Computational Physics – PH 354

Homework due on 31st March, 2022

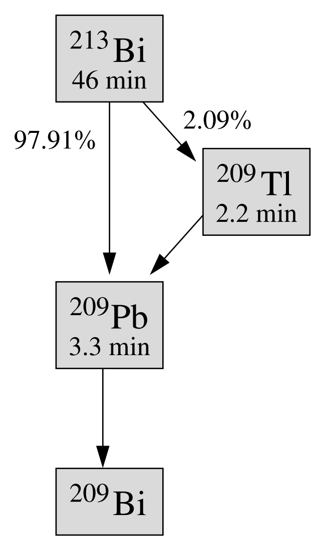
Exercise 1: Rolling dice

a) Write a program that generates and prints out two random numbers between 1 and 6, to simulate the rolling of two dice.

b) Modify your program to simulate the rolling of two dice a million times and count the number of times you get a double six. Divide by a million to get the *fraction* of times you get a double six. You should get something close to, though probably not exactly equal to, $\frac{1}{36}$.

Exercise 2: Radioactive decay chain

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 by dividing time into slices of length $\delta t = 1\,\text{s}$ 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 $p(t) = 1 - 2^{-t/\tau}$) 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.

Exercise 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. This process is called a random walk. The particle is not allowed to move outside the limits of the lattice—if it tries to do so, choose a new random direction to move in.

Write a program to perform a million steps of this process on a lattice with $L = 101$ and make an plot showing the trajectory 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.)

Exercise 4:

a) Write a program to evaluate the integral

$$I = \int_0^2 \sin^2\left[\frac{1}{x(2-x)}\right] dx$$

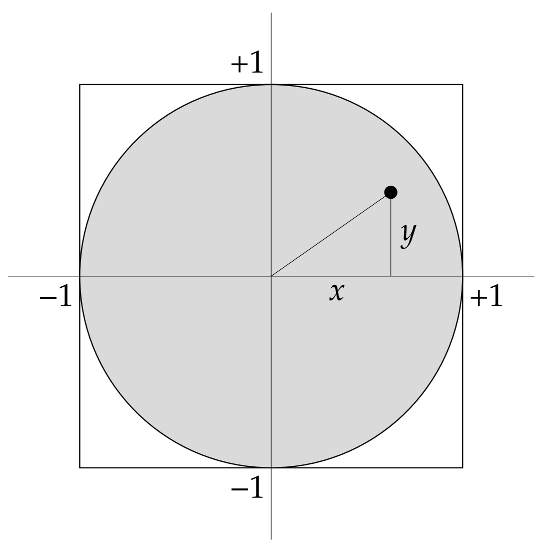
using the "hit-or-miss" Monte Carlo method with 10 000 points. Also evaluate the error on your estimate.

b) Now estimate the integral again using the mean value method with 10 000 points. Also evaluate the error.

You should find that the error is somewhat smaller using the mean value method.

Exercise 5: Volume of a hypersphere

This exercise asks you to estimate the volume of a sphere of unit radius in ten dimensions using a Monte Carlo method. Consider the equivalent problem in two dimensions, the area of a circle of unit radius:



The area of the circle, the shaded area above, is given by the integral

$$I = \iint_{-1}^{+1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y,$$

where $f(x,y) = 1$ everywhere inside the circle and zero everywhere outside. In other words,

$$f(x,y) = \begin{cases} 1 & \text{if } x^2 + y^2 \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

So if we didn't already know the area of the circle, we could calculate it by Monte Carlo integration. We would generate a set of N random points (x, y), where both x and y are in the range from -1 to 1. Then the integral can be calculated as

$$I \simeq \frac{4}{N} \sum_{i=1}^{N} f(x_i, y_i).$$

Generalize this method to the ten-dimensional case and write a program to perform a Monte Carlo calculation of the volume of a sphere of unit radius in ten dimensions.

If we had to do a ten-dimensional integral the traditional way, it would take a very long time. Even with only 100 points along each axis (which wouldn't give a very accurate result) we'd still have $100^{10} = 10^{20}$ points to sample, which is impossible on any computer. But using the Monte Carlo method we can get a pretty good result with a million points or so.

Exercise 6: Calculate a value for the integral

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

using the importance sampling formula 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

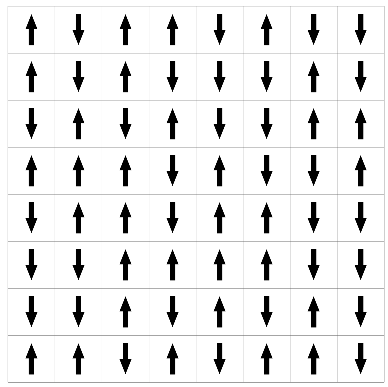
$$p(x) = \frac{1}{2\sqrt{x}}$$

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.

Exercise 7: The Ising model

The Ising model is a theoretical model of a magnet. The magnetization of a magnetic material is made up of the combination of many small magnetic dipoles spread throughout the material. If these dipoles point in random directions then the overall magnetization of the system will be close to zero, but if they line up so that all or most of them point in the same direction then the system can acquire a macroscopic magnetic moment—it becomes magnetized. The Ising model is a model of this process in which the individual moments are represented by dipoles or "spins" arranged on a grid or lattice:



In this case we are using a square lattice in two dimensions, although the model can be defined in principle for any lattice in any number of dimensions.

The spins themselves, in this simple model, are restricted to point in only two directions, up and down. Mathematically the spins are represented by variables $s_i = \pm 1$ on the points of the lattice, $+1$ for up-pointing spins and -1 for down-pointing ones. Dipoles in real magnets can typically point in any spatial direction, not just up or down, but the Ising model, with its restriction to just the two directions, captures a lot of the important physics while being significantly simpler to understand.

Another important feature of many magnetic materials is that the individual dipoles in the material may interact magnetically in such a way that it is energetically favorable for them to line up in the same direction. The magnetic potential energy due to the interaction of two dipoles is proportional to their dot product, but in the Ising model this simplifies to just the product $s_i s_j$ for spins on sites i and j of the lattice, since the spins are one-dimensional scalars, not vectors. Then the actual energy of interaction is $-J s_i s_j$, where J is a positive interaction constant. The minus sign ensures that the interactions are *ferromagnetic*, meaning the energy is lower when dipoles are lined up. A ferromagnetic interaction implies that the material will magnetize if given the chance. (In some materials the interaction has the opposite sign so that the dipoles prefer to be antialigned. Such a material is said to be *antiferromagnetic*, but we will not look at the antiferromagnetic case here.)

Normally it is assumed that spins interact only with those that are immediately adjacent to them on the lattice, which gives a total energy for the entire system equal to

$$E = -J \sum_{\langle ij \rangle} s_i s_j \,,$$

where the notation $\langle ij \rangle$ indicates a sum over pairs i, j that are adjacent on the lattice. On the square lattice we use in this exercise each spin has four adjacent neighbors with which it interacts.

Write a program to perform a Markov chain Monte Carlo simulation of the Ising model on the square lattice for a system of 20×20 spins. You will need to set up variables to hold the value ± 1 of the spin on each lattice site, probably using a two-dimensional integer array, and then take the following steps.

a) First write a function to calculate the total energy of the system, as given by the equation above. That is, for a given array of values of the spins, go through every pair of adjacent spins and add up the contributions $s_i s_j$ from all of them, then multiply by $-J$. Hint 1: Each pair of spins contributes to the total energy only once.

b) Now use your function as the basis for a Metropolis-style simulation of the Ising model with $J = 1$ and temperature $T = 1$ in units where the Boltzmann constant k_B is also 1. Initially set the spin variables randomly to ± 1, so that on average about a half of them are up and a half down, giving a total magnetization of roughly zero. Then choose a spin at random, flip it, and calculate the new energy after it is flipped, and hence also the change in energy as a result of the flip. Then decide whether to accept the flip using the Metropolis acceptance formula. If the move is rejected you will have to flip the spin back to where it was. Otherwise you keep the flipped spin. Now repeat this process for many moves.

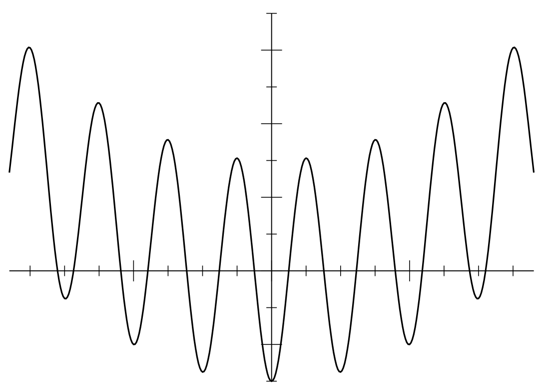
c) Make a plot of the total magnetization $M = \sum_i s_i$ of the system as a function of time for a million Monte Carlo steps. You should see that the system develops a "spontaneous magnetization," a nonzero value of the overall magnetization. Hint: While you are working on your program, do shorter runs, of maybe ten thousand steps at a time. Once you have it working properly, do a longer run of a million steps to get the final results.

d) Run your program several times and observe the sign of the magnetization that develops, positive or negative. Describe what you find and give a brief explanation of what is happening.

e) Make a second version of your program that produces the a color plot of the final snapshot of the system with different colors for plus or minus spin. Run it with temperature $T = 1$ and observe the behavior of the system. Then run it two further times at temperatures $T = 2$ and $T = 3$. Explain briefly what you see in your three runs. How and why does the behavior of the system change as temperature is increased?

Exercise 8: Global minimum of a function

Consider the function $f(x) = x^2 - \cos 4\pi x$, which looks like this:



Clearly the global minimum of this function is at $x = 0$.

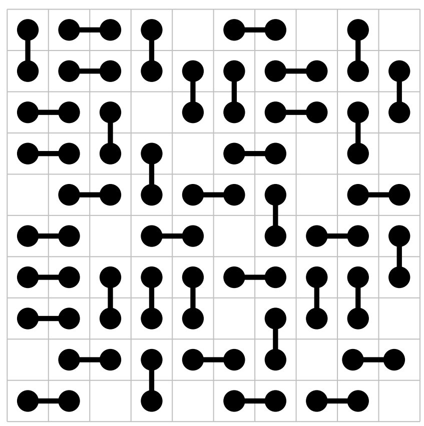
a) Write a program to confirm this fact using simulated annealing starting at, say, $x = 2$, with Monte Carlo moves of the form $x \to x + \delta$ where δ is a random number drawn from a Gaussian distribution with mean zero and standard deviation one. Use an exponential cooling schedule and adjust the start and end temperatures, as well as the exponential constant, until you find values that give good answers in reasonable time. Have your program make a plot of the values of x as a function of time during the run and have it print out the final value of x at the end. You will find the plot easier to interpret if you make it using dots rather than lines, with a statement of the form `plot(x,".")` or similar.

b) Now adapt your program to find the minimum of the more complicated function $f(x) = \cos x + \cos \sqrt{2}x + \cos \sqrt{3}x$ in the range $0 < x < 50$.

Hint: The correct answer for part (b) is around $x = 16$, but there are also competing minima around $x = 2$ and $x = 42$ that your program might find. In real-world situations, it is often good enough to find any reasonable solution to a problem, not necessarily the absolute best, so the fact that the program sometimes settles on these other solutions is not necessarily a bad thing.

Exercise 9: The dimer covering problem

A well studied problem in condensed matter physics is the *dimer covering problem* in which dimers, meaning polymers with only two atoms, land on the surface of a solid, falling in the spaces between the atoms on the surface and forming a grid like this:



No two dimers are allowed to overlap. The question is how many dimers we can fit in the entire $L \times L$ square. The answer, in this simple case, is clearly $\frac{1}{2}L \times L$, but suppose we did not know this. (There are more complicated versions of the problem on different lattices, or with differently shaped elements, for which the best solution is far from obvious, or in some cases not known at all.)

a) Write a program to solve the problem using simulated annealing on a 50×50 lattice. The "energy" function for the system is *minus* the number of dimers, so that it is minimized when the dimers are a maximum. The moves for the Markov chain are as follows:

   i) Choose two adjacent sites on the lattice at random.
   ii) If those two sites are currently occupied by a single dimer, remove the dimer from the lattice.
   iii) If they are currently both empty, add a dimer.
   iv) Otherwise, do nothing.

Perform an animation of the state of the system over time as the simulation runs.

b) Try exponential cooling schedules with different time constants. A reasonable first value to try is $\tau = 10\,000$ steps. For faster cooling schedules you should see that the solutions found are poorer—a smaller fraction of the lattice is filled with dimers and there are larger holes in between them—but for slower schedules the calculation can find quite good, but usually not perfect, coverings of the lattice.

Exercise 10: A random point on the surface of the Earth

Suppose you wish to choose a random point on the surface of the Earth. That is, you want to choose a value of the latitude and longitude such 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:

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

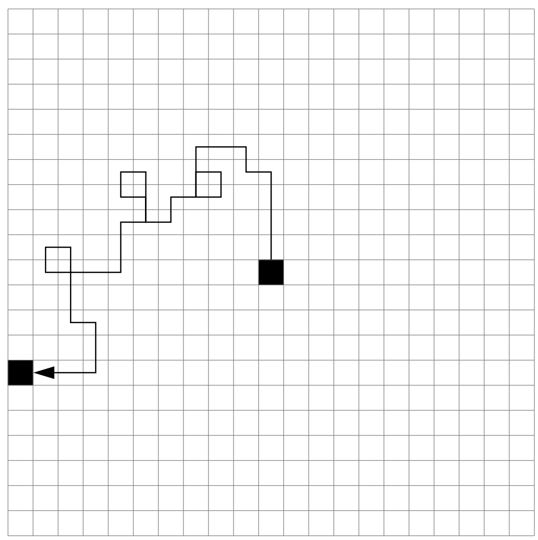
a) What are the ranges of the variables θ and ϕ? Verify that the two distributions $p(\theta)$ and $p(\phi)$ are correctly normalized—they integrate to 1 over the appropriate ranges.

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.)

Exercise 11: Diffusion-limited aggregation

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. There are various versions of DLA, 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 from square to square on the grid 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 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 Exercise 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—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.

   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 outside the edge of the system 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) 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.

   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) Now, 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 forever—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 chosen random point doesn't fall precisely on 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 one, 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.