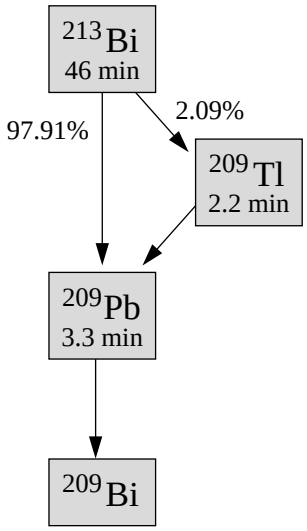
Physics 411: Homework 10

1. **Radioactive decay chain:** This exercise looks at a more advanced version of the simple radioactive decay simulation in Example 10.1.

   The isotope ^{213}Bi decays to stable ^{209}Bi via one of two different routes, with probabilities and half-lives thus:



   (Technically, ^{209}Bi isn't really stable, but it has a half-life of more than 10^{19} years, a billion times the age of the universe, so it might as well be.)

   Starting with a sample consisting of 10 000 atoms of ^{213}Bi, simulate the decay of the atoms as in Example 10.1 by dividing time into slices of length $\delta t = 1$ s each and on each step doing the following:

   (a) For each atom of ^{209}Pb in turn, decide at random, with the appropriate probability, whether it decays or not. (The probability can be calculated from Eq. (10.3) in the book.) Count the total number that decay, subtract it from the number of ^{209}Pb atoms, and add it to the number of ^{209}Bi atoms.

   (b) Now do the same for ^{209}Tl, except that decaying atoms are subtracted from the total for ^{209}Tl and added to the total for ^{209}Pb.

   (c) For ^{213}Bi the situation is more complicated: when a ^{213}Bi atom decays you have to decide at random with the appropriate probability the route by which it decays. Count the numbers that decay by each route and add and subtract accordingly.

   Note that you have to work up the chain from the bottom like this, not down from the top, to avoid inadvertently making the same atom decay twice on a single step.

   Keep track of the number of atoms of each of the four isotopes at all times for 20 000 seconds and make a single graph showing the four numbers as a function of time on the same axes.

   ☑ **For full credit** turn in a printout of your program and a copy of the graph it produces.

2. **Monte Carlo integration:** Calculate a value for the integral

$$I = \int_0^1 \frac{x^{-1/2}}{e^x + 1}\,dx,$$

using the importance sampling formula, Eq. (10.45), with $w(x) = x^{-1/2}$, as follows.

(a) Show that the probability distribution $p(x)$ from which the sample points should be drawn is given by

$$p(x) = \frac{1}{2\sqrt{x}}$$

and derive a transformation formula for generating random numbers between zero and one from this distribution.

(b) Using your formula, sample $N = 1\,000\,000$ random points and hence evaluate the integral. You should get a value around 0.84.

☑ **For full credit** turn your derivations for part (a) and a printout of your program and the answers it calculates.

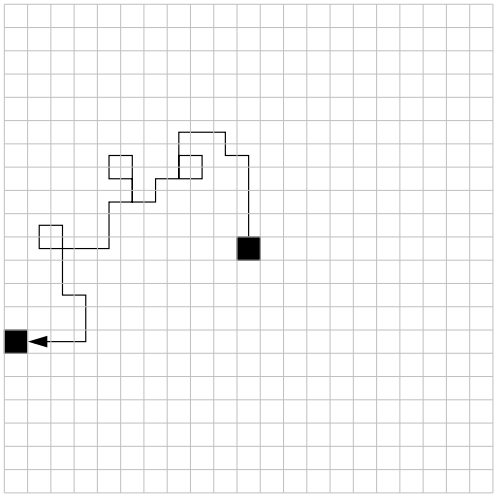
3. **Brownian motion:** Brownian motion is the motion of a particle, such as a smoke or dust particle, in a gas, as it is buffeted by random collisions with gas molecules. Make a simple computer simulation of such a particle (in two dimensions) as follows. The particle is confined to a square grid or lattice $L \times L$ squares on a side, so that its position can be represented by two integers $i, j = 0 \ldots L - 1$. It starts in the middle of the grid. On each step of the simulation, choose a random direction—up, down, left, or right—and move the particle one step in that direction. The particle is doing a "random walk." The particle is not allowed to move outside the square of the lattice—if it tries to do so, choose a new random direction to move in.

Write a program to do this calculation for a million steps of the random walk with $L = 101$ and make an animation on the screen of the position of the particle. (We choose an odd length for the side of the square so that there is one lattice site exactly in the center.)

Hint: The `visual` package doesn't always work well with the `random` package, but if you import functions from `visual` first, then from `random`, you should avoid problems.

☑ **For full credit** turn in a printout of your program.

4. **Diffusion-limited aggregation:** This exercise builds on the previous one on Brownian motion. In this exercise you will develop a computer program to reproduce one of the most famous models in computational physics, *diffusion-limited aggregation* (or DLA for short), invented by UM Professor of Physics Leonard Sander in 1981. There are various versions of the DLA process, but the one we'll study is as follows. You take a square grid with a single particle in the middle. The particle performs a random walk until it reaches a point on the edge of the square, 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 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) Make a copy of the Brownian motion program that you wrote for problem 3. This will serve as a starting point for your DLA program. Modify your program to perform the DLA process on a 101×101 lattice. Repeatedly introduce a new particle at the center and have it walk randomly until it sticks to an edge or an anchored particle.

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 neighboring squares to see if they border on the edge of the square or are occupied by an anchored particle. How are you going to do this? You should also modify your visualization code from the Brownian motion exercise to visualize the positions of both the randomly walking particles and the anchored particles. Run your program for a while and observe what it does.

(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 particles. That way you need to update the pictures on the screen only when a new particle becomes anchored. Also remove any `rate` statements that you added to make the animation smooth.

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 before they can even move anywhere.

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.

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.

(c) **Extra credit:** If you are feeling particularly ambitious, try the following. The original version of DLA was a bit different from the version above—and more difficult to do. In the original version you start off with a single *anchored* particle at the center of the grid and a new particle starts from a random point on the perimeter and walks until it sticks to the particle in the middle. Then the next particle starts from the perimeter and walks until it sticks to one of the other two, and so on. Particles no longer stick to the walls, but they are not allowed to walk off the edge of the grid.

Unfortunately, simulating this version of DLA directly takes a long time—the single anchored particle in the middle of the grid is difficult for a random walker to find, so you have to wait a long time even for just one particle to finish its random walk. But you can speed it up using a clever trick: when the randomly walking particle does finally find its way to the center, it will cross any circle around the center at a random point—no point on the circle is special so the particle will just cross anywhere. But in that case we need not wait the long time required for the particle to make its way to the center and cross that circle. We can just cut to the chase and start the particle on the circle at a random point, rather than at the boundary of the grid. Thus the procedure for simulating this version of DLA is as follows:

i. Start with a single anchored particle in the middle of the grid. Define a variable r to record the furthest distance of any anchored particle from the center of the grid. Initially $r = 0$.

ii. For each additional particle, start the particle at a random point around a circle centered on the center of the grid and having radius $r + 1$. You may not be able to start exactly on the circle, if the circle doesn't pass exactly through a grid point, in which case start on the nearest grid point outside the circle.

iii. Perform a random walk until the particle sticks to another, except that if the particle ever gets more than $2r$ away from the center, throw it away and start a new particle at a random point on the circle again.

iv. Every time a particle sticks, calculate its distance from the center and if that distance is greater than the current value of r, update r to the new value.

v. The program stops running once r surpasses a half of the distance from the center of the grid to the boundary, to prevent particles from ever walking outside the grid.

Try running your program with a 101×101 grid initially and see what you get.

☑ **For full credit** turn in a printout of your program and a snapshot showing the final state of the animation it produces. If you did the extra credit part, turn in your program and a snapshot for that part too.