Physics 411: Midterm 4

This is a take-home exam. You have **5 days** to complete it (not 2 as with the previous exams). Your answers must be handed in at or before the end of class, 11:30am on Tuesday, April 22 to receive a grade. **Answers turned in late will not receive a grade** (unless you've made arrangements with the professor to take your exam at a different time).

There are four questions, arranged roughly in order of difficulty—the later ones are harder, and also carry more points. Partial credit will be given for all parts of a question that are correctly completed.

The rules are the same as for the previous midterms. You may consult the textbook and you can use any of the on-line resources that accompany the textbook; you may use programs and functions from the on-line resources page as a starting point if you want, as well as programs you yourself have written for the homework assignments. But collaboration is not allowed. You may not discuss the exam with anyone else and you may not look up answers or use material from any other source, including friends or classmates, the internet, or other books. Everything you turn in must be your own original work and no one else's.

Paragraphs marked with a check-mark thus ☑ indicate what is to be handed in for each question.

Good luck!

1. **[8 points]** We have studied the relaxation method as way of solving partial differential equations, but there is no reason why it must be restricted to the solution of differential equations with two or more independent variables. It can also be applied to those with one independent variable, i.e., to ordinary differential equations. In this context, as with partial differential equations, it is a technique for solving boundary value problems, which are less common with ordinary differential equations but do occur, as we discussed in class.

   Consider the problem in Example 8.8 on page 390 of the book, in which a ball of mass $m = 1\,\text{kg}$ is thrown from height $x = 0$ into the air and lands back at $x = 0$ ten seconds later. The problem is to calculate the trajectory of the ball, but we cannot do it using initial value methods like the ordinary Runge–Kutta method because we are not told the initial velocity of the ball. One approach to finding a solution is the shooting method of Section 8.6.1. Another is the relaxation method.

   Ignoring friction effects, the trajectory is the solution of the ordinary differential equation

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -g,$$

   where g is the acceleration due to gravity.

   (a) Replacing the second derivative in this equation with its finite-difference approximation, derive a relaxation-method equation for solving this problem on a time-like "grid" of points with separation h.

   (b) Using boundary conditions $x = 0$ at $t = 0$ and $t = 10$, write a program to solve for the height of the ball as a function of time using the relaxation method with 100 points and make a plot of the result from $t = 0$ to $t = 10$. Run the relaxation method until the answers change by 10^{-6} or less at every point on each step.

   Note that, unlike the shooting method, the relaxation method does not give us the initial velocity needed to achieve the required solution. It gives us only the solution itself, although one could get an approximation to the initial velocity by calculating a numerical derivative of the solution at time $t = 0$. On balance, however, the relaxation method for ordinary differential equations is most useful when one wants to know the details of the solution itself, but not the initial conditions needed to achieve it.

   ☑ **For full credit** turn in your derivation from part (a), a printout of your program, and a printout of the plot it produces.

2. **[8 points]** Suppose you wish to choose a random point on the surface of the Earth. That is, you want to choose a latitude and longitude in such a way that every point on the planet is equally likely to be chosen. In a physics context, this is equivalent to choosing a random vector direction in three-dimensional space (something that one has to do quite often in physics calculations).

   Recall that in spherical coordinates θ, ϕ the element of solid angle is $\sin\theta\,d\theta\,d\phi$, and the total solid angle in a whole sphere is 4π. Hence the probability of our point falling in a particular element is

$$p(\theta,\phi)\,d\theta\,d\phi = \frac{\sin\theta\,d\theta\,d\phi}{4\pi}.$$

   We can break this up into its θ part and its ϕ part thus:

$$p(\theta,\phi)\,d\theta\,d\phi = \frac{\sin\theta\,d\theta}{2} \times \frac{d\phi}{2\pi} = p(\theta)\,d\theta \times p(\phi)\,d\phi.$$

   (a) What are the ranges of the variables θ and ϕ? Show that the two distributions $p(\theta)$ and $p(\phi)$ are correctly normalized—they integrate to 1 over the appropriate ranges.

   (b) Find formulas for generating angles θ and ϕ drawn from the distributions $p(\theta)$ and $p(\phi)$. (The ϕ one is trivial, but the θ one is not.)

   (c) Write a program that generates a random θ and ϕ using the formulas you worked out. (Hint: In Python the function `acos` in the `math` package returns the arc cosine in radians of a given number.)

   (d) Modify your program to generate 500 such random points, convert the angles to x, y, z coordinates assuming the radius of the globe is 1, and then visualize the points in three-dimensional space with small spheres (of radius, say, 0.02) using the `visual` package. You should end up with a three-dimensional globe spelled out on the screen in random points.

☑ **For full credit** turn in your answers to parts (a) and (b), a printout of your final program, and a printout of the visualization it produces.

3. **[10 points]** Consider the diffusion equation in two dimensions:

$$\frac{\partial \phi}{\partial t} = D\left[\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right].$$

(a) Replace the spatial second derivatives with finite differences on a square grid of points with separation a along both axes, and hence show that to a good approximation

$$\frac{\mathrm{d}\phi(x,y,t)}{\mathrm{d}t} = \frac{D}{a^2}\big[\phi(x+a,y,t) + \phi(x-a,y,t) + \phi(x,y+a,t) + \phi(x,y-a,t) - 4\phi(x,y,t)\big].$$

(b) Write a program that uses the FTCS method to solve the two-dimensional diffusion equation in a square system of side $L = 1$ with initial condition

$$\phi(x,y,0) = \sin^2(4\pi x)\sin^2(4\pi y)$$

and diffusion constant $D = 0.1$. Use a grid of 100×100 points and *periodic boundary conditions*, meaning the system wraps around from left to right and from top to bottom—if you go off the left side you come back on the right, and so forth. Note that there are no fixed values on any of the grid points in this system—no points that are fixed at zero or any other value. Every grid point can take whatever value it likes.

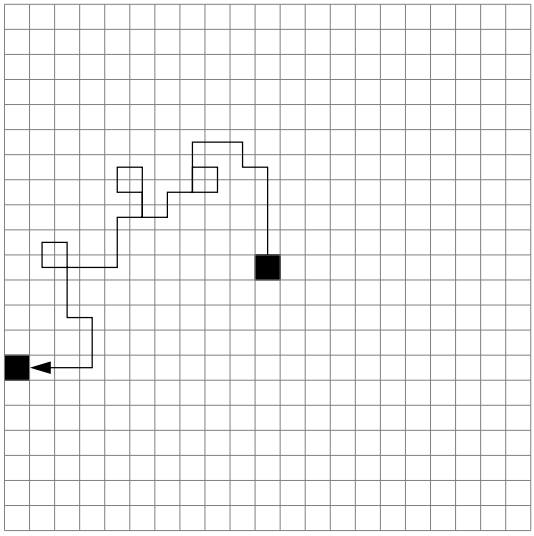
Use a time-step of $h = 10^{-5}$ and have your program make density plots of $\phi(x,y,t)$ at times $t = 0$, 0.01, 0.05, and 0.1. Use the same color/brightness scale for each density plot, which you can fix using the `vmin` and `vmax` arguments for `imshow`. Values of `vmin=0.0` and `vmax=1.0` are sensible. This will make it easier to see how the system evolves over time.

Hint: You will have to use Python's slicing/array manipulation abilities to solve this problem. If you do it the slow way, element by element, your program will not run in a reasonable amount of time. You'll also have to work out a way to do the periodic boundary conditions using slicing.

☑ **For full credit** turn in your derivation from part (a), a printout of your program, and a printout of the density plots it produces.

4. **[12 points]** In this exercise you will write a computer program to reproduce one of the most famous models in computational physics, *diffusion-limited aggregation*, or DLA for short, which was invented by UM physics professor Leonard Sander along with Thomas Witten in 1981. There are various versions of DLA, but the one we'll look at is as follows.

   You take a square grid with a single particle in the middle. On each time step of the program, choose a random direction—up, down, left, or right—and move the particle one step in that direction. This process is called a random walk. The particle continues the random walk until it reaches a point on the edge of the system, at which point it "sticks" to the edge, becoming anchored there and immovable:



   Then a second particle starts at the center and does a random walk until it sticks either to an edge or to the top, bottom, or side of the other particle. Then a third particle starts, and so on. Each particle starts at the center and walks until it sticks either to an edge or to any anchored particle.

   (a) Write a program to perform the DLA process on a 101×101 lattice—we choose an odd length for the side of the square so that there is one lattice site exactly in the center. Repeatedly introduce a new particle at the center and have it walk randomly until it sticks to an edge or an anchored particle. Add code to your program to visualize the positions of both the randomly walking particles and the anchored particles using the `visual` package.

   Hints: You will need to decide some things. How are you going to store the positions of the anchored particles? On each step of the random walk you will have to check the particle's neighboring squares to see if they are at the edge of the system or are occupied by an anchored particle. How are you going to do this?

   (b) In the interests of speed, change your program so that it shows only the anchored particles on the screen and not the randomly walking ones. That way you need update the picture on the screen only when a new particle becomes anchored. Also remove any `rate` statements that you added to make the animation smooth. This will ensure that the program runs as fast as possible.

Set up the program so that it stops running once there is an anchored particle in the center of the grid, at the point where each particle starts its random walk. Once there is a particle at this point, there's no point running any longer because any further particles added will be anchored the moment they start out.

Run your program and see what it produces. If you are feeling patient, try modifying it to use a 201×201 lattice and run it again—the pictures will be more impressive, but you'll have to wait longer to generate them.

[Extra credit] A nice further twist is to modify the program so that the anchored particles are shown in different shades or colors depending on their age, with the shades or colors changing gradually from the first particle added to the last.

☑ **For full credit** turn in a printout of your program and a printout of the visualization it produces of the DLA system.