There are two bonus points for following instructions, plus a bonus question, 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. Each of the 4 problems is worth the same amount (25 points) and parts within each problem are worth the same amount. The final score is capped at 100. You have 24 hours to complete from when you start working.

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. Please do NOT put your name elsewhere in your codes/answers. (2 points)

Problem 1 - Landing on Boardwalk:
Part a) Show analytically that the probability distribution of the sum of two independent variables is the convolution of the probability distributions of the original variables.

Part b) In the classic game *Monopoly*, players roll pairs of 6-sided dice each turn. They then move forward the number of squares that they rolled. What is the probability you land on Boardwalk on your first pass around the board on your n^{th} roll? Boardwalk is 39 squares from your starting location (and please ignore chance/community chest/go to jail. I am asking about rolls, not turns, so doubles are nothing special either). Please save the probability you are on Boardwalk during your first pass after n rolls, and report the total probability you land on Boardwalk during your first pass around the board. The output should be a text file "boardwalk.txt" and should have a separate line for each roll containing the roll number and the probability of hitting Boardwalk. You will likely want to use the result from part a). Note that I am asking only about landing on Boardwalk during your first pass, so your rolls sum to 39; you don't need to worry about hitting Boardwalk on your second pass (where the rolls would sum to 79). As a sanity check, the probability should be zero for the first three rolls and any turn past 20, since the minimum roll is 2 and the maximum roll is 12.

Problem 2 - Hunting for Planets:
One way astronomers find planets around other stars is by looking for time-varying changes in the radial velocity of those stars. As the Earth goes around the Sun, the Sun also goes around the Earth, but since the sun is a million times more massive, its velocity is a million times slower than the Earth's, or a few cm/s. Unfortunately for astronomers, the intrinsic width of spectral lines corresponds to at least a few km/s, so they have to find the center of a line to an accuracy of about a thousandth of its with. For this problem, I have provided simulated data for you in planet_data.npz, which can be read as follows:

```
import numpy as np
stuff=np.load("planet_data.npz")
```

```
v=stuff["v"]
dat=stuff["dat"]
```

You may assume that the data are Gaussian, of the form $a - b\exp(-(v - v_0)^2/2\sigma^2)$ where a is a background (close to, but perhaps not exactly, 1), b is the depth of the spectral line (this should be order unity but less than 1), v_0 is the velocity of the line center relative to us, and σ is the width of the line. I have given you v in m/s, so you might expect σ to be of order 10^3. You may also assume the noise in the data is Gaussian and uncorrelated between data points.

Part a) *Ignoring noise*, fit a Gaussian to the provided data. What are your best fit values for a,b,v_0, and σ?

Part b) We would expect the noise to be proportional to the square root of the signal, so $\langle (dat - model)^2 \rangle = n_0^2 model^2$. Estimate n_0 and report your value. You will receive partial credit if you estimate the value by looking at a region away, but to get full credit you should use your model from Part a).

Part c) Use your noise model from Part b) to re-fit the line using the noise model. What is your estimate for the uncertainty in v_0 using this noise model?

Part d) If you were an ET astronomer, could you find the Earth using the data presented here? You can assume the sun's induced velocity is 0.1 m/s. In real life, there might be 1,000 spectral lines you could use in the same spectrum. What would your (approximate) uncertainty on v_0 if you used all the lines? If you didn't get Part c), you can assume that the uncertainty on v_0 was 10 m/s.

Problem 3 - Advection Stability:

Part a) In class for advection, we saw that for advection, the *upwind* estimate for the gradient was stable, as long as we satisfied the CFL condition. The upwind gradient estimate was $\nabla(f) = (f(x) - f(x_{left}))/dx$. Derive the stability constraint for the *downwind* gradient $\nabla(f) = (f(x_{right}) - f(x))/dx$. Show that this is unstable for any value of $\alpha = vdt/dx$, where $f(x, t + dt) = f(x, t) - \alpha(f(x_{right}) - f(x))$.

Part b) We saw that the Lax method could restore stability by adding numerical diffusion. In particular, we replaced $f(t, x)$ with $((f(t, x + 1) + f(t, x - 1))/2$. Is the downwind derivative stable when we use the Lax method? You can show either analytically or with code.

Part c) We can take a more flexible approach by extending the Lax method as follows: we take $f(t) = b((f(t, x + 1) + f(t, x - 1))/2) + (1 - b)f(t, x)$. If we choose $b = 1$, we have the classic Lax solution, or $b = 0$ to give us our original solution. For the downwind gradient, with $\alpha = 1$, what value or values of b will lead to a stable solution?

Problem 4 - N-body performance:

We saw that an *n-body* simulation could calculate long-range forces efficiently using FFTs. However, in the class example, we ignored the forces between

any particles in the same grid cell. This is called a *particle-mesh* scheme and not surpisingly is not very accurate on small scales. We can do better with *particle-particle/particle-mesh* (P^3M) schemes where we also explicitly calculate the forces between particles *in the same grid cell*. In such codes, we have the freedom to set the grid size. In this problem, we will estimate the optimal grid size for P^3M codes. We will work in *three dimensions* where we have m grid cells along each side, and n^3 total particles.

Part a) If the particles are roughly evenly distributed in the grid, on average how many particles per grid cell are there?

Part b) The work to do an FFT of size p is $p \log(p)$. This remains true even in multi-dimensions as long as p is the *total* number of elements. So, *e.g.* the scaling for a 2D FFT of a 1000 by 1000 array would have $p = 1000^2 = 10^6$. If the run time to do an FFT is $ap \log(p)$ for some coefficient a, how long would it take to FFT our m by m by m grid?

Part c) If the time to brute-force calculate the forces between a set of p particles is bp^2, how long would it take to calculate the forces within a *single* cell of our grid, given n, m, and your answer to Part a)? How long would it take to calculate the forces between particles in *all* grid cells (to be clear, you only calculate the forces between particles within a cell, but you must do that within-cell calculation for every cell in the grid)?

Part d) Find the value for m that minimizes *total* run time, which will be the sum of Parts b) and c). Formally this is a transcendental equation, but you may assume that the $\log(m)$ is constant (which it very, very nearly is). If the coefficients a and b are similar, and $\log_2 m = 10$ (which it will be to within a factor of 2 for any foreseeable simulations), roughly how many particles should you have per grid cell for your optimal value of m?

Bonus: If we let our simulation run, we will of course see our particles start to collapse under gravity. What *fractional* overdensity will it take before the n^2 work in a single cell equals that of the FFT? Galaxy clusters can have densities 1000 times higher than the background, but occupy maybe 10^{-4} of the volume of space. How much slower would your code run after galaxy clusters form if you don't change m? One way around this problem is through what is called *adaptive mesh refinement* (AMR), which subdivides highly populated cells into a finer mesh just in that region but leaves sparsely populated cells in a coarse grid.