PHYS4070/PHYS7270 - Due date: Tuesday 30/4/2024, 5:00pm

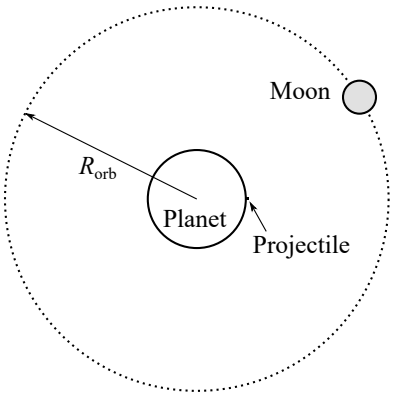
Project 2 - ODEs and Markov chain Monte Carlo

1 Numerical solution of a simple n-body problem

System setup

We will consider a system consisting of a planet and moon, interacting via gravity. We will numerically model the dynamics of a projectile launched from the planet to the moon using fourth-order Runge–Kutta (RK4) integration.

A diagram of the system is shown below:



We will work in units where the planet's mass m_{pl} and radius R_{pl} are both set to 1, and the gravitational constant $G = 1$. The moon and projectile masses and radii are:

Property	Planet	Moon	Projectile
Mass	1	0.1	10^{-24}
Radius	1	0.25	10^{-6}

The moon orbits around the planet at a constant radius of $R_{\text{orb}} = 19\,R_{\text{pl}}$.

Our system of equations will, in general, have the form:

$$m_i \ddot{\mathbf{r}}_i = \sum_{j \neq i} \frac{m_i m_j}{|\mathbf{r}_{ij}|^2} \hat{\mathbf{r}}_{ij}, \tag{1}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, and $\mathbf{r}_i = (x_i, y_i)$ (i.e. we assume that all the dynamics are restricted to the xy-plane). The energy of the system is given by:

$$E = \frac{1}{2} \sum_i m_i |\dot{\mathbf{r}}_i|^2 - \sum_i \sum_{j<i} \frac{m_i m_j}{|\mathbf{r}_{ij}|}. \tag{2}$$

Our co-ordinate system will be centred on the planet. For simplicity, we will assume that the planet is fixed in place, and hence its acceleration can be assumed to be zero for all time.

Tasks

Part 1.1 (20%)—two-body approximation:
We want to consider a projectile, launched from the surface of the planet towards the moon at some (purely radial) velocity $v_{\rm pr}$. As a first approximation, we will start by assuming that the moon does not influence the trajectory of the projectile once it has been launched.

1. Analytically calculate the initial velocity you would need to launch the projectile from the surface of the planet, such that it has zero velocity exactly as it reaches the surface of the moon (you can consider the moon to be fixed in place here). Hint: energy is conserved in this system.

2. Now we will simulate the trajectory using the RK4 method (note: you are welcome to build off the code provided in the workshop). Start your simulation with the projectile on the surface of the planet, and launch it at your calculated velocity $v_{\rm pr}$.

   (a) How long, to three significant figures in your dimensionless units, does the particle take to get to the surface of the moon in your simulation?

   (b) To make sure you can trust your simulation, perform a consistency check by comparing your results with known constants of motion for this system.

   (c) Based on the time taken for the projectile to reach the moon's surface, what initial condition, $\{\mathbf{r}_{\rm moon}, \dot{\mathbf{r}}_{\rm moon}\}$, should you set for the moon such that it passes by the projectile at exactly the time that the projectile's velocity reaches zero?

   (d) Run your code again with the moon's new initial condition. Plot the distance $r_{\rm mp} = |\mathbf{r}_{\rm moon} - \mathbf{r}_{\rm proj}|$ between the moon and the projectile as a function of time, and confirm that the minimum value is what you expect it to be.

   (e) Plot the trajectories of both the moon and the projectile, in the planet-centred coordinates. Note: you will want to set your x and y axes to equal scales, otherwise your circular orbit might look elliptical. For gnuplot users, you can use the command '`set size ratio -1`'.
   For python users, you can use:
   '`ax.set_aspect('equal')`' (for the figure handle `ax`—you can select the current figure with '`ax = plt.gca()`').

Part 1.2 (20%)—three-body problem:
We will now generalise the simulation to include the gravitational interaction between the moon and the projectile. The goal will be to perform a simplistic 'orbital transfer', such that the projectile ends up in orbit around the moon. This will require the projectile to be instantaneously 'kicked' once it becomes close enough to the moon. This kick models thrust produced by the projectile (e.g. a spacecraft) as it is being maneuvered into orbit around the moon.

1. Write out the equations of motion with this new moon–projectile interaction included. State any assumptions that you have made.

2. In a separate C++ file, build on your code from Part 1.1 to include the moon–projectile gravitational interaction. Confirm that your simulation is giving reasonable results, as you did in question 2(b) of Part 1.1. *Important:* make sure you update your whole system of equations *simultaneously*. The positions and velocities of each object at time $t + \Delta t$ should depend only on the positions and velocities at time t.

3. Run your new code for the parameters and initial conditions used in question 2 of Part 1.1.

Plot the new trajectory of the projectile. What has changed compared to its trajectory in Part 1.1? How close is the 'linearised' approximation of this dynamical system in Part 1.1 to this nonlinear calculation?

4. Once again, plot the distance $r_{\rm mp}(t)$ between the moon and the projectile. At what time does this separation first drop to a value of $r_{\rm mp} = 0.5R_{\rm pl}$ (i.e. twice the radius of the moon)?

5. Analytically determine the both the magnitude and direction of a 'kick' velocity $\Delta\mathbf{v}$ that would need to be instantaneously applied to the projectile, $\mathbf{v}_{\rm proj} \to \mathbf{v}'_{\rm proj} = \mathbf{v}_{\rm proj} + \Delta\mathbf{v}$, such that it enters a circular orbit around the moon at its current radius $r_{\rm mp}$ (assume that $r_{\rm mp}$ is small enough that we can neglect the gravitational effects of the central planet). Your answer should be expressed in terms of (i) the radial and angular separation of the moon and projectile ($r_{\rm mp}$ and $\theta_{\rm mp} = \arctan[(y_{\rm moon} - y_{\rm proj})/(x_{\rm moon} - x_{\rm proj})]$, respectively), (ii) the current (pre-kick) projectile velocity $\mathbf{v}_{\rm proj}$, and (iii) the current velocity of the moon $\mathbf{v}_{\rm moon}$. It might help to draw a diagram.

6. In your code, add in this instantaneous 'kick' velocity to your projectile at the time found in question 4 such that its motion is converted into a circular orbit around the moon at a radius of $0.5R_{\rm pl}$ for all later times.

   Run the code for a whole orbit of the moon around the central planet (following the kick) and plot the trajectories of both the moon and the particle in the planet-centred coordinates. Also plot the projectile's motion in moon-centred coordinates, and check how circular its trajectory is after the kick. As in Part 1.1, make sure to set the x and y axes to equal scales for these plots.

2 Markov chain Monte Carlo for the 2D Ising model

Background

We will consider the 2D Ising model on a square lattice, with $N_\mathrm{s} = L \times L$ sites. The Ising Hamiltonian is given by:

$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - \mu B \sum_{i}^{N_\mathrm{s}} \sigma_i, \tag{3}$$

where $\sigma_i = \pm 1$ is the spin at site i, J is the spin–spin interaction strength, μ is the magnetic moment of each spin, and B is the applied magnetic field strength. The notation $\langle i, j \rangle$ denotes a sum over pairs of nearest neighbours.

For the purposes of this assignment, we will take $B = 0$. We will also fix $J = 1$, which is equivalent to measuring energy in units of J. We will use periodic boundary conditions, such that the nearest-neighbour interactions 'wrap around' the boundaries, as shown in Fig. 1.

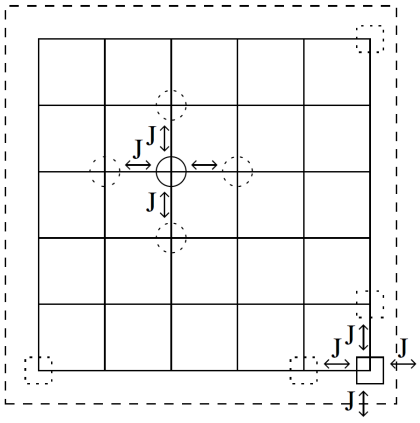


Figure 1: 2D lattice with periodic boundary conditions in both directions. Note that each interaction between two nearest neighbours is only counted once.

In the thermodynamic limit ($L \to \infty$), there is a critical point at temperature

$$\frac{k_\mathrm{B} T_\mathrm{c}}{J} = \frac{2}{\ln(1 + \sqrt{2})} \approx 2.27, \tag{4}$$

where k_B is the Boltzmann constant, which we will take to be 1. This critical point corresponds to a second order phase transition between a ferromagnetic phase (for $T < T_\mathrm{c}$) and an unmagnetised phase (for $T > T_\mathrm{c}$).

The 2D Ising model on a square lattice can be solved exactly. The exact expression for the absolute magnetisation per spin $|M| = |\mathcal{M}|/N_\mathrm{s} = |\sum_i \sigma_i|/N_\mathrm{s}$ is

$$|M| = \left[1 - \sinh\left(\frac{2J}{k_\mathrm{B} T} \right)^{-4} \right]^{1/8} \tag{5}$$

for $T < T_\mathrm{c}$, and $M = 0$ for $T > T_\mathrm{c}$.

Tasks

Building on the code you have developed in the first two worksheets for this part of the course, write a Markov chain Monte Carlo code that calculates the equilibrium properties of the Ising model on a 2D square lattice. To obtain reasonable estimates of thermodynamic quantities, you will need to perform $\gtrsim 1000$ sweeps of the lattice (following a suitable burn-in time), where one sweep corresponds to N_s attempted (but not necessarily accepted) flips. The sweeps can be done by either attempting to flip spins sequentially ($i = 0, 1, 2, 3, ..., N_\mathrm{s} - 1$) or at random. It is recommended to only save your outputs of observables once per sweep. A lattice of at least 16×16 spins is also recommended.

Ensure your submitted code is written such that:

- A particular seed for the random number generator is chosen at the beginning.
- The final state of each temperature is output in the terminal, in a form like this:

```
O - O O
- - - O
O - O O
- O - -
```

  where "O"=spin up, "-"=spin down.

- Preferably, you should perform your loops (over temperatures, then over sweeps, then over the lattice) all in a single C++ script.

Some tips:

- To track the energy per spin $E = \mathcal{E}/N_\mathrm{s} = H/N_\mathrm{s}$ and magnetisation per spin $M = \mathcal{M}/N_\mathrm{s}$ over the course of the simulation, you don't need to calculate them 'from scratch' (using the whole system) each Monte Carlo step. Instead, calculate $\mathcal{E}$ and $\mathcal{M}$ at the beginning of your temperature run, and then track the changes $\Delta\mathcal{E}$ and $\Delta\mathcal{M}$ for each accepted step, and add these to a running total, as described in the second worksheet for this topic. This should save a lot of computation time.
- It is a good idea to loop over temperatures in order (e.g. smallest to largest), so that you can use the final state of the previous temperature as the initial state for the next temperature.
- If you use a randomised configuration of spins as your initial state, make sure your chosen seed for the random number generator produces states that avoid the $\approx 50\%$ up, $\approx 50\%$ down state that can occur at low energies. This state is a local (rather than global) minimum in the free energy, and appears quite often in small lattices.
- Note that you will need to run your code longer for temperatures near T_c, due to critical slowing down (technically, you should also increase the time between outputs to avoid taking correlated samples, but in the small systems we're considering here, this isn't a significant issue).

Part 2.1 (20%)—thermodynamic properties of the 2D Ising model:
Perform a sweep over ≈ 20 temperatures in the range $0 < T < 5$, and plot as a function of T:

1. The mean energy per spin, $\langle E \rangle$, with vertical error bars corresponding to the standard error of the mean (SEM) $\sqrt{\mathrm{var}(E)/\mathcal{N}}$, where $\mathcal{N}$ is the number of samples you have

averaged the energy over. Recall that the variance of a variable X is given by $\mathrm{var}(X) = \langle X^2 \rangle - \langle X \rangle^2$.

2. The mean magnetisation per spin, $\langle M \rangle = \langle (1/N_\mathrm{s}) \sum_i \sigma_i \rangle$, with vertical error bars corresponding to its SEM $\sqrt{\mathrm{var}(M)/\mathcal{N}}$. In the same figure, plot the exact expression for $|M|$ on the same axis as your numerical result (if they don't agree, something has gone wrong!). Note that you only expect $|M|$ to agree—the sign could be flipped.

3. The specific heat per spin, $c = \mathrm{var}(E) N_\mathrm{s}/(k_\mathrm{B} T^2)$.

4. The magnetic susceptibility per spin, $\chi = \mathrm{var}(|M|) N_\mathrm{s}/(k_\mathrm{B} T)$.

Describe any important features in each of these plots. What is happening to the spin configuration as you vary T?

Part 2.2 (20%)—critical scaling behaviour of the 2D Ising model:
Repeat the calculations of $c(T)$ and $\chi(T)$ for three more grid sizes L, each differing by factors of 2 (e.g. if you used $L = 16$, you could run again with $L = 8$, $L = 32$, and $L = 64$).

1. Plot the $c(T)$ data for your different L values in a single figure. Do the same for $\chi(T)$. The susceptibility might look cleaner if you use a log scale for the vertical axis. Make sure you have enough samples for these plots to look smooth.

   What do these plots tell us? What do you expect to happen if you keep increasing the system size?

2. It can be shown that near the critical point the specific heat and magnetic susceptibility obey so-called *finite-size scaling* laws:

$$c(t) = \log(L) f\left(L^{1/\nu} t\right), \qquad \chi(t) = L^{\gamma/\nu} g\left(L^{1/\nu} t\right), \tag{6}$$

   where $t = (T - T_\mathrm{c})/T_\mathrm{c}$ is the reduced temperature, and the critical exponents for the 2D Ising model are $\nu = 1$ and $\gamma = 7/4$. The functions f and g are unknown, but importantly they are *independent of the system size*.

   By rescaling the x- and y-axes of your $c(t)$ and $\chi(t)$ plots appropriately, demonstrate that each of these datasets from your four system sizes L collapse onto a single curve (i.e. f or g) and therefore obey these finite-size scaling laws. Note that the collapse will fail away from $t = 0$. You may need to run for longer in larger lattices to get clean data near T_c.

   Based on these results, is it ever possible to reproduce the behaviour of the true thermodynamic limit using a finite simulation? What is changing as you increase the system size L? What additional information are you gaining?

3. In the thermodynamic limit, the magnetisation per spin M and the magnetic susceptibility per spin χ are expected to scale as power-laws with the relative temperature:

$$M \sim |T - T_\mathrm{c}|^{\beta}, \qquad \chi \sim |T - T_\mathrm{c}|^{-\gamma}, \tag{7}$$

   with critical exponents $\beta = 1/8$ and $\gamma = 7/4$ in two dimensions. The magnetisation scaling only applies for $T < T_\mathrm{c}$, while the susceptibility scaling applies both above and below T_c.

   Using your code, run ≈ 10 temperatures in the range $2 < T < T_\mathrm{c}$, and reproduce the above two power-laws. The most straightforward way to check this is to plot M (or χ) vs. $|T - T_\mathrm{c}|$ on a log–log scale, in which case a power-law will look like a straight line with slope equal to the exponent. Plot the expected power-law on the same plot for

comparison. You might find that you need a large lattice ($L \geq 32$) and a lot of sweeps to reproduce these power-laws.

What do these power-laws reflect is happening in the system, physically?

Part 2.3 (20%)—OpenMP implementation:
Parallelise your 2D Ising model code to run on multiple threads using OpenMP. The details of how to do this will be given in week 9's worksheet.

Once you have parallelised the code, benchmark it for $N_{\text{thread}} = \{1, 2, 4, 8\}$ (or as many as you can on your own computer) by measuring how long it takes to run in each case. Plot the run time as a function of N_{thread}, and describe what you see. (You don't have to output any Monte Carlo data for this part, so you can just pick a single temperature to run at.) For a fair comparison, ensure that all important parameters are fixed across the different runs (temperature, number of sweeps etc.). Note that you may find you get a better (relative) speed-up for increasing N_{thread} if you use a larger lattice of spins.

Note: Part 2.3 will be graded differently to the other parts of this assignment (see the marking guide below).

3 Format and grading

You should submit your assignment as a combination of (i) a written report, and (ii) any scripts (C++, gnuplot, python, bash etc) used to compile, run the simulations, and plot the results. All code should be written to compile and run as-is.

3.1 Report structure

Your report should be split into two sections (one for ODEs, one for Monte Carlo). In each half, you may want to write a short introduction outlining the problem and the main goals, as well as a short conclusion summarising what you have done and what you have learned. However, no references are needed.

Make sure to include:

1. All the results, plots and answers the questions ask you to produce, including any important steps from your working.
2. A description and explanation of everything you've done to obtain your results/data/figures.
3. Interpretation and analysis of your results.

3.2 Marking guide

Within each of the five parts (each worth 20% of the overall assignment), your report will be marked on the following criteria:

1. **Correctness (30%):** Are your answers to the questions correct? Did you produce all of the plots that were asked for? Is your code correct, or are there bugs?
2. **Explanation (30%):** Did you describe all your steps? Have you described the data shown in your figures? Does the writing flow logically, or are steps missing?
3. **Interpretation and insight (30%):** Did you demonstrate an understanding of your simulation/results? Did you provide any physical interpretation? Did you identify/discuss any interesting/unexpected features of your results?
4. **Code quality (10%):** Is your code well-written? Is it easy to understand? Is it commented, where necessary?

Note: For part 2.3 (the OpenMP implementation), these criteria will instead be weighted as $40\% + 10\% + 10\% + 40\%$, respectively.

4 Submission

You should submit your assignment on Blackboard. Your submission should include:

1. A pdf of your report, with filename 'surname-studentid.pdf'.
2. All C++ scripts you used to obtain your results, clearly labelled as e.g. 'part1_1.cpp', 'part1_2.cpp', 'part2_1.cpp', 'part2_2.cpp', 'part2_3.cpp'.
3. Your plotting scripts—either gnuplot or python scripts (or any other plotting program you might have used).

Preferably, you should submit these as a compressed file (e.g. .zip).