1 Numerical Integration

Answers to $\star$ question must be submitted to WebCT, as an extensively commented program and any additional text by Monday 4/11/13 4:30pm (usual late penalties will apply).

In solving physics problems, it is frequently the case that once an integration is involved it is no longer possible to carry on solving analytically and we need to turn to numerical integration. There are many functions available in C++ libraries such as the GNU Scientific Library (GSL) or from Numerical Recipes to do this - but before relying on a "black box" integration we will use some simple methods which can be readily programmed by you!

We will start with using the Trapezium rule which approximates the area under a curve between two given points by a trapezium -to remind yourself of this see, for example:
http://mathworld.wolfram.com/TrapezoidalRule.html
http://numericalmethods.eng.usf.edu/topics/trapezoidal_rule.html

1. When testing a computational method it is a good idea to start with a problem for which the answer is known from another source.

   Consider the integral
   $$\int_0^1 \frac{1}{(1+x)^2}\mathrm{d}x$$

   (a) Work out the answer analytically.

   (b) Write a program to evaluate this integral which uses the Trapezium rule. It should work over a range of n where n is the number of intervals you have used to divide your integral up.

      i. Output the width (h) of the interval and the result of the integration, and the difference from the exact result.

      ii. Determine how the error (approximately) varies with the value of h.

2. ⋆

   (a) Evaluate analytically the integral

   $$\int_0^2 \mathrm{e}^{-x}\sin(x)\,\mathrm{dx}$$

   (b) Generate a set of data for use with gnuplot or Matlab which will enable you to plot the function $\mathrm{e}^{-x}\sin(x)$ in the range $[0,\pi]$ (you should include a labeled figure in your report).

   (c) Write a program that uses the Trapezium rule to evaluate the above integral. Make your program write a table to a text file showing the value of the integral against number of sub intervals used.

   (d) Produce a plot from this showing the log_{10} of the number of intervals against the log_{10} of relative error against the analytic value of the integral. How accurate can the Trapezium method calculate the answer to? Comment on what you see. Make sure your program increases the subinterval ranges logarithmically (you will need to calculate up to at least 10^8 intervals, this will take alot of computing time).

   (e) An analytic expression for the Trapezium methods error exists (find it somewhere and cite source or derive it if you like), how does the error of your trapezium method compare to this?

   (f) Now using an appropriate online resource, write a program (either a separate program or an extension of the previous one with option on what method to run) that uses Simpson's rule to evaluate the integral. The program should have a user input to define how many significant figures the result should be accurate to. You must make it clear how you have checked the accuracy in the code and report.

   (g) Repeat the error comparison as before producing a log_{10} plot of number of intervals against log_{10} of the relative error. Compare to an analytic expression for the error of the Simpsons rule (find it somewhere and cite or derive it if you like) and the Trapezium method.

3. ♠ (the ♠ means that fourth years must submit this question in addition). This is a physically motivated problem which involves a change of variables to evaluate the integral numerically . Consider a simple pendulum of length l. It is oscillating with a maximum angle of from the vertical of θ_m. You know that if θ_m is small the pendulum undergoes simple harmonic motion with period $T_0 = 2\pi\sqrt{l/g}$, where g is the acceleration due to gravity. Here we investigate how the period changes when the amplitude is no longer small.

(a) From conservation of the energy, E, where

$$E = \frac{1}{2}m(l\dot{\theta})^2 + mgl(1 - \cos\theta),$$

show that the period of oscillation $T(\theta_m)$ can be expressed as

$$\frac{T(\theta_m)}{T_0} = \frac{\sqrt{2}}{\pi}\int_0^{\theta_m} \frac{\mathrm{d}\theta}{\sqrt{\cos\theta - \cos\theta_m}}$$

Your working should be coherent in your WebCT submission.

(b) Why is this form not a good one to choose for the computational integration? (You may find it helpful to plot the integrand.)

(c) Show in your notebook that it is possible to re-express the integral as

$$\frac{T(\theta_m)}{T_0} = \frac{2}{\pi}\int_0^{\pi/2} \frac{\mathrm{d}\psi}{\sqrt{1 - \sin^2(\theta_m/2)\sin^2(\psi)}},$$

Why is this form preferable?

(d) This last integral is actually a function known in the literature as an elliptical integral. However, we will not use this information here. Expand the integrand in powers of $\sin^2(\theta_m/2)$ and integrate it term by term, to obtain the coefficients a and b in the expansion:

$$T(\theta_m)/T_0 = 1 + a\sin^2(\theta_m/2) + b\sin^4(\theta_m/2) + \cdots$$

(e) Evaluate $T(\theta_m)/T_0$ numerically using Simpson's Rule for $\theta_m = 0.1; 0.2; \pi/4; \pi/2$ and $3\pi/4$. Compare your answers in *tabulated form* with the results of the series expansion in the last section. Which works better?

2 Code Libraries - GSL and numerical integration

To complete the following questions you will need to update your git repository you first created backin worksheet 1, the `compmod2013` folder. Change to this directory and use the `git pull` command to update your folder to the latest version.

If you still have not got the correct folder then you can get the latest copy by first going to your home directory, then use the `git clone git://lnx0.sr.bham.ac.uk/compmod2013` command.

2.1 Why a scientific code library?

Beforehand we used Trapezium and Simpsons rules for numerical integration. These algorithms work well for smooth functions which dont vary very much , but their constant step size makes them a poor choice in general as functions may vary rapidly in one region and be almost asymptotic in others.

There are many efficient and robust algorithms for numerical integration but it would be time consuming and error-prone to continually re-implement these across multiple scientific projects. Instead scientists have helped create libraries with standard implementations of tried and tested algorithms.

By using these libraries we can reduce the amount of time it takes to create our programs and be more confident that they are doing what we think they are doing (although you should never assume any piece of code is without bugs!).

Next we will learn how to use the many routines available in the GNU Scientific Library (GSL), specifically those for numerical inegration. In future worksheets we will make further use of GSL to solve other types of problems.

2.2 What is the GNU Scientific Library (GSL)?

GSL is a popular collection (library) of routines for scientific computing used widely in the scientific community (e.g. CERN and LIGO). Because it is free software you can download a copy for use in your own projects free of charge as long as you retain the original licenses distributed with the code. You can download a copy of the software from:

`http://www.gnu.org/software/gsl/`

A comprehensive manual is available free online here:

`http://www.gnu.org/software/gsl/manual/html_node/`

WARNING: The online manual refers to a version of GSL which is later than the one installed on phymat. A few of the code examples found online will therefore not work (mainly those in the differential equations section). A version of the manual appropriate to the GSL installation on phymat can be accessed with the `info` command e.g.

```
$> info gsl
```

To find out how to use `info` consult its `man` page:

```
$> man info
```

In your code comments you should indicate which version of GSL you used. You can find out which version of GSL you are using with

```
gsl-config --version
```

You will need to consult the GSL manual regularly to successfully complete the worksheets so you should familiarise yourself with how it is structured.

2.3 Compiling a C++/GSL program

You should find an example C++ program which uses GSL in:

```
[Add path to your compmod git folder]/worksheet2/w2_example.cpp
```

If you open this file in an editor you should see that there is nothing special about it, it looks like and is valid C++ source code.

Although GSL is implemented in C (not C++) it has been written so that all the GSL functions and structures it exposes through its Application Programming Interface (API) will always work when used in C++ code.

To compile the example program we must invoke g++ with some extra options. These options tell the compiler where to find the files it needs to build and link the program.

For convenience the GSL installation provides a program called `gsl-config` which tells us what these are for GSL. We run this program with the following options:

```
gsl-config --libs
gsl-config --cflags
```

The output of these programs should be added to the g++ command you normally use to compile C++ files e.g.

```
g++ [GSL options] -o [path to output program] [path to input C++ file]
```

To make life easier instead of typing out all those compiler options each time we ideally want to be able to type one command that does it all for us. We can do this by adding a function to the shell that we can call. We define the function in a file called `.bashrc` in your home directory. This file is loaded and run everytime you log in to phymat 2 or 3 and can be used to create custom functions and automate tasks on login. We have written a function for you called `gsl-build` which can be found in the `compmod2013/misc` folder, which you should see if you have updated your git folder correctly. To add this function to your `.bashrc` file call the following command,

```
cat [Add path to your compmod git folder]/misc/gsl-build.bash >> ~/.bashrc
```

Use the `more ~/.bashrc` command to see the contents of your `.bashrc` file, you should see the function in there. Remember that the `.bashrc` file is only run when you log in, so to be able to use `gsl-build` without having to log out and back in again use the command `source ~/.bashrc`, this runs the file as a source file and compiles any commands in it.

Sometimes you may get an error that pops up saying that access to `.bashrc` is denied. If this does happen we need to change the permissions on the file. You do this using the command `chmod 775 ~/.bashrc`.

You should use `gsl-build` it in the same way you use g++ e.g.

```
gsl-build -o output_binary input_source_file.cpp
```

4. In your lab book you should quote the g++ compiler options for GSL in full and explain the meaning of the options starting with
    (a) -L
    (b) -l
    (c) -I

2.4 Example: Special Functions and Physical Constants

The example program above we will print the values of some special functions and physical constants using GSL routines.

We can compile the example program using:

```
gsl-build -o w2_example w2_example.cpp
```

This will produce an executable called `w2_example`. When you run it it should print something like this to the terminal:

```
 *Airy Function, Ai(x=0.8,mode=0)*
GSL status  = success
Ai(0.8) = 0.169846317444364847
      +/- 5.17737398721610777e-17

 *Gravitational Constant,G*
G = 6.67299999999999986e-11
```

5. By copying the example program and consulting the GSL manual write a program which prints out the value of
    (a) Regular cylindrical Bessel function of zeroth order for $x = 0.8$ (Look at section 7 Special Functions).

(b) The mass of a muon (in SI/MKS units) (HINT: Look at the section of the manual called 40 Physical Constants).

You should note that most sections of the online manual have a sub-section with code examples e.g. 7.3.3 Special Functions Examples .

2.5 Numerical Integration with GSL

We now return to the topic which originally motivated us to consider using a scientific library.

6. * Consider the integral from before:

$$\int_0^2 e^{-x}\sin(x)dx$$

(a) Using the GSL manual (17 Numerical Integration) write a program which uses a GSL numerical integration routine to evaluate the above integral. Record all relevant output from this routine, including the approximation of the integral and an estimate of the error in this approximation. It should be obvious from the commenting in your source code that you understand how the GSL routine works.

(b) How many times is the function $f(x) = e^{-x}\sin(x)$ called in the trapezium and Simpson's method compared to the GSL routine?

7. * Now consider the integral

$$\int_0^{2\pi} x\sin(30x)\cos(x)dx$$

(a) Use Maple/Mathematica/Matlab/Mathcad to determine the value of this integral

(b) Plot the function in the range 0 to 2π, what is difficult about this integral for the computer?

(c) Use an *appropriate* GSL routine to evaluate this integral. Quote the result and the error estimate.

2.6 More Fun GSL Problems (Non-Assessed)

You are encouraged to try at least one of the following problems depending on which one seems most interesting to you out of solving linear algebra using matrices, computing discrete fourier transforms for diffraction patterns or calculating the energy eigenvalues of a quantum harmonic oscillator.

2.6.1 Linear Algebra

If you have looked through the GSL documentation you will see that there is a lot more functionality on offer than just numerical integration. These exercises are intended to give you a better working knowledge of GSL and how to solve some physics problems computationally.

Solving linear algebra problems is a common one in physics and GSL provides various ways of solving them. Typically we formulate our linear equations into the form,

$$y = M.x$$

where y and x are vectors and M a matrix. The matrix M can be decomposed into other matrices that can be more helpful to solve the equations such as LU and QR decomposition. These are complicated and time consuming to program from scratch, luckily GSL has these ready for us to use (`http://www.gnu.org/s/gsl/manual/html_node/Linear-Algebra.html`).

8. Take this system of linear equations,

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 2 \quad (1)\\ 4x_1 + 3x_2 + x_3 &= 3 \quad (2)\\ 2x_1 + 2x_2 + 3x_3 &= 5 \quad (3) \end{aligned}$$

   (a) Out of QR and LU decomposition, which is the more suitable method to use to solve this set of equations?

   (b) Find the values of x_1, x_2 and x_3 that satisfy this set of linear equations.

2.6.2 Fourier Transforms

As you may have gathered from year 2 math lectures Fourier Transforms are very important in physics. They take something represented in the spatial or temporal domain and transforms them into a new basis, the frequency domain.

Fourier Transforms are used extensively for calculating optical effects. Transforming a signal into the frequency domain allows us to work with the individual frequency components a signal is made of as often the time or spatial domain view of it is difficult to work with. For example permittivities for dielectric materials are often dependant on frequency. Knowing what refractive index ($n = \sqrt{\epsilon}$) light composed of multiple frequencies will experience in a material is only possible by knowing its constituent frequencies, hence working with a signal in the frequency domain is often easier.

Fourier Transform however are continuous, computers are discrete. Therefore calculating the Fourier Transform of data stored on a computer requires a Discrete Fourier Transform (DFT). GSL contains functions for computing DFT's however there are many mathematical and computational tricks that allow DFT's to be calculated much faster than a direct calculation.

These methods are known as Fast Fourier Transforms, a complicated topic which we will only scratch the surface of here. The various functions provided by GSL for FFT can be found here, http://www.gnu.org/software/gsl/manual/html_node/Fast-Fourier-Transforms.html.

FFT's work by providing the FFT function with an array of data that you wish to transform. So you first need to write your data into a `double[2*n]` array. It is very important that `n` is some power of 2 as the algorithms for FFT require the data to be 2^n long. The reason for the 2 factor is that the real and complex part of your number is stored in alternating array positions, i.e.

```
double data[2*n];

data[0] = Real part of z_1
data[1] = Imaginary part of z_1
data[2] = Real part of z_2
data[3] = Imaginary part of z_2
...
data[n] = Real part of z_n
data[n+1] = Imaginary part of z_n
```

To access the real and imaginary parts some macros (See here http://www.cprogramming.com/tutorial/cpreprocessor.html) are defined in a small FFT helper file we provide you with. To get it goto your `Compmod2012` folder and type `git pull`, this should update your folder with the latest version of files. In the `misc` folder you should see `gsl_fft_helper.h`. Include this in your code by copying the file to the local directory and using an `#include` statement.

If you look in this file 2 macros called `Re` and `Im` are defined. Once you have declared your data array as described above, you can use the macros like this

```
double data[2*n];

Re(data,0) = 1;
Im(data,0) = 1; // z_0 = 1+i
Re(data,1) = 0;
Im(data,1) = -1; // z_1 = -i
...
```

Read the GSL documentation on computing complex FFT's at http://www.gnu.org/software/gsl/manual/html_node/Fast-Fourier-Transforms.html. A short example is given at the bottom of the page.

9. Here we take you back to year 2 math, where you were (hopefully!) taught that a Fourier Transform can be used to calculate the far-field Fraunhofer diffraction pattern from a

source. If you look back at your notes you should find that the intensity of the diffraction pattern along a distant plane from the source is

$$I(X) \approx 2\pi \left| \tilde{a}\left(\frac{2\pi}{\lambda d}X\right)\right|^2.$$

Working through the whole derivation is not required here, we are just interested in the final result. $\tilde{a}$ is the Fourier Transform of the source function, λ the wavelength of the source, d the distance from source to plane and X is the distance on the plane from the origin.

(a) Write a program that calculates a source function that resembles a double slit, this should be in the complex `double` array form as described above. The user should be able to enter `n` for the size of the FFT, check the input for errors.

(b) Compute the complex FFT of the source data. Plot the intensity, what is wrong?

(c) Use the `fftshift` function in `gsl_fft_helper.h` to shift your data. Plot the data again, what has it done?

(d) Compare your intensity result to other double slit sources is it similar? What other diffraction pattern can you create?

2.6.3 More Matrices and Eigenvalues

Matrices are a very useful tool for numerically computing answers to problems. Previously we saw how matrices can be used to easily solve linear algebra problems, using matrices again we will look again at a more interesting problem now calculating eigenvalues of a matrix.

For this problem you can find a helpful header file in your `compmod2012/misc` folder called `gsl_matrix_helper.h` (run the command `git pull` to update your `compmod2012` folder). Include this in your program by copying it to your local directory and using a `#include` statement at the top of your `cpp` file. You can now call two helper functions `printMatrix` and `matrix_pow`, read the code file for more information on them.

10. The Hamiltonian $\hat{H}$ for a simple harmonic oscillator is

$$\hat{H} = -\frac{\hat{p}^2}{2m} + \frac{\omega^2\hat{x}^2}{2} \tag{4}$$

Where the position and momentum operators $\hat{x}$ and $\hat{p}$ can be described in terms of the Ladder operators $\hat{a}$ and $\hat{a}^\dagger$

$$\hat{x} = \sqrt{\frac{\hbar}{2m\omega}}(\hat{a}^\dagger + \hat{a}) \tag{5}$$

$$\hat{p} = -i\sqrt{\frac{\hbar m\omega}{2}}(\hat{a} - \hat{a}^\dagger) \tag{6}$$

The Ladder operators can be expressed as $n \times n$ matrices like so,

$$\hat{a}^\dagger = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots \\ \sqrt{1} & 0 & 0 & 0 & \dots \\ 0 & \sqrt{2} & 0 & 0 & \dots \\ 0 & 0 & \sqrt{3} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\hat{a} = \begin{pmatrix} 0 & \sqrt{1} & 0 & 0 & \dots \\ 0 & 0 & \sqrt{2} & 0 & \dots \\ 0 & 0 & 0 & \sqrt{3} & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Note that the $\sqrt{N}$ factor carries on in the same diagonal fashion.

The harmonic oscillator eigenvalue problem is one with well known analytic solutions. This exercise is set out to see if you can correctly use matrices to produce the same eigenvalues. You will also want to read the GSL documentation on how to use GSL matrices and vectors at `http://www.gnu.org/software/gsl/manual/html_node/Vectors-and-Matrices.html` as well as the section on eigensystems `http://www.gnu.org/software/gsl/manual/html_node/Eigensystems.html`.

(a) Write a program that calculates the matrices $\hat{x}^2$ and $\hat{p}^2$. (You can assume for simplicity in calculations that $\hbar = m = \omega = 1$)

(b) Calculate the the harmonic oscillator Hamiltonian matrix using $\hat{x}$ and $\hat{p}$

(c) Using an appropriate method from GSL calculate the eigenvalues of $\hat{H}$ and output them to the console in ascending order. You should calculate up to a user inputted number of eigenvalues

(d) Compare these to the analytic values for the harmonic oscillator energy eigenvalues

(e) Another type of oscillator is the anharmonic oscillator, where the pertubation to the harmonic oscillator Hamiltonian is

$$\hat{H}_{an} = \alpha \hat{x}^4 \tag{7}$$

Where α is the constant of anharmonicity.

i. Allow for a user specified α ranging from 0 to 1

ii. Compute the new Hamiltonian and its energy eigenvalues. Comment on the perturbed eigenvalues.