PHYS 4023
Version 1

Homework Assignment #5: Quadrature & Random Numbers
Due Nov. 21st

Quadrature:

1. (Problem 7.2) Carry out an error analysis for the mid-point rule (use the Taylor series similar to what we did in class keeping terms up to 2nd order). This should include both the one panel and the composite case. Then implement midpoint rule in Python for the usual case $f(x) = 1/\sqrt{x^2+1}$ ($a = 1, b = 1$) for $n = 51$. Compare with the analytic answer as well as with the other methods (e.g., endpoint)

2. (Problem 7.14) We will study the *Glasser function*, a function that exhibits many oscillations and thereby necessitates adaptive integrator to compute accurately:

$$G(x) = \int_0^x \sin(t\sin(t))dt$$

   (a) Use an adaptive Simpson's method to plot $G(x)$ from $x = 0$ to $x = 20$.

   (b) Plot the number of integration points n needed for each x value (for an absolute tolerance of 10^{-12}); use a logarithmic scale on the y-axis.

   (c) Locate the first 20 minima of $G(x)$ after the origin.

3. (Problem 7.25) In code 3.3 (i.e., psis.py), we computed the values of Hermite Polynomials and their derivatives; then, in problem 5.18 we saw how to computer their roots, by modifying code 5.4 (i.e., legroots.py). Following the approach outlined in in section 7.5.3 on arrives at the following formula for the weights:

$$c_j = \frac{2^{n+1}n!\sqrt{\pi}}{[H_n'(x_j)]^2}$$

   which is the Gauss-Hermite analog of Eqn. 7.117. Implement a general routine for Gauss-Hermite quadrature; feel free to use 'numpy.sort()' and 'numpy.argsort()'. The factorials in our equation for the c_j's can cause problems, but our root solver doesn't work for too large values of n anyways, so don't go above $n = 20$. Compare your results with the results from 'numpy.polynomial.hermite.hermgauss()'.

4. (Problem 7.26) Carry out the change of variables $u = \sqrt{x}$ for the integral:

$$I = \int_0^\infty \frac{x^2+x}{\sqrt{x}} e^{-x} dx$$

   and observe that you can compute it using Gauss-Hermite quadrature, if you also realize the integrant is now even. If you solved problem 7.25, use the function you developed there and compare results with 'numpy.polynomial.hermite.hermgauss()'.

5. (Problem 7.28) Apply the trapezoidal rule to the following integral:

$$I = \int_0^\pi \sqrt{x}\cos(x)dx$$

   (a) Print out the answer for increasing values of n, checking to see which digits are no longer changing. Do you understand why this function is slow to converge?

   (b) Carry out the change of variables $u = \sqrt{x}$ and then apply the trapezoidal rule again. Compare the convergence behavior with that in part (a).

6. (Problem 7.54) In quantum statistical mechanics, the total number of particles takes the form:

$$N = A(k_B T)^{\alpha+1} \int_0^\infty dx \frac{x^\alpha}{e^{x-w} \pm 1}$$

where $x = \epsilon/(k_B T), w = \mu/(k_B T)$, the plus sign corresponds to a Fermi gas, the minus sign to a Bose gas, and α is an exponent related to whether or not the gas is relatvistically ($\alpha = 2$ or $\alpha = 1/2$. Textbooks typically address only the degenerate (vanishing T) or classical (huge T) limits. Here we will focus on intermediate temperatures, doing the integrals numerically. Plot the value f the integral for bosons (w from -30 to 0) and fermions (w from -10 to $+20$) with a logarithmic scale on the y-axis; in each plot include results for both α cases. You can use any method you want to compute the integrals, but it's probably easiest to employ Gauss-Legendre quadrature up to some large energy cutoff/right endpoint of integration.

Random Numbers:

1. (Problem 7.31) Reproduce both panels of Figure 7.5. Then, make a pairwise plot using the numbers provided by 'random.random()'.

2. (Problem 7.32) We now introduce the *equidistribution* test for random number generators. Split the region from 0 to 1 into, say, $n_b = 10$ bins of equal size. Generate $n = 10^5$ samples in total and count the populations in each bin, n_j, where $j = 0, 1, ..., n_b - 1$. We expect to find n/n_b samples in each bin (so 10^4 in our example). Compute the following statistic:

$$\chi^2 = \frac{n_b}{n} \sum_{j=0}^{n_b-1} \left(n_j \frac{n}{n_b}\right)^2$$

for the three generators used in problem 7.31. A "good" χ^2 is equal to the number of degrees of freedom. Since $\sum_j n_j = n$, the n_b populations are not fully independent, so we actually have $n_b - 1$ degrees of freedom.

3. (Problem 7.35) This chapter is focused on numerical integration, but the ability to draw samples from a desired $w(x)$ distribution, which we developed when introducing Monte Carlo Techniques (section 7.7) is much more general. In this problem, we will see how to generate "synthetic data" from a desired distribution. The following weight function is an example of the *Cauchy distribution*:

$$w(x) = \frac{1}{10\pi[(x-2.5)^2 + 0.01]}$$

Apply the inverse-transform sampling method from section 7.7.3.2 to generate Cauchy-distributed samples , $\mathfrak{x}_i = g^{-1}(\mathcal{U}_i)$.

   (a) Produce a histogram of raw counts, i.e., the y-axis should be the population of x's in a given narrow 'bin'. You should use 100 bins of equal width from -9 to 14 for 10^4 Cauchy-distributed samples. Use the appropriate *NumPy/Matplotlib* functionality for the visualization.

   (b) Make a plot of: (i) the $w(x)$ you started from, Eqn. 7.258, and (ii) the raw counts normalized such that the total area in the previously produced histogram is one.

   (c) To understand how important the tails are, generate $n = 10^3, 10^4, ...10^8$ Cauchy-distributed samples and compute each sample mean (no need to plot anything). For each n, compare the answer with the corresponding sample mean using n normally distributed samples (for $\mu = 2.5$ and $\sigma^2 = 0.01$).

4. (Problem 7.40) The one-dimensional integral $\int_0^\infty e^{-x} x^2 dx$ is of the form $\int_0^\infty w(x) f(x) dx$ for $w(x) = e^{-x}$ and $f(x) = x^2$. Implement the metropolis algorithm for 10^8 total steps, measuring f for every 100 steps. (To keep the x's positive, you may interpret $w(x < 0) = 0$, allowing you to simply reject any move that tries to make x negative.) Observing that you can also employ Gauss-Laguerre quadrature, recompute this integral using $n = 2$ together with the function 'numpy.polynomial.laguerre.laggauss()'.