PHYS4070 Project 2: ODEs & Markov chain Monte Carlo

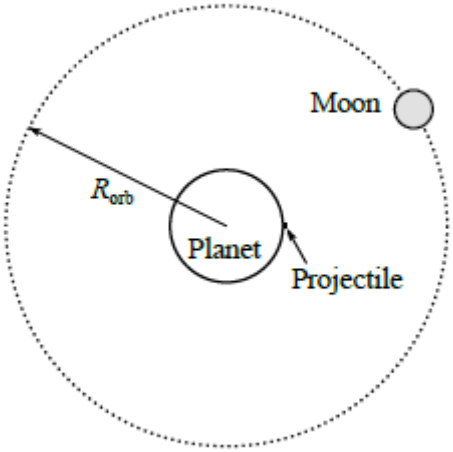
Semester 1 2023, 2 May

Hugh McDougall, 4302007

Contents

Part 1 - Numerical Solution of a Simple N-Body Problem

In this part of the project, we will use forward numerical integration via the Runge-Kutta method to simulate simple orbital mechanics around a fixed mass. The system to be modelled is a moon and planet of finite size with an arbitrarily small projectile being launched between them in a coplanar orbit.



Orbital System

Simplifications across all simulations include:

- No rotational effects on any of the bodies (the planet is considered to be static)
- Ignoring any extra-planetary effects (solar gravity etc)
- No atmospheric effects about the moon or planet
- The planet being arbitrarily heavy such that it may be held at a fixed position
- All bodies being treated as perfectly smooth spheres
- The time of all manoeuvres performed by the projectile (launch, orbital burns, rotations etc) are arbitrarily fast.

Part 1.1 – The Two Body Approximation

In this section we neglect the effect of the moon's gravity, treating the orbits of the moon and projectile as being independent paths under the influence of the planet's static gravitational potential. In this section, our goals are to:

- Validate our numerical integration scheme
- Estimate the launch properties (launch time, velocity) required to encounter the moon

Part 1.1.1 – Analytical Approximations

As a first order approximation, we can ignore the effect of the moon's pull on the projectile to estimate the launch velocity required to reach the moon's orbital radius at rest. Under

this simplification, and assuming $m_{planet} \gg m_{proj}$, the projectile's energy is conserved, making it a simple matter to estimate the purely radial velocity:

$$\Delta E = 0$$
$$m_{proj}\left(\frac{v_0^2}{2} - \frac{\mu M_{planet}}{R_{planet}}\right) = m_{proj}\left(\frac{0}{2} - \frac{\mu M_{planet}}{r_{final}}\right), \qquad r_{final} = r_{moon} - R_{moon}$$

Where lowercase 'r' indicates orbital position and capital 'R' is the size of the object. Using natural units and noting $M_{planet} = 1$:

$$v_0 = \sqrt{2\left(\frac{1}{R_{planet}} - \frac{1}{r_{moon} - R_{moon}}\right)}$$

For $R_{planet} = 1$, $R_{moon} = 0.25$ and $r_{moon} = 19$, this gives:

$$v_0 \approx 1.376$$

This gives a flight time on the order of 10's of time units:

$$t = \frac{r_{moon} - R_{planet}}{r_0} \approx 13$$

> *Note: the analytical solution to the flight time, found by integrating* $\frac{dt}{dr} = \frac{1}{v(r)}$*, comes to* ≈ 89.7 *time units, of similar order.*

We can also find the velocity required by the moon to maintain its stable circular orbit at $r_{moon} = 19\, R_{planet}$ by a simple force balance:

$$\frac{v_{moon}^2}{r_{moon}} = \frac{\mu M_{planet}}{r_{moon}^2}, \qquad \Rightarrow v_{moon} = \sqrt{\frac{1}{r_{moon}}} = \frac{1}{\sqrt{19}} \approx 0.229$$

Corresponding to an orbital period of:

$$T_{moon} = \frac{2\pi r_{moon}}{v_{moon}} = 520.4$$

For the projectile and moon's paths to meet at the correct time, this requires the moon to be $\theta_0 \approx -62.1^o$ behind the encounter at time of launch for a counter-clockwise orbit. The initial position and velocity of the moon are then, in (x, y) coordinates:

$$\vec{\mathrm{r}}_{moon}(\mathrm{t} = 0) = r_{moon}\begin{pmatrix}\cos(\theta_0)\\ \sin(\theta_0)\end{pmatrix}, \qquad \vec{\mathrm{v}}_{moon}(\mathrm{t} = 0) = v_{moon}\begin{pmatrix}-\sin(\theta_0)\\ \cos(\theta_0)\end{pmatrix}$$

Where $r_{moon} = 19$, $v_{moon} \approx 0.229$ and $\theta_0 \approx -62.1^o$. It is these values that we will validate with numerical integration in the next section.

Part 1.1.2 – Numerical Simulation of the Two Body Problem

Our goal is to simulate the kinematic behaviour of the orbital system using forward integration. To simplify our integration to a first-order problem, we describe the system in the form of a state vector $\vec{X}$, containing the positions and velocities of all bodies, and the derivative of which is given by a first order ODE:

$$\vec{X} = \begin{pmatrix} \vec{r}_i \\ \vdots \\ \vec{v}_i \\ \vdots \end{pmatrix}, \qquad \frac{d}{dt}\vec{X} = \begin{pmatrix} \vec{v}_i \\ \vdots \\ \vec{a}_i \\ \vdots \end{pmatrix} = f(\vec{X})$$

In a gravity-only system, the forces / accelerations depend only on the body positions:

$$\frac{d}{dt}\vec{r}_i = \vec{v}_i, \qquad \frac{d}{dt}\vec{v}_i = \vec{a}_i = \frac{1}{m_i}F_i(\vec{r}_1, \vec{r}_2 \ldots)$$

In this section, where we ignore interaction between the bodies, the forces depend only on each body's position relative to the planet, fixed at $(x, y) = (0,0)$:

$$\frac{d}{dt}\vec{v}_i = -\frac{\vec{r}_i}{|\vec{r}_i|^3}$$

Integration is then performed using the fourth order Runge Kutta method, a fixed step-size integrator which uses a nested series of estimates of $\frac{d}{dt}\vec{X} = f(\vec{X})$ to minimize truncation error in each step:

$$\vec{X}(t + \Delta t) = \vec{X}(t) + \left(\frac{\vec{k}_1}{3} + \frac{\vec{k}_2}{6} + \frac{\vec{k}_3}{6} + \frac{\vec{k}_4}{6}\right)\Delta t$$

$$\begin{aligned} \vec{k}_1 &= f(\vec{X}) \\ \vec{k}_2 &= f\left(\vec{X} + \frac{\Delta t}{2}\vec{k}_1\right) \\ \vec{k}_3 &= f\left(\vec{X} + \frac{\Delta t}{2}\vec{k}_2\right) \\ \vec{k}_4 &= f(\vec{X} + \Delta t\vec{k}_3) \end{aligned}$$

We choose this method for its simplicity and good scaling over more complicated adaptive step-size methods like RK45 or bounded error methods like symplectic leap-frog.

Required Step Size

We are using the fourth order Runge Kutta method, which has approximately fourth order global error. Taking the size of the projectile, $R_{proj} = 10^{-6}$, as a scale for significant position error, we can estimate a decent step size at $\Delta t \approx 0.03$:

$$10^{-6} \approx |\Delta r| \approx \Delta t^4, \qquad \Rightarrow \Delta t \approx 10^{-\frac{3}{2}} \approx 0.03$$

Which we round to $\Delta t = 0.01$ for neatness. For the simulation timescale estimated in part 1.1.1, this will require 1000's of iterations per simulation.

Code Structure

The numerical simulation is carried out in a c++ script, `simulation.cpp,` which contains the runtime code for both this section and section 1.2. Important inputs, such as whether to use the 2-body approximation, the maximum runtime and the output location for results can be changed via command line inputs. See the `readme.pdf` file provided with this report for details.

System parameters (e.g. sizes and masses of each body, desired orbital radii) and the analytical approximations from part 1 are stored in `parameter.cpp` file. The number and properties of the bodies provided is hardcoded, but the simulation code is set up to be as general as possible, i.e. to run the integration scheme for as many bodies as are provided. For a demonstration of this, see section 1.3.

The system vector is described as a cpp vector of vectors, and overloaded utilities for more easily handling these (e.g. addition and multiplication as required by the RK4 method) are stored in `sysvec_utils.cpp.` The integration step is calculated with the `runge_int()` function, while the $\frac{dX}{dt} = f(x)$ function is calculated by `f_planet()`. Both are stored in the `forces_and_integrators.cpp` file.

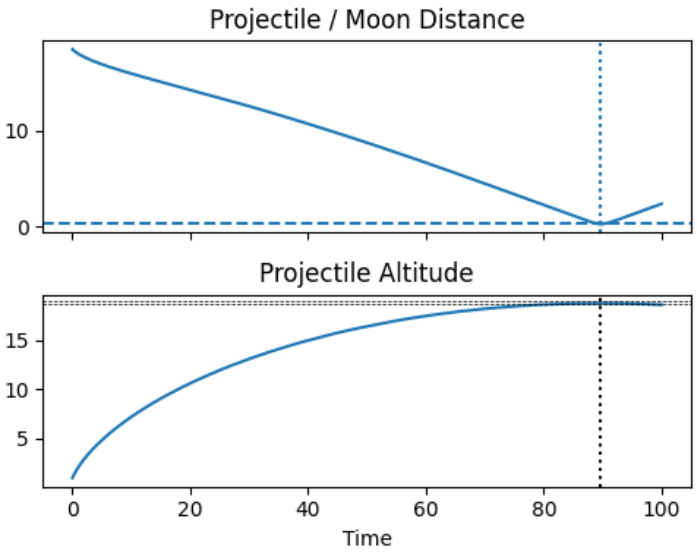
By default, all results are saved to the '`./results`' folder. The '`_compile_and_run.sh`' shell script should generate results for all parts, which contains python scripts (e.g. `_plot_1a.py`) for producing the plots in the following sections.

Results

Running our simulation for the given parameters up to $t = 100$, we find our results in close (within 1%) agreement with our predictions:

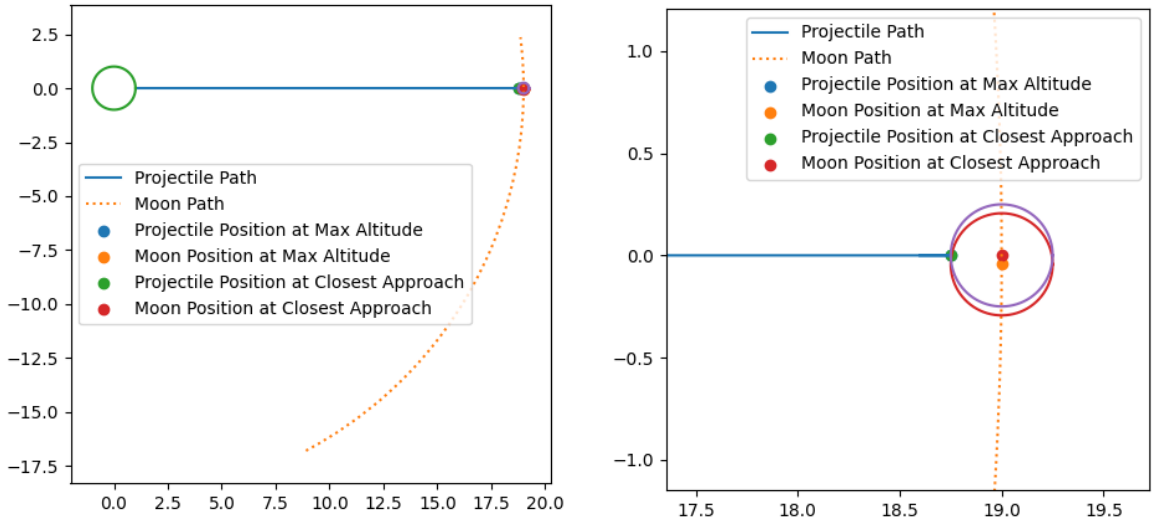
	Estimated	Simulated
Maximum Altitude	18.7	18.75
Time to Maximum Altitude	89.7	89.51
Closest Approach	0.25	0.25
Time to Closest Approach	89.7	89.70

Simulation Results



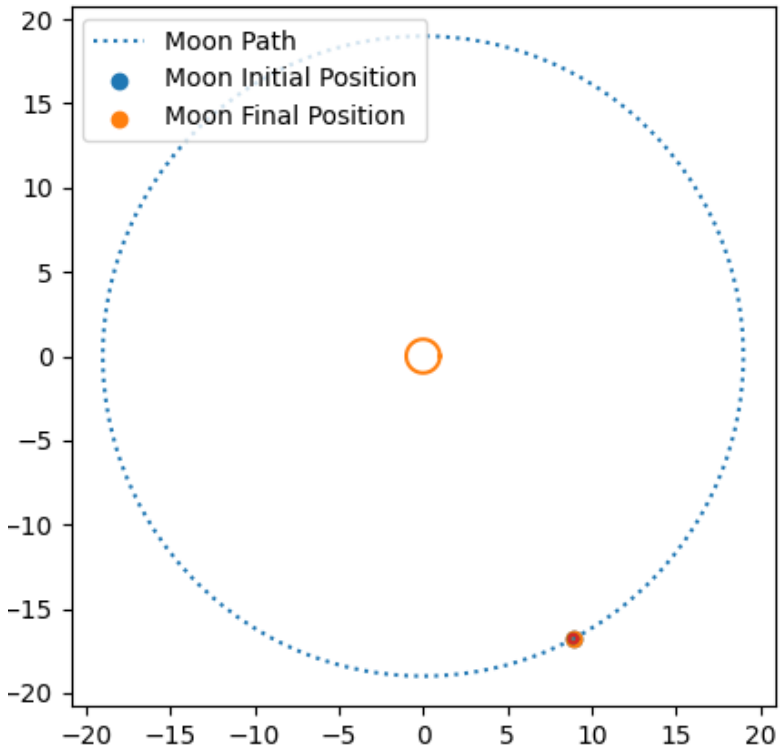
Projectile Distances Without Body-Body Interaction

For a perfect result, we would expect the point of closest approach to be the same as the point of maximum altitude, but truncation error leads to a small difference between the two:



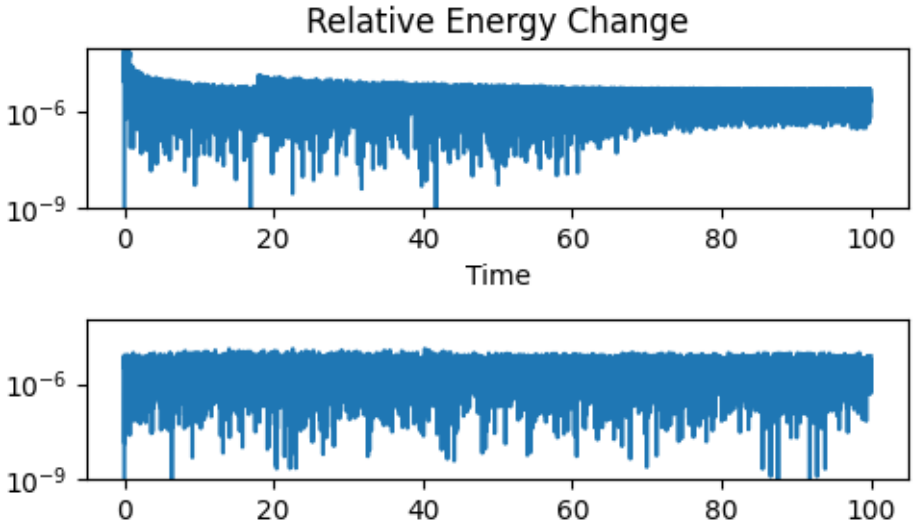
Orbital Paths Up to $t = 100$. Zoomed out (left) and zoomed in (right)

We can validate the accuracy of our simulation by simulating the moon's orbit for one full rotation, up to $T = 520.4$, a readily available analytical result. We can see the error in position is small.



Single Orbit of Moon

For a non-interacting system, energy is conserved for each body. We can confirm the accuracy of our simulation by noting that the fractional energy change of each object, $\frac{\Delta E}{E(t=0)}$, is small, only of order $\approx 10^{-6}$.



Fractional Energy Change for Projectile (Top) & Moon (Bottom)

Part 1.2 – Three Body Problem

In this section we add in the gravitational interaction between the bodies and model a simple two-stage orbital transfer into a circular orbit about the moon.

Part 1.2.1 – Equation of Motion

Defined as a first order system with a system state vector as in part 1.1, the equation of motion is the same as before:

$$\frac{d}{dt}(\vec{X}) = f(\vec{X})$$

But now with the force terms, stored in the second half of $\frac{d}{dt}(\vec{X})$, having an extra term depending on the positions of all other bodies:

$$\frac{d}{dt}(\vec{X})_{i>N} = \vec{a}_i = \frac{1}{m_i}F_i = -\frac{\vec{r}_i}{|\vec{r}_i|^3} + \sum_{i\neq j} m_j \frac{-\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|^3}$$

Part 1.2.2 – Code Structure

The new body-body interaction is calculated in function `f_nbody()`. To allow this function to scale more easily, this function takes a vector bools, LIVE, which tracks whether each body in the system is massive enough to warrant calculating the effects of its gravity. For the purposes of this section, both the moon and projectile are treated as live, even though the projectile contributes negligibly to the system behaviour.

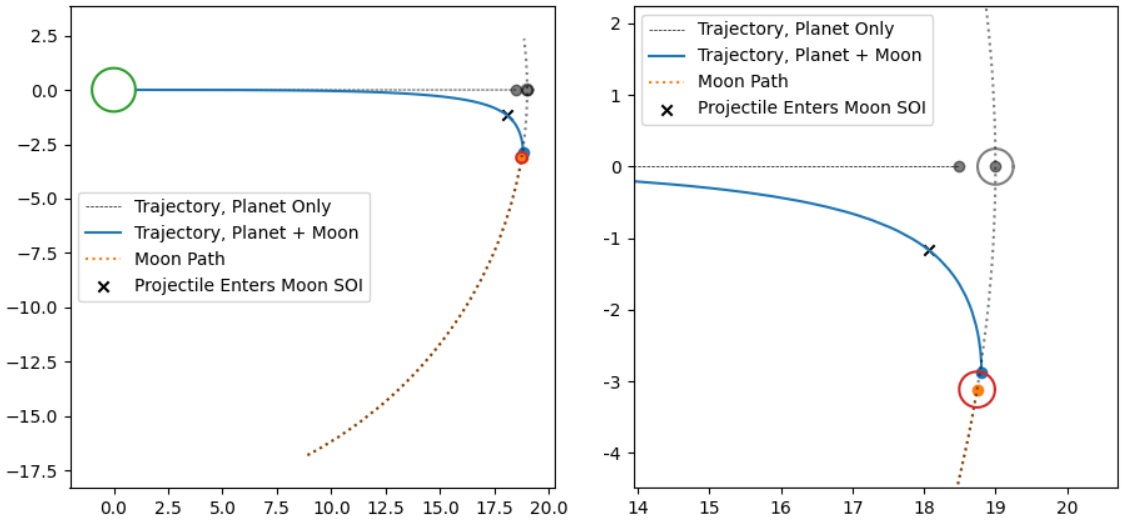
Because RK4 integration requires derivatives be in the form $\frac{d}{dt}\vec{X} = f(\vec{X})$, the at runtime this function is compacted into a single input single output function with:

```
f = [M,LIVE](sysvec X) { return f_nbody(X,M,LIVE);};
```

Part 1.2.3 – Trajectories with Moon Influence

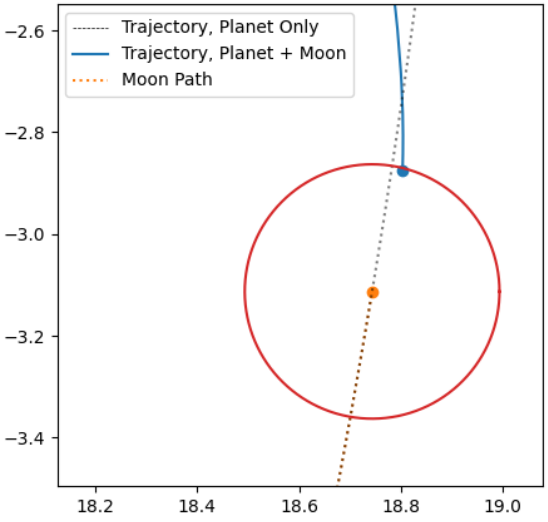
Including the pull of moon causes the projectile to veer- off its previously straight trajectory. The largest deviations occur after the projectile enters the moon's sphere of influence, where the moon's pull is stronger than that of the planet, a distance of $|\vec{r}_{proj} - \vec{r}_{moon}| \approx 4.5$.

The projectile entering the moon's potential well increases its velocity, shortening the time to impact from $t = 89.51$ to $t = 76.07$



Projectile Trajectories with Moon Gravity

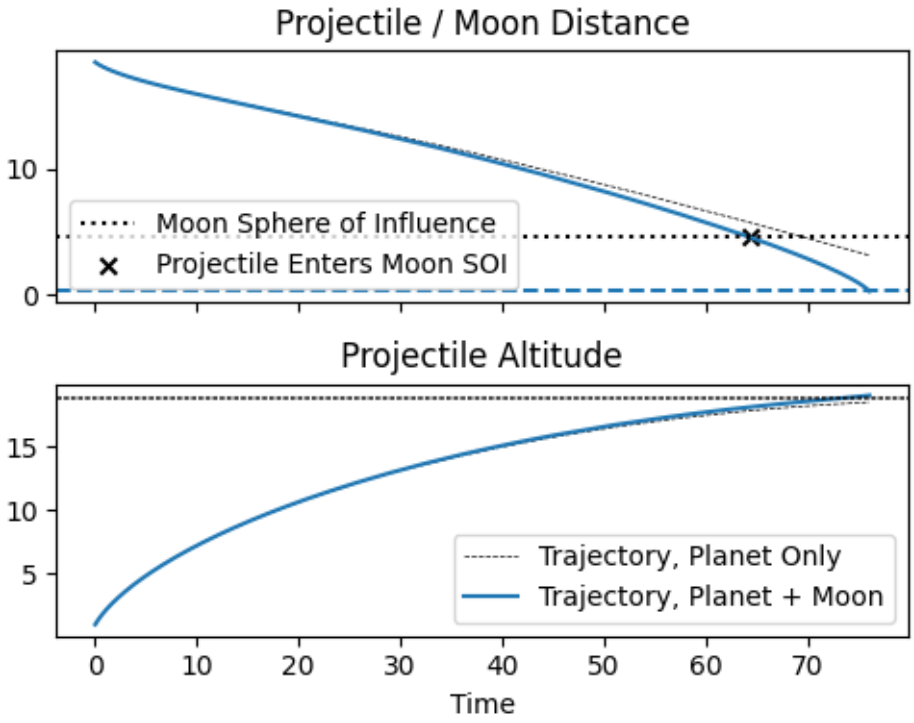
Another interesting feature is that the extra acceleration causes the projectile to 'slingshot' slightly beyond the moon's orbit, colliding while travelling almost anti-parallel to the moon's orbit.



Projectile Moon Impact Position with Moon Gravity

Part 1.2.4 – Impact & Altitude with Moon Influence

As seen with the trajectories, adding the influence of the moon causes use to approach the moon faster than the non-interacting case (top figure), and to slightly exceed the moons $r_{moon} = 19$ orbit (bottom figure).



Projectile Distances with Body-Body Interaction

Part 1.2.4 – Derivation of Orbital Kick

To break into a circular orbit about the moon, we need to apply an (approximately) instantaneous kick of delta-V when we reach the desired distance of $|\vec{r}_{proj} - \vec{r}_{moon}| = 2R_{moon}$. To maintain a circular orbit, we need a relative velocity, $|v_{orb}|$ such that the centrifugal force counteracts the moons gravity:

$$\frac{|v_{orb}|^2}{|\vec{r}_{proj} - \vec{r}_{moon}|} = \frac{m_{moon}}{|\vec{r}_{proj} - \vec{r}_{moon}|^2}$$

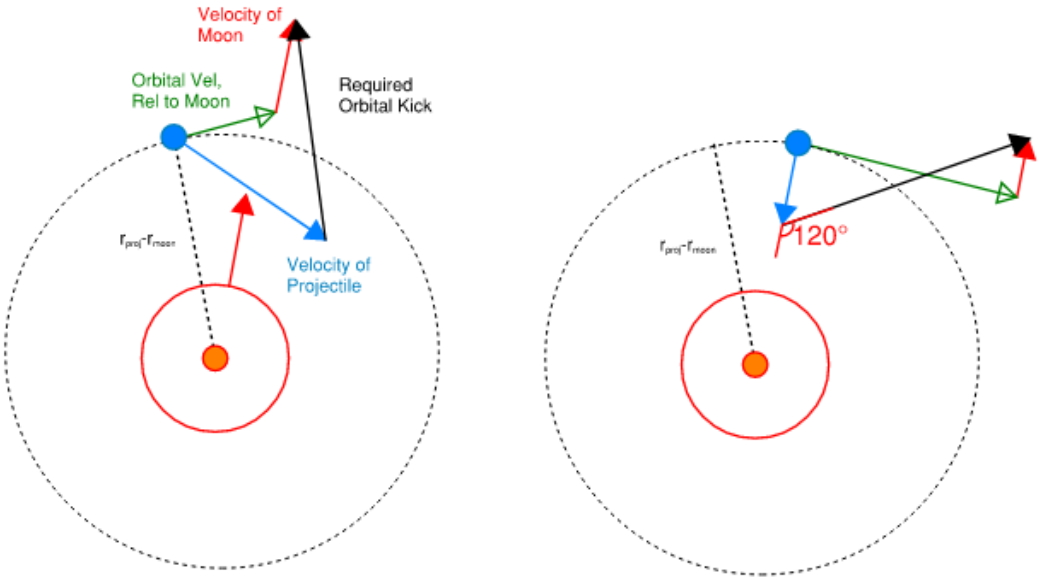
$$|v_{orb}| = \sqrt{\frac{m_{moon}}{|\vec{r}_{proj} - \vec{r}_{moon}|}}$$

We're looking to orbit at $|\vec{r}_{proj} - \vec{r}_{moon}| = 2R_{moon} = 0.5$ for required orbital velocity:

$$|v_{orb}| = \sqrt{\frac{m_{moon}}{2R_{moon}}} = \sqrt{\frac{0.1}{0.5}} \approx 0.447$$

This velocity needs to be pointing perpendicular to the radial to the moon:

$$\vec{v}_{orb} \perp \vec{r}_{proj} - \vec{r}_{moon}, \qquad \Rightarrow \vec{v}_{orb} = \pm\sqrt{\frac{m_{moon}}{|\vec{r}_{proj} - \vec{r}_{moon}|}}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\frac{\vec{r}_{proj} - \vec{r}_{moon}}{|\vec{r}_{proj} - \vec{r}_{moon}|}$$



Geometry of Orbital Kick, general (Left) & Approximate Analytical (Right)

The $\pm$ represents the fact that we can orbit either clockwise or counter clockwise about the moon. If we begin at a velocity $\vec{v}_0$ relative to the rest frame, and we need to achieve the above velocity relative to the moon, our total orbital kick is:

$$\begin{aligned}\vec{v}_{kick} &= (\vec{v}_{orb} + \vec{v}_{moon}) - \vec{v}_0 \\ &= [\vec{v}_{moon} - \vec{v}_0] \pm \sqrt{\frac{m_{moon}}{|\vec{r}_{proj} - \vec{r}_{moon}|}} \cdot \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\frac{\vec{r}_{proj} - \vec{r}_{moon}}{|\vec{r}_{proj} - \vec{r}_{moon}|}\end{aligned}$$

In our code, we test the positive and negative case and select whichever orbital kick requires the smaller delta-V, and apply this instantaneously at the first simulation step at which $|\vec{r}_{proj} - \vec{r}_{moon}| < 2R_{moon}$. Calculation of the kick is performed in `orbital_kick()` in the `forces_and_integrators.cpp` file.

Analytical Approximation

We know our energy is conserved since launch and that the moons gravity dominates when nearby, and so:

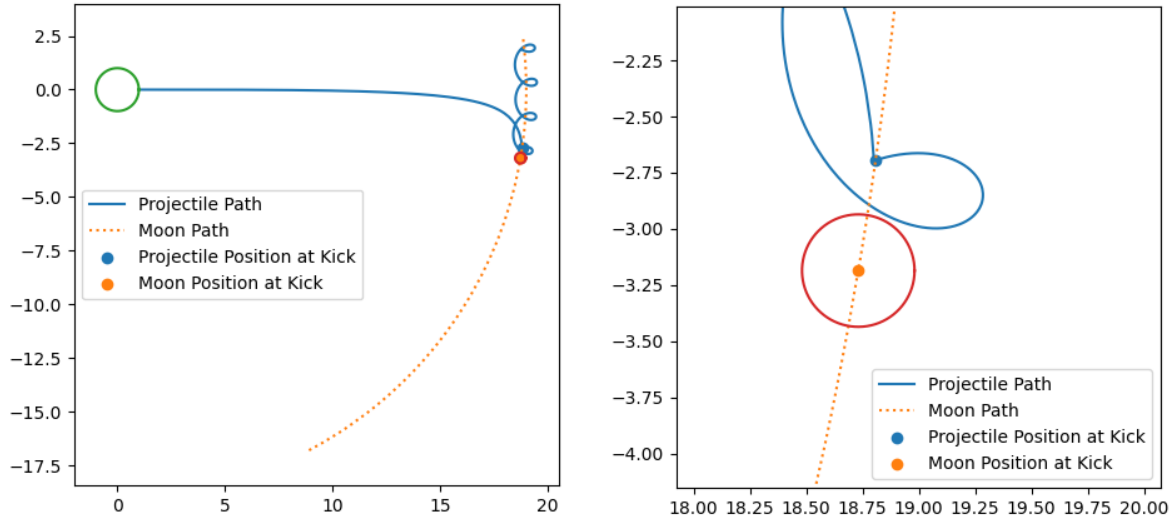
$$\frac{v_0^2}{2} - \frac{m_{moon}}{2R_{moon}} \approx E = \frac{v_{launch}^2}{2} - \frac{1}{R_{planet}}, \qquad \Rightarrow v_0 \approx \sqrt{v_{launch}^2 - \frac{2}{R_{planet}} + \frac{m_{moon}}{R_{moon}}} \approx 0.542$$

From the previous section, we know that we approach the moon almost radially at impact. The kick then needs to bring our radial speed in line with the moon and achieve orbit. As a first approximation, these two effects are at right angles:

$$\Delta\vec{v}_{kick} = \begin{pmatrix} |v_0| + |v_{moon}| \\ |v_{orb}| \end{pmatrix}, \qquad \Rightarrow |\Delta v_{kick}| \approx 0.891, \qquad \theta_{v_{kick}} \approx 120.1^o$$

This represents a 'worst case', and so we expect a kick of $|\Delta v_{kick}| \leq 0.891$

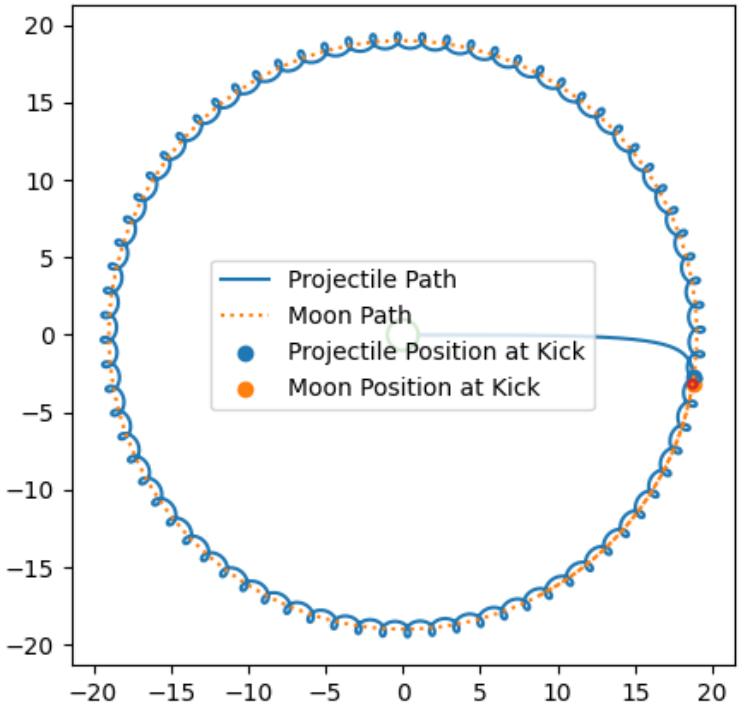
Part 1.2.5 – Orbital Paths with Kick



Orbital Trajectories to t=100 For Orbital Kick. Full (Left) & Zoomed (Right)

The magnitude and angle of the kick are in line with our estimates. Because we approach at a slight angle instead of dead-on, the required Δv is smaller, and the angle sharper as we focus more on decelerating and less on accelerating into the required tangential velocity.

	Expected	Simulated
Time of Kick	n/a	75.74
Magnitude of Kick	<0.891	0.757
Angle of Kick, Rel. To Prograde Dir.	$\approx 120.01^o$	138.74^o

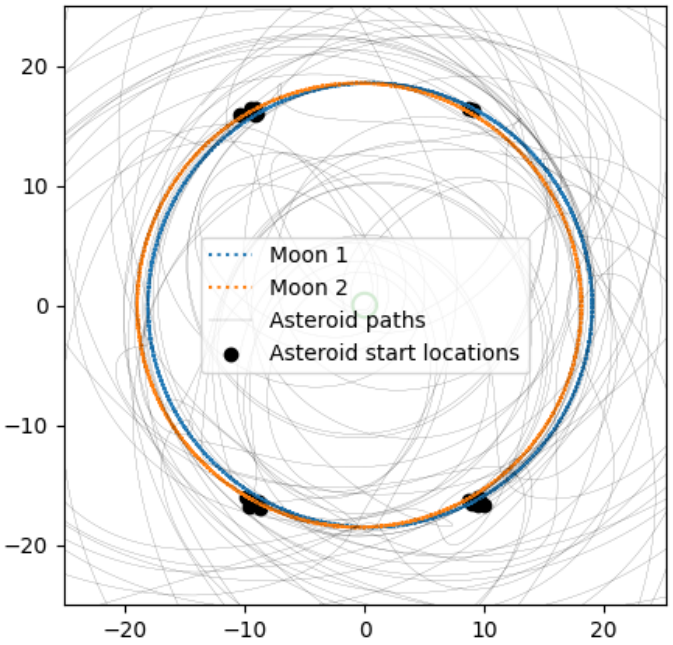


Trajectory Post-Kick for One Moon Orbit

Part 1.3 (Bonus) Many-Body Demonstration

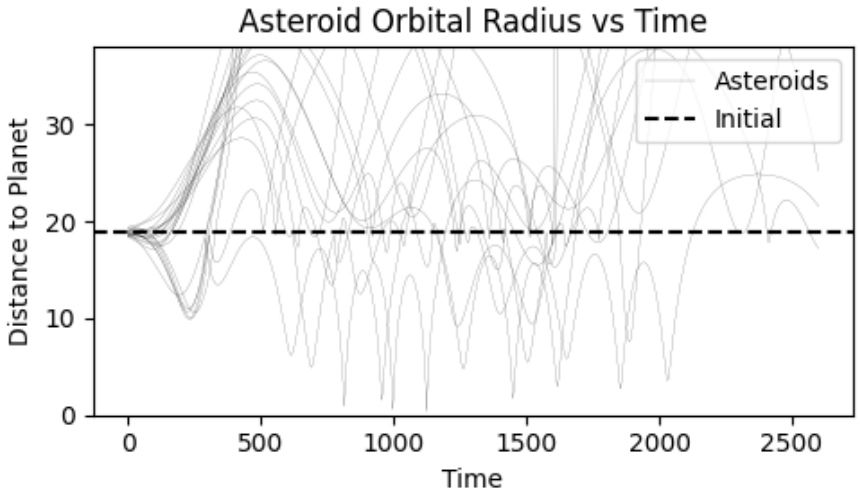
For the bulk of this report, we have only looked at the relatively simple case of two significant masses, but our method can be scale up to more complex systems with more chaotic behaviour. To demonstrate this, this section shows results for the same geometry, but now with two moons on opposite sides of their orbits and an array of asteroids beginning at each moon's Lagrange points ($\pm 60^o$ from the moon at the same orbital radius).

Parameters for this simulation are defined in `manybody_example.hpp`.



Orbital Paths in Many-Body System

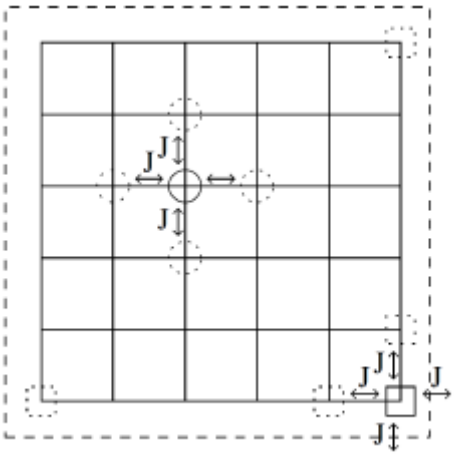
In a two-body system, the Lagrange points are stable equilibria, with the lateral pull of the moon and sun cancelling out against the asteroids' centrifugal motion. By adding a second moon, the stability is broken and the asteroids rapidly diverge into wildly varying orbits.



Asteroid Orbital Radii, Showing the Rapid Divergence Between Bodies of Similar Initial Conditions

Part 2 - Markov Chain Monte Carlo for the 2D Ising Model

In this section, we simulate the statistical properties of a 2D grid of spins, analogous to the magnetic spin-alignment that occurs in metals. The grid in question, if of side length 'L', has $N_s = L \times L$ spin sites, σ_i, in a square lattice that can be aligned either 'up' ($\sigma = +1$) or 'down' ($\sigma = -i$).



Grid Layout

This leads to a total energy 'E" and total spin 'M":

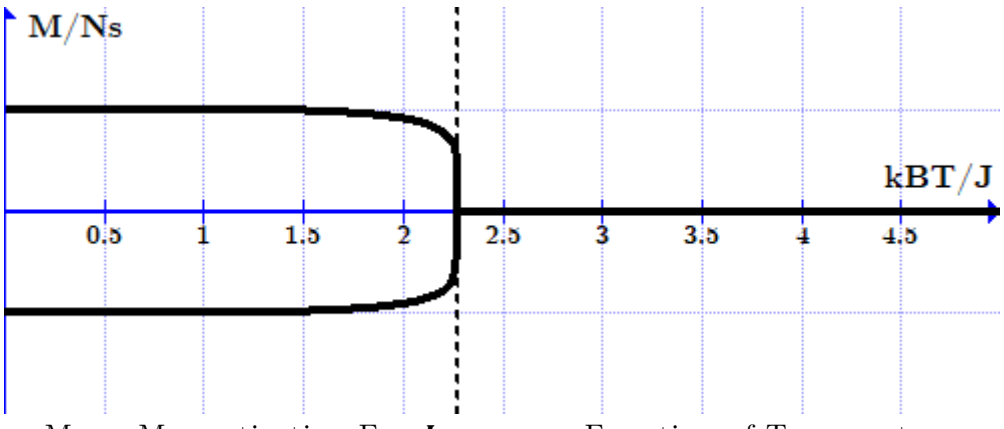
$$E = -J\sum_{\langle i,j\rangle}\sigma_i\sigma_j - \mu B\sum_i \sigma_i\,, \qquad M = \sum_i \sigma_i$$

Where $\langle i,j\rangle$ represents the set of all neighbouring pairs of points in the grid. To remove edge-effects, we 'wrap' the grid on itself such that the columns and rows of opposite sides of the grid are considered neighbours, i.e. $i = L + 1 \equiv i = 1$.

In this project, we are concerned only with the self-sustaining properties of the grid, and so exclude the external magnetic field ($B = 0$). This leaves only the first term, which creates a low energy state when sites are aligned with their neighbours, and a high energy state when anti-aligned. The minimum-energy states are then $M = \pm N_s$, i.e. a uniform grid of all spin-up or all spin down.

However, the randomizing influence of thermal energy means that the minimum energy state is not the highest probability state at finite temperature. Instead, the magnetisation of an actual lattice is temperature-dependent, with mean magnetisation obeying:

$$\langle M\rangle = \pm N_s\left(1 - \sinh^{-4}\left(\frac{2J}{k_B T}\right)\right)^{\frac{1}{8}}$$



Mean Magnetisation For $\boldsymbol{L \to \infty}$ as a Function of Temperature

There exists a critical temperature, $T_{crit} \approx \frac{2}{\ln|1+\sqrt{2}|}$, at which a phase transition occurs. Below this temperature, the grid readily settles at or near its positive or negative minimal energy uniform states. However, this structure rapidly breaks down as $T \to T_{crit}$ until a sharp transition occurs beyond which the thermal noise supresses any uniform structure in the spins, rapidly devolving the lattice into a mess of random spin directions for $|M| \approx 0$.

To simulate the grid, we make use of Markov Chain Monte Carlo (MCMC) methods, a statistical technique in which we randomly vary the system parameters to achieve a series of snapshot 'samples' that are representative of the system's overall behaviour. In MCMC, a 'walker' follows random path through various system states (a Markov Chain) in which, at each step, a new random configuration is proposed and either accepted or rejected with probabilities weighted by how likely the proposal state is (hence Monte Carlo). The general procedure for simple MCMC is, at each step:

1. Choose a random grid element to flip (make a proposal)
2. Determine the resultant energy change, ΔE
3. If $\min\left(1, exp\left(-\frac{\Delta E}{k_B T}\right)\right) > r,\ r \sim U(0,1)$, the proposal is accepted
4. If not remain at the current position and add that to the chain

Because samples should be statistically independent for a representative distribution, several steps are taking between the drawing of each sample. The sampling phase is also typically preceded by a series of burn-in / warmup iterations to move towards a high-likelihood state.

MCMC is often employed in optimizing high dimensional systems by way of physical analogy, navigating walkers through parameter space until they reach regions of low energy, with energy being an analogue for 'goodness of fit'. In our case, energy and temperature are not analogues, but actual physical properties of the system.

The advantage of MCMC over direct grid search integration is that we need not waste resources evaluating low-likelihood states, and can instead focus on sampling only the configurations of highest interest. Our case requires only a few thousand samples be drawn with $\approx N_s$ steps between each sample, giving a cost that scales with $\mathcal{O}(N_s)$. By contrast, a full grid search of all possible states would scale with $\mathcal{O}(2^{N_s})$.

Though more robust than direct optimization, MCMC can still become pinned in local energy wells or show unrepresentative behaviour in multimodal distributions. As such, it is common practice to launch multiple independent 'chains', each with their own burn-in and sampling phase, and collate their results after the fact.

Code Structure

The code for this section of the assignment is contained in three cpp files / executables:

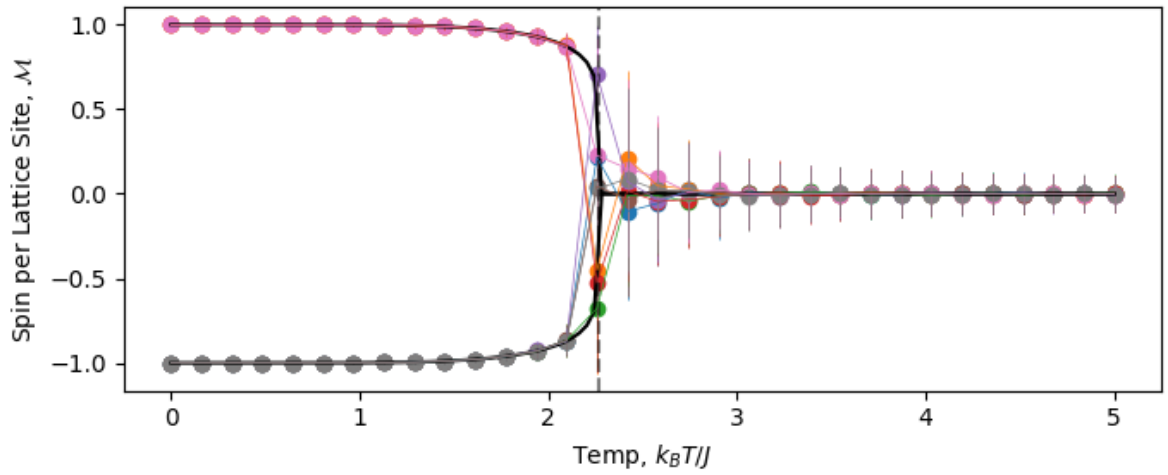
- `singlegrid.cpp`, which runs a single MCMC chain for a grid of fixed size at fixed temperature and describes the resultant state
- `temp_sweep.cpp` which calculates performs progressive MCMC chains over a range of temperatures and saves the mean and variance of energy and spin for each temperature and each chain
- `singlegrid_parallel.cpp`, which utilizes openMP to parallelize the MCMC process for benchmarking the impacts of multi-threading.

The results as generated by `temp_sweep.cpp` are by default, saved to the "`./results`" folder, and the file can have its run parameters (grid size, MCMC chain parameters, output folder etc) input via command line. See the `readme.pdf` in the docs folder for details. Programs in `c++` are only used for the numerical 'heavy lifting' of the MCMC routines: all plots and calculations are performed with `python` scripts contained in the results folder. By running the `_compileandrun.sh` shell script, all output files should be generated with the correct naming conventions for use with these plotting scripts.

All methods relating to the grid (calculations of energy state, energy change associated with a bit flip etc) are stored in the `grid.hpp` file. The MCMC procedures are stored in the `monte_carlo.cpp` file, and general utilities for random number generation and vector calculations are stored in `rand_utils.cpp` and `vector_utils.cpp`.

Methods

To achieve good convergence and uncorrelated samples, we perform $L \times L$ bit flips per sample, drawing 10^4 samples per chain with 10^3 burn-in steps. To account for the multi-modal spin-function at low temperature, we run 8 independent chains per run for all results in parts 2.1 and 2.2:



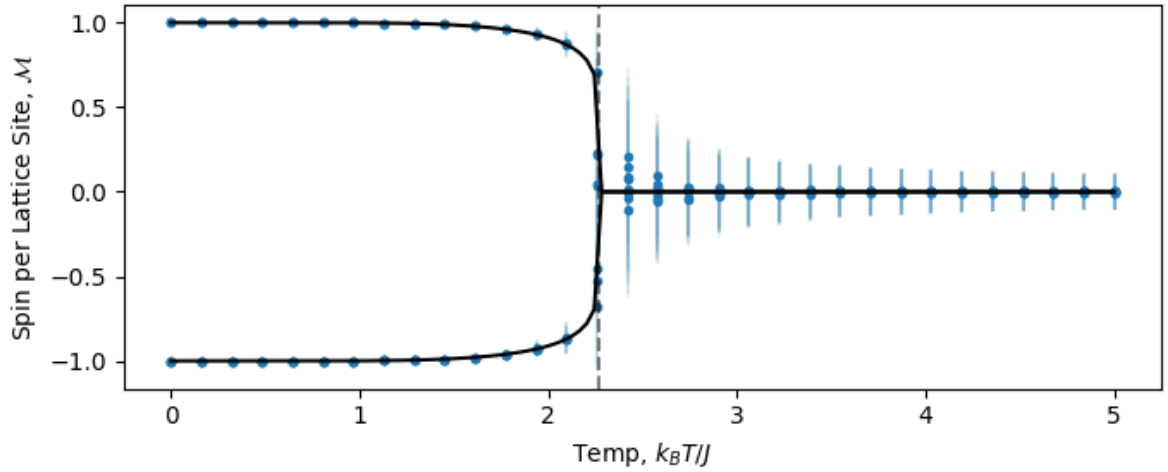
Spin-Walks for 8 Chains Over Temperature Sweep For $\boldsymbol{L = 16}$ Grid, Showing Mode Migration at $\boldsymbol{T \approx T_{crit}}$

Each chain is initiated at the lowest temperature of the sweep in a randomly selected all-up or all-down state, and the end state of this MCMC run is used as the input for the proceeding state. To combine these results, we take the minimum variance estimate across all chains for average values, e.g.:

$$\langle M(T_i)\rangle = \frac{1}{\sum_k w_i^k}\sum_k \langle M^k(T_i)\rangle w_i^k, \qquad w_{-i}^{\ k} = \frac{1}{VAR\left(M^k(T_i)\right)}$$

Where superscript 'k' indicates the chain number. The variance in this mean is calculated with similar weighting. Running multiple chains also allows us to calculate the spread in variances, and it is this spread that is used to provide error bars for $c(T)$ and $\chi(T)$.

Because of the high total sample number, the uncertainty in the mean (which scales with $\left(N_{samples\ per\ chain} \cdot N_{chain}\right)^{-\frac{1}{2}}$) is extremely low, making them difficult to see in most plots. In all plots, error bars with flat caps indicate the uncertainty in the mean values, while the transparent un-capped error bars represent the spread in the overall population of samples.



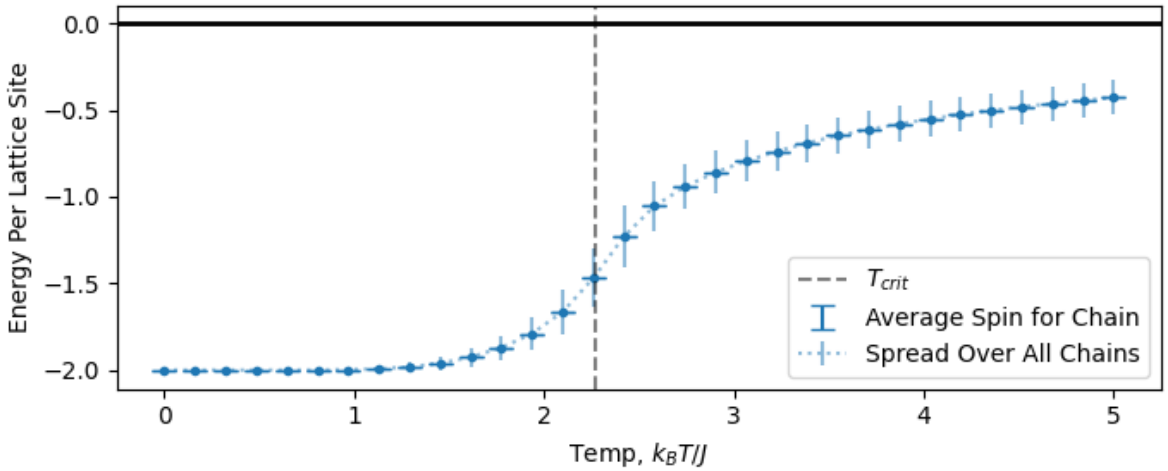
Spin-Distributions for 8 Chains Over Temperature Sweep For $\boldsymbol{L = 16}$ Grid
Transparent uncapped bars are population spread for each chain, and capped bars (too narrow to see) are measurement uncertainties within each chain

Part 2.1 Thermodynamic Properties of the 2D Ising Model

All plots in this section are for an $L = 32$ grid, with 8 chains at 10^4 samples and 10^3 burn-in steps, with $32 \times 32 = 1024$ proposed bit flips at randomly selected grid locations for each sample.

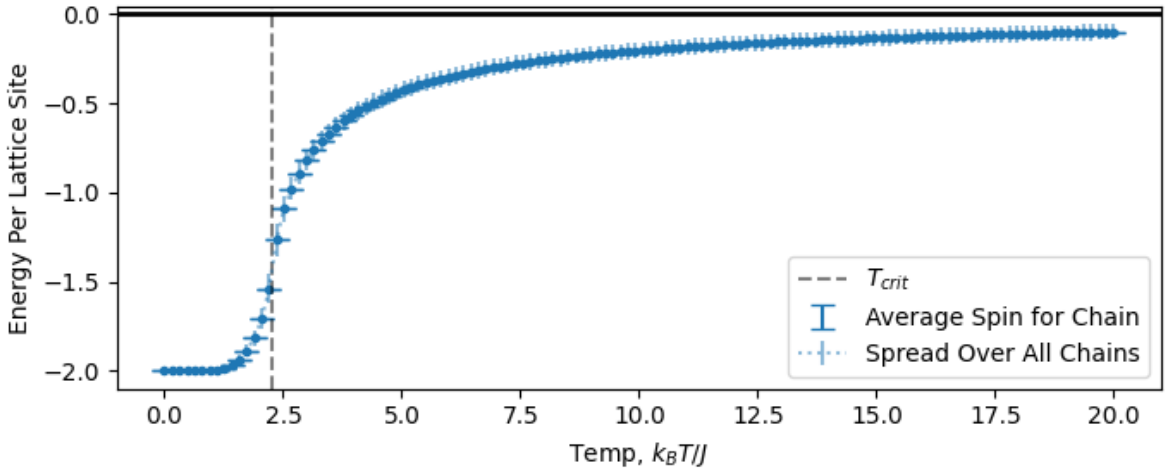
Part 2.1.1 Energy Vs Temperature

Plotting energy against temperature for the grid, we see the initial minimal energy state, with all spin sites aligned parallel to one another, steadily evolve into a higher energy state with a faster increase at $T \approx T_{crit}$. This increase is something similar to the change in latent energy we see in other physical phase transitions, representing a rapid change in the system properties over a small range of temperatures. The energy variance is small at low temperatures, as perturbations to the consistent lattice state are small and short-lived.



Specific Energy Vs Temp for an $\boldsymbol{L = 32}$ Grid

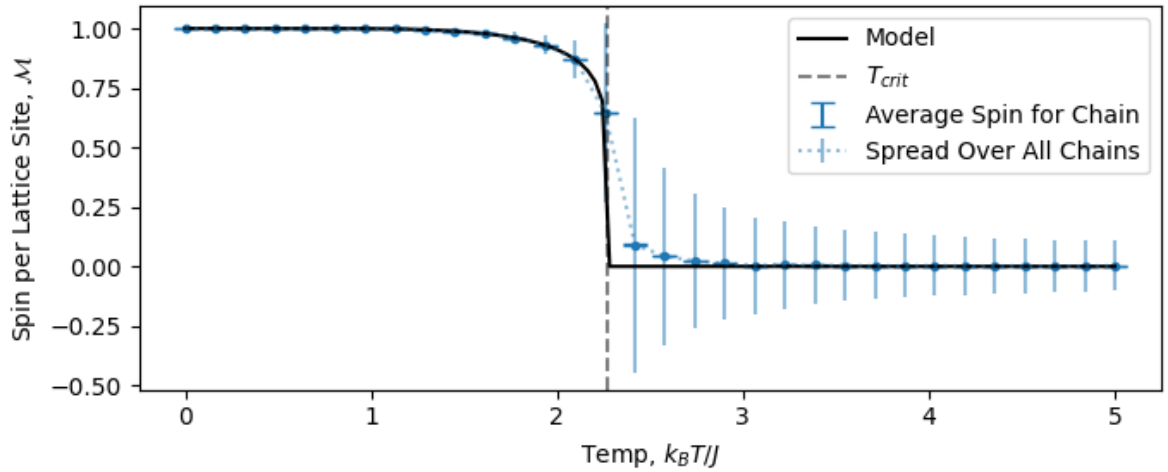
Extending our plot out to higher temperatures, we can see that the limit $T \to \infty$ give $\mathcal{E} \to 0$, which corresponds to a purely random state of every spin half up and half down neighbours on average. The mean energy is strictly negative, as spins still have an overall tendency to align parallel to one another.



Specific Energy Vs Temp for an $\boldsymbol{L = 32}$ Grid to High Temperature

Part 2.1.2 Spin vs Temperature

Plotting the absolute spin against temperature, we successfully recover the predicted behaviour, including the sharp phase change. In our finite grid, the phase change is not instantaneous, but instead has a continuous, though sharp, progression across the phase boundary:



Magnitude of Spin Per Cite Vs Temperature for an $\boldsymbol{L = 32}$ Grid

Another interesting feature is the behaviour of the spin variance. As with energy, there is little to no variance at low temperatures as the grid is 'frozen' into its minimal energy state: any bit flip would produce an increase in total energy, and the sharp scaling of the proposal acceptance means that these are overwhelmingly rejected. At $T \to \infty$, the grid approaches near-random spins equally aligned up and down. In this regime, perturbations are readily accepted, but lack structure and are quickly washed out by thermal noise. Near the transition temperature, variance rapidly increases as small perturbations to the grid are moderately accepted, but survive long enough to have consistent structure before dying out.

We can predict the variance in the $T \to \infty$ regime by noting that each site's spin obeys a binary distribution, with:

$$Var(\sigma_i) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$$

The resulting variance of the entire lattice's spin is then:

$$Var(M) = N_s \cdot Var(\sigma_i)$$

Such that the spin obeys a random distribution:

$$M = 0 \pm \sqrt{\frac{N_s}{4}} = \pm\frac{L}{2}$$

Or, normalizing to spin per lattice cite:

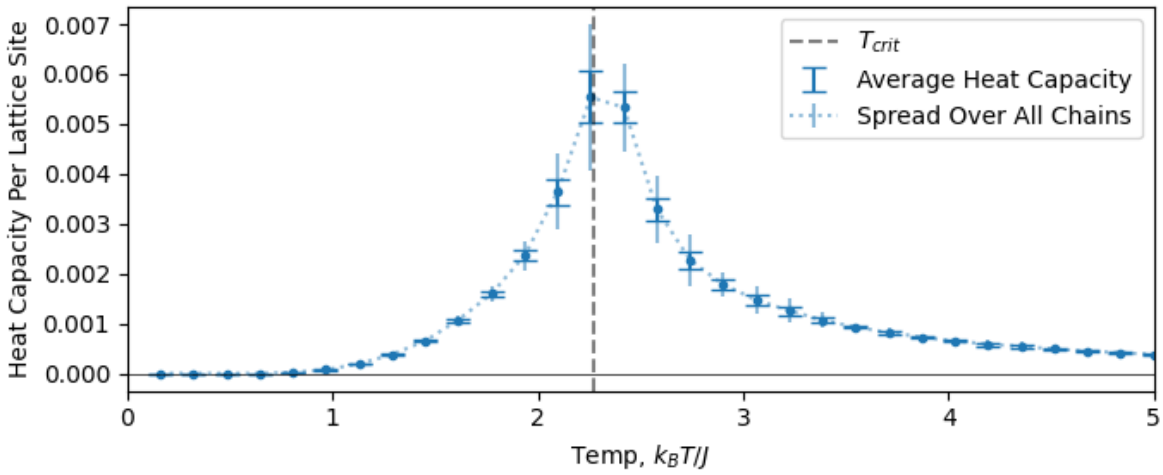
$$\mathcal{M} = \frac{M}{N_s} = \pm\frac{1}{2L}$$

Part 2.1.3 Specific Heat vs Temperature

We can also calculate the specific heat capacity of the system with, in our natural units:

$$c(T) = \frac{N_s}{T^2} Var(\mathcal{E})$$

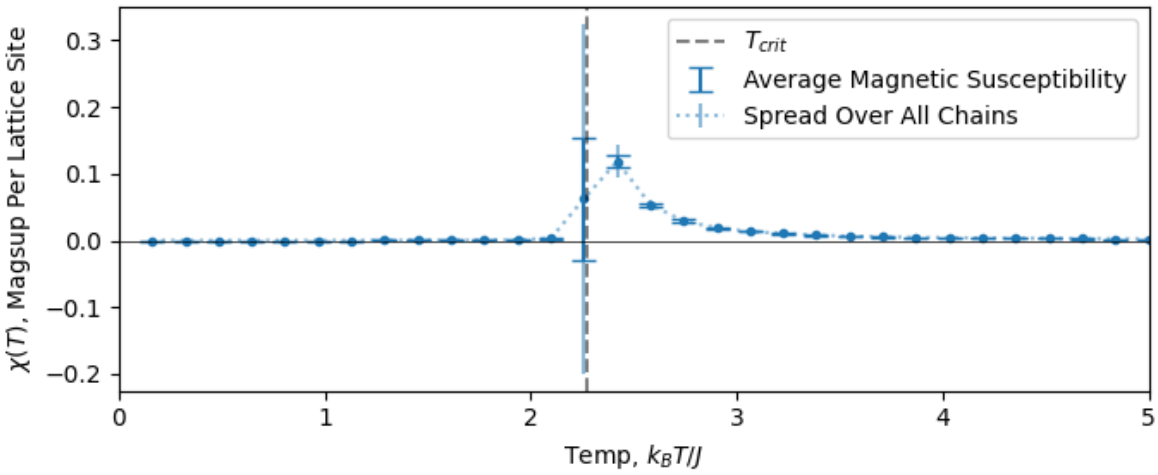
Doing so, and estimating error based on the spread in this value across our 8 chains, we recover the distribution we would expect given $c(T) = \frac{d\mathcal{E}}{dT}$: a peak that flattens to zero at $T = 0$, trends towards it slowly as the energy levels out at $T \to \infty$, and sees a sharp peak in the critical region where the energy changes most rapidly:



Part 2.1.4 Magnetic Susceptibility vs Temperature

Following a similar procedure as above, but now for magnetic susceptibility:

$$\chi(T) = \frac{N_s}{T} Var(\mathcal{M})$$

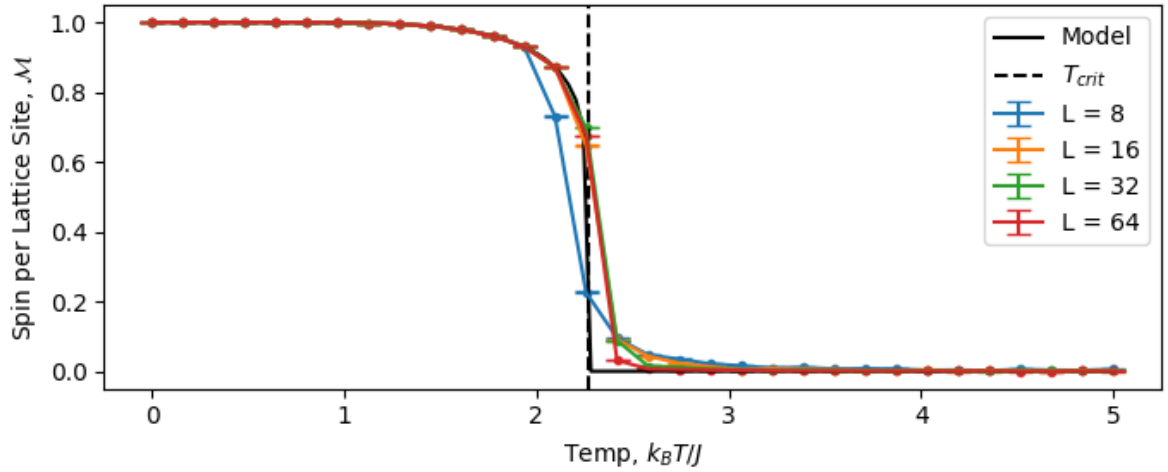


Magnetic Susceptibility Vs Vs Temperature for $\boldsymbol{L = 32}$ Grid

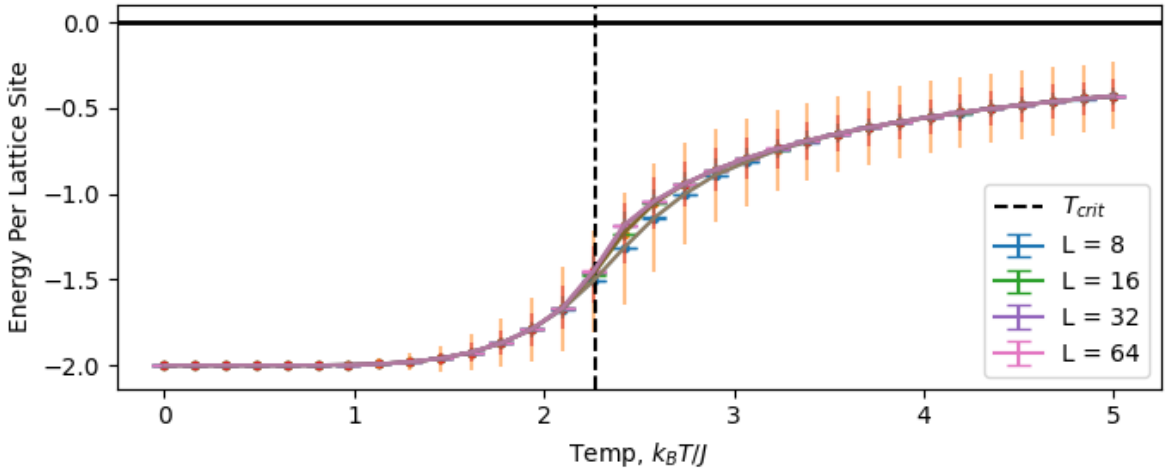
We see a similar trend to before, but with much faster damping at $T \to \infty$. Again, this is in keeping with our intuition: $\mathcal{M}$ settles out to $\mathcal{M} = 0$ rapidly past the phase transition, and so its derivative becomes flat. On a similar note, $\chi(T)$ becomes poorly conditioned at $T = T_{crit}$ as the grid becomes extremely susceptible to small changes in magnetisation, corresponding to the $\frac{d\mathcal{M}}{dT} \to \infty$ phase transition.

Part 2.2 Critical Scaling Behaviour of the 2D Ising Model

In this section, we examine the effect of the finite grid size on the system behaviour. The general trend, after normalizing, is that smaller grids lead to smooth transitions across the phase boundary. As we move from $L = 8$ to $L = 64$, both spin and energy tend towards the non-smooth analytical model for $L \to \infty$:



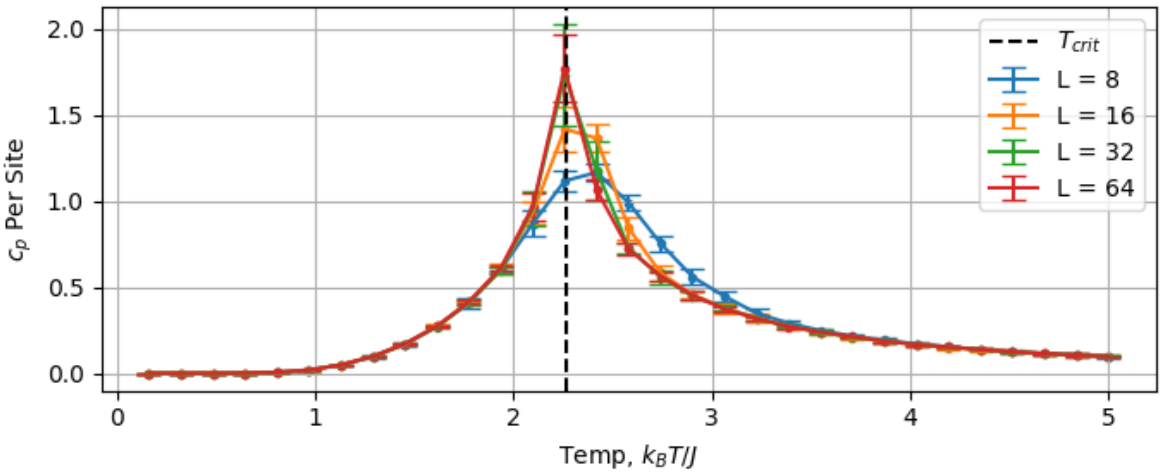
Spin Vs Temp for Multiple Grid Sizes



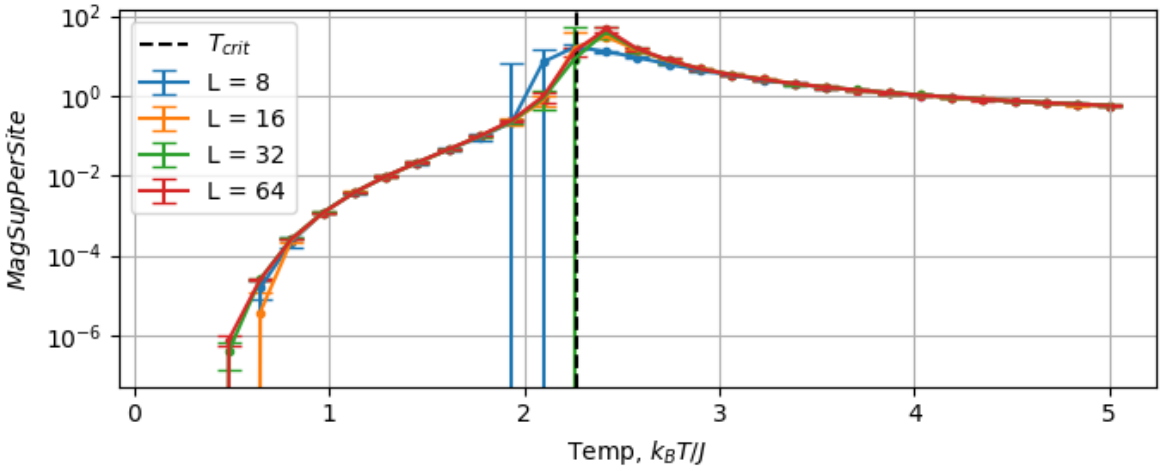
Energy vs Temp for Multiple Grid Sizes

Part 2.2.1 Behaviour of $c(T)$ & $\chi(T)$ Without Re-Scaling

Plotting the heat capacity and magnetic susceptibility for various grid sizes, we see behaviour in keeping with what we observed for their $\mathcal{M}(T)$ and $\mathcal{E}(T)$ plots: general agreement as we move away from the critical temperature, but with smaller grids having smoother curves near $T = T_{crit}$:



Heat Capacity Vs Temp For Multiple Lattice Sizes



Magnetic Susceptibility Vs Temp For Multiple Lattice Sizes (Log Scale)

Part 2.2.2 Behaviour of $c(T)$ & $\chi(T)$ With Re-scaling

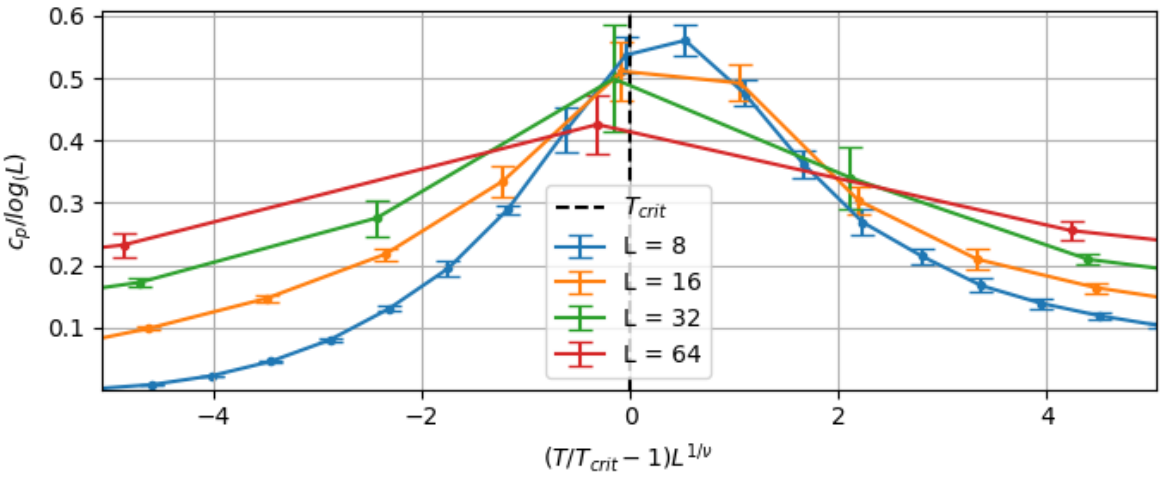
In order to develop a consistent, scale-invariant, model of $c(T)$ & $\chi(T)$ near $T = T_{crit}$, we can use the fact that they are known to obey finite scaling laws:

$$c(T) = \ln(L) \cdot f\left(L^{\frac{1}{\nu}}t\right), \qquad \chi(T) = L^{\frac{\gamma}{\nu}} \cdot g\left(L^{\frac{1}{\nu}}t\right)$$

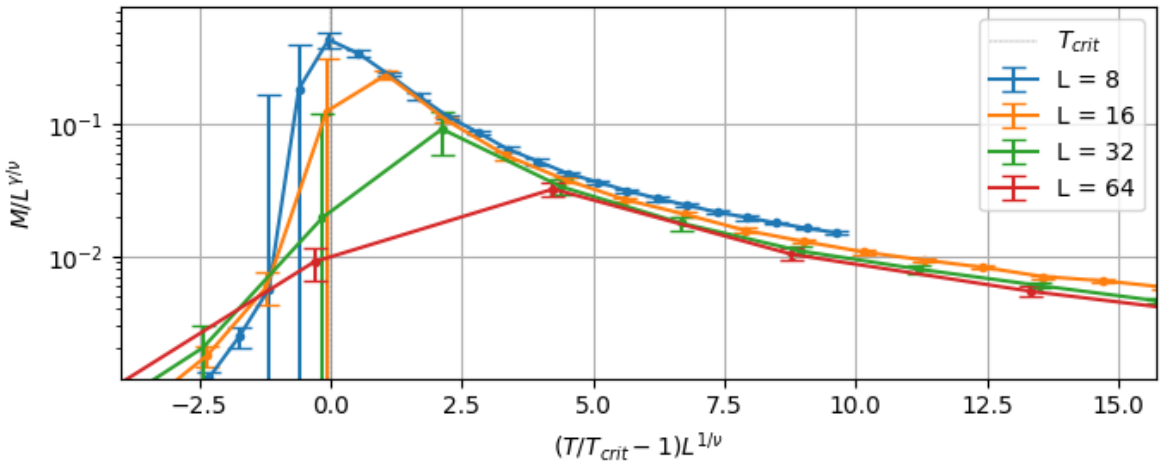
Where:

$$\nu = 1, \qquad \gamma = \frac{7}{4}, \qquad t = \frac{T - T_{crit}}{T_{crit}}$$

If we rescale our axis accordingly, plotting $\frac{c(T)}{\ln(L)}$ and $\frac{\chi(T)}{L^{\frac{\gamma}{\nu}}}$ against $L^{\frac{1}{\nu}}t$, we can see better consistency between the curves near the phase transition:



Heat Capacity, Rescaled, near $\boldsymbol{T = T_{crit}}$



Magnetic Susceptibility, Rescaled, near $\boldsymbol{T = T_{crit}}$

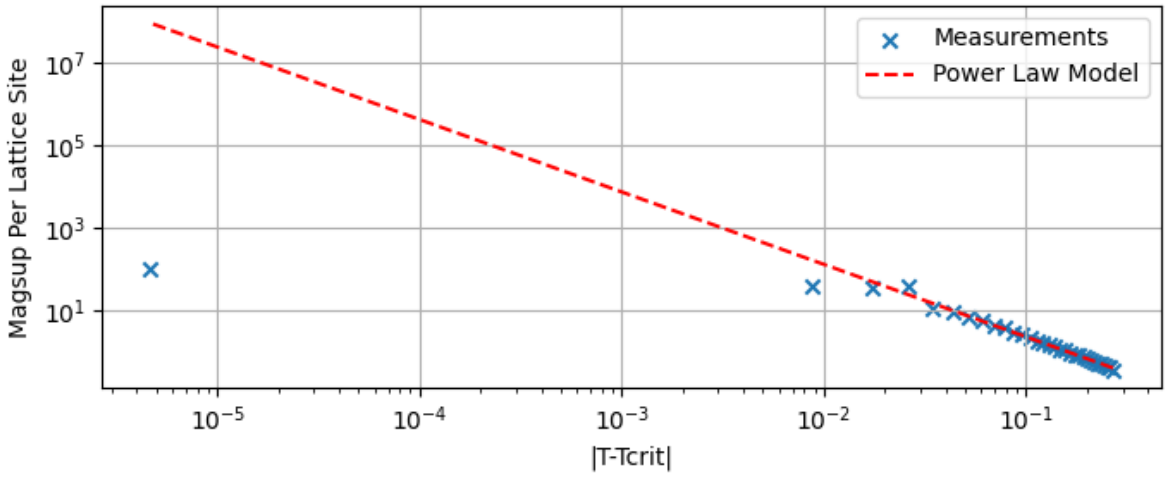
Note that some disagreement is expected, as we are dealing with values that are inherently variable and poorly conditioned.

Part 2.2.3 Power Law Scaling Near $T = T_{Crit}$

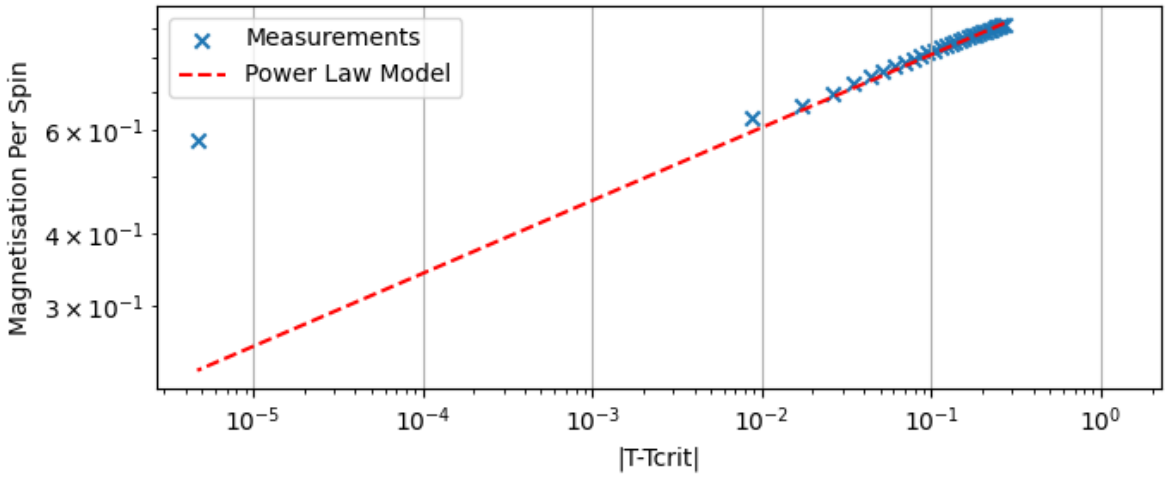
In addition to the finite scaling laws, we can also approximate the rapid changes in the system state near the phase transition with power laws, e.g.:

$$\mathcal{M} \propto |T - T_{crit}|^{\beta}, \qquad \chi(T) \propto |T - T_{crit}|^{-\gamma}, \qquad \beta = \frac{1}{8}, \qquad \gamma = \frac{7}{4}$$

These laws are derived in the limit $L \to \infty$, but we can validate them by simulating a sufficiently large grid, with results here being for $L = 128$. Because we are concerned only with the behaviour along a single branch for $\mathcal{M}$, we here run only a single chain, drawing 10^5 samples per temperature and sweeping over 32 temperatures in the range $T \in [2.0, T_{crit}]$.

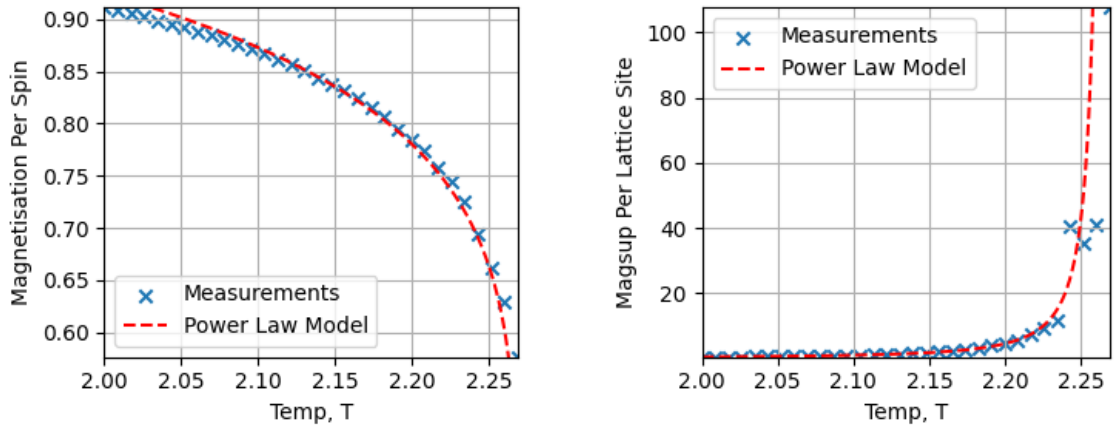


Power Law Scaling of Magnetic Susceptibility Near $\boldsymbol{T = T_{crit}}$



Power Law Scaling of Magnetisation Near $\boldsymbol{T = T_{crit}}$

Removing the log-scale, we can get a sense of how these laws reproduce the actual curves as seen in previous sections. As $\chi(T)$ goes to zero at $T \to 0$, the strictly decreasing power law doesn't have much room to diverge, but the finite $\mathcal{M} \to 1$ trend conflicts with the strictly increasing magnetisation power law. While $\mathcal{M}(T)$ flattens out, the power law continues indefinitely.



Power Law Scaling, no Log Scale

Variations at $T \to T_{crit}$ are best explained by the inherent variability introducing noise to our measurements.

Part 2.3 OpenMP Implementation

In this section, we test the effect of multithreading with `openMP` on the MCMC running time. For the parallelization process, we follow the procedure as nominated in the tutorial sheets, i.e. for each MCMC sample:

1. Isolate the even rows, $i \in \{0,2,\dots\}$, of the grid
2. Perform an `openMP` loop over all such rows
3. Within each loop, perform a sweep over all elements in that row, $j = \{0,1 \dots L-1\}$
    a. For each element, calculate the energy change for a flip
    b. Accept or reject this flip per the standard Metropolis hastings procedure
    c. If accepted, flip the bit, but do not update the grid's energy or spin tally. Instead, accumulate the change in a thread-specific count
4. Repeat steps 1-3 for the odd numbered rows
5. Have each thread apply its accumulated energy and spin changes to the grid
6. Have the master thread save the resulting grid properties to the output stream

Note that, unlike the standard MCMC procedure, sweeps are sequential over the row indices instead of being randomized.

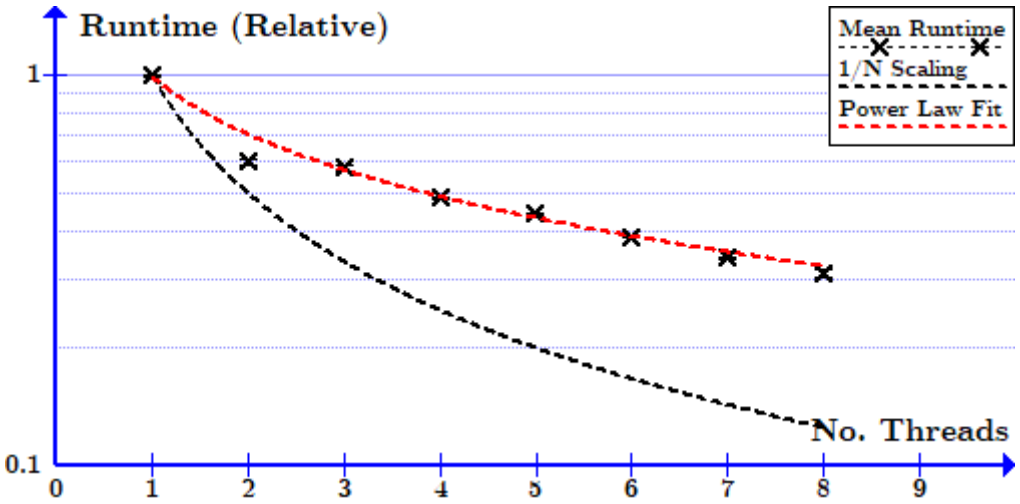
To achieve consistent results, we need to test for a sufficiently large grid, and all results here are presented for an $L = 64$ grid undergoing independent MCMC runs at 10^4 samples and 10^3 burn-in iterations, consistent with parts 2.1 and 2.2. The multi-threaded MCMC implementation is called in `singlegrid_parallel.cpp`, and is defined in the `monte_multithread()` function in contained in the `monte_carlo.cpp` file. The temperature for all runs is $T = 0.75T_{crit}$, a value that results in non-trivial structure in the final grid, without being so close to the phase transition as to give poorly conditioned results. To check for variability, each thread number is run for 5 independent chains and their run times averaged.

Running for up to 9 threads (one more than the CPU core count of the testing device), we get the results as shown below.

No. Threads	Mean Runtime (s)	Standard Deviation (s)	Run Times (s)				
1	3.60	0.04	3.56647	3.56329	3.55582	3.6334	3.66359
2	2.16	0.08	2.23361	2.03133	2.10355	2.26225	2.17654
3	2.08	0.09	1.93331	2.09397	2.18006	2.03464	2.17774
4	1.75	0.01	1.71752	1.76037	1.7544	1.75142	1.74297
5	1.59	0.01	1.58881	1.56499	1.59031	1.60242	1.60194
6	1.39	0.02	1.38113	1.37441	1.38476	1.38847	1.42348
7	1.23	0.03	1.19926	1.19529	1.23869	1.24133	1.26044
8	1.12	0.05	1.05697	1.06508	1.17098	1.13496	1.18354
9	5.77	0.04	5.76335	5.73598	5.74428	5.75997	5.85115

MCMC Runtimes

As expected, we see a general reduction in runtime as we use more resources, though with diminishing returns as more resources are being taken up by the overhead of thread management. Past the CPU count (i.e. at $N \geq 9$) performance massively drops, as there aren't enough devices available to parallelize across. We also see better performance at thread numbers that are powers of two as compared to the general trend, i.e. $N \in \{2,4,8\}$, as these allow the loop, which runs over $2^{\frac{L}{2}}$ rows, to be evenly divided between the threads. An equal amount of work for each thread means better efficiency, as all threads finish at roughly the same time, and no runtime is lost waiting for threads to synchronize.



Effect of Multi-Threading on Run-Time (Log-Scale)