Three-body simulation using numerical integration.

A dissertation submitted in partial fulfilment of the requirements for

B.Sc. Honours Physics.

Jacob Aniyan, 2023

Contents

[Abstract 4](#_Toc133579327)

[1. Introduction 4](#_Toc133579328)

[1.1 Background 4](#_Toc133579329)

[1.2 Chaos Theory and stability of system 4](#_Toc133579330)

[1.3 Historical Developments in the Three-Body Problem 5](#_Toc133579331)

[1.4 Numerical Integration Method 6](#_Toc133579332)

[1.5 Relevance today 8](#_Toc133579333)

[1.6 MATLAB vs Python 9](#_Toc133579334)

[1.7 Project Plan 9](#_Toc133579335)

[2. Specification 9](#_Toc133579336)

[2.1 Overview 9](#_Toc133579337)

[2.2 Functionality Required 10](#_Toc133579338)

[2.3 Restrictions 10](#_Toc133579339)

[3. Top level Design and Algorithms 10](#_Toc133579340)

[3.1 Overall Structure 10](#_Toc133579341)

[3.1.1 Overall Structure Flowchart 10](#_Toc133579342)

[3.2 ODE45 algorithm 11](#_Toc133579343)

[3.3 Motion function 11](#_Toc133579344)

[3.3.1 Motion function flowchart 12](#_Toc133579345)

[3.4 Plotting final positions 12](#_Toc133579346)

[3.4.1 Plot Flowchart 13](#_Toc133579347)

[4. Development and Testing 13](#_Toc133579348)

[4.1 Initial state of two-body system 14](#_Toc133579349)

[4.1.1 Implementation of initial state 14](#_Toc133579350)

[4.2 Two-body motion function development 14](#_Toc133579351)

[4.2.1 Motion function development flow chart 15](#_Toc133579352)

[4.2.2 Motion function implementation 15](#_Toc133579353)

[4.3 ODE45 implementation 16](#_Toc133579354)

[4.4 Plotting and animation of two body system 17](#_Toc133579355)

[4.5 Two-body system in 3D space 19](#_Toc133579356)

[4.5.1 3D position and velocity components 19](#_Toc133579357)

[4.5.2 3D plot 20](#_Toc133579358)

[4.6 Three-body system 21](#_Toc133579359)

[4.6.1 Three-body motion function 21](#_Toc133579360)

[4.6.2 Initial state of three-body system 22](#_Toc133579361)

[4.6.3 Three-body 2D plot 23](#_Toc133579362)

[4.6.4 Three-body 3D plot 24](#_Toc133579363)

[4.7 Testing 24](#_Toc133579364)

[4.7.1 Integration tolerances 25](#_Toc133579365)

[4.7.2 ODE function type 26](#_Toc133579366)

[4.7.3 Fixed vs variable timestep 26](#_Toc133579367)

[4.8 Centre of Mass 27](#_Toc133579368)

[4.9 Development overview 28](#_Toc133579369)

[5. Results and Analysis 28](#_Toc133579370)

[6. Discussion 30](#_Toc133579371)

[7. Conclusion 30](#_Toc133579372)

[8. References 31](#_Toc133579373)

[9. Appendix 32](#_Toc133579374)

[9.1 Two-body simulation in 2D Full Script: 32](#_Toc133579375)

[9.2 Two-body simulation in 3D Full Script: 34](#_Toc133579376)

[9.3 Three-body simulation in 2D Full Script: 36](#_Toc133579377)

[9.4 Three-body simulation in 3D Full Script 39](#_Toc133579378)

## Abstract

A two-body simulation was developed in MATLAB and used to build a three-body simulation. Each were first developed in 2D followed by 3D. An established method was also used in which the distances between the centre of mass (C.O.M.) and each body was plotted over time. This was analysed qualitatively for characteristics of ejection and chaos in three body systems of varying mass. These methods can be used in the field of celestial mechanics to investigate the formation of binary systems further and the relationship between chaos theory and the three-body problem.

# 1. Introduction

## 1.1 Background

Ever since Newton devised the laws of motion and the universal law of gravitation in his pivotal work, Principia, the question of describing the motion of three or more bodies has been vital for physicists concerned with celestial mechanics. An analytical solution can be found concerning an isolated two body system by considering the interaction between each body and the centre of mass. No such general solution exists for the three-body problem. However, extremely accurate solutions have been developed over the centuries, especially in recent decades as computational power has greatly increased. Finding accurate solutions to the n-body problem was essential when launching space missions within the solar system. These simulations helped to model star clusters, triple stellar systems, and the evolution of galaxies.

## 1.2 Chaos Theory and stability of system

The three-body problem or more generally the n-body problem is considered chaotic because although the system is deterministic, small variations in the initial variables of position and momentum can have drastic, unpredictable effects on the outcome. A statistical approach is therefore taken to predict the outcome of the system more accurately. For example, concepts such as phase space are useful when studying chaotic systems. Phase space is a representation of all possible outcomes a system can exhibit over time. In the case of the three-body system, this means all configurations of the position and momentum can be mapped out uniformly due to its pseudo-random properties. Using statistical methods, the probability of a system being in a particular part of a phase space can be found.

The stability of the system depends on various factors. For example, N-body systems with a greater range in mass disintegrate faster than bodies of similar mass (van Albada, 1968). Lower masses are ejected while the higher mass bodies remain in the system. This was one of the first studies to analyse the formation of binary star systems through disintegration of multiple bodies, which has important implication for the dynamics of star clusters. Although this study gave valuable insights into this matter, it was limited by the computational power of the late 60s and more accurate models of star clusters have since been produced.

Angular momentum also plays a major role in determining whether a system disintegrates or when one of the bodies are ejected and it was shown that almost all triple systems are dynamically unstable (Standish Jr, 1972). A study was able to classify six types of motion in a three-body simulation (Szebehely, 1971) and it has been shown that systems that exhibit “Lagrangian Equilibrium Configurations” (discussed later) and “Revolution” motion are considered stable. Revolution motion is essentially a binary system being orbited by a third body at a large enough distance away, where the binary acts as a “nucleus”. After the simulation ran for a long enough time, almost all the systems fell into the “Escape” class, where the ejected body followed a hyperbolic orbit around the binary and escaped orbit. Throughout the literature, there is consistent evidence that three body systems are almost always unstable and lead to ejection or escape. In light of this, a more recent study has looked at extracting useful information from these ejections (Stone and Leigh, 2019). By analysing the phase space of the “escape” class of three-body systems, regions where ejections were likely to occur were identified and used to predict the possible motions of the binary left behind. This has implications for the modelling of binary systems formed from triple star systems in star clusters, where ejections frequently occur.

## 1.3 Historical Developments in the Three-Body Problem

Euler made early developments on the three-body problem in 1762, where he was able to find the colinear positions of orbit in a three-body system. These are positions along the plane of the first two bodies, where a third body would be stable due to the resultant forces acting on it being balanced. As an important advancement in celestial mechanics, it was the first special case in which the three-body problem could be solved analytically and led to the use of L1, L2 and L3 points around the Sun-Earth orbit, where satellites are placed in stable, fixed orbits.

The restricted three-body problem was also produced by Euler in which one of the three bodies is assumed to have a negligible mass and moves within a two-body system. This is useful in any system where one of the masses are very low compared to the other two masses. For example, the Earth orbiting the Sun with Jupiter or satellites orbiting Earth as it orbits the Sun. Although useful in these applications, they are nonetheless only approximate solutions which fail when the masses are similar or when over a long enough time the low mass body will affect the system substantially.

Special cases of the three-body problem were found with analytical solutions. Lagrange points refer to fixed points in a circular orbit of a secondary body around a primary body in which a third body can inhabit in a stable orbit over time (Fitzpatrick, 2012). Stability of the third body in the system occurs due to the resultant forces of the other two bodies and centrifugal force being in balance (Bistafa, 2021). The first three points L1,L2 and L3 lie along the plane of the primary and secondary body while Lagrange points L4 and L5 form equilateral triangles with the other two masses (See Fig.1). These latter two points are found to be stable when the secondary mass is “less than about 4 percent” (Fitzpatrick, 2012) of the mass of the primary assuming the third mass that inhabits these points have negligible (or very low) mass. This is observed naturally in the solar system, where L4 and L5 points of the Sun-Jupiter orbit contains many asteroids called “Trojans” (Fitzpatrick, 2012). This was an important development in celestial mechanics which identified places in the Earth’s orbit to position spacecraft without using large amounts of fuel. These special cases are unfortunately limited and still fail to provide a general solution to the three-body problem.

A screenshot of a computer

Description automatically generated with medium confidence

**Figure** : Lagrange Points relating to Earth’s orbit.

Source: NASA/WMAP Science Team

Periodic orbits in the three-body system were also found. These are systems that return to their initial positions and momentums after a certain time. An interesting case was found where three bodies of equal masses orbit each other in a figure eight pattern (Chenciner and Montgomery, 2000). Many such periodic systems have been found since and provide good analytical solutions to special cases, however, these are rarely observed in nature and so provide little use practically.

## 1.4 Numerical Integration Method

Modern numerical integration methods to study the N-body problem were later developed in the 1960s (Sverre, 2003) and this is still used today as an accurate model of systems with ≥ 3 bodies. Various assumptions can be made about the system. For example, the bodies can be considered a point in space rather than having dimensions or they can also be modelled as 2D or 3D objects. Collisions can be ignored or accounted for, and masses can be equal or varied. All these parameters affect how the system behaves.

This method involves knowing the initial positions and velocities of each body and differentiating to find corresponding accelerations (Valtonen and Karttunen, 2006). Since the mass of each body is known, using Newton’s second law and gravitational law, the resultant force acting on each body can be found. At a time-step, 2 of the bodies are considered fixed, and the resultant force and hence motion of the third body is calculated using its initial position. This can be represented as a lower order Taylor series:

(1)

Where is the new position of the body, is the current position, is the resultant force (and hence acceleration) on the body and is the backward difference where:

(1.1)

This is the difference between the current resultant force and previous resultant force, **,** over the preceding time step (Valtonen and Karttunen, 2006). This lower order Taylor series is chosen because it saves on computational cost and as a consequence, allows a lower time step to be set providing greater accuracy.

Each body has its motion calculated sequentially while the other two are fixed in that time-step. In reality, the accelerations of each body are determined simultaneously. The source of error is therefore the time-step in which the other 2 bodies are fixed, where they would otherwise be moving. Reducing this time-step reduces the error in orbital motion, however, the computational time increases and so there needs to be a compromise between accuracy and time taken to compute the simulation. Increasing the number of bodies also increases the force polynomials involved, which further increases computation time (Sverre, 2003).

The equations of motion for the two-body problem can be found using Newtons 2nd law and law of gravitation:

For masses m1 and m2:

Equating (1) and (2):

(3) simplifies to:

In vector form:

The same is derived for mass 2:

These are the resulting accelerations for the two bodies. The new velocities can be found by integrating the accelerations and adding the initial known velocities:

The new positions can also be found using the new velocities:

These new positions can be plotted over each time step to produce a two-body simulation.

For the three-body problem, this method of integration to find the final velocities and positions remain the same, however the way in which the acceleration is calculated is different.

In this case, each body experiences two attractive forces from the other two bodies.

The notation for force used here will refer to the mass on which the force is exerted on. For example, F12 is the force exerted on mass 1 by mass 2.

Consider a three-body system with masses m1, m2 and m3. For m1, Newton’s second law and law of gravitation can be expressed.

The total force acting on m1 by the other two massescan be expressed as follows.

This can be simplified to find the acceleration of m1.

For m2 and m3 the same can be derived.

In an N-body problem, increasing the number of bodies increases this computational cost by N2 (Greengard, 1990). The rapid advance of computational power from the 1960s to present has remedied this issue, however it still imposes limitations on the larger astrophysical systems that can be modelled such as large star clusters or galaxies. Further developments were made to reduce computational power using the Barnes-Hut algorithm, which reduced the computational cost from N2 to N log N (Barnes and Hut, 1986). This was achieved by dividing space into “tree-structured” cubic cells where each cell which contains more than one body is split into eight sub-cells. Essentially, this reduced a particular group of N bodies in a space to a centre of mass approximation. Although a compromise on accuracy was made, this greatly reduced the computational cost which had implications not only for the modelling of large astrophysical structures, but other simulations modelling N particles.

To save on computation time, the “individual time-step method” was implemented (Aarseth 1971). For each body, a different time-step could be used at various points along the orbit. Generally, a faster change in acceleration requires a smaller time-step and vice-versa (Valtonen and Karttunen, 2006).This greatly reduces computational power for a small loss in accuracy and is therefore a valuable method especially with high N value systems and bodies that exhibit highly elliptical orbits.

## 1.5 Relevance today

Centuries after Newton to this day, there has not been a general analytical solution for the N-body or three-body problem. However, it has been shown that accurate approximations using numerical integration and the recent advancements in computational simulations have served just as useful in observation and modelling of celestial mechanics. These methods have given insight into the properties of this chaotic system, revealing patterns in a seemingly random process. They have also proven to be an instrumental tool in space missions, which furthered the field of astrophysics in a fundamental way.

The N-body problem remains relevant today with new developments being made as recently as three years ago. As bigger structures in the universe are modelled, there will be a greater demand placed on these simulations when the N value goes up. For example, simulating the collision of the Andromeda galaxy with the Milky Way (Cox and Loeb, 2008) required careful consideration in the number of particles (N) as a limitation in computation. A more recent study (Panamarev et al, 2019), was able to simulate one million bodies when simulating the centre of the Milky Way in high resolution. As computational power and efficiency increases with time, physicists will be able to model larger astrophysical structures in greater detail. This has implications in the field of cosmology with the study of the structure of the universe and shows it is a worthwhile endeavour to address computational limitations and efficiency methods.

## 1.6 MATLAB vs Python

At the start of the project, I opted to use python for my three-body simulation. However, after 2 weeks I decided to switch to MATLAB. My previous experience with the language was advantageous and it is particularly useful for arrays, which is involved in simulating motion 2D and 3D space. MATLAB also includes built in ODE solvers whereas on python they need to be installed.

## 1.7 Project Plan

I will organise my work by starting with the simulation of the two-body problem with fixed masses. In this simulation, I will need to use the centre of mass as fixed reference frame and use Newton’s laws of motion and universal law of gravitation. This should not be as difficult to simulate since there is already an analytical solution. This will also allow me to build the programming techniques needed later. Using the assumptions of bodies stated in the previous section, I can build on the two-body simulation to create Euler’s restricted three-body simulation because the same principles will apply. I will model the Sun-Earth-Jupiter system to scale and compare these results with previous simulations to check its validity.

If this restricted three-body system is valid, I can move on to the numerical integration approach by implementing equations (1) and (1.1). At first, this will be fixed, point-like masses on a 2D plane and no collisions. Following this, I can make properties such as position, mass and velocity of each body variable. These variables are important for the aim of my project because adjusting these will allow me to study the possible emergent systems of order.

Once the three-body numerical integration simulation is produced, I can progress onto an N-body system. The same variables will be adjustable as before, except the complexity of the computations will now increase. I will need to experiment at this stage with the maximum N value I can use practically. The greater this N value is, the better the range I will be able to study. I will be able to study how emergent order arises differently in different values of N. If, however I am not able to transition from the three-body to N-body simulation or if I am not able to get sufficiently high N values to notice a change in patterns, I will limit the scope of my experiment to just a three-body or low N-body system and change the other variables such as mass, position or velocity. If this does not allow me to explore order in chaotic systems sufficiently, I could also take an alternative project route by changing the gravitational constant and observing the effects on the system. If Python does not seem feasible to learn within the first couple weeks, I will switch to MATLAB.

# 2. Specification

## 2.1 Overview

The purpose of this software is to simulate a three-body gravitational system with physical parameters controlled by the user. Mass, initial positions and velocities of the three bodies can be set by the user along with the volume of space in which they appear. The user selects the duration of the simulation and is provided with an animation of the motion of the three bodies in either 2D or 3D space. Centre of mass is also animated, and the user can access the distance from it to each body over time. Collisions will be ignored, and the objects will be treated as point masses.

This software will allow for easy investigation of chaoticity in three body systems, however, will require some background knowledge in MATLAB programming language to control the initial parameters.

## 2.2 Functionality Required

The requirements for the simulation will be characterised by the features that allow the user to observe orbits as they change over time. The programming language used to build this software will be MATLAB and will be compatible on Windows and MacOS.

* The user must be able to set a volume or area of space which will be represented by a graph. This must graph will plot the motion of the objects over time as an animation.
* The user must be able to set the initial positions and velocity components of the three bodies according to the dimensions of the simulation and each object must have a defined mass set by the user in kilograms. Each mass must be distinguishable by colour.
* User must be able to set a time duration in which the simulation will run.
* Centre of mass must also be displayed and animated on the graph, distinguishable from the bodies.
* Simulation must calculate the distance from centre of mass to each body over time duration. These must be stored as three separate variables (for each mass) with a distance value for each timestep.

## 2.3 Restrictions

Objects will be assumed to be point masses, so will not have any dimensions and collisions will not be possible. This simplifies the code and takes less computational power to process. The equations of motion will also be simplified since conservation of momentum, mass and energy do not need to be accounted for. Many celestial mechanics simulations use point masses because the radius of celestial objects is much smaller than the distances between them and so it can be assumed to be negligible. It must be noted that this assumption may not be accurate during very close encounters.

The software is also restricted in simulating direct collisions and will fail when it encounters a singularity in the solution. For example, if two objects are directly headed for each other, the simulation will stop just before they meet. However, the margin for this is very small and direct collisions are rare.

# 3. Top level Design and Algorithms

## 3.1 Overall Structure

The overall structure of the three-body simulation will be built on the two-body simulation. Firstly, an initial set of variables which define the gravitational constant, mass values and total mass followed by sets of arrays which describe the initial positions and velocity of each body. This will be compiled into an initial state vector, which will be input to the numerical integrator ODE45 MATLAB function along with timespan and the motion function. The function takes time and the state vector as inputs and through calculation of acceleration, finds the change in velocities over time for each body as output. Final positions of each body are defined along with centre of mass and these values are plotted inside a for loop which runs to the time set. This plot is paused every 0.01s to create an animation of the motion of the three bodies and centre of mass.

### 3.1.1 Overall Structure Flowchart

Diagram

Description automatically generated

Figure (3.1)

## 3.2 ODE45 algorithm

The ODE45 algorithm in MATLAB is based on the Runge-Kutta 4th order numerical integration method (MATLAB doc1). It is considered a versatile ordinary differential equation (ODE) solver and is recommended as the first option to try. The motion function, which calculates acceleration, is input into the ODE45 function along with timespan and the initial state vector. At each step, it determines what the optimal step size should be to minimize error and hence uses variable step sizes. This means when objects are close together with high rate of acceleration, it will take smaller step sizes to minimise the increase in error and larger step sizes when objects are further away. It provides an efficient way to use numerical integration while giving accurate solutions. Tolerances for error can be set by the user and input into the ODE45 function.

## 3.3 Motion function

The role of the motion function is to calculate the acceleration using newton’s law of gravitation and compile it with the initial velocities into a new state vector, which can be integrated over a timestep using the ODE45 solver. It first defines the positions and velocities of each body in the state vector y. The distances between each body are then found using the norm function. These distances are used to calculate the accelerations derived earlier. Finally, the accelerations and the initial velocities are compiled into a new state vector called “dydt”.

### Diagram Description automatically generated3.3.1 Motion function flowchart

Figure(3.2)

## 3.4 Plotting final positions

The final positions can be called and defined after using the ode45 solver. These are then plotted in a for loop which has the same duration as the timespan which was set earlier. Each body is represented as a circle and the axes are defined. Finally, the for loop is paused for a given time to animate the plot so the user can observe the motion of the objects.

### 3.4.1 Plot Flowchart

Diagram

Description automatically generated

Figure(3.3)

# 4. Development and Testing

This section will detail the development and testing of the code from the two-body problem to the three-body problem. The latter was largely built on the former, with the key difference being in the motion function. Namely, the acceleration calculation that was derived earlier and additions to the initial state vector. Therefore, the development of the two-body system in 2D will be discussed first, followed by the same system in 3D. Then, the three-body system development in 2D and 3D. The numerical integration method remained the same for both simulations. Much of the development for the two-body system was taken by existing code ([orbital mech 1]), although it had been altered especially for developing the three-body system.

## 4.1 Initial state of two-body system

The initial state of the system comprises of mass, position and velocity variables compiled in an array called the initial state vector. Gravitational constant is defined and set as global along with mass variables as these will be used in the motion function later. Total mass is also defined to be later used in calculation of COM. The user inputs values for mass in kg; position and velocity are in the form of arrays corresponding to 2 or 3 dimensions. Lastly, the timespan can also be set by the user in the initial state.

### 4.1.1 Implementation of initial state

A simple 2D case of the two-body problem was considered. Two masses with positions and velocities were defined in arrays which contained x and y components.

Text

Description automatically generated

Figure(4.1.1)

This sets up the variables which will be compiled into the initial state vector, y0, as the positions followed by velocity arrays of each body. It organises each component of position and velocity to be called later when working with acceleration calculations and finding distance between objects.



Figure(4.1.2)

The timespan was defined as 0 to t in steps of 0.1.



Figure(4.1.3)

Originally, the timespan had no fixed timestep, which resulted in the animation of the plot slowing down when the objects came close together. This is due to the nature of the ode45 algorithm which takes smaller timesteps as the objects travel closer together. The fixed time step of 0.1 was a later addition for the sake of accurately animating the objects. Later testing shows that there is no compromise on accuracy when doing this, but there are less time steps taken in total.

## 4.2 Two-body motion function development

Before the ode45 algorithm could be used, an ode function needed to be created. Essentially, the ode function serves to call initial positions and velocity components from the initial state vector. The distance between the two bodies is then calculated and used to find the acceleration of each body. These accelerations are compiled into an output array along with the initial velocities, and this is input into the ode45 algorithm, which integrates the acceleration over the time step to find the new velocity and positions.

### 4.2.1 Motion function development flow chart

Diagram

Description automatically generated

Figure(4.2.1)

### 4.2.2 Motion function implementation

The function in fact has two inputs. These are the state vector, y, and the current time, t. However, since ‘t’ is not used in the function itself, it is ignored and replaced with a placeholder ‘~’ instead as is MATLAB convention. The purpose of the function is to lay out how velocity and position change over time using the acceleration equations derived earlier or the two-body problem.

The global variables for the gravitational constant and masses were defined. This allowed the use of these variables in both the function and in the main script.

Text, table

Description automatically generated

Figure(4.2.2)

Next, the components of the state vector y were defined. Since this is a 2D simulation to begin with, the x and y components of positions of each body were from 1 to 4 and for velocities 5 to 8:

Text

Description automatically generated

Figure(4.2.3)

The vector form calculation for accelerations in equation () and () were implemented into the function. However, the distance between the objects needs to be defined and used in this equation.

A picture containing graphical user interface

Description automatically generated

Figure(4.2.4)

This is found using the norm function in MATLAB, which finds the magnitude of the displacement between both masses as r. Acceleration of each mass can then be found, where R2-R1 is the vector displacement between mass 2 and mass 1 and vice versa. Finally, the initial velocities v1, v2 and the accelerations a1, a2 are compiled into the final state vector, ‘dydt’. This output is the input to the ode45 function and solved over time, t.



Figure(4.2.5)

The motion function is a key part of the simulation and is last of the three inputs required to use the ode45 numerical integrator. Since it would only be called once in the script, it was decided to leave the function at the end rather than save it as a separate file.

## 4.3 ODE45 implementation

MATLAB has useful documentation on the ode45 numerical integrator [MATLAB doc 1]. It has three inputs, which are the motion function (odefun), initial state vector (y0) and timespan (tspan). It also has two outputs, which are the current time (evaluation points), t, set as a column vector and the solution state vector, y.

The general form of the function is shown below in F(fig.4.3.1)



Fig(4.3.1)

The same format was used in the case of this simulation.

Text

Description automatically generated

Although the plotting function had not been written yet, this provided a set of results which can be tested. The output variable y produced a 301x8 double data type with initial conditions described in earlier.

Text, letter

Description automatically generated

Fig(4.3.2)

The table in fig(4.3.3) shows the first 8 timesteps and their corresponding solutions. Initial timestep and conditions are shown on the first row.

Table

Description automatically generated

Fig(4.3.3)

Each column corresponds to a component in the state vector and each row is a timestep, with 300 timesteps being taken after the first row. The positions of each body over time are the first four columns and the velocities of each body are the last four.

## 4.4 Plotting and animation of two body system

After results had been obtained for final positions and velocities of each body in the state vector y, the former had to be defined. The final positions are plotted over time on a 2D plane, where the motion of the bodies can be observed visually. For this to be achieved, the plot had to be inside a for loop, where the length of the for loop was equal to the output time from ode45 (i.e., the length of the timespan). The plot would only show the final state of the system however, without a pause function within the loop. This is essential for the animation aspect of the simulation.

Final positions of the state vector y were called and defined. The centre of mass of the system, Rm was also defined.

A picture containing diagram

Description automatically generated

Fig(4.4.1)

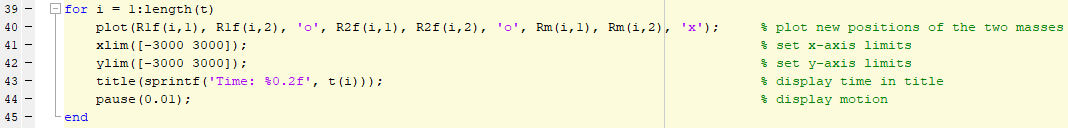
A figure was also created, and its dimensions set.

Text

Description automatically generated with medium confidence

Fig(4.4.2)

The for loop could then be written. The index, i, was set from 1 to the length of t and the first statement was the plot function. Each component of each body at i was plotted as a circle, while the centre of mass components was plotted as an ‘x’ mark. The x and y axis limits were then set, and the title of the figure printed the current time index. Finally, the pause function was set to 0.01s. This controls the speed of the animation and how smooth it appears.



Fig(4.4.3)

The for loop proved successful in plotting the motion of the two-body system over time. A screenshot of the plot at time 2.20 can be seen in Fig(4.4.4). This concludes the development of the two-body system in a 2D plane. Developing it into a 3D simulation largely involved changing the components of the array and the plotting segment of the code.

Although the plot is simple, it is valuable to the user to see visually how the initial system evolves so they can change the initial state and investigate the orbital mechanics of the system. It is also another key part of the specification that has been met.

Chart, scatter chart

Description automatically generated

Fig(4.4.4)

## 4.5 Two-body system in 3D space

To introduce an extra dimension, two key aspects of the code needed to be changed. Firstly, the position and velocity components needed an extra z component, which affects how they are called in the state vector. Secondly, a new plot function was required which called all three components of each body. This section will detail the development of adding an extra dimension to the simulation and much of this development carried over to changing the 2D simulation of the three-body system into 3D.

### 4.5.1 3D position and velocity components

Each position and velocity array gained an extra z component. Although the state vector remained the same, the number of components that made it up had now changed from 8 to 12.

Text, letter

Description automatically generated

Fig(4.5.1)

Calling the final position had also been adjusted to account for this.



Fig(4.5.2)

In the motion function, the calling initial positions and velocities were adjusted in the same way, however, the acceleration calculation remained the same.

Text

Description automatically generated

Fig(4.5.3)

### 4.5.2 3D plot

MATLAB has a 3D plot function called plot 3d. The final positions were plotted the same way as in the 2D plot, but with an extra z coordinate and corresponding time index. Centre of mass was treated the same way.

Text

Description automatically generated

Fig(4.5.4)

When running the code, the simulation seemed to work successfully in 3D. Unfortunately, it was difficult to distinguish the plot as 3D without grid lines, and so these were later added by using the ‘grid on’ setting.

Chart

Description automatically generated

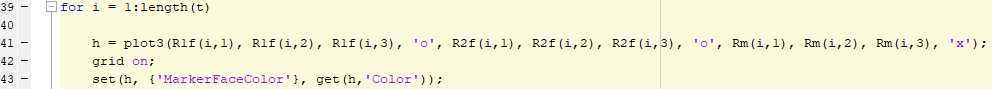
Fig(4.5.5)

The circles were also filled by setting the plot equal to a variable and setting the marker face to their respective colours.

Chart, radar chart

Description automatically generated

Fig(4.5.6)



Fig(4.5.7)

The two-body system had successfully been developed into a 3D model. The same method of development was used in the three-body system.

## 4.6 Three-body system

As shown in the derivations earlier, the acceleration calculation for the three-body system was more complicated than that of the two-body. The motion function was therefore the first to be developed. For simplicity, a 2D simulation was first built and later converted to 3D. Changes were then made to the initial conditions and state vector to introduce the third body with mass, m3. Finally, the final position of the third body was defined and plotted along with the other two bodies as another circle.

### 4.6.1 Three-body motion function

A third body will have its own position and velocity components that will need to be called and defined in the function. This means the state vector will now have 12 components in total. New variables R3 and v3 were defined in the function.

Text

Description automatically generated

Fig(4.6.1)

These will later be defined outside the function in the set of initial state variables.

Instead of the one r value that was used for the two-body system, there were 6 distinct values that were required for the acceleration calculation. Each body has two r values associated with it, which are the displacements between it and the other two bodies. The norm function was still used here, and the notation refers to which body the distance is taken from. For example, “r12” refers to the distance from m1 to m2.

Text

Description automatically generated

Fig(4.6.2)

These distances were used respectively in the acceleration equations derived earlier, which were implemented into the function.

Letter

Description automatically generated with medium confidence

Fig(4.6.3)

The output state vector ‘dydt’ was altered to match the input state vector y.



Fig(4.6.4)

No changes were made to the ode45 function since the same numerical method is being used on more variables.

### 4.6.2 Initial state of three-body system

The initial state was next altered to include the third body. Mass, position, and velocity were set, and the initial state vector was extended to include these. An additional final position had to be defined after the ode45 function and the centre of mass calculation also had to be adjusted to include the third body.

Firstly, the mass of the third body was introduced along with its addition to the total mass. Then, its initial position and velocity components were set. The variables were all included in the initial state vector, which now contained 12 components.

Text, letter

Description automatically generated

Fig(4.6.5)

The extra two position components in the output state vector were called and defined as “R3f”. An extra term (m3\*R3f) was also added to the C.O.M. calculation.

Text

Description automatically generated with medium confidence

Fig(4.6.6)

### 4.6.3 Three-body 2D plot

Including the final position of the third body into the plot function was relatively simple, as it was added the same way the other two bodies were. It was set as another circle, which MATLAB assigned a distinct colour to automatically.

Text

Description automatically generated with medium confidence

Fig(4.6.7)

Chart, scatter chart

Description automatically generated

Fig(4.6.8)

### 4.6.4 Three-body 3D plot

The three-body system was next developed into a 3D plot using the same method used to convert the two-body system. Position and velocity variables were given an extra component, and this time the state vector had a total of 18 components. The new initial state of the system is shown in Fig(4.6.9)

Text

Description automatically generated

Fig(4.6.9)

The final positions were defined to match this arrangement of components.

Text

Description automatically generated with medium confidence

Fig(4.6.10)

Components were extended within the function too, while the acceleration calculation remained the same.

Text

Description automatically generated

Fig(4.6.11)

Finally, the “plot3” function was used again and the components were extended on the final positions in the same way. Fig(4.6.12) shows the portion of the function that plots the third body and centre of mass.



Fig(4.6.12)

## 4.7 Testing

Three key areas of the three-body system were tested. These are integration tolerances, ODE function type and fixed vs variable timestep. Integration tolerances served to set limit on the relative and absolute tolerances of the numerical integration. This affected aspects such as collisions, where singularities would be encountered in the solutions and how they were dealt with. ODE functions were investigated to find which was most suitable to the three-body problem because different types provided different levels of accuracy. Timestep by default is variable in the ode45 function. However, this produced slower animations with inconsistent frame rates. This was compared with the fixed timestep, which provided a smoother animation, but produced less data.

### 4.7.1 Integration tolerances

Integration tolerances were required when the simulation encountered singularities. This occurs in “stiff” problems, where at certain points in the solution there are sharp discontinuities [MATLAB doc 2]. It corresponds to rapid changes in acceleration when the masses encounter at very close proximity. At certain proximities, the masses encounter singularities in the solution. Since they were treated as point masses, the acceleration tends to infinity, where there otherwise would be a collision. Relative and absolute tolerances are two types of errors set by the integral solver. Relative tolerance sets the accuracy relative to the solution value, while absolute tolerance sets the accuracy that any value in the solution can take [Stan 1]. These are low enough to maintain high accuracy but high enough to avoid singularities where the simulation fails.

Implementation of these error tolerances was carried out using the ‘odeset’ function [MATLAB doc 3]. An ‘options’ variable was first created, and it contained the odeset function. Relative and absolute tolerances could be set within the function and the variable was included within the ode45 function.

Text

Description automatically generated with medium confidence

Fig(4.7.1)

Lower tolerance limits produced more accurate solutions. These values were chosen after some testing.

Without these settings, the simulation would sometimes fail completely when objects came very close to each other. After implementing this setting, most warning messages that stopped the simulation due to singularities ceased, however, direct collisions remained a point of failure. The conditions for direct collisions were set by the initial conditions shown in fig(4.7.2). This was done in 2D for simplicity, but the same issue applies to the 3D system.

Text, table

Description automatically generated

Fig(4.7.2)

This creates a symmetrical system, where the objects fall directly into each other. The timespan was set to 0:0.1:50. A warning message appears and completely stops the simulation at t = 20.40s. This could not be fixed; however, it is worth noting that direct collisions such as in this case are rare.

Increasing the x component of the velocity of mass 1 by only 1 ms-1 and keeping all else same produced no warning message, and the simulation ran with no issue.

When deciding what relative and absolute tolerance to set, the last position coordinates of mass 1 were tested. The initial conditions were controlled, with the y velocity component of mass 1 being 1, but all else being the same as in Fig(4.7.2). The last 5 position x-y position coordinates were recorded and compared (see Fig(4.7.3)). These conditions of the system were ideal because accuracy is most important in close encounters of the masses, where large changes in acceleration occurs.

1.Table

Description automatically generated 2. Table

Description automatically generated 3.Table

Description automatically generated

Fig(4.7.3)

Tables 1,2 and 3 show the last 5 positions of mass 1 with integration tolerances of (RelTol = 1E-8, AbsTol = 1E-10), (RelTol = 1E-10, AbsTol = 1E-12) and (RelTol = 1E-12, AbsTol = 1E-14) respectively.

Significant differences in final positions were observed. It was found that the setting of RelTol = 1E-10, AbsTol = 1E-12 provided the best compromise. Higher tolerances gave less accurate results and lower tolerances increased computation time significantly. It was also found that decreasing the tolerances even further (to RelTol = 1E-14, AbsTol = 1E-16) reintroduced the warning message and stopped the simulation.

### 4.7.2 ODE function type

MATLAB has different built in ODE solvers that are suitable for different problems. ODE45 is considered the most versatile and was first used in this code. Other types were also tested for accuracy when encountering the warning for integration tolerances at singularities. ODE15s and ODE23s are suited for stiff problems that require low accuracy. ODE45 is suited to non-stiff problems and provides relatively higher accuracy. The “stiffness” of a problem refers to the amount of variation in nearby solutions. For example, when the masses have a close encounter with each other, acceleration varies slowly at first and then varies rapidly in a short amount of time. This could be considered a stiff problem. The differences in the ODE solvers are most apparent during these close encounters as this is when accuracy in the solution is most important.

I set the initial state as in fig(4.7.2) but with the v1i array had 1 for its y component instead of 0. The integration tolerances were constant at RelTol = 1E-6, AbsTol = 1E-8 and since this test was carried out at an earlier stage in the development, the gravitational constant, G, is 6.67E-11 instead of 6.6743E-11. ODE45, ODE15s and ODE23s were tested by recording the last five positions of mass 1.

1.Table

Description automatically generated 2.Table

Description automatically generated 3.Table

Description automatically generated

Fig(4.7.4)

Tables 1,2 and 3 show the last 5 positions of mass 1 using ode45, ode23s and ode15s respectively.

All three runs of the code left one body ejected with the other two bodies forming a binary system. Interestingly, mass 1 was in the opposite side of the axis when using ode15s and much further away than the other two runs.

### 4.7.3 Fixed vs variable timestep

ODE45 uses variable timestep by default. This means during the animation the objects appear to slow down as they gravitate closer together and speed up as they move away. As acceleration increases at a faster rate, the timesteps reduce and each frame covers a shorter distance. Adding a fixed timestep fixed this issue. However, in total there were significantly less timesteps recorded because they were constant regardless of how close the masses encountered each other. Objects were now appearing to move faster as they gravitated towards each other as you would expect in a real system. Accuracy was not reduced significantly as testing shows, the last positions of masses that had close encounters were identical.

A variable time step corresponded with an array for 0 to t with no specified time step. A fixed time step involved an array with a specified time step. For example, 0:0.1:t is from 0s to ts in steps of 0.1.

The test was done for 0 to 15s with variable and fixed timesteps of 0.1. The last 5 positions of mass 1 were recorded.

1.Table

Description automatically generated 2. Table

Description automatically generated with medium confidence

Fig(4.7.5)

Table 1 and 2 show the last 5 positions of mass 1 using variable and fixed timesteps respectively.

Less timesteps were taken with the fixed timesteps. Further testing showed that in systems with closer encounters, more timesteps were taken than in systems where masses are further apart. Unfortunately, in very close encounters such as the initial state conditions used to test the ODE types, the simulation takes extremely long to complete to the point where it effectively stops, because so many timesteps are being taken. Due to these reasons, a variable time step of 0.1 was decided to be appropriate for users to see the interaction in the animation. Higher fixed time steps resulted in even fewer total timesteps taken, and the simulation would end quickly, however these may be useful when observing systems over a long time span.

## 4.8 Centre of Mass

Centre of mass was calculated over each timestep. This provided a quantitative method of analysing the chaoticity of the system. The distance between each body and the centre of mass was plotted against time. The figure was hidden, however, to not interfere with the animation, but the data for each body was saved as a variable and could be plotted on excel.

To find the distance between each body and the C.O.M. over time, the norm function was not sufficient. Instead, the ‘vecnorm’ function had to be used.

A screenshot of a computer

Description automatically generated with medium confidence

Fig(4.8.1)

The difference between the final position of each body and the final position of the C.O.M. is found as a positive scalar set by the first ‘2’ following the vector difference. The second ‘2’ sets which dimension the positive scalar is to be found through. ‘2’ represents rows while ‘1’ represents columns. Since the position values were expressed in rows this was set as 2.

Next, a hidden figure was created, where these distances would be plotted against the current time step, t.



Fig(4.8.2)

Each of the ‘Rcom’ variable produced a set of distances over the time step, which could then be plotted on excel.

## 4.9 Development overview

The three-body system successfully meets the specification detailed earlier. The system is very flexible and allows a variety of avenues to approach the problem. Studying how the distance between each body and the centre of mass changes over time has potential to provide insights into the chaoticity of the system.

In retrospect, a user interface could have been developed to make this program accessible to a wider range of audience. At present, a certain level of knowledge in MATLAB and the classical mechanics is required to both operate the program and gain any utility out of it. A more intuitive way to set the initial conditions could be programmed in the future. For example, the masses could be dragged and dropped into their initial positions and their velocity components could be represented by an arrow which could be adjusted to the desired direction and amount.

# 5. Results and Analysis

Distances between each body and the centre of mass were taken over time and plotted on excel. Various initial states were studied, and the time taken for the three-body system to lead to ejection was observed. Time of periodicity of the system was also observed. The error in distance measurements is equal to the error in the final position. These correspond to the integration tolerances, where the absolute error in the solution is ±1E-12.

The majority of three body systems of equal masses result in an ejection of one mass leaving a binary system. However, systems where one mass is one or two orders of magnitudes different produce interesting results.

For initial conditions:

Text

Description automatically generated



The following data was produced.

Fig(5.1)

This shows the distance from m1 to the C.O.M. over time. The orbit exhibits periodicity.

Fig(5.2)

This shows the same system with the distances of all three masses to the C.O.M. overlayed on one plot over time.

The same system was studied. This time all three masses were equal, and an ejection of a third body occurred. All other variables in the initial state stayed the same.

Text

Description automatically generated



The following data was produced for all three masses.

Fig(5.3)

Graph shows distance between each body and C.O.M. over time. Mass 2 is ejected after around t = 60s.

# 6. Discussion

Unfortunately, the results and analysis section is not extensive in the data recorded due to time restraints. However, valuable insights were made into the three-body problem regarding the nature of ejections and its link with mass. Even a difference of only one magnitude in mass for one of the bodies was enough to produce a relatively stable, periodic system. Within this periodicity there are still signs of chaotic behaviour. Small and unpredictable fluctuations can be observed in fig(5.1) when mass 1 is closer to the C.O.M.

Using a larger time step will allow the observer to speed the simulation and use much longer timespans. However, the larger the timestep is, the more difficult it is to plot the distances of the masses over time. Certain characteristics of the plot correspond to its physical behaviour. For example, when the data points are more spread out, the mass is accelerating faster. This is because in a given fixed time step its final position is further away from the centre of mass than previous steps.

For the same initial state, but with equal masses, the system experienced an ejection of mass 2. Chaos seems to be associated with the three masses coming closer to the centre of mass at the same time. This can be seen in fig(5.3), where there is high fluctuation in the periods after the times in which all three masses are closer to the C.O.M. At each of these points, a new mass seems to be furthest in no particular order from the C.O.M. until one of them is completely ejected. This is contrasted with the periodic orbit seen in Fig(5.2), where when mass 2 and 3 are closer to the C.O.M., mass 1 is further away and vice versa. More investigation is needed to confirm this, however.

# 7. Conclusion

A three-simulation was successfully produced, which can be used with sufficient background knowledge of MATLAB and classical mechanics. This has many applications in celestial mechanics for studying the formation of binary systems in particular. A method of studying the chaotic characteristics of these systems inspired was also developed. Limited data was produced and analysed qualitatively. Although the three-body problem has been extensively studied in the past especially in the advent of computer simulations in the past decades, there is potential for future investigations into its relationship with chaos theory.

# 8. References

* Aarseth, S. (1971). Direct Integration Methods of the N-Body Problem. *International Astronomical Union Colloquium, 10*, pp. 118-132
* Barnes, J. and Hut, P. (1986). A hierarchical O(NlogN) force-calculation algorithm. *nature*, 324(6096), pp.446-449
* Bistafa, Sylvio R. (2021). Euler's three-body problem, *Euleriana*: *1*(2), p. 181, Article 6.
* Chenciner, A. and Montgomery, R. (2000) A remarkable periodic solution of the three-body problem in the case of equal masses. *Annals of Mathematics*, pp. 881-901
* Cox, T.J. and Loeb, A., 2008. The collision between the Milky Way and Andromeda. *Monthly Notices of the Royal Astronomical Society*, *386*(1), pp.461-474.
* Figure 1: <https://solarsystem.nasa.gov/resources/754/what-is-a-lagrange-point/>
* Fitzpatrick, R. (2012). *An Introduction to celestial mechanics*. Cambridge: Cambridge University Press.
* Greengard, L. (1990). The numerical solution of the n-body problem. *Computers in physics*, 4(2), pp. 142-152.
* Panamarev, T., Just, A., Spurzem, R., Berczik, P., Wang, L. and Arca Sedda, M., 2019. Direct N-body simulation of the Galactic centre. *Monthly Notices of the Royal Astronomical Society*, *484*(3), pp.3279-3290.
* Standish Jr, E.M. (1972). The dynamical evolution of triple star systems. *Astronomy and Astrophysics, 21,* p.185.
* Sverre, J.A. (2003) *Gravitational N-Body Simulations*. New York: Cambridge University Press.
* Szebehely, V. (1971). Classification of the motion of three bodies in a plane. *Celestial Mechanics, 4*, pp.116-118
* Valtonen, M. and Karttunen, H. (2006). *The Three-Body Problem.* Cambridge: Cambridge University Press.
* van Albada, T.S., (1968). Numerical integrations of the N-body problem*. Bulletin of the Astronomical institutes of the Netherlands, 19,* p479
* [MATLAB doc 1] <https://uk.mathworks.com/help/matlab/ref/ode45.html>
* [MATLAB doc 2] <https://uk.mathworks.com/help/matlab/math/solve-stiff-odes.html>
* [MATLAB doc 3] https://uk.mathworks.com/help/matlab/ref/odeset.html
* [orbital mech 1] <https://orbital-mechanics.space/the-n-body-problem/two-body-inertial-numerical-solution.html>
* [Stan 1] <https://mc-stan.org/docs/2_22/stan-users-guide/control-parameters-for-ode-solving.html>

All websites last accessed 28/04/2023.

# 9. Appendix

## 9.1 Two-body simulation in 2D Full Script:

Text

Description automatically generated

A picture containing timeline

Description automatically generated

In Text:

close all;

clear all;

%% Initial variables

%{

G = Gravitational Constant

m1 and m2 = mass (in kg) of first and second object respectively.

R1 and R2 = positions of masses (in m)

v1 and v2 = velocities (in ms^-1) of the masses

a1 and a2 = accelerations (in ms^-2) of the masses

%}

global G

global m2

global m1

G = 6.6743E-11;

m1 = 5E+17;

m2 = 10E+17;

M = m1+m2;

R1i = [-2000 0];

R2i = [-1000 0];

v1i = [20 -70];

v2i = [45 100];

y0 = [R1i R2i v1i v2i]; %initial state vector

tspan = 0:0.1:10;

[t,y] = ode45(@motion, tspan, y0); % this solves the set of ODEs in the motion function over tspan for state vector y0

R1f = y(:, 1:2);

R2f = y(:, 3:4);

Rm = (m1.\*R1f+m2.\*R2f)./M;

f = figure; % create a new figure

f.Position(1:2) = [170 140]; % figure size config

f.Position(3:4) = [840 630];

for i = 1:length(t)

plot(R1f(i,1), R1f(i,2), 'o', R2f(i,1), R2f(i,2), 'o', Rm(i,1), Rm(i,2), 'x'); % plot new positions of the two masses

xlim([-3000 3000]); % set x-axis limits

ylim([-3000 3000]); % set y-axis limits

title(sprintf('Time: %0.2f', t(i))); % display time in title

pause(0.01); % display motion

end

function dydt = motion(~,y)

global G

global m1

global m2

R1 = y(1:2); % define y state vector components

R2 = y(3:4);

v1 = y(5:6);

v2 = y(7:8);

r = norm(R2-R1); % distance between masses

a1 = G.\*m2.\*(R2-R1)./r.^3; % acceleration derived from F = ma for both masses

a2 = G.\*m1.\*(R1-R2)./r.^3;

dydt = [v1; v2; a1; a2]; % Change in velocities over time

end

## 9.2 Two-body simulation in 3D Full Script:

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

In Text:

close all;

clear all;

%% Initial variables

%{

G = Gravitational Constant

m1 and m2 = mass (in kg) of first and second object respectively.

R1 and R2 = positions of masses

v1 and v2 = velocities (in ms^-1) of the masses

a1 and a2 = accelerations (in ms^-2) of the masses

%}

global G

global m1

global m2

G = 6.6743E-11;

m1 = 10E+17;

m2 = 7E+17;

M = m1+m2;

R1i = [0 0 -45];

R2i = [1000 30 -20];

v1i = [25 0 0];

v2i = [-30 50 30];

y0 = [R1i R2i v1i v2i]; %initial state vector

tspan = 0:0.1:30; % time span of the simulation

[t,y] = ode45(@motion, tspan, y0); % this solves the set of ODEs in the motion function over tspan for state vector y0

R1f = y(:, 1:3);

R2f = y(:, 4:6);

Rm = (m1.\*R1f+m2.\*R2f)./M;

f = figure; % create a new figure

f.Position(1:2) = [170 140]; % figure size config

f.Position(3:4) = [840 630];

for i = 1:length(t)

h = plot3(R1f(i,1), R1f(i,2), R1f(i,3), 'o', R2f(i,1), R2f(i,2), R2f(i,3), 'o', Rm(i,1), Rm(i,2), Rm(i,3), 'x'); % plot new positions of the two masses and COM

grid on;

set(h, {'MarkerFaceColor'}, get(h,'Color'));

xlim([-3000 3000]); % set x-axis limits

ylim([-3000 3000]); % set y-axis limits

zlim([-3000 3000]); % set z-axis limits

title(sprintf('Time: %0.2f', t(i))); % display time in title

pause(0.01); % displays motion

end

%% Acceleration Calculation

function dydt = motion(~,y)

global G

global m1

global m2

R1 = y(1:3);

R2 = y(4:6);

v1 = y(7:9);

v2 = y(10:12);

r = norm(R2-R1); %distance between masses

a1 = G.\*m2.\*(R2-R1)./r.^3; %acceleration derived from F = ma for both masses

a2 = G.\*m1.\*(R1-R2)./r.^3;

dydt = [v1; v2; a1; a2]; %Change in velocities over time

end

## 9.3 Three-body simulation in 2D Full Script:

A picture containing text

Description automatically generated

Text

Description automatically generated with medium confidence

In text:

close all;

clear all;

%% Initial variables

%{

G = Gravitational Constant

m1, m2, m3 = mass (in kg) of first, second and third object respectively.

R1, R2, R3 = positions of masses

v1, v2, v3 = velocities (in ms^-1) of the masses

a1, a2, a3 = accelerations (in ms^-2) of the masses

%}

global G

global m1

global m2

global m3

G = 6.6743E-11;

m1 = 10E+16;

m2 = 10E+16;

m3 = 10E+16;

M = m1 + m2 + m3;

R1i = [0 0]; % initial conditions

R2i = [1000 1000];

R3i = [-1000 -1000];

v1i = [0 1];

v2i = [20 0];

v3i = [0 40];

y0 = [R1i R2i R3i v1i v2i v3i]; % initial state vector

tspan = 0:0.1:50; % time span of the simulation

options = odeset('RelTol', 1e-6, 'AbsTol', 1e-8); % sets integration tolerances to avoid singularities

[t,y] = ode45(@motion, tspan, y0, options); % this solves the set of ODEs in the motion function over tspan for state vector y0

R1f = y(:, 1:2); % final positions

R2f = y(:, 3:4);

R3f = y(:, 5:6);

Rcom = (m1.\*R1f + m2.\*R2f + m3.\*R3f)./M; % centre of mass

Rcom1 = vecnorm(R1f - Rcom, 2, 2); % distance from centre of mass to each object

Rcom2 = vecnorm(R2f - Rcom, 2, 2);

Rcom3 = vecnorm(R3f - Rcom, 2, 2);

figure('visible','off'); % the Rcom(1,2,3) distances are recorded over each timestep

plot(t, Rcom1, t, Rcom2, t, Rcom3);

f = figure; % create a new figure

f.Position(1:2) = [170 140]; % figure size config

f.Position(3:4) = [840 630];

for i = 1:length(t)

h = plot(R1f(i,1), R1f(i,2),'o', R2f(i,1), R2f(i,2),'o', R3f(i,1), R3f(i,2),'o', Rcom(i,1), Rcom(i,2),'x'); % plot new positions of the three masses and COM

grid on;

xlim([-3000 3000]); % set x,y axis limits

ylim([-3000 3000]);

title(sprintf('Time: %0.2f', t(i))); % displays time in title

pause(0.01); % displays motion of three bodies over time

end

%% Acceleration Calculation

function dydt = motion(~,y)

global G

global m1

global m2

global m3

R1 = y(1:2); % define position components in state vector y

R2 = y(3:4);

R3 = y(5:6);

v1 = y(7:8); % define velocity components in state vector y

v2 = y(9:10);

v3 = y(11:12);

r12 = norm(R1-R2); % distances between each object

r13 = norm(R1-R3);

r21 = norm(R2-R1);

r23 = norm(R2-R3);

r31 = norm(R3-R1);

r32 = norm(R3-R2);

a1 = -G.\*m2.\*(R1-R2)./r12.^3-G\*m3.\*(R1-R3)./r13.^3; % acceleration derived from F = ma for both masses

a2 = -G.\*m3.\*(R2-R3)./r23.^3-G.\*m1.\*(R2-R1)./r21.^3;

a3 = -G.\*m1.\*(R3-R1)./r31.^3-G.\*m2.\*(R3-R2)./r32.^3;

dydt = [v1; v2; v3; a1; a2; a3]; % change in velocities over time

end

## 9.4 Three-body simulation in 3D Full Script

In text:

close all;

clear variables;

%% Initial variables

%{

G = Gravitational Constant

m1, m2, m3 = mass (in kg) of first, second and third object respectively.

R1, R2, R3 = positions of masses

v1, v2, v3 = velocities (in ms^-1) of the masses

a1, a2, a3 = accelerations (in ms^-2) of the masses

%}

global G

global m1

global m2

global m3

G = 6.6743E-11;

m1 = 10e17;

m2 = 10e17;

m3 = 10e17;

M = m1 + m2 + m3;

R1i = [0 0 0]; % initial conditions

R2i = [500 420 200];

R3i = [-500 -420 -200];

v1i = [65 80 -78];

v2i = [100 -120 -30];

v3i = [-150 90 50];

y0 = [R1i R2i R3i v1i v2i v3i]; % initial state vector

tspan = 0:0.1:100; % time span of the simulation

options = odeset('RelTol',1e-10,'AbsTol',1e-12); % sets integration tolerances to avoid singularities

[t,y] = ode45(@motion, tspan, y0, options); % this solves the set of ODEs in the motion function over tspan for state vector y0

R1f = y(:, 1:3); % final positions

R2f = y(:, 4:6);

R3f = y(:, 7:9);

Rcom = (m1.\*R1f + m2.\*R2f + m3.\*R3f)./M; % centre of mass

Rcom1 = vecnorm(R1f - Rcom, 2, 2); % distance from centre of mass to each object

Rcom2 = vecnorm(R2f - Rcom, 2, 2);

Rcom3 = vecnorm(R3f - Rcom, 2, 2);

figure('visible','off'); % the Rcom(1,2,3) distances are recorded over each timestep

plot(t, Rcom1, t, Rcom2, t, Rcom3);

f = figure; % create a new figure

f.Position(1:2) = [170 140]; % figure size config

f.Position(3:4) = [840 630];

for i = 1:length(t)

h = plot3(R1f(i,1), R1f(i,2), R1f(i,3),'o', R2f(i,1), R2f(i,2), R2f(i,3), 'o', R3f(i,1), R3f(i,2), R3f(i,3),'o', Rcom(i,1), Rcom(i,2), Rcom(i,3), 'x'); % plot new positions of the three masses and COM

grid on;

set(h, {'MarkerFaceColor'}, get(h,'Color')); % fill circles with colour

xlim([-12000 12000]); % set x,y,z axis limits

ylim([-12000 12000]);

zlim([-12000 12000]);

title(sprintf('Time: %0.2f', t(i))); % display time in title

pause(0.01); % displays motion of three bodies over time

end

%% Acceleration Calculation

function dydt = motion(~,y)

global G

global m1

global m2

global m3

R1 = y(1:3); % define position components in state vector y

R2 = y(4:6);

R3 = y(7:9);

v1 = y(10:12); % define velocity components in state vector y

v2 = y(13:15);

v3 = y(16:18);

r12 = norm(R1-R2); % distances between each object

r13 = norm(R1-R3);

r21 = norm(R2-R1);

r23 = norm(R2-R3);

r31 = norm(R3-R1);

r32 = norm(R3-R2);

a1 = -G.\*m2.\*(R1-R2)./r12.^3-G\*m3.\*(R1-R3)./r13.^3; % acceleration derived from F = ma for both masses

a2 = -G.\*m3.\*(R2-R3)./r23.^3-G.\*m1.\*(R2-R1)./r21.^3;

a3 = -G.\*m1.\*(R3-R1)./r31.^3-G.\*m2.\*(R3-R2)./r32.^3;

dydt = [v1; v2; v3; a1; a2; a3]; % change in velocities over time

end