ME 701 – Development of Computer Applications In Mechanical Engineering
Homework 2 – Programming Basics with Python – Due 9/29/2014

Instructions: Designate who did what problems; see below for team assignments. For each problem, state all assumptions and clearly indicate a final answer where applicable, following the "solution template."

Problem 1 – Nonlinear Regression

In NE 495, you might recall having learned about the *semi-empirical mass formula* (SEMF), which fits nuclear mass data to a function of the atomic number (i.e., number of protons) Z and mass number (i.e., number of protons and neutrons) A. Traditionally, the SEMF is given the form

$$B(A,Z) = a_v A + a_s A^{2/3} + a_c \frac{Z^2}{A^{1/3}} + a_a \frac{(A-2Z)^2}{A} + \frac{a_s}{A^{1/2}} \left[(-1)^{Z \bmod 2} + (-1)^{(A-Z) \bmod 2} \right] .$$

Your task is to determine the 5 coefficients a_v, a_s, a_c, a_a, and a_s by fitting the given function to the mass data provided online.

Deliverables:
1. Short summary of how you solved the problem.
2. The coefficient values.
3. The root mean square error.
4. A plot of the measured data and your fitting function.
5. Your very well-documented, very clean code.

BONUS: (Here, bonus means I'll write really nice letters of recommendation...). Look up a tool called "Eureqa" for performing *symbolic regression*. Unlike regular regression, which determines values for coefficients of a prescribed model, symbolic regression finds the model *and* any coefficients. See if you can use Eureqa to determine your own (better?) SEMF.

Problem 2 – Numerical Integration

We often go into (and, perhaps, come out of) engineering classes thinking that calculus and, by extension, differential equations are the most important math for doing what we need to do. On paper yes—in practice no. Can computers compute df/dx? Or $\int f(x)dx$? Not generally, and so we need to go from dx to Δx and hope for the best.

In this exercise, we'll examine numerical quadrature, i.e., numerical integration. Consider a function $f(x)$ and the definite integral

$$I = \int_a^b f(x)dx \approx \sum_{n=1}^{N} w_i f(x_i) dx ,$$

where x_i are the *abscissa* and w_i are the weights of integration. You are probably familiar with the *mid-point rule*, which lets $w_i = \Delta = (b-a)/N$ and $f(x_i) = f((i-1/2)\Delta)$. In the limit that $\Delta \to 0$, the result is just the formal definition for an integral you learned in calculus. For brevity, we'll shorten $f(x_i)$ to f_i.

You will consider three different N-point quadratures:

1. Midpoint rule (evenly spaced), with $I \approx \Delta \sum_j^N f_j$
2. Simpson's rule (evenly spaced, with $I \approx \frac{\Delta}{3}\left(f_0 + 2\sum_{j=1}^{N/2-1} f_{2j} + 4\sum_{j=1}^{N/2} f_{2j} + f_N\right)$
3. Gauss-Legendre quadrature (see, e.g., `http://pomax.github.io/bezierinfo/legendre-gauss.html`)

Unlike the midpoint and Simpson's rules, Gauss-Legendre has unevenly-spaced abscissa, and the abscissa and weights can be found at the given site. You will apply these quadratures to the three integrals

1. $\int_1^{-1} x^5 dx$
2. $\int_5^{-5} \frac{1}{x^2+1}$
3. $\int_0^{\pi} \frac{x \sin x}{1+\cos^2 x}$ (which, by the way, equals $\pi^2/4$)

Deliverables:

1. Compute and plot the *relative, absolute error* of your integration, for each quadrature, and for each integral as a function of N for $N = 1, 2, \ldots, 10$. Use a log scale for the y-axis. You should have three three plots: one per function, with three curves per plot.
2. Comment on any strange or unexpected results.
3. Your very well-documented, very clean code. By the way, if you can make use of, e.g., `scipy`, for expediting some of this, that is acceptable and encouraged (though knowing how to set up your own integrator can be useful in the field...)

Problem 3 – The Sieve of Eratosthenes

Straight from Wikipedia:
"A prime number is a natural number which has exactly two distinct natural number divisors: 1 and itself.

To find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

1. Create a list of consecutive integers from 2 through n: (2, 3, 4, ..., n).
2. Initially, let p equal 2, the first prime number.
3. Starting from p, enumerate its multiples by counting to n in increments of p, and mark them in the list (these will be 2p, 3p, 4p, etc.; the p itself should not be marked).
4. Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.

When the algorithm terminates, all the numbers in the list that are not marked are prime."

Deliverables:

1. Implement a function named `find_primes(n)` for providing all of the prime numbers less than n (where n is arbitrary). Your function should handle all values of n up to 10^5 in a reasonable amount of time (i.e., a few minutes at most).
2. Name the file `primes.py`.
3. **BONUS**. Although a fully correct function gets full credit, I'll give bonus points to the fastest implementation.