Assignment 8: Curve Fitting and Error Analysis

due: April 23, 2019 – 11:59 PM

GOAL: The first part of the exercise includes understanding how to fit noisy data with a low order polynomial to tease out the general trend. The second part looks for oscillatory behavior using Fourrier analysis and again filtering out high frequency (wiggly) noise by a high frequency cut-off.

I Introduction

Statistical measure are used to determine if a model (or parameterization) is good and justifiable fit to the data. All good science requires this particularly when the data is noisy. Machine learning is dramatic example of this! The common test is fit and test the χ^2.

The `chis square` or χ^2 is the sum over the square of the residual for error , $r_i = y_i - F(x_i, \theta_j)$, of the fit relative the measure measurements y_i when the parameterize model function $F(x, \theta_n)$ with parameters $\theta_0, \theta_1, \cdots$ weighted by the expected mean error σ_i for each measure at x_i. This is the one dimensional example. In machine learning the there is general huge vector of measured or marked training data $\vec{x}_i$ and target vector of outcomes: ($\vec{Y} = \vec{F}(\vec{x}, \theta_n)$). (e.g. Images in and number out). Here we have one function outcome. This fit is found by minimizing the chi square for this fit:

$$\chi^2 = \sum_{i=1}^{N_0} \frac{r_i^2}{\sigma_i^2} = \sum_{i=1}^{N_0} \frac{(y_i - F(x_i, \theta_n))^2}{\sigma_i^2} \tag{1}$$

The number of fitting parameters θ_n with $n = 0, 1, 2, \cdots, N$ should general be much less that the input data $N + 1 < N_0$ if the model is well chosen. The question is does the fit justify the theory or model for the data. Because non-linar fits and the estimates of standard deviation σ_i is very tricky, here we do a very simple case.

We consider only the case where the model is parameterized by a linear sum of parameters ($\theta_n = c_n$) for known functions.

$$F(x, c_n) = \sum_{n=0}^{N} c_n f_n(x) \tag{2}$$

Also we assume standard deviations, $\sigma_i's$, known and Gaussian distributed. Now gradient descent is linear algebra. Things are hardly ever this simple but this is the common starting point for modeling and error analysis! In this case the `goodness of fit` is measured by the reduced chi square per degree of freedom

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

Note that for $N = N_0 + 1$, one should always get a perfect fit so $\chi = 0$ independent of the assumed errors telling us nothing useful. As a rule of thumb $\chi^2_{reduced} \gg 1$ is a bad fit. Either the model is wrong or the variance is underestimated. If $\chi^2_{reduced} = O(1)$, the model is a good fit assuming the variance estimates are correct. If $\chi^2_{reduced} < 1$, the model is "over-fitting" the data or the error variance has been overestimated.

II 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 calculate the χ^2 of our fit to see if you are able to reliably extract information from the fit.

You should use the file `Complete_TAVG_summary.txt` to test of the code. The problem here is to fit the N_0 data values in the time interval [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 the data with an exact fit ($N + 1 = N_0$, $\chi^2 = 0$) given by:

$$f(x) = \sum_{j=0}^{N_0-1} y_j \prod_{i \neq j} \frac{x - x_i}{x_j - x_i} \tag{4}$$

   (You should avoid doing this product with $O(N^2)$ operations! There are references online on how to do this.)

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), and $N = 3$ (cubic). The chi square is now defined to be

$$\chi^2 = \sum_{i=0}^{N_0} \frac{(y_i - f(x_i))^2}{\sigma_i^2} \quad , \quad \chi^2_{reduced} = \chi^2/(N_0 - N - 1) \tag{5}$$

Visually a good $\chi^2_{reduced}$ means the fit generally goes through the error bars.

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

$$\sum_{m=0}^{N} A_{nm} c_m = b_n \tag{6}$$

defined by

$$A_{nm} = \sum_{i=0}^{N_0-1} \frac{x_i^{n+m}}{\sigma_i^2} \quad , \quad b_n = \sum_{i=0}^{N_0-1} \frac{x_i^n y_i}{\sigma_i^2} \tag{7}$$

for all $n, m = 0, 1, 2, ..., N$. You can solve this linear system with Gaussian elimination using the code from Problem Set #4.

The deliverables for this exercise are:

- A fit to the entire data set in the range [1900 : 2015] using Eq. 4.This may be expensive; you can use a smaller range: 32 to 64 points is ok. Submit a plot of the fit against the data.
- A fit from minimizing a χ^2 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 described in part 3 of this assignment.

II.1 Coding Exercise #2: Discrete Fourier Series Fit.

This problem revolves about the double pendulum [2]. 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
```

[1]By the way the proof of Eq.6 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 $c = \{c_0, \cdots, c_N\}$ and a large number of measurements $y = \{y0, \cdots, y_{N_0-1}\}$. 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 matrix $A = M^T M$.

[2]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

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:

$$f(k) = \sum_{n=0}^{N-1} c_n e^{2\pi ikn/N} = a_0 + \sum_{n=1}^{N-1} \left[a_n \cos(2\pi kn/N) + b_n \sin(2\pi kn/N)\right]. \tag{8}$$

For the right hand side, I have used $c_n = a_n - ib_n$. This formula only works so well for the special case [3] that $f(k)$ is purely real! The c_n's can be defined by:

$$c_n = \frac{1}{N}\sum_{k=0}^{N-1} f(k) e^{-2\pi ikn/N} = \frac{1}{N}\sum_{k=0}^{N-1} f(k)[\cos(2\pi kn/N) - i\sin(2\pi kn/N)] \tag{9}$$

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'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 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. 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 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 8, 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`.

[3]To see this take the real part of RHS of Eq. 8: $\frac{1}{2}(c_n e^{2\pi ikn/N} + c_n^* e^{2\pi ikn/N}) = \frac{1}{2}(c_n + c_n^*)\cos(2\pi kn/N) + i\frac{1}{2}(c_n - c_n^*)\sin(2\pi kn/N)$ 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. 8.

- 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, 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 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.

III Extra Credit

First redo the fit in **Coding Exercise #2** above by selecting a few dominate modes in your low pass filter and minimizing the χ^2. Since the data is very good you may set $\sigma's = 1$ through out.

Also show the filter on dominate modes in the full FFT is NOT identical to the χ^2 fit of these same modes.

IV Comment for your project

This chi square test should be use in many (of not all) of your projects to fit to the runtime of your algorithms as function of problem size n. In reality determining the scaling behavior to larger systems on complex hardware is not easy because running the same code over and over does not give the same wall clock performance for a variety of reasons. It is a good idea to treat the computer as system with random noise and at least give fit to this system. Do to this you must run many trials at each size n and compute the mean and standard deviation before fitting to scaling using the χ^2 method. The new feature is the multiple runs are done not only to get fitting data BUT also to give estimates of the standard deviations σ_k to define the χ^2.

In class we will give a code as a warm up exercise to try this out. We can choose an example from our past homework to study scaling as function of size n to do a careful estimate of the scaling of complexity (or time) $T[n]$. To do this run run multiple trial runs $k = 1, 2, \cdots, N_{trials}$ and fit $T[n]$ to the mean time and the standard deviations:

$$T_{mean}[n] = \frac{1}{N_{trials}} \sum_{k=1}^{N_{trials}} T_k[n] \quad , \quad \sigma_n^2 = \frac{1}{N_{trials} - 1} \sum_{k=1}^{N_{trials}} (T_k[n] - T_{mean}[n])^2 \tag{10}$$

The fitting now requires a model assumption such as $T[n] = c_0 + c_1 n^\theta + c_2 nlogn + \cdots$ and fitting you data by minimizing

$$\chi^2 = \sum_n \frac{(T_{mean}[n] - T[n])^2}{\sigma_n^2} \tag{11}$$

This is non-linear fit in c_0, c_1, θ_0, c_2 but gnuplot will do this fit for you. In class let's find a good example to try this on.