1 Using the GNU Scientific Library (GSL)

Answers to ⋆ question must be submitted to webct, as an extensively commented program and any additional text by Wedesday 26/10/11 5pm(usual late penalties will apply).

1.1 Git Repository

To do some of the problems this week you will need to update your git repository you first creeated back in worksheet 1, the `compmod2011` 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/compmod2011` command.

1.2 Why a scientific code library?

In Worksheet 2 we used the Trapezium and Simpsons rules for numerical integration. For the integrals we considered Simpson's rule required fewer function evaluations than the Trapezium rule to reach the same level of accuracy.

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!).

In this Worksheet 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.

1.3 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.

1.4 Compiling a C++/GSL program

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

`$COMPMOD_REPO/worksheet3/w3_example.cpp`

Where `$COMPMOD_REPO` is the directory of you git folder. Copy this file to a convenient place in your home directory.

If you open this file in an editor you should see that there is nothing special about it, it looks like and is legal 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 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]
```

For your convenience we have created a shell function, `gsl-build` , which includes these options so you do not have to type them every time you want to compile a GSL program. To add this to your shell copy the function definition from the example code to your startup script by issuing the following command:

```
$> cat $COMPMOD_REPO/misc/gsl-build.bash >>~/.bashrc
```

You only need to do this once. You should use it in the same way you use g++ e.g.

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

1. 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

1.5 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 w3_example1 w3_example1.cpp
```

This will produce an executable called `w3_example1`. 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
```

2. 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 $x = 0.8$ (HINT: 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 .

1.6 Numerical Integration with GSL

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

3. * Consider the integral from Worksheet 2:

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

   (a) Using your code from last week, use the trapezium method to solve the above integral. Output the results to a table with the number of sub intervals used. Produce a plot showing the log of the number of intervals against the log of relative error. Use upto 10^8 sub intervals.

   (b) 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.

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

4. * 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 ti 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.

1.7 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.

1.7.1 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).

5. 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.

1.8 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 `Compmod2011` 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.

6. 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?

1.9 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 `compmod2011/misc` folder called `gsl_matrix_helper.h` (run the command `git pull` to update your `compmod2011` 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.

7. 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 $\hat{a}^\dagger$

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

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

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 $\bar{h} = m = \omega = 1$)

(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.