I Assignment 6: Curve Fitting and Code Organization

due: March 27, 2017 – 11:59 PM

GOAL: The First part is an exercise to integrate pieces from Problem Set #2 and Problem Set #3. The goal is to link together several source files, xxx.c with their xxx.h files and compile using a Makefile. The Second part is to understand how to fit noisy data with a low order polynomial to tease out the general trend. The Third part looks for oscillatory behavior using Fourrier analysis and again filtering out high frequency (wiggle) noise by a high frequency cut-off. This ability to organize a more complex software will be useful in your project, especially when different members of a team are writing separate components.

I.1 Coding Exercise #1: Polynomial Fits and Error Estimate

Recall in class we used gnuplot to show that temperature is rising in the data for the interval [1900 : 2015] with the commands:

```
plot "Complete_TAVG_summary.txt" using 1:2:3 with errorbars
f(x) = a + b*(x-1750)
fit [1900:2015] f(x) "Complete_TAVG_summary.txt" using 1:2:3 via a,b
replot f(x)
```

This exercise is to write your own program to do a polynomial fit to a discrete data file with N_0 values y_i, x_i, σ_i for $i = 0, 1, 2, \cdots N_0 - 1$. With errors (e.g. σ_i) you should give the chi square of our fit to see if you are able to reliably extract information of the fit.

You should use this file `Complete_TAVG_summary.txt` to test of the code. The problem here is to fit the N_0 data values in the time intervale [1900 : 2015] of the file `Complete_TAVG_summary.txt`. The program should allow for any reasonable polynomial fit to $y = f(x) = c_0 + c_1 x + c_2 x^2 + \cdots c_N x^N$ for $N = 0, 1, 2, .., N_0 - 1$

1. First fit it taking the number of paramters $N+1$ to equal to the number of data points ($N+1 = N_0$), the chisqr is zero and the exact fit is given by

$$f(x) = \sum_{j=0}^{N_0-1} y_j \prod_{i \neq j} \frac{x - x_i}{x_j - x_i} \tag{1}$$

(You should avoid doing this product with $O(N^2)$ operations!)

2. Next take the number of parameters to be much less than N_0 ($N \ll N_0$) and get a least square fit. Here don't let N be too large. Only try $N = 0$ (constant), $N = 1$ (linear), $N = 2$ (quadratic)

$N = 3$ (cubic) $N = 4$ (quartic). The chi square is defined to be

$$\chi^2 = \sum_{i=0}^{N_0} \frac{(y_i - f(x_i))^2}{\sigma_i^2} \tag{2}$$

The reduced chi square is

$$\chi^2_{reduced} = \chi^2/(N_0 - N - 1) \tag{3}$$

It tells you how well on average can shoot through the error bars for each point x_i, remembering that given $N + 1$ parameters chi-square would be zero for that many points.

We can find the values of $c's$ that minimize the chi-square by solving the N+1 linear equations [1],

$$\sum_{m=0}^{N-1} A_{nm} c_m = b_n \tag{4}$$

where have defined,

$$A_{nm} = \sum_{i=0}^{N_0-1} \frac{x_i^{n+m}}{\sigma_i^2} \quad , \quad b_n = \sum_{i=0}^{N_0-1} \frac{x_i^n y_i}{\sigma_i^2} \tag{5}$$

for all $n, m = 0, 1, 2, ..., N$. Just do this again with Gaussian elimination using the code from Problem Set #3.

The deliverables for this exercise are:

- A fit to the entire data set in the range [1900 : 2015] using Eq. 1. If this it too difficult you can use a smaller range: 32 to 64 points is ok. Submit a plot of the fit against the data.
- Fit using to minimize chisqr for N = 0,1,2,3. Plot each fit against each other and report the value of χ^2 and $\chi^2_{reduced}$.
- You will probably want to use a Makefile as describe in part Three below.

(If you are really bored you might find it fun to go the web to find more data but this is not part of the assignment to pass in. If there are no errors just set all $\sigma_i = 1$ in the chi-square fit.)

[1] By the way the proof of Eq.4 take one line of algebra. Namely it is equivalent to setting all derivatives with respect c_n,

$$\frac{1}{2}\frac{\partial \chi^2}{\partial c_n} = -\sum_{i=0}^{N_0} \frac{x_i^n(y_i - \sum_m c_m x_i^m)}{\sigma_i^2} = \sum_m \sum_{i=0}^{N_0} \frac{x_i^n x_i^m c_m}{\sigma_i^2} - \sum_{i=0}^{N_0} \frac{x_i^n y_i}{\sigma_i^2} = 0 \; .$$

to zero! See Lecture on `Curve Fitting` for more details. If you start with an over complete constraints trying to fit, $Mc \simeq y$, where there is a small vector of constants $c = \{c_0, \cdots, c_{N-1}\}$ and a large number of measurements $y = \{y0, \cdots, y_{N-1}\}$. M is $N \times N_0$ non-square matrix. The the least square problem is give by the linear algebra problem $Ac = A^T y$ for the $N \times N$ square matricm $A = M^T M$.

I.2 Coding Exercise #2: Discrete Fourier Series Fit.

The coding problem is to consider the double pendulum. Analyze this with a discrete Fourier series and replot it by a filter that cuts off high frequencies. Compare with low order polynomial fits. 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. This run the double pendulum. You can try various initial condition but one you must use when you turn it in is the file `Trace.dat` You can plot it 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 slides. There are $2^{13} = 8192$ in the file. The problem is to read this file in fit the first $N = 1024$ data points 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:

$$f(k) = \sum_{n=0}^{N-1} c_n e^{2\pi ikn/N} = a_0 + \sum_{n=1}^{N-1} \left[a_n \cos(2\pi kn/N) + b_n \sin(2\pi kn/N)\right]. \tag{6}$$

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

$$c_n = \frac{1}{N}\sum_{k=0}^{N-1} f(k)e^{-2\pi ikn/N} = \frac{1}{N}\sum_{k=0}^{N-1} f(k)[\cos(2\pi kn/N) - i\sin(2\pi kn/N)] \tag{7}$$

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.

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

In this exercise we we're going to implement a simple *low pass filter*. In its simplest form, a low pass filter takes a signal and cuts out any high frequency contribution (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? That's right, there isn't one. Thus, use a low pass filter and get rid of it!

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

1. Use the function you defined for the first part of the exercise to convert $f(k)$ to frequency, a_n and b_n.

[2]To see this take the real part of RHS of Eq. 6: $\frac{1}{2}(c_n e^{2\pi ikn/N} + c*_n\, e^{2\pi ikn/N}) = \frac{1}{2}(c_n + c_n^*)\cos(2\pi kn/N) + i\frac{1}{2}(c_n - c_n^*)\sin(2\pi kn/N)$ inplies $c_n = a_n = ib_n$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. 6!

2. Zero out an appropriate set of a_n and b_n's corresponding to high frequencies. This may give something away: If you wanted to remove the top half of the frequency space, you'd set a_n and b_n to zero on the range $N/4 < n < 3N/4 - 1$.
3. Transform back using equation 6, plugging in the modified a_n and b_n. You should implement the *inverse Fourier transform* in a separate function as well.

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

- The original data $f(k)$, where $f(k)$ is the first 1024 data points in column 3 or 5 of `Trace.dat`.
- The data after removing the top half of the frequency space.
- 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. 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 and a write-up for the first part of the assignment where you describe where the redundancy is in the discrete Fourier transform—the plots should support your write-up! Feel free to make `lowpass.cpp` also print out a file containing the a_n's and b_n's.
- Plots and a write-up for the second part of the assignment where you describe the effect of a low-pass filter as you remove more and more frequencies.

I.3 Coding Exercise #3:

The first part is an exercise to integrate pieces from Problem Set #2 and Problem Set #3 to make a nice package, BU `Metaphysical Integrator`. We can put it on github in our first effort to push

Mathematica out of business! The application should take a function $f(x)$ and integrate between $[a, b]$ by tranform it to [-1,1], and do the intergral over a fixed set of intervals N for trapezoidal, N for simpson and N order for Gaussian. We will use Makefiles to organize the `Metaphysical Integrator` package and the program request the user to select one of a ten function to integrate on an interval $[a, b]$and to select on of 3 methods of refinement to N, and perform the integral. You should write documentation on how to use the tool and how it is implemented in software. This is a nice exercise in **software engineering** and will give you practice in preparation of our expectations for the final project.

The other parts are simple exercise to fit data to polynomials and/or Fourier series.

Ok this is the truth:

```
Good coding style is very important... even though no one agrees on what it is!
```

One useful trick we're going to emphasize in this problem is organizing code into separate source files. Let's consider the routines we wrote during the last problem set to find the zeroes of Legendre polynomials. It would be useful to put them all in the same source file—say `legendre.cpp`—and then make them accessible in any program you would wish to write, for example some `main.cpp`. The function definitions (`getLegendreCoeff`, `getLegendreZero`) would go in `legendre.cpp`, and their declarations would go in a header file `legendre.h`. You could then call the Legendre functions from any other main file by adding an include statement:

```
#include "legendre.h"
```

As a more general presentation to this idea, give a look at: https://www.cs.cf.ac.uk/Dave/C/node35.html It is also a good practice to put in a header file guard: https://en.wikipedia.org/wiki/Include_guard.

The next very useful tool is a `Makefile`. In simple form, a makefile contains instructions on how to compile and link many files for you automatically: the precursor to what a modern IDE will do as a basic functionality. A simple (and silly) example is with guards on header files is in `HelloMake.tar` on `github`.

In this problem set you will combine the code you have written in Problem Set #2 and #3 to write a general piece of code to integrate an arbitrary function on an arbitrary interval using a choice of order of Gaussian quadrature. In reality, this shouldn't be a big deviation from Coding exercise #2 on the last problem set. In fact, let's spell out what you have to change:

In this exercise the minimun is to use pairs of .cpp and .h files

```
legendre.cpp            legendre.h
integrate.cpp           integrate.h
getFunction.cpp         getFunction.h
main.cpp
```

The `legendre.*` files should contain routines related to Legendre polynomials, such as generating them and finding the zeroes of them.

The `integrate.*` files should contain routines related to integration via the trapezoidal rule and Gaussian quadrature. You can imagine that the Gaussian quadrature routines should call routines defined in `legendre.cpp`.

The `getFunction.*` files have been supplied for you, in fact. We've defined five functions for you, and a function `getFunction` that returns *function pointers* to you based on an `enum` type that you pass to it. There's a sample program `main.cpp` that shows you how to use function pointers—learn well from it!

In this exercise you are to combine components for Problem Set #2 and Problem Set #3 and introduce a new main file that takes a function from `getFunction.cpp`, then integrates it over a user-supplied interval $[a, b]$ using either trapezoidal rule or Gaussian quadrature (of user-supplied order n), via one of two functions defined in `integrate.cpp`, by taking a function pointer as input. The Gaussian quadrature routine should depend on routines defined in `legendre.cpp`.

You can feel free to split this up more: for example, it may be nice to have a separate pair of `*.cpp/*.h` for gaussian elimination, and still another for finding zeroes of polynomials. That's up to you! What's most important is you *at least* split up the code into the above seven files, and you use a makefile to manage compilation.

The code should compile by running, from the command line, `make` in the source directory. A sample test-run of the code could be:

```
> ./main
Welcome to the  Metaphysical Integrator:
What function would you like to integrate?

1: x^2
2: x log|x|
3: exp[ - x^2]
4: sin(x)/x
5: 1/(1+x^2)

Which do you choose? (Enter 1-5)
> 1
What is the lower limit:
> 2.0
What is the upper limit:
> -1.0
The upper limit is less than the lower limit. What is the upper limit:
> 5.0
Integrate with (1) Trapeziodal, (2) Gaussian (Enter 1-2):
> 1
How many points do you want to use: (Enter N)
> -5
```

```
A negative number of points does not make sense. How many points do you want to use?
> 4
The result of int^a_b f(x) is 2.150000000000E+01.
Do you want a new function: Y/N
> N
Thanks for using the Metaphysical Integrator!
Contributions welcome send cash  to Suisse bank
```

In this routine, using the trapezoidal rule corresponds to using N points in equation **??**. Using Gaussian quadrature corresponds to an Nth order integration on the supplied interval, remembering the rule given in equation **??** for transforming an integral on $[a, b]$ to an integral on $[-1, 1]$.

You only need to do what's listed above. Some food for thought on other ways to extend this program: You could put together a class of functions like `f(x)  = a + b*x + c*x*x` and select coefficients. Of if you a lot of functions as for `f1(x)` and `f2(x)` and ask if you with to integrate and a follow a tree with option like `f(x) =  f1(x) *f2(x)` or `f(x) =  f1(x)/f2(x)` or `f(x) =  f1(f2(x))` etc. This will give more than 10 function all at once. Of course the profession version which require mega`$'s` to get would have a slick interface and a parser!

The deliverables for this exercise are:

- Your *own* implementations of at least the above seven code files, with perhaps more if you so desire.
- A functioning makefile which compiles your integration code.

Feel free to go above and beyond with this program—as long as I can integrate the above five functions, I'm going to be happy!

II Submitting Your Assignment

This assignment is due at 11:59 pm on Monday, March 27. Please e-mail a **tarball** containing the assignment to the class e-mail, bualghpc@gmail.com. Include your name in the tarball filename.

If you're not familiar with tar, here's a sample instruction that perhaps I would use:

```
tar -cvf evan_weinberg_asgn6.tar [directory with files]
```

You may want to include other files. **Do NOT include a compiled executable!** That's a dangerous, unsafe practice to get into.