0 Logging in and getting started

- under Start - Programs - School of Physics & Astronomy - Open Text Exceed 14 x64, select **Exceed (passive)**
- under Start - Programs - ITS Applications, select **PuTTY** (you can also just press Start and then type 'putty' to access it).
- In PuTTY load the profile for host **phymat2** or **phymat3**.

  If no such profile exists, enter the following:
  - hostname: phymat2.adf.bham.ac.uk or phymat3.adf.bham.ac.uk
  - connection type: ssh
  - under the menu tree on the left, connection → SSH → X11, enable Forwarding.
- 'open' the connection and enter your username and password (the same as for your mybham login) when prompted
- Type **xclock&** in to your phymat2 window, and ask a demonstrator if this does not produce a new window with a clock in.
- In a Unix system, when typing commands into a terminal, these always refer to a specific folder, the so-called 'current working directory'. You can print the name of this folder with the command 'pwd', and change the folder with the command 'cd'. When you connect to phymat2 or phymat3, the session starts in your so-called 'home' folder, which should be called something like '/home/adf/YOURID' with 'YOURID' replaced by your login name.
- In your home folder type:
  ```
  git clone git://lnx0.sr.bham.ac.uk/compmod2013
  ```
  which will create the folder 'compmod2013' with example and help files that might be useful during the course.

0.1 Programming language and Compiler

The programming for the course should be done in C++ on UNIX (not quincy) - although, other compilable languages can be used if agreed with me first. If you are rusty we would recommend for this C++ tutorial:
http://cplusplus.com/doc/tutorial/
Or this course material:
http://www.ep.ph.bham.ac.uk/courses/cppintro/index.html.

A key point is that in order to compile code on phymat you need to type compilation instructions on the command line. In the simplest case this will have the form:
```
g++ prog.cpp
```
where prog.cpp is the program you wish to compile. Many useful options can be passed to the compiler via so-called *flags*. For example, a much more useful compiler command is:

```
g++ -Wall -o prog.out prog.cpp
```

where prog.out will be your executable and '-Wall' causes the compiler to print warnings when it detects possible errors in the code. To run your program (assuming it compiled successfully) you then need to type: `./prog.out`

More information about the compiler options be found on WebCT for this module in 'Computer info' and generally on the web. In your lab book you should quote some common g++ compiler options and explain the meaning of the options '-lm', '-Wall' and '-O2'.

0.2 Numerical algorithms

For understanding numerical algorithms a good starting point is Numerical Recipes
http://www.nrbook.com/a/bookcpdf.php (note this is an old version, which is free to view).

You may prefer the 'black box' approach and the GNU Scientific library can be used (we will spend a session learning how to incorporate that).

0.3 Unix

You will find that knowing a few UNIX commands will help in designing and running your programs. A few suggestions for Unix tutorials are:
Unix help Edinburgh server: http://unixhelp.ed.ac.uk/
Unix FAQs: http://www.faqs.org/faqs/unix-faq/faq/part1/preamble.html
but you will find many more on the web.

0.4 Graphics

For graphical output we recommend to save the data to be plotted in a file and then use a plotting package such as:
Matlab http://www.mathworks.co.uk/ or
Gnuplot http://gnuplot.info/ or
Grace http://plasma-gate.weizmann.ac.il/Grace/.
Example scripts for Matlab and Gnuplot are provided and there is help on using Grace for displaying real-time output.

0.5 Word processing equations!

You will have probably found by now that equations can be infuriating to type with conventional word processing packages - in fact there is a useful program designed to deal with this - *Latex*. A ready-to-write Latex template for a report is available on WebCT. An introduction to Latex can be found at:
http://www.andy-roberts.net/misc/latex/index.html

and a reference manual is available at:
http://home.gna.org/latexrefman/.

Another good resource is the Latex Wiki Book:
http://en.wikibooks.org/wiki/LaTeX
and it's helpful to bookmark the link to the comprehensive list of symbols in Latex:
http://www.ctan.org/tex-archive/info/symbols/comprehensive/.

For writing Latex documents you can use a simple text editor, such as Notepad++ http://notepad-plus-plus.org/ on Windows or **gedit** on Unix. The Windows machines in P9 should also have the Texnicenter IDE http://www.texniccenter.org/ installed which you might find useful.

It is *not* required that you submit your work formatted using Latex - however, you may find that it is more time efficient to learn how to use it now (particularly if you will be doing a theoretical or mathematics project next year).

0.6 WebCT

Work will need to be submitted via WebCT, please check that you are enrolled, and if not e-mail me at adf@star.sr.bham.ac.uk

The text part of any submission needs to be either postscipt (ps) or pdf. If you are using windows, then CutePDF will generate the pdf for you, otherwise use pdflatex or dvips for LaTeX.

1 Know your computer's limitations!

Answers to ⋆ questions must be submitted to WebCT, as an extensively commented program (source code) and any additional text by the Monday 21/10/2013 4:30pm (usual late penalties will apply). Should there be a problem with submitting to WebCT please send me an e-mail adf@star.sr.bham.ac.uk

You are *not* required to answer the other questions on this sheet - however, they are essentially warm-up questions for the starred question, and have the advantage that you can ask the demonstrators for detailed help.

1. ⋆ This is the first problem that you will need to submit to WebCT. And is to familiarise you with unix, and the idea of typing commands rather than selecting them from menus. Listed below are a number of unix commands. You should type these in to your unix bash shell and submit a pdf file which includes the output and what the command has done.
   (a) mkdir compphys
   (b) cd compphys
   (c) cat > file1.txt [rtn] this is my first file [rtn][ctrl-c]
   (d) ls
   (e) more file1.txt
   (f) xclock & (what does the "&" do?)
   (g) whoami
   (h) man ls
   (i) top (use [ctrl-c] to exit the program once you have seen what it does)
   (j) work out how to **kill** your xclock
   (k) ps -u [username]
2. (a) Check you can compile and run a program: Write a program in your chosen language which sends "Hello World" and on the next line "My name is *student*" to the screen.
   (b) Extend the program so it can now take an input such that it can also print to the screen "your name is *demo*" where one of the demonstrators have typed in the additional name *demo*.
3. Knowing your computer and compiler: it is extremely important when using a computer for mathematical modelling to understand its strengths and limitations - in the first instance you need to know what are the maximum integer sizes it can work with? What limits its accuracy/precision? (remind yourself and write in your laboratory book a working definition for these, you may use the web or a reference text).
   (a) Write a program which finds the largest (positive) integer your compiler represents. Your program should compute an estimate by increasing an integer in a loop. Compare that result to the limit given by the C++ libraries, see:

http://www.cplusplus.com/reference/clibrary/climits/
Then using the fact that the computer works in binary, deduce how many digits are used to store a integer number on the computer.

(b) Write a program that (approximately) finds the largest number that can be represented in single and double precision (float and double in C++). Again, your program should derive the estimate from performing mathematical operations on a float and double variable, and then you should compare your results to the value given by the library constants.

(c) Computer programs often suffer from round-off errors. Find the number closest to 1 that can be differentiated from 1 in both single and double precision - carry out this investigation in both single and double precision. Your program output should appear neatly tabulated with labeled columns.

(d) If you carry out a sum analytically, the order that you sum the numbers in cannot matter: this is not *necessarily* true computationally. Write a program to sum

$$S_{up} = \sum_{n=1}^{N} \frac{1}{n} \qquad \text{and} \qquad S_{down} = \sum_{n=N}^{1} \frac{1}{n}.$$

Explore a wide range of N in both single and double precision and tabulate your results. Justify which sum is more accurate and why this is the case.

4. $\star$ You will need to submit a C++ program file, which should be well commented *and* a pdf (or postscript) file describing the results as well as the maths parts of the questions.

You may ask for demonstrator help with the boxed part of the question.

The "Silver Ratio", see
http://mathworld.wolfram.com/GoldenRatioConjugate.html
is given by $\phi = \frac{-1+\sqrt{5}}{2}$, and is the reciprocal of the more commonly quoted Golden ratio.

In this example we will calculate multiples of the silver ratio by different recurrence methods, and compare to the direct multiplication (calulcated in double precision), which we will treat as our reference value.

(a) write code which will calculate ϕ^N (where N is a user input) and outputs the tabulated code to a file for $n = 1 \cdots N$.

(b) show that $\phi^{n+1} = \phi^{n-1} - \phi^n$ is satisfied. (*this will require you to typset some equations in order to submit to WebCT- now is the time to give Latex a try!*)

(c) Hence we can calculate ϕ^n by subtraction:

$$\phi^{n+1} = \phi^{n-1} - \phi^n \tag{1}$$

you will need to specify two initial values, $\phi^0 = 1$ and $\phi^1 = \phi$ and can then use Eq.1 as a recursion relation.

(d) Use Eq. 1 in both float and double and compare with the answer you calculated in double by directly multiplying up to ϕ^n. You should go up to approximately $N = 50$.

(e) What happens? (*this should be included as text in your write up*)

(f) Why? Show that there are two numbers which satisfy the recursion relation, and use this to explain your results.

For an interesting assortment of places where the golden ratio can be found see http://www.goldenmeangauge.co.uk/

5. ♠**Year 4 only**

Accumulating roundoff errors often limits the ability of a program to perform accurate calculations. Your task is to compute the spherical Bessel function j_l. There exists a recursion relation

$$j_{l-1}(x) = \frac{2l+1}{x} j_l(x) - j_{l+1}(x)$$

In addition the value

$$j_0(x) = \frac{\sin(x)}{x}$$

is known.

(a) Working with fixed x (which should be an input parameter) and choosing arbitrary values for 'large' (50 is sufficient) l (say $j_l(x) = j_{l+1}(x) = 1$). Write a program which will calculate $j_5(x)$ for $x = 0.1, 1, 10$, using the exact value of $j_0(x)$ to scale your results. *I.e.* the ratio of the terms in your recursion relation should be correct and to get the absolute value you only need to work out the scaling at one point, $j_0(x)$.

(b) Check your results, for instance by using Maple. (Make sure you compare with the correct Bessel function!) Include which check you did in your submission.

(c) If we instead use the recursion relation for *increasing* l and use as an additional initial parameter:

$$j_1(x) = \frac{\sin(x) - x\cos(x)}{x^2}$$

we can evaluate, in principle the values of higher l Bessel functions. Tabulate and comment on the results (you may find it helpful to read up on Bessel and Neumann functions!)

2 Root finding

The rest of the worksheet aims to show you the standard methods for numerically finding the roots of an equation $f(x)$. That is, the values of x for which $f(x) = 0$. You will also need to use error analysis to understand the performance of each method.

It is often helpful when looking for the roots of an equation to have an idea of what the function looks like e.g. does it have multiple roots close by? How many roots are we looking for? To understand some of the issues involved in root finding we will stick to 1 dimensional problems. Higher dimensional problems can be solved in similar ways but come with their own set of issues, the difficulty of being able to visualise them being one of these.

Bisection method

The easiest (but relatively inefficient) way to tackle the problem is the *bisection method.* We want to find the root(s) of $f(x) = 0$. To start we 'bracket the root' by finding (either by guesswork or brute force) two values of x ($x_b < x_t$) such that $f(x)$ is positive for one of the values and negative for the other. As $f(x)$ must pass through 0 to go from positive to negative there must be at least one root in the interval between these two points. This constraint on x_b and x_t is equivalent to $f(x_b)f(x_t) < 0$.

We proceed by selecting the centre of the interval $x_m = \frac{(x_b+x_t)}{2}$ and test to see if

$$f(x_t)f(x_m) < 0$$

If this is true then x_m is a suitable lower bound (which is higher than x_b and so is more constraining) so we set $x_b = x_m$, else x_m is a suitable upper bound and so we set $x_t = x_m$. We continue repeating this until the interval containing the root reaches a certain size (i.e. if we want an accuracy of 3 decimal places then $x_t - x_b < 10^{-3}$).

Newton-Raphson Method

Another common method of root finding known as the Newton-Raphson method is more efficient under a wide range of circumstances. Starting with an initial guess for the root the next guess is given by,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Figure 1 shows what the algorithm does for 2 iterations. With an initial guess x_0 the tangent of the function at that point is found (red line) and followed down to the x-axis to give the next guess x_1. This is done again (blue line) to find the following guess x_2 and so on, to get better and better guesses. To see this is true you can just write down the function $y(x)$ of a straight line that goes through the point $(x_n, f(x_n))$ and is tangent to $f(x)$ at that point. Then x_{n+1} is simply the value of x when $f(x) = 0$.

With a reasonable first guess and a suitable function the series of guesses will converge to the root. What we mean by '*reasonable*' and '*suitable*' for this method will be expanded on in question 2.

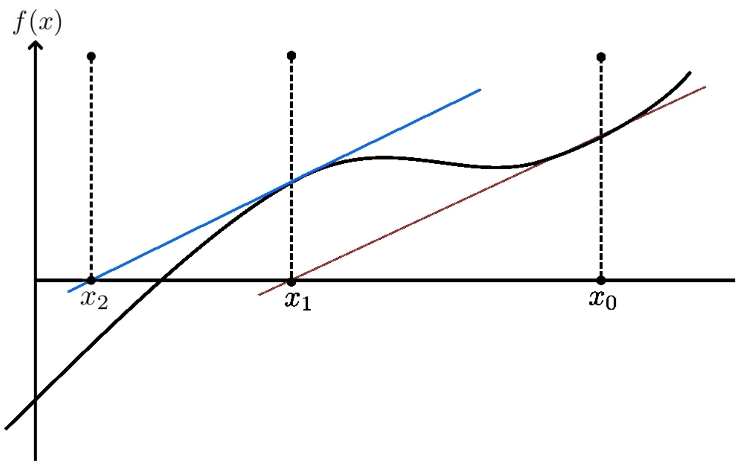
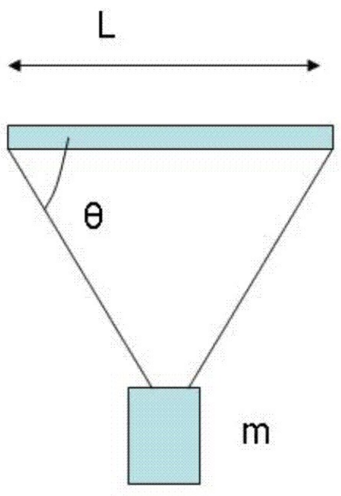


Figure 1: given an initial guess, x_0, the above diagram shows how the next approximations of the root are found by the Newton-Raphson method.

6. $\star$ Our problem is a classical mechanics one, which is easy to set up but cannot be solved exactly analytically. Consider a mass m suspended from a bench of length L by two equivalent springs, of unstretched length $L/2$ and of spring constant k



Show, by balancing forces, that the equation you need to solve is

$$kL(\tan\theta - \sin\theta) = mg$$

Assuming $m = 5.5$kg and $L = 0.6m$, $k = 850$N/m find θ
using the bisection method outlined above. Tabulated output data should include: value of theta found, the size of the uncertainty, number of times you needed to apply the bisection method.

7. $\star$ In this question we will explore the performance of various methods to understand why you would use different algorithms depending on your problem.

i) Consider the bisection method, we can analytically estimate the uncertainty in the value of the root after 1 iteration if we know the error of the result from the previous iteration. Derive the relation between the error at a step $n + 1$, ϵ_{n+1}, in terms of the error at step n, ϵ_n.

ii) By Taylor expanding the function $f(x)$ around x_n and considering $f(x_{root})$ show the relation between ϵ_n and ϵ_{n+1} for the newton-raphson method. Discuss the conditions needed for the newton raphson method to converge.

iii) Find the solution to

$$x^3 + 7x^2 - 6x + 15 = 0$$

both by using the bisection method the Newton-Raphson method. For the Newton-Raphson method try a few different initial guesses and choose 2 starting points that show different initial behaviours of the algorithm. Plot a graph that compares the evolution of the errors for these 3 runs and discuss the result, keeping in mind the analysis of the first parts of this question. Make sure you have found the solution to a high enough accuracy that this plot is meaningful.