# Introduction to Computing in the Oxford Physics Course

Being able to write a computer program is an important skill for all physicists. In physics, computing is mainly required for the following types of task:

1. Analysis of experimental data
2. Solving numerical problems such as differential equations
3. Controlling scientific instruments and acquiring data from them

We aim to teach you all these skills during your time in Oxford. In the first year we focus on the first two aspects. The third aspect is generally learnt by doing experiments in the teaching labs, where you will experience some data acquisition software during first year practicals. You will spend more time writing your own code for data acquisition and control in the second and third year lab experiments.

Physicists use many different computing packages, and our philosophy is to expose you to a range of different programming environments during your time in Oxford, so that, like professional physicists, you learn to choose which you prefer for different tasks. In the first year we teach you two programming languages, RStudio (which uses an underlying language called R) and Matlab. Your first lab classes will be introductory exercises in both languages.

Many modern computing languages are interchangeable in the sense they can all be used to carry out the tasks above, although there are a few differences between languages, which you will learn about in the computing lectures. In the first year, we encourage you to use RStudio for data analysis, whereas Matlab is used to train you mainly in solving numerical problems.

Here we provide some introductory resources that use free and friendly web-based tools to get you used to the basics of computing. We also provide resources for a third language, Python, which we do not formally teach, but is easy to learn, and popular with scientists due to its versatility. Python and R/RStudio are open source which means they are free to download and use, whereas Matlab needs a license and cannot be downloaded in its full form before you arrive in Oxford. All three systems are essentially independent of operating system i.e. it will not make a difference if you try something on a Mac at home and use a Windows machine in Oxford, or vice versa.

*We strongly recommend that you do EITHER*

- *BOTH the online introductions to Matlab and R/RStudio. Then download R/RStudio and do the short data analysis exercises provided.*
- *OR the CodeAcademy Python course*

**Physics and Philosophy student?** You will need to do an exercise in R when you start lab work in Oxford in the second year, so best to focus on this. However, if you are keen to learn to code, we encourage you to do the Python course which should equip you well for anything you might need to do later on.

**Matlab**

An online introduction to Matlab, Matlab Onramp, is available here:

https://matlabacademy.mathworks.com/

The course is web-based (you must register with the provider) and uses an interface similar to Matlab. It covers the basics of programming in Matlab and develops many of the skills you will need for the first year computing course. The course providers suggest that the course will take about two hours. You can work through the sections in any order.

If you do want to try out the full version of Matlab before you come to Oxford, a free 30-day trial can be downloaded.

https://uk.mathworks.com/programs/trials/trial_request.html

**RStudio (R)**

The recommended introductory course is called Try R:

http://tryr.codeschool.com/

Like the Matlab course, this uses a web interface that is similar to R, but does not require you to download the full package. It provides a simple introduction to many of the key concepts, starting at a very basic level. The sections must be worked through in order, but they are all short.

R can be downloaded from here:

https://cran.r-project.org/

The R interface is relatively simple, and very similar to the Try R environment, with a window to enter commands (called the "command line") and with plots appearing in an adjacent window. Once you have completed TryR online, you should download and install R and do the data analysis exercises that we provide (**at the end of this document for now, but will need a link**) in the command line.

In the Physics Teaching Labs we use RStudio, which is a more "friendly" interface for programming in R, and provides more "point and click" functionality. The underlying language is exactly the same. RStudio can be downloaded here (note that you will need to download R first)

https://www.rstudio.com/products/rstudio/download/

**Python**

CodeAcademy run a good introductory Python course:

https://www.codecademy.com/

This online course is longer (13 hours) and goes further than the others. It also provides more general background to programming than the Matlab and R courses, though all are equally good at introducing their language. For this reason, if you take this course you will be well prepared to start our computing course.

**Notes**

- We regret that we are not able to provide IT support before you attend the introductory lectures at the start of Michaelmas Term, however there are many web resources available beyond the ones we have listed here.
- We do not recommend any specific operating systems or types of computer. We have Windows, Mac and Linux systems available for undergraduate use, and our students buy many different types of personal computer.

# Introductory Data Analysis with R

Before starting, you should have completed the TryR online course, and downloaded R to your own computer. Most of the concepts are discussed in our bridging material on *Data Analysis and Statistics*, so it is also a good idea to look at this beforehand.

These exercises use a dataset from the famous Michelson-Morley experiment, in which the speed of light in air was measured in 1879. The dataset is conveniently already stored in R, where it is called "morley". To look at it, you can just type

```
morley
```

into the command line. Because the dataset contains 100 numbers and a header, you probably won't be able to see it all. A more useful way to quickly inspect the data is to try

```
head(morley)
```

which only shows the first few lines of data. The first column gives the row number, and the other three columns are labelled. Each row represents a speed (labelled `Speed`) recorded in a particular "run" (`Run`), and in this case the data are divided into five groups of 20, each labelled an "experiment" (`Expt`). The speeds are in units of kms$^{-1}$ after subtracting 299 000 kms$^{-1}$. You can see the speed data by typing:

```
morley$Speed
plot(morley$Speed)
```

One way to obtain a visual overview of data is to produce a histogram. In a histogram, rectangles are used to represent frequency, with the area of each rectangle proportional to the frequency. A histogram of the Michelson Morley data can be generated with:

```
hist(morley$Speed)
```

If you want to edit the histogram bins, colour and/or labels, these can be added by expanding the `hist()` command as follows:

```
hist(morley$Speed,breaks=25,col="darkgray",xlab="speed - 299 000 km/s")
```

The concepts of mean and median are probably familiar from school maths, and are also mentioned in our *Data Analysis* bridging material. They can be displayed (remembering to include the offset subtracted earlier), with the following commands:

```
mean(morley$Speed) + 299000
median(morley$Speed) + 299000
```

R includes functions that make it very simple to work out the standard deviation and variance of a data set (again, these are explained fully in *Data Analysis*). To save time typing, we can define a variable called Speed before calculating its variance and standard deviation:

```
Speed = morley$Speed
var(Speed)
sd(Speed)
```

Finally, the `summary` command is a useful way to get an overview of a dataset. This lists the minimum, maximum, median, mean, first and third quartiles. (If you don't understand any of these quantities, check *Data Analysis* again.)

```
summary(Speed)
```

Note that the `summary(morley)` command also works (try it), although it is not particularly helpful for this dataset. To continue working out some of the quantities in *Data Analysis*, how would we calculate the standard error in the speed measurement? There is no single command for standard error, but it is easy to calculate, using the `length` function to work out the number of data points

```
N=length(Speed)
sd(Speed)/sqrt(N)
```

**Reference**

This material is mainly taken from *Scientific Inference: Learning with Data* by Simon Vaughan (CUP, 2014)

http://www.star.le.ac.uk/sav2/stats.html