into three sections:

- Things you liked about the course. Let us know what parts you found useful. Do you feel you have learned new skills from the course? Do you understand the motivations and role of scientific computing? Was it fun?!
- Things you didn't like about the course. Was the course too easy or too hard? Did you feel under too much pressure? Was the course too long? Where the tasks unhelpful? Was the course poorly motivated and/or explained?
- Things you would change. If you were teaching this course yourself, what would you change? How would you teach this course differently?