Assignment 8 : FFT and the Double Pendulum

due: April 16, 2020 – 11:59 PM

GOAL: The first part of the problem is program the general Fast Fourier Transform (FFT) with complex arithmetic. The second part looks for oscillatory of the double pendulum Fourrier analysis and filtering out high frequency (wiggly) noise by a high frequency cut-off to get a smooth fit.

The fast Fourier transform is a classic algorithm very important to analyzing signals of all kinds. It was invented by Gauss (who else!) but credited to Cooley and Tukey who rediscovered without knowing Gauss' earlier application: https://en.wikipedia.org/wiki/Fast_Fourier_transform

```
Gauss, Carl Friedrich,"Theoria interpolationis methodo nova tractata",
Werke, Band 3,265{327 (Königliche Gesellschaft der Wissenschaften, Göttingen, 1866)

Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine
calculation of complex Fourier series". Math. Comput. 19:
297{301. doi:10.2307/2003354.
```

This is a classic devide and conquer algorithms doing a specific matrix vector operation in $O(NlogN)$ instead of the straight forward $O(N^2)$. See the class lecture notes for details.

I Coding Exercise #1: Program the Complex FFT

The discrete Fourier Transform on N points and its inverse are

$$f_k \;= \sum_{n=0}^{N-1} c_n e^{2\pi ikn/N} \quad , \quad c_n \;= \frac{1}{N}\sum_{k=0}^{N-1} f_k e^{-2\pi ikn/N} \tag{1}$$

respectively. On GitHUB at `HW8code/FFT` there is already code for the slow FT. This exercise is simply to convert it into a FFT

The deliverables for this exercise are:

- Run the code `MainFFT.ccp` and verify that the slow $O(N^2)$ tranforms and its inverse does give the correct result. You can use small data set for this test.
- Add to `MainFFT.ccp` the recursive FFT routines describe in the class lectures and test them against the slow FT.
- Now put in timers and run comparison speed test of slow FT vs FFT for large range of sizes $N = 2^n$ with $n = 1, 2, 3, \cdots, 10$ and plot result to *prove* empirically demonstrate the scaling of $O(N^2)$ and $O(NlogN)$ respectively.

II Coding Exercise #2: Discrete Fourier Series Fit.

This problem revolves about the double pendulum [1]. We will analyze the path tracked by a double pendulum and replot it after applying a filter that cuts off high frequencies and compare it with low order polynomial fits to the data. Which is more appropriate? Here we have no error bars—the data is treated as perfect. It is generated by the program `dbl_pendulum.cpp` on GitHub. While you should feel free to play with the program, ultimately your analysis in this assignment should be based on the data in `Trace.dat`. You can plot the data in gnuplot. For example, to plot the sine of the displacement angle, try running the commands:

```
plot [0:1023]    "Trace.dat"  using 1:3 with lines
replot  "Trace.dat" using 1:5  with lines
```

to see the first $2^{10} = 1024$ time slices. There are $2^{13} = 8192$ in the file. The problem is to read this file and fit the first $N = 1024$ data points of the sine of the displacement angle (columns 3 and 5) to a Fourier series. Let k index the N data points, $k = 0, 1, ..., N-1 = 1023$, and f_k denote the displacement angle for data point k. The Fourier series is defined by:

$$f_k = \sum_{n=0}^{N-1} c_n e^{2\pi ikn/N} = a_0 + \sum_{n=1}^{N-1} \left[a_n \cos(2\pi kn/N) + b_n \sin(2\pi kn/N)\right]. \tag{2}$$

For the right hand side, I have used $c_n = a_n - ib_n$. This formula only works so well for the special case [2] that f_k is purely real! The c_n's can be defined by:

$$c_n = \frac{1}{N}\sum_{k=0}^{N-1} f_k e^{-2\pi ikn/N} = \frac{1}{N}\sum_{k=0}^{N-1} f_k[\cos(2\pi kn/N) - i\sin(2\pi kn/N)] \tag{3}$$

where the f_k are, again, the data for the sine of the displacement angle in `Trace.dat`. How do you extract a_n and b_n from c_n? We told you above! You should take advantage of the C++ complex number class which you can include by using `#include <complex>`. The complex class includes functions to extract the real and imaginary part of a complex number, too.

[1] By the way the double pendulum this is a simple example of a chaotic system. (see https://youtu.be/PrPYeu3GRLg.) To see a beautiful video why this relevant to weather see https://youtu.be/aAJkLh76QnM

[2] To see this take the real part of RHS of Eq. 2: $\frac{1}{2}(c_n e^{2\pi ikn/N} + c_n^* e^{2\pi ikn/N}) = \frac{1}{2}(c_n + c_n^*)\cos(2\pi kn/N) + i\frac{1}{2}(c_n - c_n^*)\sin(2\pi kn/N)$ which implies $c_n = a_n - ib_n$. For real series we have a relation $c_n = c_{-n}^*$ so there are actually only $2N+1$ parameters on both sides of Eq. 2.

NOTATION NOTATION:

- The notation above using n and k probably suggests a spatial transform of the data c_n from a positions x_n on grid , $0 \leq x_n = n\ h < Nh$, to amplitudes,f_k, for wave numbers, $k = 1 - N/2, \cdots N/2$, where for convenience in code both are both n and k are fixed up to be `integers`.

  Actually real wave number, $\kappa = 2\pi/\text{wavelength}$, is $\kappa = k/(N\ h) \in [-1/2, 1/2]$.

- In this problem we have a time series so $t_n = \Delta t\ n$ where Δt is the time intervals between measurements and k now refers to frequencies.

  Again technically the proper angular frequency is $\omega = 2\pi$ frequency $= 2\pi/T$ where T is the time (period) of rotation through by 2π radians. So $\omega = k/(N\Delta t)$.

Okay, well, we've given you some equations. What do we want you to do to them?

In this exercise we're going to implement a simple *low pass filter*[3] In its simplest form, a low pass filter takes a signal and cuts out any high frequency contributions (above some threshold frequency). This is important in audio processing, for example: as a simple example, the human ear can't hear any frequency over 20kHz. If you're compressing audio (say an mp3), what's the point in saving any data corresponding to frequencies above 20kHz? Thus, use a low pass filter and get rid of it!

To implement a low-pass filter, you should follow these steps:

1. Filter on the power (absolute value square of amplitude) of complex modes: $|f_k|^2 = f_k^* f_k$ for a give frequency. Use the function you defined for the first part of the exercise to convert f_k to frequency, a_n and b_n.

2. Zero out an appropriate high frequencies. This may give something away: If you wanted to remove the top half of the frequency space, for example **non-zero** on the range $N/4 < k < 3N/4 - 1$ for low frequencies[4]

3. Transform back using equation 2, plugging in the modified a_n and b_n.

You should compare how the data looks as you apply a more and more aggressive low pass filter. Compare, for example:

- The original data a_n, b_n (or real and imaginary parts of $c_n^* = a_n + ib_n$) the first 1024 data points in column 3 or 5 of `Trace.dat`.

- The data after removing the top half of the frequency space.

[3]See https://en.wikipedia.org/wiki/Low-pass_filter

[4]**NOTE:** We can make this more intuitive by mapping to an integer frequencies index, $\omega = (k + N/2)\%N - N/2$, symmetric around $k = 0$ with low positive and negative values: $\omega \in [-N/2, N/2]$. A symmetric low pass filter is keeping $|\omega| < \epsilon N$. Try $\epsilon = 4/N$ for example!

- The data after removing the top 75% of the frequency space.
- The data after removing the top 87.5% of the frequency space.
- ...And so on.

Make some plots and submit a brief qualitative write-up on how the data looks after applying a more and more aggressive low pass filter. A leading question: at what point does the low pass filter start looking bad?

The deliverables for this exercise are:

- At least one source file, `lowpass.cpp`, which prints to file the data in column 3, and to a separate file the data in column 5, before and after applying the low-pass filters described above. In each file, the first column should be the time (which is column 2 of `Trace.dat`), then the subsequent columns should be the values of f_k for increasingly aggressive low pass filter. Feel free to split the code into multiple files as you see fit—bear in mind that we'll be revisiting Fourier transforms in upcoming assignments, so the more effort you put into writing clean code now, the less pain you'll go through later! Don't forget a makefile, too.
- Plots the filtered curves for the motion of data in column 3 and 5 with a filter vs the original with now filter.

III Extra Credit & Extra Fun if you have spare time!

- Apply the filter **Coding Exercise #2** to the temperature data of HW7, which will do very nicely when you cut-off the vast majority of the high frequency noise.
- First **Coding Exercise #2** above by selecting a VERY few dominate modes in your low pass filter and minimizing the χ^2. There probably about 3 dominate frequencies. Since the data is very good you may set $\sigma' s = 1$ through out.
- Also you might show the filter on dominate modes in the full FFT is NOT identical to the χ^2 fit of these same modes. Curious?

Can do any filter problem you like and entertain the class with plot in the **zooming meetings**.