There are 5 problems, each of which is worth 20 points. There are two bonus points for following instructions, plus a couple of bonus questions, which can make up for mistakes elsewhere. All codes/work should be put into a zip/tar file and emailed to me. Do NOT put your answers on github. You may refer to notes/old codes and use google, but you may not consult anyone (including online) besides the TAs and the instructor. All problems are worth the same amount, and parts within each problem are worth the same amount. The final score is capped at 100.

Problem 0:

Please put everything into a directory. Include a file called "myinfo.txt" that has your name and ID in it, so I can match grades to students at the end. (2 points)

Problem 1: Computer Division Computers are very good at multiplying, adding, and subtracting (which is just adding with a sign flip), but division is another matter. You will often hear that division is "slower", but without quantifying how much slower. In this problem, we will look at how computers can divide floating point numbers. Since y/x can be written as $y * (1/x)$, we will in fact restrict ourselves to just calculating $1/x$.

a) For starters, we will only consider $1/x$ for x in the range of 0.5 to 1.0. Explain why this is not actually a limitation given how computers store floating point numbers.

b) Let's assume that we have an *approximate* value for $1/x$, which we'll call $\tilde{x_i}$. If we have $x * \tilde{x_i} = 1 + \delta$, show analytically that the first order improvement to $\tilde{x_i}$ is $\tilde{x_i} \to \tilde{x_i} * (1 - \delta)$.

c) Current CPUs start with a lookup table where they have stored $1/x$ for 64 entries between 0.5 and 1. Write a routine called *inv_guess* that generates such a lookup table and returns the nearest entry of $1/x$ from that table when given an input x in $[0.5, 1)$.

d) Write another routine called *inv* that returns $1/x$ accurate to nearly machine precision. Your routine should start with $\tilde{x_i}$ from the lookup table, and repeatedly apply the algorithm in part b). Explain why a sensible stopping point for this code when inverting *double precision* numbers would be something like $|\delta| < 10^{-8}$, assuming you return the updated value for $\tilde{x_i}$ using that δ.

e) How many iterations does *inv* typically need to converge? How many *total* multiplications is it doing?

f) Write a similar routine called *inv2* that uses the second-order expansion to calculate the update to $\tilde{x_i}$. How many iterations does this typically need to converge? How many total multiplications does the second-order method need? Would you expect first-order or second-order to be faster?

Comments: Technically the IEEE standard does not allow for replacing y/x with $y * (1/x)$, but the GCC option $-ffast-math$ turns it on. Compilers also often replace division by constants with multiplication by the inverse at compile time, so writing $x = x/3$ in your code probably isn't any slower than

$x = x * 0.333333.....$ That may depend on optimizations you use, though.

Problem 2: Volume of an n-Sphere Calculating the volume of an $n-$dimensional sphere analytically can be somewhat annoying. We'll use the power of computers to make it easier! Recall that the region inside an n-sphere of radius 1 is $x^2 + y^2 + z^2 + ... \leq 1$.

a) Generate a large number of random points uniformly distributed between -1 and 1. Find the fraction with $|r| < 1$ in both 2 and 3 dimensions. Show that that fraction times the volume of the $n-$cube bounded by $[-1, 1]$ gives you the area of a circle/volume of a sphere.

b) Show analytically that the PDF of the sum of two independent random variables is the convolution of their PDFs.

c) Show analytically that the PDF of $y = x^2$ where x is uniform on $[0, 1]$ is $1/(2\sqrt{y})$.

d) Using the results in parts b) and c) (don't worry if you didn't answer those parts - you can still use the quoted results), write a routine that calculates the volume of an n-sphere with radius=1 in *arbitrary* dimensions. It should do so using convolutions to find the PDF of $r^2 = \sum^{n_{dim}} x_i^2$ (where x_i is uniform on $[-1, 1]$), find the fraction of that probability that has $r^2 < 1$, and from that scale to find the volume of the sphere. Please use

```
def sphere_vol(ndim,dx=0.001):
```

for your prototype, and it should return a floating point number[1].

Problem 3: PDE Stability

We saw in class that satisfying the CFL condition was necessary but often not sufficient for PDE solvers to be stable. In this problem, we will consider stability of different updates for the advection equation, $\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} = 0$. As a reminder, the simplest discretized advection solver looks like:

$$f(t + \delta t, x) = f(t, x) - \delta t v \frac{\partial f}{\partial x}$$

.

a) We saw in class that trying to solve the advection equation with the "upwind" derivative $\frac{\partial f}{\partial x} \approx \frac{f(x)-f(x-\delta x)}{\delta x}$ was stable as long as $\alpha \equiv v\delta t/\delta x <= 1$. Show analytically that the downwind derivative $\frac{f(x+\delta x)-f(x)}{\delta x}$ is unstable.

b) We also saw that using the Lax method could restore stability. As a reminder, the Lax method was to approximate $f(t, x)$ by $\sim (f(t, x+\delta x)+f(t, x-\delta x))/2$. First, if $f(x, t) = \exp(ikx)$ at some time t, what is the amplitude of $f(t + \delta t, x)$ if we use both the Lax method and the downwind derivative? Feel free to leave this as a complex number, but group the real term(s) together and

[1] Wikipedia has a list of the volumes you can use to check your code is producing correct results.

the imaginary term(s) together. To be explicit, Lax+downwind would give us:

$$f(t+\delta t, x) = \frac{f(t, x+\delta x) + f(t, x-\delta x)}{2} - \alpha\left[f(t, x+\delta x) - f(t, x)\right]$$

c) Plot the absolute value of this amplitude as a function of $\theta \equiv k\delta x$ from 0 to 2π for $\alpha = (0, 0.25, 0.5, 0.75, 1)$. Name your plot *stability.png*. Which (if any) of these values of α will lead to a stable solver?

d) Solve for the critical value of α below which the solution is stable. You can do this either numerically or analytically. If you do it numerically, please try to express your answer in terms of a famous number (if you aren't sure, you can google your answer. It will show up if you search on the first few digits). It's slightly tricky analytically - I suggest you express the squared amplitude as a function of α and θ. That should give you a quadratic as a function of $\cos(\theta)$, and the critical value happens when the square root ($\sqrt{b^2 - 4ac}$) in the quadratic equation (which is only a function of α) equals zero (you can accept this fact without proving it on the exam). That expression will be a perfect square if everything has gone well, which will let you write down the answer in closed form.

e) Write/modify an advection code to use this downwind/Lax scheme. Please name your code *advect_downwind_lax.py*. Plot your solution, starting from a boxcar, using periodic boundary conditions for values of α a little below and a little above the critical value (you may feel free to pick values of α inspired by part c) as well).

Bonus: Explain why the plots in part e) look the way they do, based on your curves from part c). In particular, the way the solution grows when you choose an unstable value for α has a characteristic behavior that is different from what we have seen before; explain that behavior.

Problem 4: Rutherford Scattering In Rutherford scattering, α-particles go through a thin gold foil. Most pass through unimpeded, but occasionally an α has a near-direct collision with a nucleus. When that happens, the α can be deflected by a large angle. The key physics is that if the nuclei the α's scatter off of are point sources, the intensity of scattered α's as a function of deflection angle θ should go like $\sin(\theta)^{-4}$. Look in *rutherford_foil.npz* for data you will use to reproduce this result.

The experimental setup is a detector is placed at an angle θ relative to the nominal direction of the beam of α-particles. A particle detector runs for some length of time, and reports the number of particles it counted during that length of time. You can read in the data as follows:

```
import numpy as np
data=np.load('rutherford_foil.npz')
angles=data['angles_foil']
counts=data['counts_foil']
tobs=data['tobs_foil']
```

a) Data at low deflection angles are swamped by the signal from the unscattered beam and so **only use data with** $|\theta| \geq 10$. We first need to convert to counts per second from raw counts. Recalling that the variance of a Poisson process is equal to the expected number of events (so the uncertainty in the raw counts should be roughly equal to the square root of the number of raw counts), print the counts per second and the uncertainty in that number for each angle you will use (there should be a total of 8 data points).

b) The simplest model we can use is that the count rate should be $c\sin(|\theta - \theta_0|)^\beta$ where c is an uninteresting overall amplitude, β should be close to -4, and θ_0 is the actual angle of the beam. This should be close to 0, but of course the realities of experimental physics means it won't be exactly 0. Using Newton's method/Levenberg-Marquardt and analytic derivatives (do not forget about the absolute value inside the sin), report your best-fit values *and errors* for c, θ_0, and β. If all has gone well, you should indeed get something close to -4 for β.

c) What is the total χ^2 at your best-fit parameters? Do you believe your error bars from part b)? If you did not get b) to work, answer this question assuming you got 1) a χ^2 of 6, and 2) a χ^2 of 50.

d) One of the big uncertainties in the data is the actual position of the detector (see Figure 1 to see why this might be the case). Assume the uncertainty in the angles is 0.1 degrees, *i.e.* $\theta_{true} = \theta_{reported} + \delta\theta$ and $\sigma(\delta\theta) = 0.1$ degrees. Analytically, fitting models when you have errors in both x and y can be hard, but MCMC can handle this without much pain. Update your model to be $d_i = c\sin(\theta_i + \delta\theta_i - \theta_0)^\beta$ where each value of θ has its own $\delta\theta$. Of course, we know that $\delta\theta$ should be small, so your likelihood for the MCMC should include the usual contribution from the difference between the data and the fit plus $\sum\left(\frac{\theta_i}{\sigma_\theta}\right)^2$. We now have more parameters than data points, but that extra term in the likelihood will save us! What is your new best-fit value/error for β? As a sanity check, if you set σ_θ to something very small, you should reproduce your results from b).

bonus: The instrument beam has finite width, so what we see should really be the convolution of the "true" signal with the instrument beam. Beam data (without the scattering foil) are contained in *rutherford_nofoil.npz*. Find a model for the beam that fits the beam data reasonably well. Redo the Markov chains accounting for the beam convolution and report your new value/uncertainty for β. Did this make a difference for your residuals?

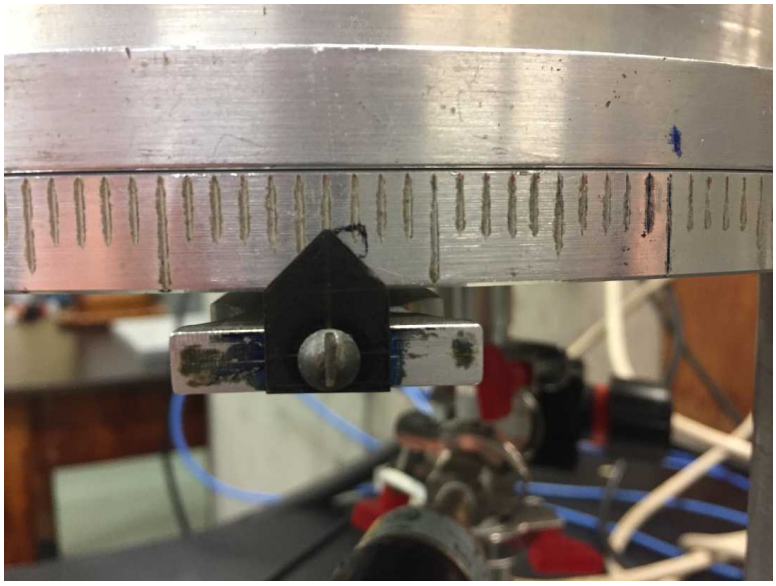

Figure 1: Rutherford angle setup. The position of the arm below the steel cylinder sets the angle between the detector and the beam. Each vertical tick mark represents one degree. The one in black corresponds to zero degrees, so the angle here is nominally 14 degrees. The angle is set manually, so there will always be some error between the angle we meant to use and the actual angle we set the instrument at. Hopefully from this picture you can get a sense for why 0.1 degrees is a reasonable uncertainty for our angle values.