PHYS4070 Project 2: ODEs and Markov chain Monte Carlo

Semester 1 2023, 2 May

Hugh McDougall, 4302007

Contents

[Part 1 - Numerical Solution of a Simple N-Body Problem 2](#_Toc133024843)

[Part 1.1 – The Two Body Approximation 2](#_Toc133024844)

[Part 1.1.1 – Analytical Approximations 3](#_Toc133024845)

[Part 1.1.2 – Numerical Simulation of the Two Body Problem 4](#_Toc133024846)

[Part 1.2 – Three Body Problem 7](#_Toc133024847)

[Part 1.2.1 – Equation of Motion 7](#_Toc133024848)

[Part 1.2.2 – Code Structure 7](#_Toc133024849)

[Part 1.2.3 – Trajectories with Moon Influence 8](#_Toc133024850)

[Part 1.2.4 – Impact & Altitude with Moon Influence 9](#_Toc133024851)

[Part 1.2.4 – Derivation of Orbital Kick 9](#_Toc133024852)

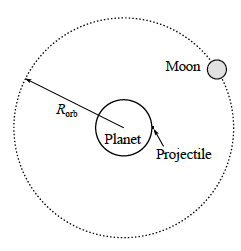
[Part 1.2.5 – Orbital Paths with Kick 11](#_Toc133024853)

[Part 1.3 (Bonus) Many-Body Demonstration 12](#_Toc133024854)

[Part 2 - Markov chain Monte Carlo for the 2D Ising Model 13](#_Toc133024855)

# 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 non- rotating)
* 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 , the projectile’s energy is conserved, making it a simple matter to estimate the purely radial velocity:

Where lowercase ‘r’ indicates orbital position and capital ‘R’ is the size of the object. Using natural units and noting :

For , and , this gives:

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

*Note: the analytical solution to the flight time, found by integrating , comes to time units, of similar order.*

We can also find the velocity required by the moon to maintain its stable circular orbit at by a simple force balance:

Corresponding to an orbital period of:

For the projectile’s and moon to meet at the correct time, this requires the moon to be ~ behind the encounter at time of launch. The initial position and velocity of the moon are then, in coordinates and for a counter-clockwise orbit:

Where , and . 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, containing the positions and velocities of all bodies:

The derivative of which is a function of its state:

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

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 :

Integration is then performed using the fourth order Runge Kutta method, a fixed step-size integrator which uses a nested series of estimates ofto minimize truncation error in each step:

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 fourth order global error (approximately). Taking the size of the projectile, , as a scale for significant position error, we can estimate a decent step size at :

Which we round to for neatness. For the simulation timescale estimated in part 1.1.1, this will require 1000’s of iterations per simulation, which is well within reason.

**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 function is calculated by f\_planet(). Both are stored in the forces\_and\_integrators.cpp file.

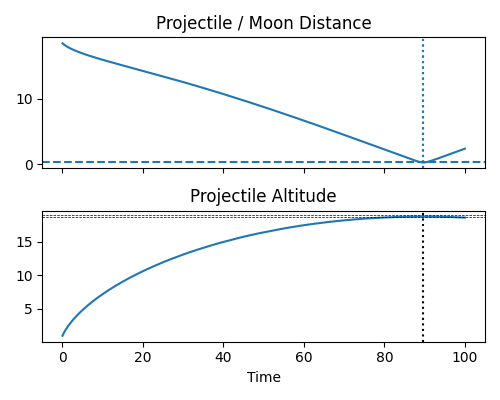
By default, all results are saved to the ‘./results’ folder. The ‘compile\_and\_run.sh’ shell script should generate results for all parts, which contains python scripts (e.g. \_plot\_1a.py) for producing the plots in the following section.

#### Results

Running our simulation for the given parameters up to , 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 |

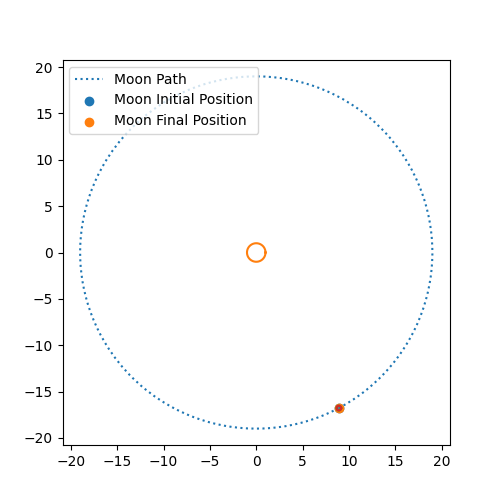
**Simulation Results**



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:

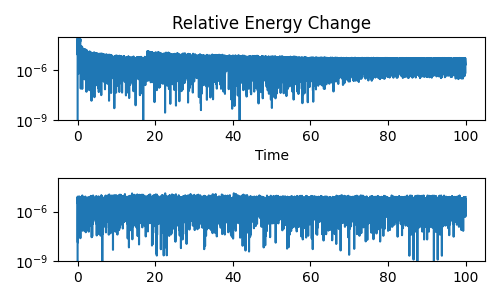
|  |  |
| --- | --- |
|  |  |
| **Orbital Paths Up to . Zoomed out (left) and zoomed in (right)** | |

We can validate the accuracy of our simulation by simulating the moon’s orbit for one full rotation, up to , a readily available analytical result. We can see the error in position is minor.



**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, , is of order .

  
**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:

But now with the force terms, stored in the second half of , having an extra term depending on the positions of all other bodies:

### 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 every 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 , 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);};

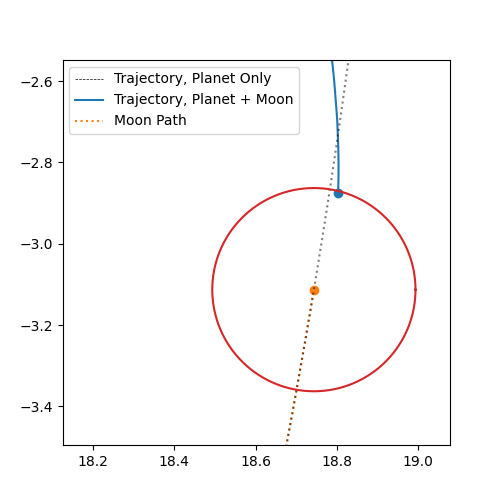
### Part 1.2.3 – Trajectories with Moon Influence

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 .

Entering the moon’s potential well causes increased velocities. At the time of impact () the planet-only trajectory would still be on approach to the moon’s orbit.

|  |  |
| --- | --- |
|  |  |
| **Projectile Trajectories With Moon Gravity** | |

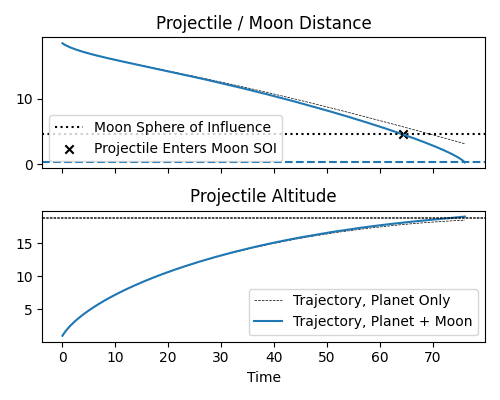
Another interesting features is that the extra accelerating causes the projectile to ‘slingshot’ slightly beyond the moon’s orbit, colliding while travelling almost anti-parallel to the moon.



**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 orbit (bottom figure).

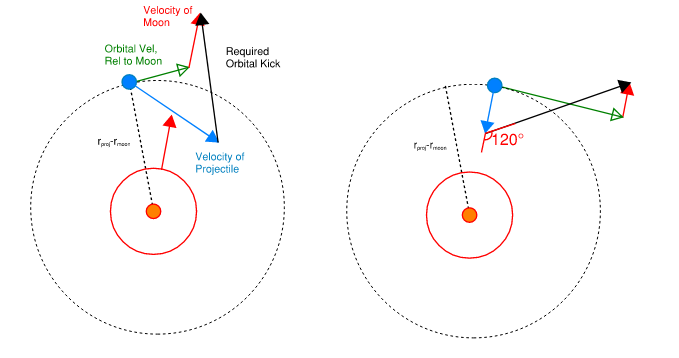


### Part 1.2.4 – Derivation of Orbital Kick

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 . To maintain a circular orbit, we need a relative velocity such that the centrifugal force counteracts the moons gravity:

We’re looking to orbit at for required orbital velocity:

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

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

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

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 . 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:

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:

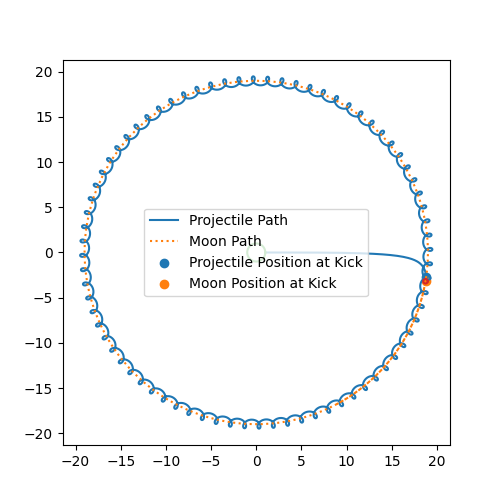
This represents a ‘worst case’, and so we expect a kick of

### 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 is smaller, and the angle sharper as we focus more on decelerating and less on accelerating into the required tangential velocity.

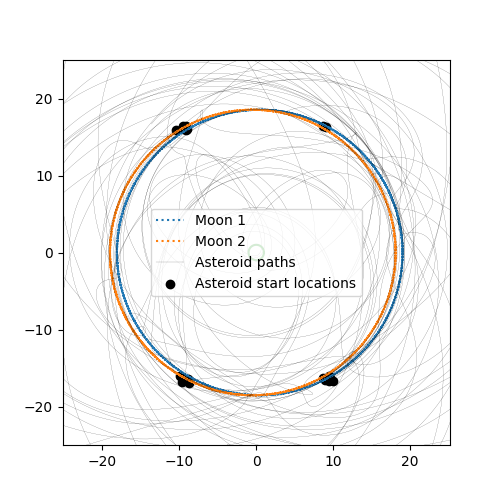
|  |  |  |
| --- | --- | --- |
|  | **Expected** | **Simulated** |
| **Time of Kick** | n/a | 75.74 |
| **Magnitude of Kick** | <0.891 | 0.757 |
| **Angle of Kick*,* Rel. To Prograde Dir.** |  |  |



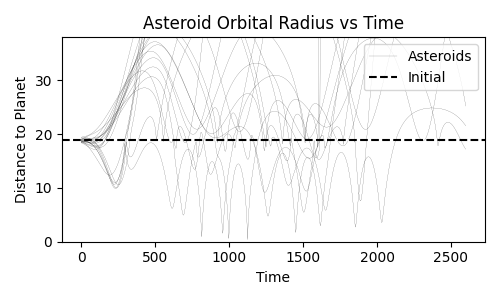
**Trajectory Post-Kick for One Moon Orbit**

## Part 1.3 (Bonus) Many-Body Demonstration

For the bulk of this report, we only have the (relatively) simple case of two significant masses, but our method can be scale up to show 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 ( from the moon at the same orbital radius). Parameters for this simulation are defined in manybody\_example.hpp.



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.



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

## Part 2.1 Thermodynamic Properties of the 2D Ising Model

-Plot of mean and var of

## Part 2.2 Critical Scaling Behaviour of the 2D Ising Model

## Part 2.3 OpenMP Implementation