Computational Physics

Project 4

UiO : University of Oslo

Hunter Wilhelm Phillips (*hunterp*)

November 19, 2018

GitHub Repository at

https://github.com/robolux/Computational_Physics

Abstract

This project explores the implementation of different integration methods to build a working model of the solar system. Object oriented programming is utilized to develop a class structure for the simulation. The computation of the celestial bodies orbits was the core part of the algorithmic development. This requires the integration of Newton's law of motion and for this project; Forward Euler and Velocity Verlet were chosen. The project started with modeling a two-body system (Earth-Sun) and progressed until a complete solar system was simulated. Throughout the development, multiple facets of the system were verified including conservation of energy, determining stable velocity for circular orbit, and time step overshoot analysis. Benchmarks were ran comparing the two chosen integrators, which resulted in showing the inaccuracies of the Forward Euler method. Various experiments were conducted with the algorithm such as changing the mass of Jupiter then analyzing the result on the associated celestial bodies and calculating the perihelion precessions of Mercury verifying that there are indeed 43. The results confirmed that an integrator and the associated law of motion coupled with object orientation produces a stable simulation of the solar system.

1. Discussion of Problem Statement

In this project, the goal is to successfully simulate the solar system through the laws of motion. To achieve this two integrators were chosen, Forward Euler and Velocity Verlet to solve the coupled differential equations (DE). First, the Earth-Sun system is set to be analyzed with the resulting discretized DE's being used as the basis for computation. Second, the algorithms are to be tested and tuned for optimal performance. This includes transferring variables into class structures to transition towards object orientation. To produce a circular orbit, the initial value for velocity must be changed to allow for stability, even with varying time steps. To verify that the conservation of energy and angular momentum is held; kinetic, potential, and angular momentum are to be calculated for analysis. A comparison between Forward Euler and Velocity Verlet is to be completed resulting in the decision to only use Velocity Verlet in the rest of the project. Escape velocity is next to be calculated and a custom β modifier is experimented with to change the gravitational force. A three-body problem is simulated next that includes the Sun, Earth, and Jupiter with varying masses of Jupiter to send the system into a disarray. A final model of the solar system that includes all planets is then modeled to verify the algorithm successfully handles the celestial bodies in our solar system. Finally, the perihelion precession of Mercury is to be calculated through the use of a relativistic correction of Newton's gravitational force and comparing it to the uncorrected original equation. Through this, the challenges at hand have been laid out and the theory and methods explaining the solution are described in the following section.

2. Discussion of Theory & Methods

2.1 The General Ising Model

Assuming that the formation of a d-dimensional lattice is coupled in a set of Λ adjacent lattice positions. For each lattice elemental location, $k \in \Lambda$, there is a representation of the spin of the element ↑ and ↓ in the form of a discrete variable $\sigma_k \in \{+1, -1\}$. Each individual spin configuration σ is connected to a specific state of the lattice.

If there are two elemental locations, i and j, in the set, one of them will have an interaction J_{ij}. This same characteristic site will also have an external magnetic field h_j coupled with it. The energy of the system specified can be given by the following Hamiltonian as seen in (1):

$$E = H(\sigma) = -\sum_{\langle ik \rangle} J_{ij}\sigma_i\sigma_j - \mu\sum_j h_j\sigma_j \tag{1}$$

Where the first summation is over pairs of adjacent spins

$\langle ij \rangle$ displays that i and j are nearest neighbors

μ is the magnetic moment

The Boltzmann distribution gives us the probability of a certain configuration as shown in (2):

$$P_\beta(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z_\beta} \tag{2}$$

Where Z_β is the normalization constant

The functioning $\beta = (k_B T)^{-1}$

This showcases that an increase in temperature directly affects the possibility of finding a system in a particular configuration in a negative manner.

This leads into the characterization of a magnetic system and the most important expected values to be calculated. The first is mean energy as seen in (3).

$$\langle E \rangle = \sum_{i=1}^{M} E_i P_\beta(\sigma) = \frac{1}{Z}\sum_{i=1}^{M} E_i e^{-\beta E_i} \tag{3}$$

Where M signifies the number of possible configurations

The second is mean magnetization as seen in (4).

$$\langle \mathcal{M} \rangle = \sum_{i=1}^{M} \mathcal{M}_i P_\beta(\sigma) = \frac{1}{Z}\sum_{i=1}^{M} \mathcal{M}_i e^{\beta E_i} \tag{4}$$

Where $\mathcal{M}_i = \sum_{j\in\Lambda} \sigma_j$ is represented for all configurations σ

The third is magnetic susceptibility as seen in (5).

$$\chi = \frac{1}{k_B T}(\langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2) \tag{5}$$

The fourth is heat capacity as seen in (6).

$$C_V = \frac{1}{k_B T^2}(\langle E^2 \rangle - \langle E \rangle^2) \tag{6}$$

These reduced equations form the basis for the project and allow for the specific model to be deduced. The Ising model can be classified as having a ferromagnetic interaction if $J_{ij} > 0$ and for the purposes of this simulation, this case will be used.

2.2 Simplified Ferromagnetic Ising Model

To ease the complexity of this project, the general Ising model as described in *Section 2.1* will be simplified. The first step is to assume that there is no external magnetic field in play and with this assumption, (1) has it second summation go to zero, leaving (7):

$$E = H(\sigma) = -\sum_{\langle ik \rangle} J_{ij} \sigma_i \sigma_j \tag{7}$$

The coupling constant J is assumed to be a constant which enables the removal of it from within the summation forming (8).

$$E = H(\sigma) = -J \sum_{\langle ik \rangle} \sigma_i \sigma_j \tag{8}$$

The final assumption is that the model is ferromagnetic in nature which means that neighboring spins are aligned, resulting in lower energies.

2.3 Case: L = 2 × 2 Ising Model

To prepare for later comparisons with numerical results, the analytical expressions for an L = 2 x 2 case are explored. Using (3), (4), (5), and (6); the following, Table 1, shows all of possible states for an L = 2 x 2 case.

State	Symmetries	Energy (J)	Mean Magnetization
↑ ↑ ↑ ↑	1	-8	4
↑ ↑ ↑ ↓	4	0	2
↑ ↑ ↓ ↓	4	0	0
↑ ↓ ↓ ↑	2	8	0
↓ ↓ ↓ ↑	4	0	-2
↓ ↓ ↓ ↓	1	-8	-4

Table 1: Case L = 2 x 2 all 16 Possible Configurations

To compute all the associated physical quantities as shown above, the closed form expressions can be given by the following partition function (9):

$$Z = \sum_{i=1}^{1} 6e^{-\beta E_i} = e^{\beta 8J} + 12 + 2e^{-\beta 8J} + e^{\beta 8J} = 4\cosh(8\beta J) + 12 \tag{9}$$

This leads to the expected energy forming into (10):

$$\langle E \rangle = -\frac{\partial}{\partial \beta} \ln Z = -\frac{\partial}{\partial \beta} \ln(4\cosh(8\beta J) + 12) = -8J \frac{\sinh(8\beta J)}{\cosh(8\beta J) + 3} \tag{10}$$

Using (4) the mean magnetization for this particular system can be computed into (11)

$$\langle \mathcal{M} \rangle = \frac{1}{Z}(-4e^{8\beta J} - 8e^{0} + 8e^{0} + 8e^{8\beta J} = 0 \tag{11}$$

This leads to the expected absolute magnetization coming out as (12):

$$\langle \mathcal{M} \rangle = \frac{1}{Z}(-4e^{8\beta J} - 8e^{0} + 8e^{0} + 8e^{8\beta J} = \frac{4 + 2e^{8\beta J}}{\cosh(8\beta J) + 3} \tag{12}$$

The expected value for specific heat is characterized in (13):

$$\langle C_V \rangle = \frac{1}{k_b T^2} \frac{\partial^2}{\partial \beta^2} \tag{13}$$

By inserting (10) into (13) the expected specific heat analytical equation (14):

$$\begin{aligned} \langle C_V \rangle &= \frac{1}{k_b T^2} \frac{\partial}{\partial \beta} \left(-8J \frac{\sinh(8\beta J)}{\cosh(8\beta J) + 3} \right) \\ &= \frac{1}{k_b T^2} \left(64J^2 \frac{\cosh(8\beta J)}{\cosh(8\beta J) + 3} - 64J^2 \frac{\sinh^2(8\beta J)}{(\cosh(8\beta J) + 3)^2} \right) \end{aligned} \tag{14}$$

Rewriting (5) into (15) gives the basis for the susceptibility of magnetism:

$$\chi = \frac{1}{k_B T} \sigma_{\mathcal{M}}^2 \tag{15}$$

By combining (11) and (12) into (16), the second integral part for the susceptibility of magnetism can be derived:

$$\sigma_{\mathcal{M}}^2 = \langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2 = \frac{32}{Z} \left(e^{8\beta J} + 1 \right) - 0 = \frac{8 + 8e^{8\beta J}}{\cosh(8\beta J) + 3} \tag{16}$$

Finally, by inserting (16) into (15), (17) can be formulated:

$$\chi = \frac{8 + 8e^{8\beta J}}{k_B T (\cosh(8\beta J) + 3)} \tag{17}$$

These analytical results are the basis for future comparisons with numerically obtained results.

3. Discussion of Algorithms

3.1 Ising Class

To expand on the object orientation programming approach explored in Project 3, the same class structure theory was implemented into this project. All computations and writing to files is completed within the class structure enabling easy future expandability of the coding framework. This allows for seamless calling from unit testing and user interface scrips. The most important

functions held within are *metropolis* and *solve*, with the following sections going into detail further about their functionality.

3.2 Metropolis

The key facet to the successful simulation of the Ising model is a core algorithm that can produce random samples from a probability distribution. The traditional form of direct sampling is beyond difficult to achieve due to the complete partition function being needed to express the probability distribution. The beauty in the Metropolis algorithm is that it only requires a function proportional to the distribution density.

Metropolis Implementation

```
def metropolis(self,w):

  L = int(self.L)
  for iterable in xrange(L*L):
      i,j = nmp.random.randint(1,L+1,2)
      dE_t = 2.*self.spin_mat[i,j]*(self.spin_mat[i,j+1] + \
          self.spin_mat[i,j-1] + self.spin_mat[i+1,j] + self.spin_mat[i-1,j])
      if dE_t <= 0:
          self.flip(i,j)
          self.E += dE_t
          self.M += 2.*self.spin_mat[i,j]
          self.accepted += 1
      else:
          if nmp.random.random() <= w[8+int(dE_t)]:
              self.flip(i,j)
              self.E += dE_t
              self.M += 2.*self.spin_mat[i,j]
              self.accepted += 1
```

A proposed move is only implemented with the basis of a transition probability coupled with an acceptance probability. The main advantage of this approach is that the transition probability does not need to be know.

3.4 Monte Carlo

Each Cycle of the Monte Carlo simulation is fairly straightforward and essentially is a framework for obtaining mass amount of data points from *metropolis*. A simple snippet from

solve can be seen below, showing a Monte Carlo simulation loop, which calls metropolis for each Monte Carlo cycle to be performed until the maximum number of cycles is reached.

Monte Carlo Implementation

```
while cycle < MCc:
    self.metropolis(w)
    avg[0] += self.E; avg[1] += self.E**2
    avg[2] += self.M; avg[3] += self.M**2
    avg[4] += abs(self.M)
    cycle += 1
self.average(avg,T,cycle,filename)
```

3.4 Parallelization

When running larger test cases with lattice sizes reaching up to 100 x 100 it was recommended that we use a pool based multiprocessing framework to parallelize the code. This would enable each thread on the CPU to solve a specific case. At first this seemed fairly trivial, although it ended up being incredibly frustrating and complex. Since this was a pure Python implementation, *mpi4py* (uses the *MPI* C library) and *multiprocessing* were analyzed for use. The main drawback of *mpi4py* is that its documentation is not as good as pure *MPI*, leaving a lot of questions unanswered. This led to *multiprocessing*; with the specific function, *pool*, being explored further.

All was going well until both libraries produced a *PicklingError* upon execution. Upon further research into the documentation it appears that bound methods cannot be pickled. In *mpi4py* the underlying *MPI* C library uses cpickle while *multiprocessing* uses pickle, both failing to properly pickle the methods for parallelization. This would require a complete program rewrite and defeat the purpose of the beautiful object orientation currently in use.

Since this was not feasible to implement in a reasonable amount of time, the runs were just performed in a successive order, which worked fine for obtaining the results. However, the issue was explored deeper for future projects. It appears that there is a fork of *multiprocessing* called *pathos.multiprocessing* that uses dill to serialize almost everything, including bound methods [1]. It appears that this library would work and be able to create appropriate parallel pools with the dill pickled methods.

4. Discussion of Results

4.1 System Specifications

A few notes about the system running the algorithms before analyzing the results should be helpful in gauging performance between differing hardware as seen in *Table 2*.

Operating System	Mac OS X
Processor	i7 - 2.2 GHz
Memory	8 GB 1600 MHz DDR3
Hard drive	128 GB SSD

Table 2: System Specifications

With the testing conditions being established we can now dive into the data recorded.

4.2 Ising Model with L = 2 x 2

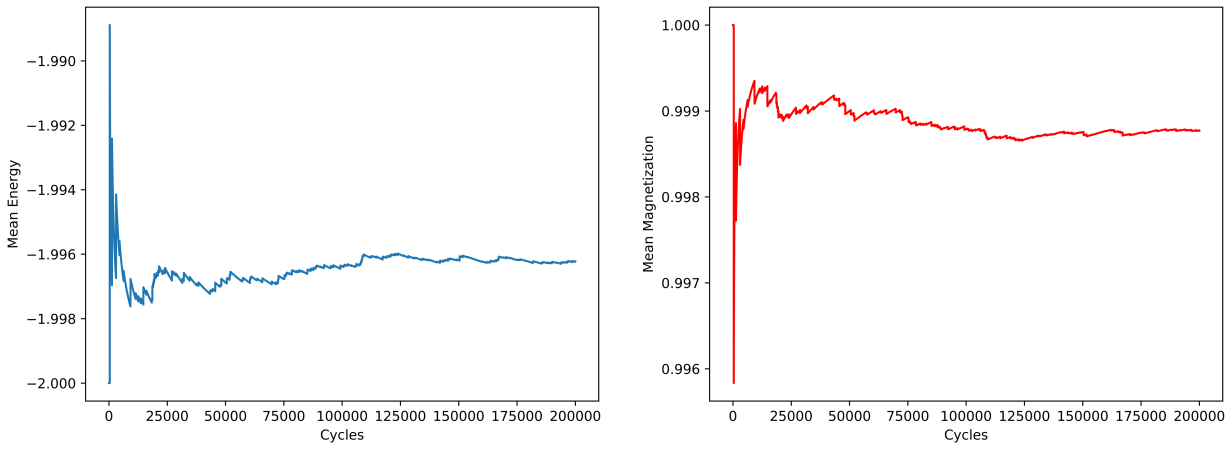
The first test of the project was to use a simple 2 x 2 lattice with a $T = 1.0$ across a range of Monte Carlo cycles in the set of [1e3, 1e4, 1e5, 1e6]. The mean energy $\langle E \rangle$, mean magnetization $\mathcal{M}$, specific heat C_V, and susceptibility χ were calculated to provide units of comparison. These numerical results were then compared with the analytical solution as showcased below in *Table 3*.

M	$\langle E \rangle$	C_V	χ	$\mathcal{M}$
1e3	-1.99600	0.031936	0.001996	0.99900
1e4	-1.99580	0.033529	0.005090	0.99845
1e5	-1.99604	0.031617	0.004133	0.99865
1e6	-1.99595	0.032318	0.004039	0.99865
Analytical	−1.99598	0.03208	0.004010	0.99865

Table 3: Comparison between Monte Carlo simulations at different length and the analytical solution for a L = 2 × 2 system

It appears that a lot of Monte Carlo cycles does improve the solution but only to a small extent. For the sake of choosing a number that achieves a good agreement, it appears that 1e5 cycles is a stable and solid choice. To verify this decision and provide a buffer zone for convergence, a test was run with 2e5 cycles as seen below in Figure 1.

Figure 1: Random Spin Orientation of L = 2 x 2 over 2e5 cycles with T = 1.0
Mean Energy (left) and Mean Magnetization (right)



It can be seen that the solution stabilizes nicely, validating the decision to provide a doubling multiplier of convergence space, resulting in 2e5 being the optimal amount of Monte Carlo cycles for this particular case

4.3 Ising Model with L = 20 x 20

With the verification that the model works on a lattice size of 2 x 2, it was time to expand it into a lattice size of 20 x 20. It can be assumed that all *Figures* in *Section 4.3* are with a lattice size of 20 x 20. The first task was to compute and plot *Figures 2 – 5* for analysis.

Figure 2: Mean Energy with T = 1.0 and 1e4 Monte Carlo Cycles

Ordered (Left) and Random (Right)

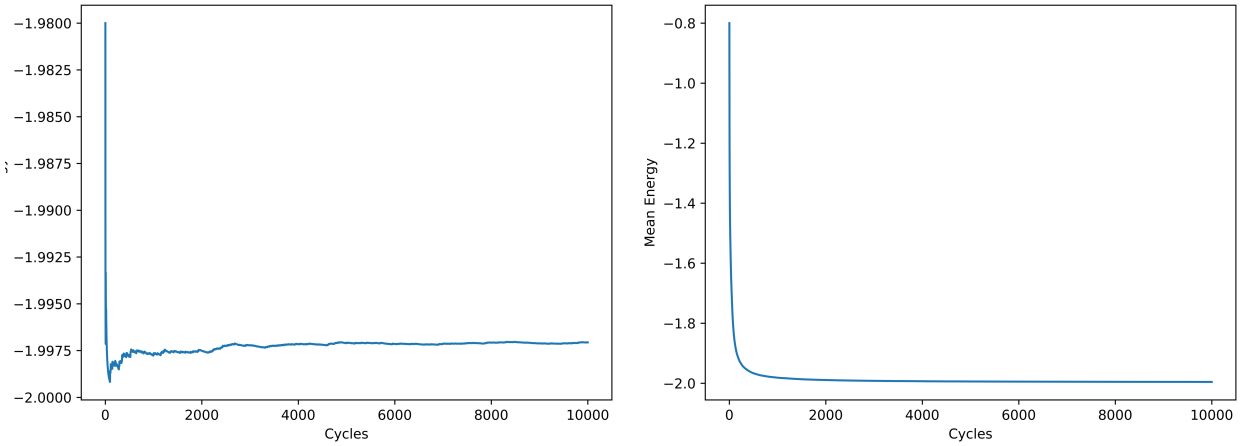


Figure 3: Mean Energy with T = 2.4 and 1e4 Monte Carlo Cycles

Ordered (Left) and Random (Right)

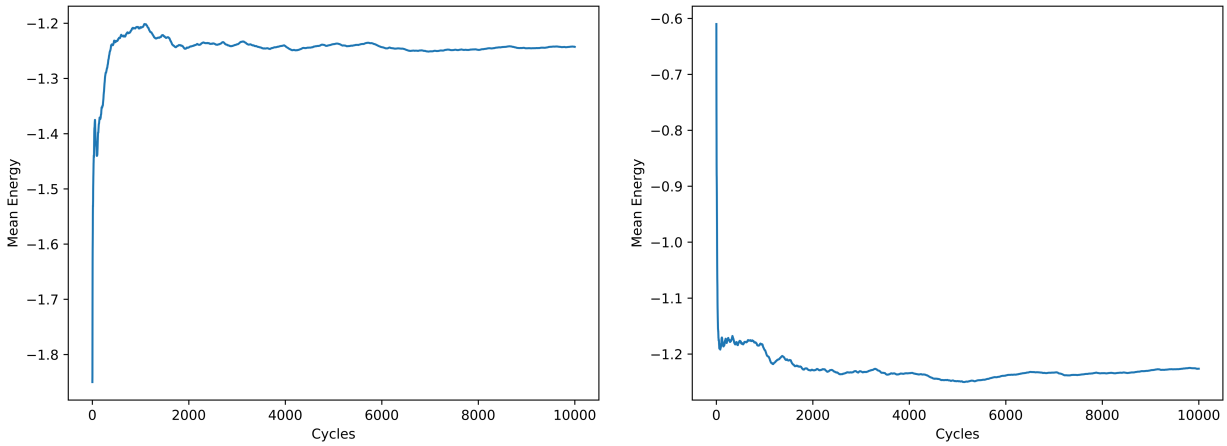


Figure 4: Mean Magnetization with T = 1.0 and 1e4 Monte Carlo Cycles Ordered (Left) and Random (Right)

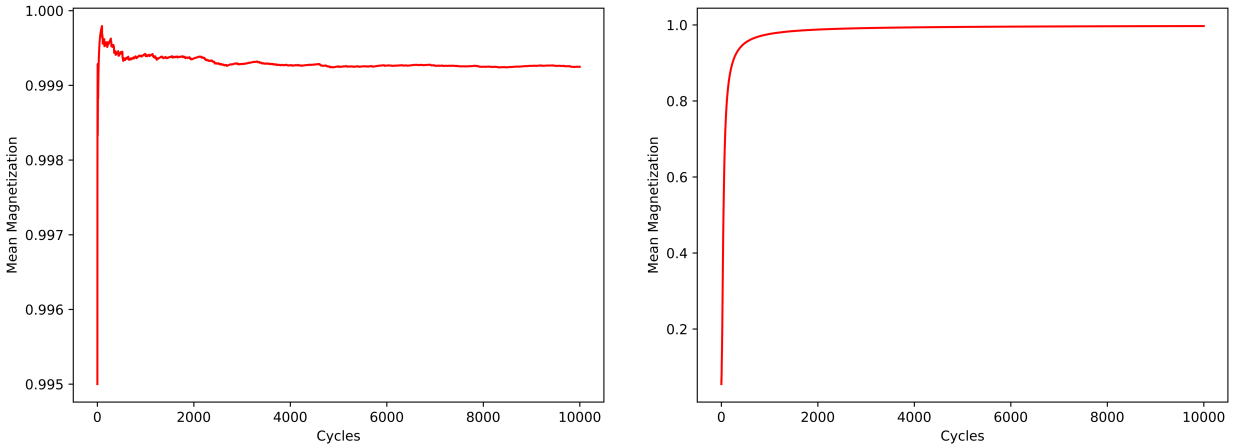
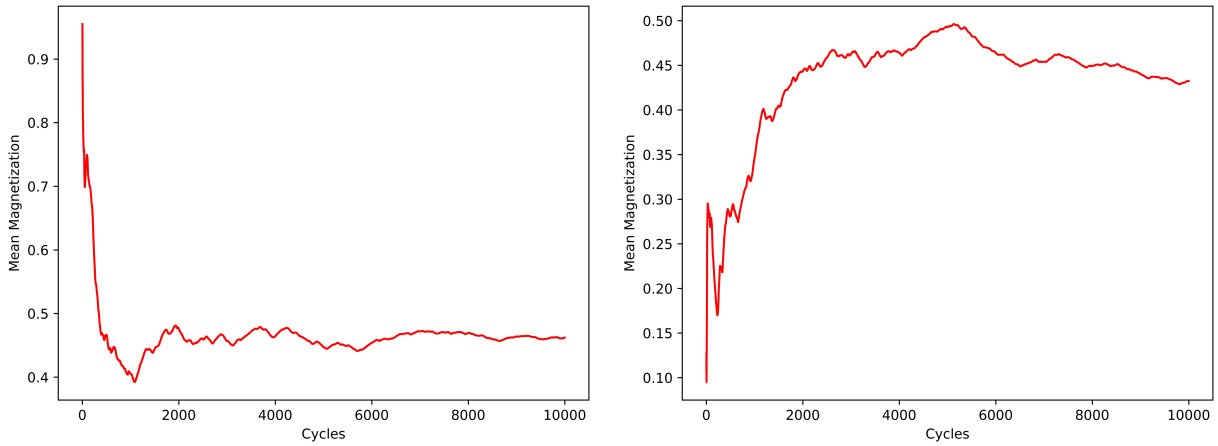


Figure 5: Mean Magnetization with T = 2.4 and 1e4 Monte Carlo Cycles Ordered (Left) and Random (Right)



The first comparison is between the random and ordered spin orientations with respect to mean energy. With a $T = 1.0$ as seen in *Figure 2*, it can be seen that the random spin orientation produced a smoother gradient slope towards the desired solution when compared to the ordered spin orientation. With a $T = 2.4$ as shown in *Figure* 3, both spin orientations appear to converge to the solution, but with greater noise. This can be attributed to the fact that the higher temperature (T) retains a slightly more unstable environment.

The second comparison is between the random and ordered spin orientations with respect to mean magnetization. The same effect can be seen in Figure 4 that was seen in *Figure 2* with a $T = 1.0$. The solution is stable in both and the random configuration initialized case converges quite smoothly. With a $T = 2.4$ the solution has a harder time maintaining stability, although manages to converge within reason as seen in *Figure 5*.

To reach an equilibrium situation for these cases, it would appear that for a 20 x 20 lattice with a $T = 1.0$, that 2000 Monte Carlo cycles would be appropriate. For a 20 x 20 lattice with a $T = 2.4$, that 4000 Monte Carlo cycles would be an acceptable stability. It is interesting to note that the temperature roughly doubles and the resulting amount of Monte Carlo cycles needed follows suit. This is the expected outcome and to verify the accepted configuration states, *Figures 6 & 7* were plotted for visual analysis.

Figure 6: Accepted Configurations with T = 1.0 and 1e4 Monte Carlo Cycles Ordered (Left) and Random (Right)

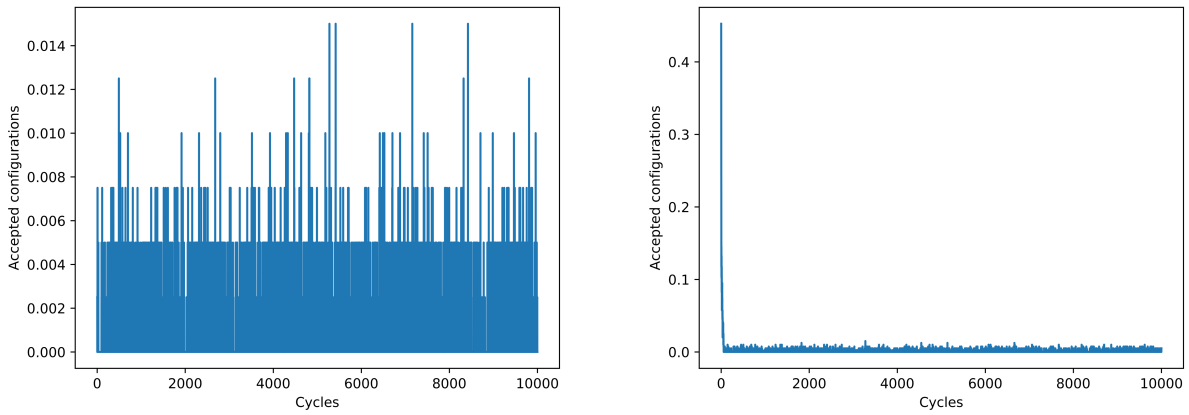
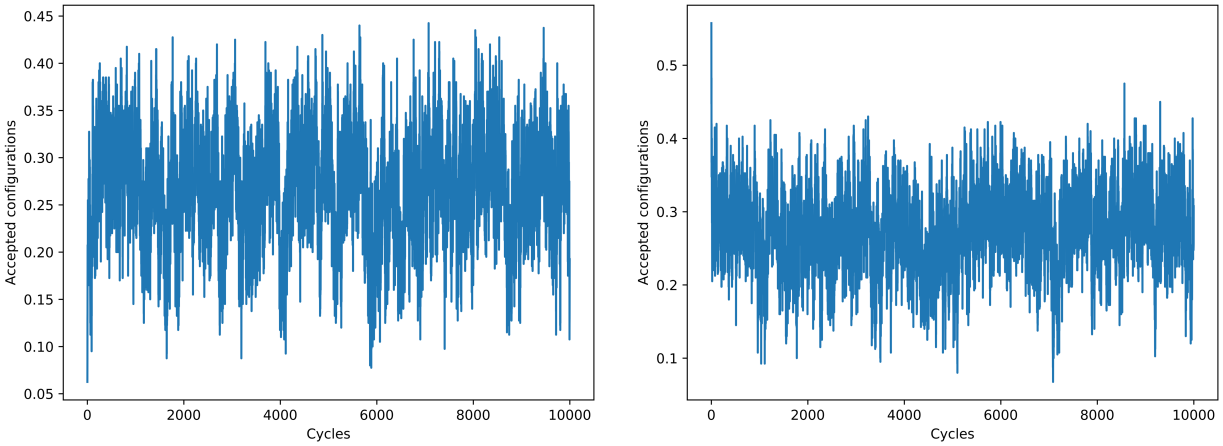


Figure 7: Accepted Configurations with T = 2.4 and 1e4 Monte Carlo Cycles Ordered (Left) and Random (Right)



When the equilibrium state has been reached, the expected rate of accepted configurations should decline. This is elegantly displayed in *Figure 6*, as the small number of accepted configurations for a $T = 1.0$ is predicted with a more stable lower temperature. When this temperature is raised to $T = 2.4$, the amount of accepted configuration drastically increases as shown in *Figure 7*. This is indicative of the large amount of chaos with respect to higher temperatures and the model currently being studied in the simulation.

4.4 Analyzing Probability Distribution

To look at the behavior of the L = 20 x 20 system, *Figures 8 & 9* were produced to analyze the probability distributions.

Figure 8: Ordered (Left) and Random (Right) Probability T = 1.0

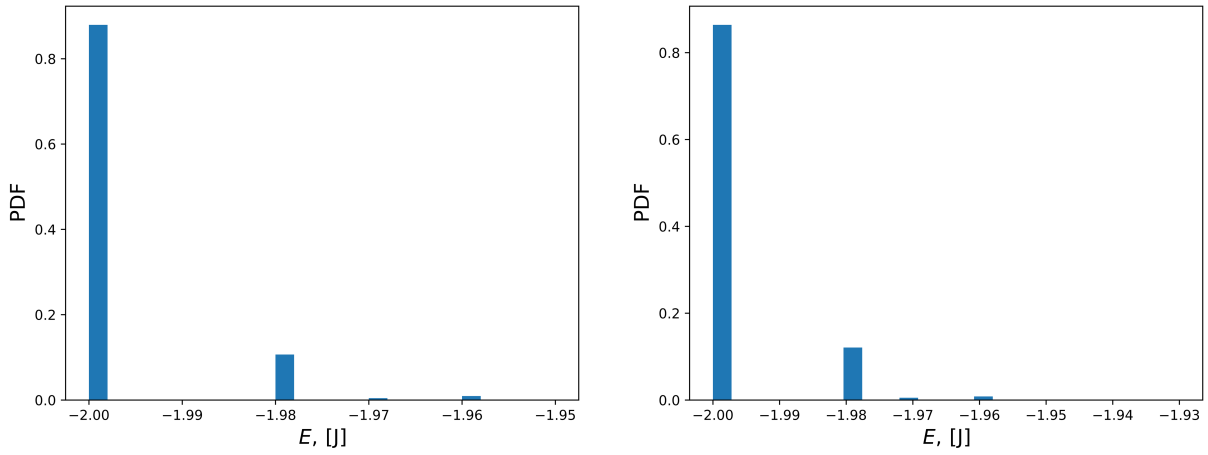
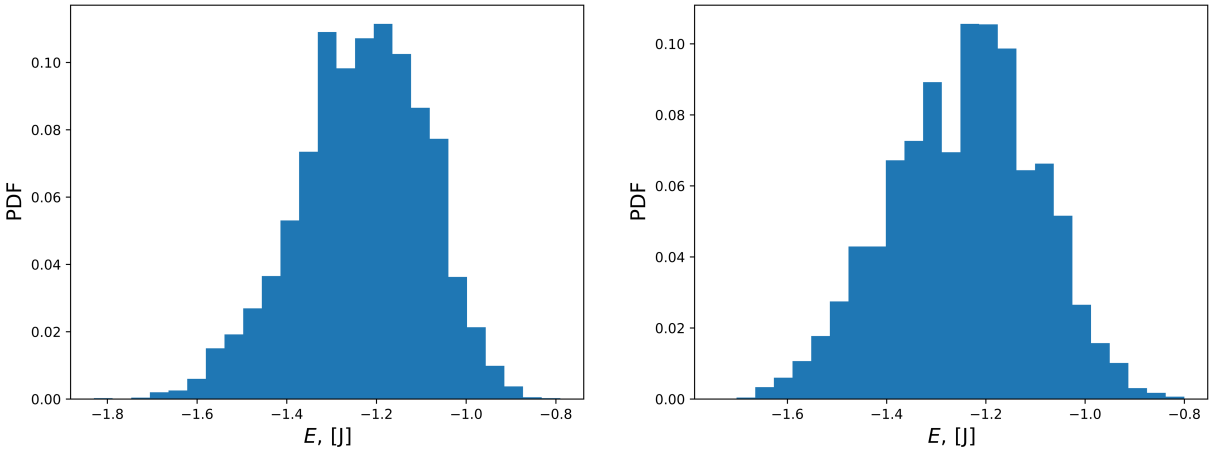


Figure 9: Ordered (Left) and Random (Right) Probability T = 2.4



It can be seen that the distribution of the energy states with $T = 1.0$ in *Figure 8* produce a right positive skew distribution in both ordered and random spin orientations. The distribution of energy states with $T = 2.4$ in *Figure 9* produces a normal distribution in both ordered and random spin orientations. To further analyze these results, the variance is calculated as shown in *Table 4*.

	Variance
Ordered, T = 1.0	0.00781
Random, T = 1.0	0.02611
Ordered, T = 2.4	0.14240
Random, T = 2.4	0.14346

Table 4: Calculated Variance for Probability Distributions

The calculated variance for $T = 1.0$ is much tighter due to the nature of the curve, but this is expected with the volatility of higher temperatures in this simulation such as $T = 2.4$ in this case. Upon inspection of *Figures 8 & 9* these values obtained are well within reason as being valid. It is worth noting that these values show the possibility of allowing a curve to be fit to the distribution for a varying computational mean energy.

4.5 Numerical Analysis of Phase Transitions

The algorithm developed was applied to larger lattice sizes of 40, 60, 80, and 100 in order to get the model as realistic as possible. For each lattice case the mean energy $\langle E \rangle$, mean magnetization $\mathcal{M}$, specific heat C_V, and susceptibility χ were plotted for visual analysis. The temperature range was set to be in the set [2,0, 2.3] with incremental steps of 0.05. These conditions resulted in *Figures 10 – 13* being generated.

Figure 10: Mean Magnetization vs Temperature with a range of Lattice Sizes

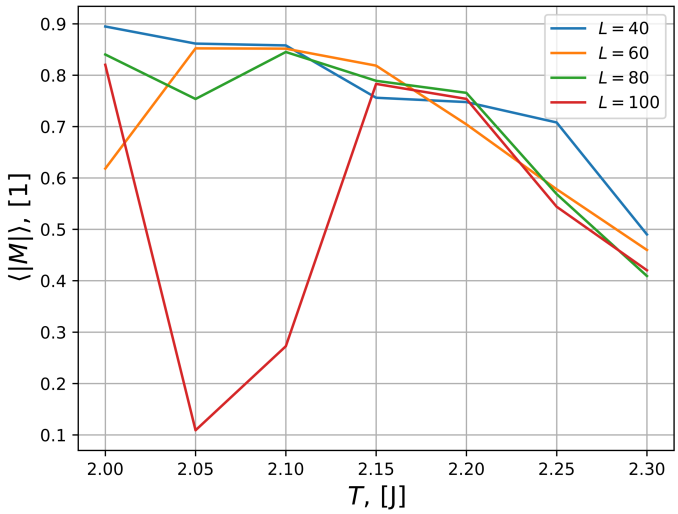


Figure 11: Mean Energy vs Temperature with a range of Lattice Sizes

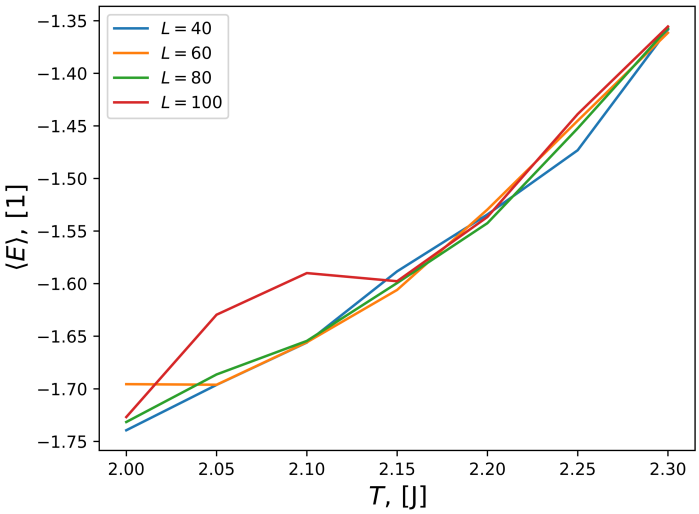


Figure 12: C_v vs Temperature with a range of Lattice Sizes

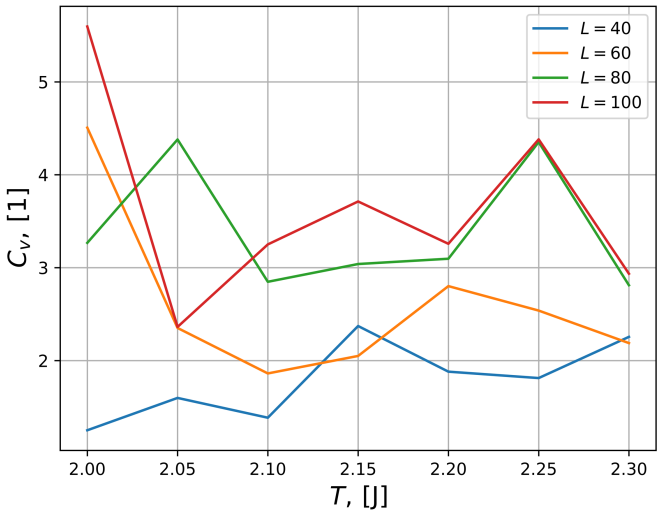
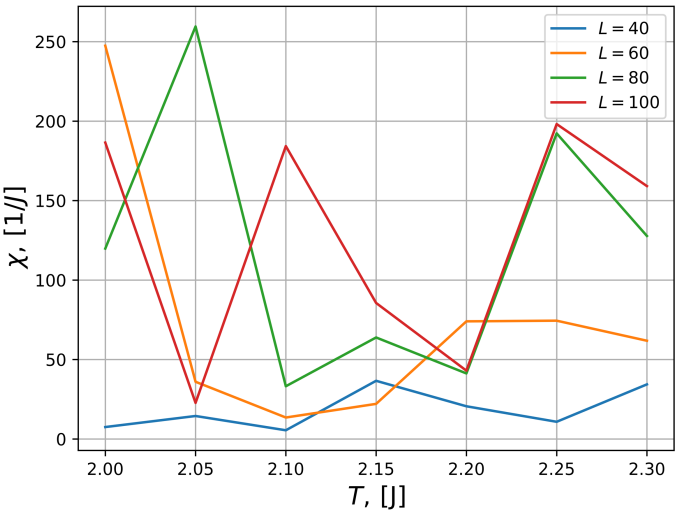


Figure 13: χ vs Temperature with a range of Lattice Sizes



In *Figure 10* the reducing mean magnetization is shown reducing with respect to an increasing temperature, which is the expected result. The energy is expected to rise with an increase in temperature and that is validated in *Figure 11*. There is a possible phase transition being seen in *Figure 10* with the lattice size of 100 x 100, which that same point lying on the x-axis (2.05) being indicated upon in *Figures 11-13*. To further refine these results a smaller temperature increment would be more optimal although the computational time drastically increases with these additional computations. The simulation for example already takes approximately 90 minutes to calculate a L = 100 x 100 matrix with a step size of T = 0.05. By reducing the temperature step size this would quickly balloon out of control.

4.6 Critical Temperature Validation

According to Onsager [2] the critical temperature is as described below in (18):

$$\frac{kT_C}{J} = \frac{2}{\ln(1+\sqrt{2})} \approx 2.269 \text{ with } \nu = 1 \tag{18}$$

Using:

$$T_C(L) - T_C(L=\infty) = aL^{-\frac{1}{\nu}} \tag{19}$$

Inserting L_i and L_j into (19) and subtracting, (20) can be obtained:

$$\begin{aligned} T_C(L_i) - T_C(L_j) &= a\left(L_i{}^{-\frac{1}{\nu}} - L_j{}^{-\frac{1}{\nu}}\right) \\ a &= \frac{T_C(L_i) - T_C(L_j)}{L_i{}^{-\frac{1}{\nu}} - L_j{}^{-\frac{1}{\nu}}} \end{aligned} \tag{20}$$

By selecting the global maximum for each L as described in the lattice set $L = [40, 60, 80, 100]$ it was computed that $a = 0.63$. Rearranging (19) concludes in the calculation of T_C as seen below in (21):

$$\begin{aligned} T_{C_\infty}(L) &= T_C(L) - \frac{a}{L} \\ &= 2.252 \end{aligned} \tag{21}$$

When comparing the values obtained in (18) and (21) it can be seen that they are within the error bound of 0.01 and this is an acceptable result. It is assumed that with a smaller temperature step size, this could be refined, but at the cost of a massive computational time.

5. Conclusion

Through this project, a successful simulation of the solar system was created with an object-oriented framework. When programming the algorithm, object orientation was implemented to ease code reusability and allow for smoother development. The two integrators initially chosen, Forward Euler and Velocity Verlet, were comprehensively tested against each other. These tests

showed that Forward Euler has the benefit of running two times the speed of Velocity Verlet. This however comes at the cost of very poor accuracy and an inability to hold orbital paths unless a tiny step size is chosen. With these weights taken into consideration, Velocity Verlet was chosen for all remaining tests, since the precision of the simulation was integral to the success of this project.

The conservation of angular momentum and energy were validated to be preserved within the simulation to a high degree of accuracy. The center of mass of the Sun was also observed to be stable across long periods of time. In an effort to observe a three-body system taken to the extreme, Jupiter's mass was altered until the system collapsed. This effect was interesting when Jupiter's mass has a multiplier of 1000 with the resulting mass being close to that of the Sun, resulting in them actually orbiting each other.

With the simulation verified to be running correctly, a complete model of the solar system was ran, which resulted in the expected orbits being produced. It was then observed that the precession of Mercury's perihelion can only be accurately explained with the introduction of general relativity. The project concludes in an effective simulation of the solar system using ordinary differential equations executed using an object-oriented structure.

6. Future Work and Thoughts

Since my passion is in robotics and their interactive design, this project gave me an idea. It is fairly trivial to calculate a navigational path using a circular path planning algorithm [3]. Another class I am taking here at UiO is INF3490 and we just completed a Machine Learning (ML) project. By combining ML with a weighting system for swarm robots, a collaborative network could be generated with rotating bodies being fused into the ML algorithm. Each planet being represented by a robot, they can generate circular paths through their relative orbits and weights being assigned by the ML to achieve the goal at hand. This could be for example a particular search pattern in rescue operations. If for example a certain planet finds a target, the weight in the ML would be increased, and the remaining "planet" robots would start being attracted to the circular path of the best performing robot.

I thoroughly enjoyed the project and found it to be enjoyable, yet challenging at the same time. One future idea to possibly implement into the project is allowing the students to create their own solar system to simulate a science fiction environment. This would entail creating unique planets and their associated properties to characterize unique orbital paths. I usually find that allowing for some creativity even in math and science heavy projects allows for students to show their individuality.

7. References

https://github.com/uqfoundation/pathos

Onsager, L., Crystal statistics. I. A two-dimensional model with an order-disorder transition, Physical Review, Series II, 65 (3-4), pp. 117-149 (1944).

[1] Hjorth-Jensen, Morten. "Project 3." *Computational Physics FYS3150*, Fall 2018. University of Oslo. Lectures.

[2] "HORIZONS Web-Interface." NASA, NASA, ssd.jpl.nasa.gov/horizons.cgi#top.

[3] Han, Sung-Min, et al. "Mobile Robot Navigation by Circular Path Planning Algorithm Using Camera and Ultrasonic Sensor." 2009 IEEE International Symposium on Industrial Electronics, 2009, doi:10.1109/isie.2009.5213204.