Computation Of Galaxy Structure Formation Using Runge-Kutta Methods

April 2019

Abstract. Though Newton's laws have been known for over 300 years, in the instance of many body problems analytical solutions to these equations have remained elusive. With the advent of silicon-based computing in the 1960s and 70s however physicists had a new tool at their disposal to investigate such previously intractable problems. Investigation of the formation and behaviour of cosmic structures is one such area where breakthroughs were made soon after the introduction of this tool.

This report describes the implementation of an analytical solution to such a many body problem, specifically applying the developed algorithm to compute the formation of Galaxy tails. A variable-step size fourth order Runge-Kutta method was used and it was shown that many of the observable features of Galaxies could be reproduced using a reduced N-body problem, where test-particles only felt a force due to a small number of large perturbing galaxy masses.

Introduction. Toomre and Toomre [2] demonstrated the importance of gravity in affecting the formation of cosmic features. Specifically, they showed that many features of pairs of galaxies can be explained as relics of close encounters of the two in the past. Before this paper it was widely thought that tidal effects were to blame for features such as bridges and tails, therefore this paper was as significant step forward in astronomy.

The fourth-order variable step-step method implemented in [2] was replicated for this report, and the findings, that a number of galaxy features can be produced via close encounters of galaxy centres, were reproduced. Specifically, spiral and tail structures of galaxies were shown to be produced by the close elliptical approach of a perturbing galaxy, and the characteristics of the produced features were shown to depend on the closest approach of the perturbing mass and the magnitude of the perturbing mass.

Analysis. The motion of bodies under the influence of gravity is described by Newton's law of gravitation:

$$F = \frac{GMm}{r^2}$$

By integrating the acceleration as determined by the force above, and similarly integrating the velocity, the motion of a body after a given time period can be calculated exactly as:

$$\int \frac{F}{m} dt = v$$

$$\int v dt = x$$

In the non-infinite limit, these integrals can be expressed as:

$$\Delta v = \frac{F}{m} \Delta t$$

$$\Delta x = v \Delta t$$

These equations can straightforwardly be implemented to determine the motion of bodies under gravity as:

$$v_{i+1} = v_i + \Delta v = v_i + \frac{F_i}{m} \Delta t$$

$$x_{i+1} = c_i + \Delta x = x_i + v_i \Delta t$$

This is Euler's first order method, in which the force and velocity at the i^{th} step are used to calculate the $i + 1^{th}$ velocity and position by multiplying by the time interval. This is a promising starting point, as it gives a method which at least in principal can total describe a system of N bodies interacting via gravity. In practice however there are a number of issues which make implementation of Euler's method for N-bodies impractical.

Primarily, the scaling of N-body problems very soon becomes intractable if the gravitational interaction of every body with every other body is included. In particular, if every gravitational force is accounted for, the number of force calculations required per ith step in an algorithm scales as N(N-1). Fortunately, systems can often be described accurately by only including the gravitating action of the most massive bodies. The orbits of stars in a galaxy is one such system, where, since the mass of a galaxy is approximately $10^{12} M_{solar}$, the motion of an individual stars is overwhelmingly determined by the attraction of the central galactic mass:

$$F_{centralmass} = \frac{GM_{solar} \times 10^{12}}{10^5} \approx M_{solar} 10^7$$

$$F_{nearestneighbour} = \frac{GM_{solar}}{5} \approx M_{solar}$$

If the closest neighbour distance for stars is taken to be on the length scale of light years, which is approximately five orders of magnitude smaller than the distance of stars to the centre of the galaxy, the force due to the concentrated mass at the centre of a galaxy is much greater than nearest neighbour attraction.

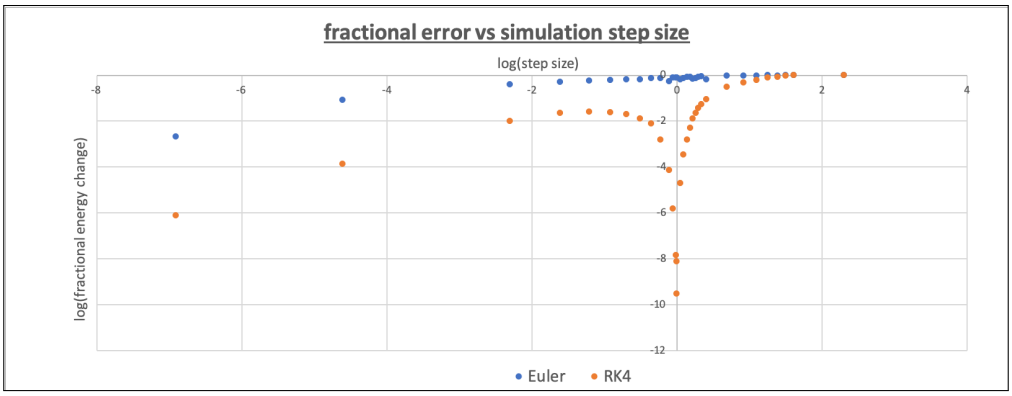


Figure 1: Plot showing the deviation of the computed solution from the exact solution for a circular orbit after a fixed amount of simulated time, in terms of energy. Physically there should be no change in energy with time for a circular orbit, so a lower fractional change is a more accurate solution.

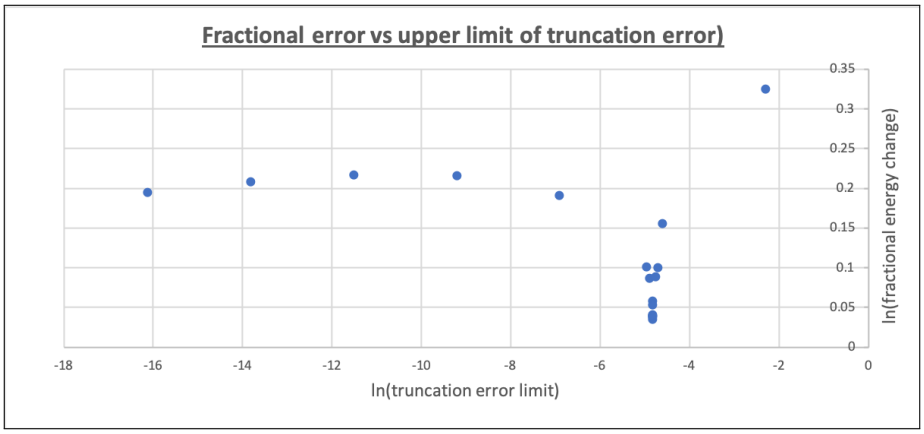


Figure 2: Fractional energy change after 2000 units of time vs upper truncation error limit.

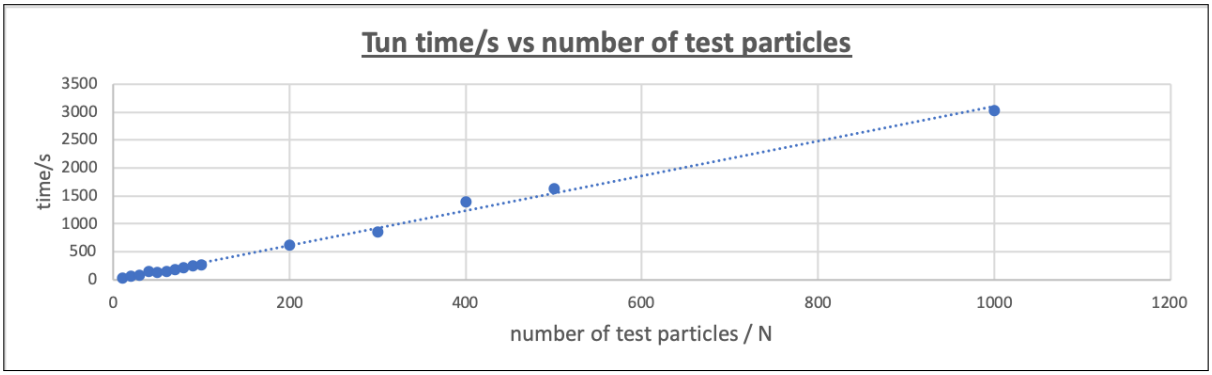


Figure 3: Run time vs number of simulated test particles for adaptive step size program.

If only the gravitational force from a few massive bodies in a system is accounted for the problem scales as nN, where there are n massive bodies and N 'testbodies.' This model, known as a reduced body problem, is implemented in this project.

The scaling of Euler's method is also hindered by the requirement of a small step size, Δt. In simple cases with a small number of test bodies N this is not a problem, however in problems that require N to be of order 10^2 or 10^3 the necessity of a small Δt for Euler's method leads to impractical running times for simulations. In these cases it becomes necessary for larger step sizes to be used if systems are to be probed in an efficient and accurate manner.

The Runge-Kutta, or 'RK' method provides a suitable improvement upon an Euler algorithm. In particular it is possible to achieve a fourth order method using RK (RK4), which means the error is proportional to the step size raised to the fourth power—much better than Euler's method in which the error is proportional to step size to squared. This therefore allows for a much larger step size to be used.

Implementation. A number of different algorithms were implemented and compared in order to produce a program capable of simulating galaxy interactions with sufficient accuracy and practical run program run times. Primarily, it was found, that a fourth order RK method was the most appropriate algorithm for this task. A fourth order RK method is much more scalable than an Euler method as accurate computations can be made using large steps sizes. In particular, even though more computations are required to simulate one step compared to an Euler or other lower order method, the gain in accuracy from using such a method is such that larger step sizes can be used, leading to an overall more efficient algorithm.

Figure one shows this behaviour, and the fractional error for a simulated circular orbit after a set amount of simulated time for an Euler algorithm and RK4 is plotted. The fractional energy was calculated by comparing the deviation of the computed orbit radius after a given amount of simulated time to the analytical solution at that time, which is simply the starting radius as circular orbits are stationary with time. This error falls much quicker as step size is reduced using an rk4 method, and the minimum energy error computed using rk4 is smaller than the minimum computed using Euler by over 3 orders of magnitude. Additionally, this minimum occurs at a relatively large step size. In terms of simulating a galaxy, to compute the orbits of 10 test particles over 1000 time units, with an average orbits energy error of 0.0012 using RK4 took 14.0s, with an average step size=0.68 time units, while to achieve the similar accuracy for the same situation using Euler took over 350s requiring a step size of less than 0.001.

The scalability of the program was also improved by implementing an adaptive step mechanism. This algorithm consisted of a scaling mechanism which would adjust the step size so that the error in coordinates at the $i^{th}+1$ step were below a set upper bound. This is a beneficial as it means that the program runs more efficiently, adapting the step size to the simulated system at each point in simulated time, meaning that small time steps are not used when large was would suffice or so large time steps are not used when smaller steps are required for an accurate simulation. In contrast, if a constant step size is used and an upper bound on the error of simulated system is required at every time point, the step size must be set to the smallest step size required at any point in simulated time. This means that for many points in a simulation using a constant step size, a smaller step size than what is needed is used, leading to longer run times. In practice, the adaptive step algorithm was also useful as it allowed the upper bound on the error of the simulation to be set without needing to probe the simulation every time the initial conditions were changed in order to find the required step size for the required error limit.

Once the RK4 and adaptive step size algorithms were implemented the program was probed and characterised in more detail. The figures 2 and 3 show the error in the coordinates after 2000 time units for a circular orbit, vs the upper truncation error limit, and the running time for the program vs the number of particles. It was found that there was a minimum in the energy error for an upper limit of truncation error set to 0.008008, and therefore this value was used in the simulation of galaxy interactions. Note that this value of the truncation error limit is the round-off error, and hence is machine dependent. The time required to simulate n particles was found to grow linearly, as expected. This meant that run times of a few hundred seconds were required to simulate systems of hundreds of particles, which was reasonable and practical for the required use.

Results and Discussion. Galaxy interactions were simulated using two Galaxy centres, one of which was initialised at rest at the origin with a number of test-particles in circular orbits.

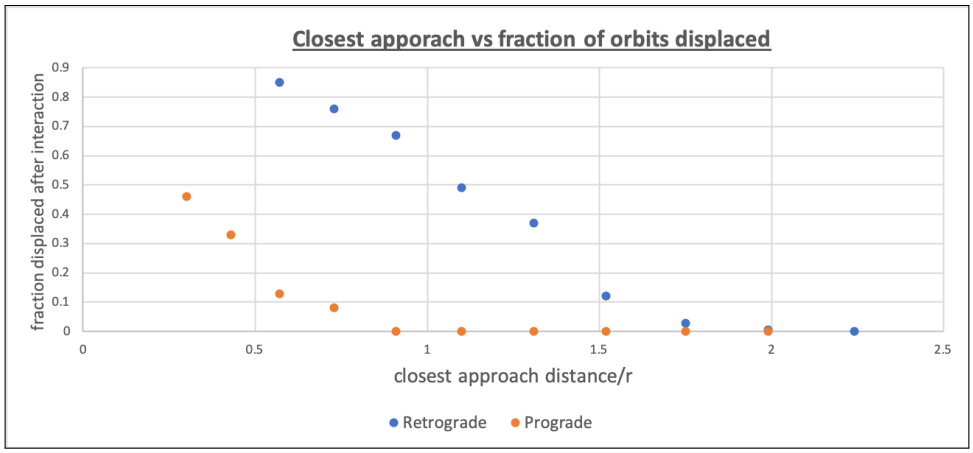
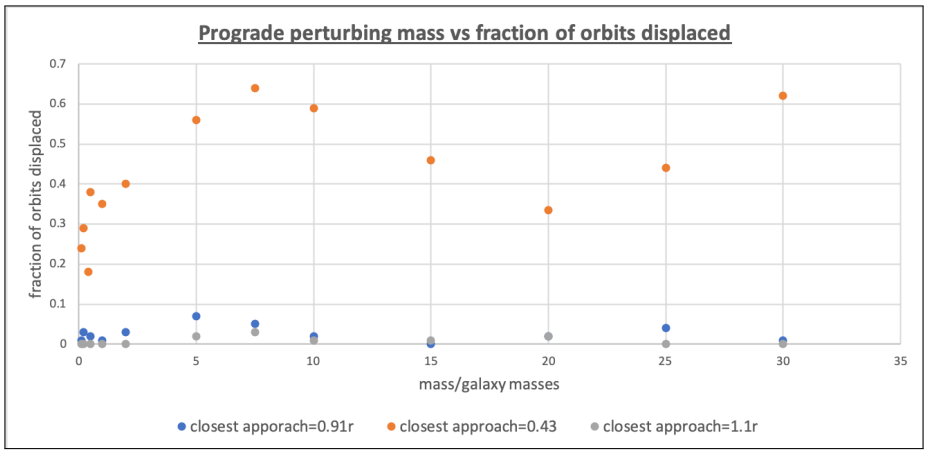
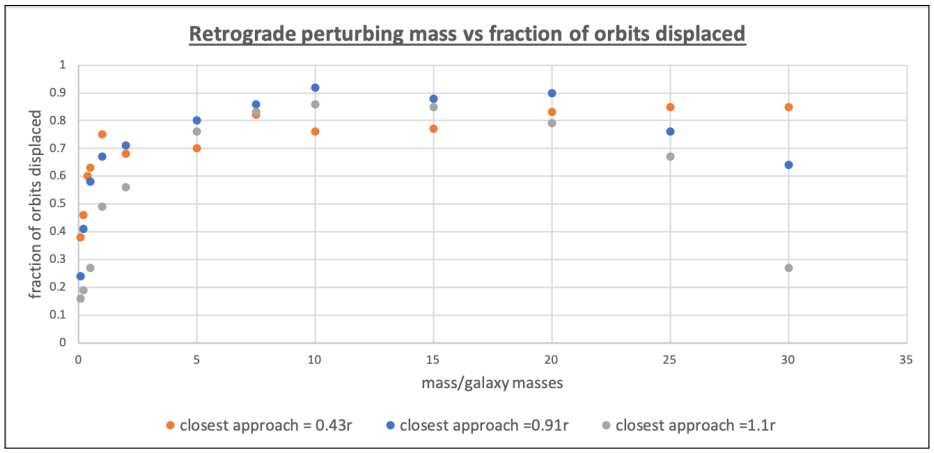


Figure 4: Effect of closest approach distance on the fraction of orbits displaced after the perturbing galaxy has passed.



(a) Fraction of orbits displaced against mass of perturbing galaxy for a prograde approach.



(b) Final galaxy shape

Figure 5: Fraction of orbits displaced against mass of perturbing galaxy for a retrograde approach.

The mass scale of the galaxies was set as $M = 10^{12}$ solar masses, which is approximately the mass of the Milky Way, with the origin galaxy mass set as constant for all simulations at 1M. The test-particles were given a random position in the range of 0.3 to 1.4 units of distance, 'r,' and a random mass in the range of 0.2 to 1.2 solar masses, m. The units of r were kiloparsecs, in line with the fact that the Milky way is approximately 50 kiloparsecs wide, and hence the units of time were of the order of 10 million years. This time scale was chosen so that the total time simulated was of the order of the age of the universe. This is convenient because it means that any galaxy structures formed through this simulation are shown to be stable on time scales of the universe, and therefore that observations in nature of the features simulated can be accredited as relics of close encounters. Note however that shorter total simulated times would in theory have sufficed to see the formation of the galaxy structures studied.

The effect of a perturbing galaxy on a system of orbits was probed by setting an incident galaxy's initial coordinates so that it had an elliptical path around the origin galaxy. The effect on the shape of the orbits around the origin galaxy and the number of bound orbitals after the perturbing had passed was investigated using a number of plots and quantitative measurements. Figure 4 shows the effect of closest approach distance of the perturbing galaxy on the number of bound orbits after the galaxy interaction. The number of bound orbits decreases significantly as the perturbing galaxy gets closer. This is as expected due to the inverse square relation for forces due to gravity. Additionally, it was found that retrograde approaches, with respect to the direction of orbit of the test-particles, were much more disruptive to the origin galaxy. This is also as expected as, because gravity is an attractive force, when the approach of the perturbing galaxy is opposite to the sense of rotation of the orbits, the force due to the perturbing galaxy acts for a longer period of time in a direction pulling the orbitals away from the origin galaxy.

The effect of varying the mass of the perturbing galaxy had a similar effect on the fraction of displaced orbits for relatively small perturbing masses: as mass increased the number of displaced orbits increased. However as the mass of the perturbing galaxy was increased to approximately an order of magnitude more than the central galaxy, and greater, the interaction becomes more complex, with orbitals not necessarily being completely displaced from the system due to the large gravitational hold of the perturbing mass. Figure 5 shows this behaviour in the case of a prograde and retrograde approach.

Analysis of the galaxy interaction by quantifying the bound and unbound orbit is useful for general characterisation of galaxy interactions, however to analyse the formation of structures the qualitative observation of the galaxy form is much more instructive. Therefore a number of plots were made to observe the formation of galaxy features with time, and spirals and tails were found to be readily produced with a variety of parameters.

A number of relationships between the perturbing galaxy approach and the evolution of the origin galaxy features can be identified. Most generally it was observed that features of the galaxy were more pronounced when the perturbing mass was greater. For example, the galaxy tails produced by larger masses tended to have a higher density and occupied a larger volume, as shown in figure 6. Here, the tail produced at 25M consists of hundred of stars and is approximately 50 kiloparsecs wide compared to a mass of M which produces only an outline of the tail with a few constituent stars. Note however the general shape is similar for all three masses in figure 6, though the length scale decreases as mass increases, showing the general higher density at higher perturbing mass relation. Figure 8 shows the formation of a spiral structure, which also follow this pattern. Specifically, the tails in the (c) series cover a greater area than the (a) series.

The structures formed in figures 7, 8 and 9 are qualitatively different from one another, illustrating the general rule that as approach distance increases, the structures become more spiral dominated, whereas at small approach distances the structures are more tail dominated. Figure 7 shows structures produced by an intermediate closest approach of 1.1r, and thus the galaxies formed have two distinct tails in a spiral pattern, but they are skewed with one covering a larger area, which is reminiscent of a tail-dominated structure.

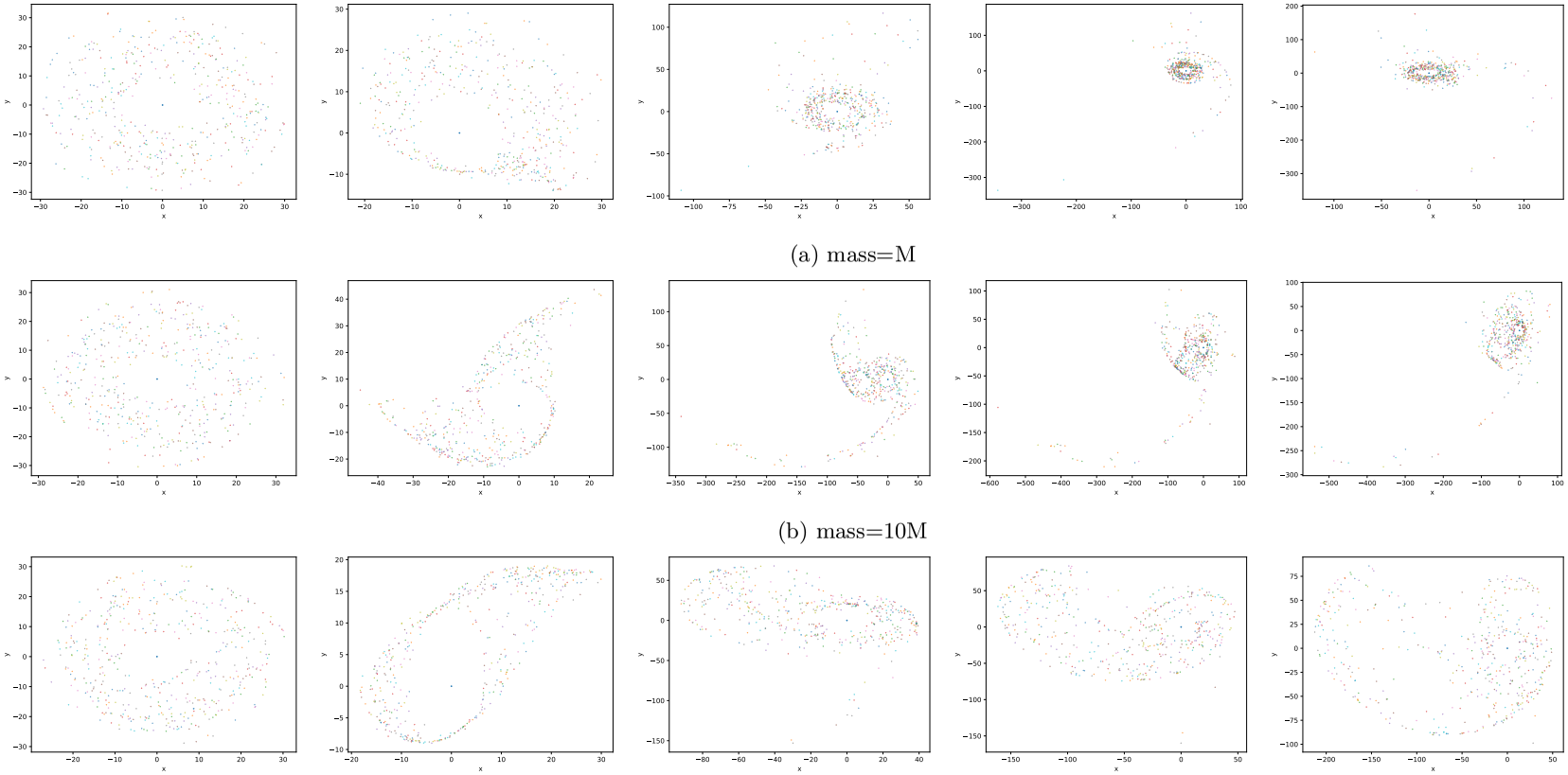
Figure 9 shows the galaxy structures formed by the retrograde approach of a perturbing galaxy. This category of perturbing mass approach was found to be distinct in the characteristics to prograde approaches, and tails formed by such an approach were of a different form to the prograde case. The most notable characteristic of the retrograde approach was the lesser role that the mass of the perturbing galaxy and dis-

tance of closest approach play in determining the shape of the galaxy after the interaction. It was found that larger masses and closer distances would displace more orbits, but the shape did not vary in any distinguishable way with such variation of parameters. In terms of the shape of structures formed, the tails spread over a large area in the direction facing the initial position of the perturbing galaxy.

These general observations seem reasonable given the inverse square law governing gravitational interactions. The fact that familiar spiral and tail structures which are observed in nature are reproduced by this model similarly means it seems likely this mechanism is involved in the formation of these structures. The simulation could be improved in some aspects however. For example, the homogeneous distribution of initialised test particles could be replaced by a more physical distribution, such as the ones outlined in [1, 3]. Additionally, dark matter is not accounted for in this model. The inclusion of such an additional force due to dark matter will be significant for the shape of galaxy tails at large distances from the origin, and for the simulation of tails on long time periods. The program could in theory be modified to account for such an additional force, by modifying Newton's central force law.

However, the simulation of galaxy interactions using a reduced body analysis has been shown to be an effective general way to model the formation of galaxy features, and the reproduction of general galaxy spirals and tails is a measure of the success of this model. Though further work such as the simulation of specific observed galaxies in addition to the modifications to the algorithm aforementioned would be a useful tests of the range of application for this model, it is clear from this demonstration that close approach galaxy interactions are an important physical mechanism for the study of galaxy structures.

word count:2600

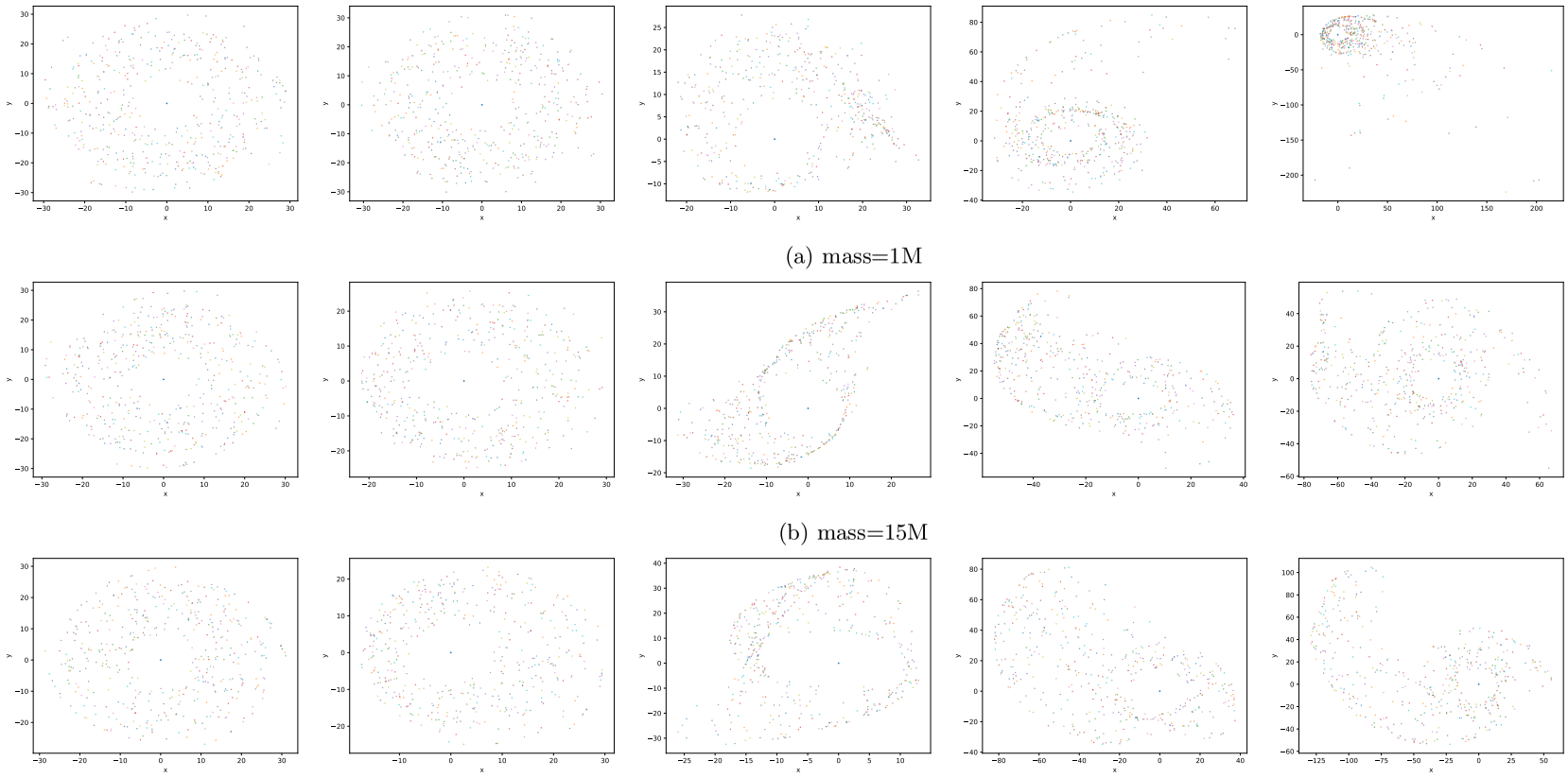


(a) mass=M

(b) mass=10M

(c) mass=25M

Figure 6: The formation of galaxy tails for a prograde approach of 0.91r. Time is increasing from left to right, in increments of 200T.

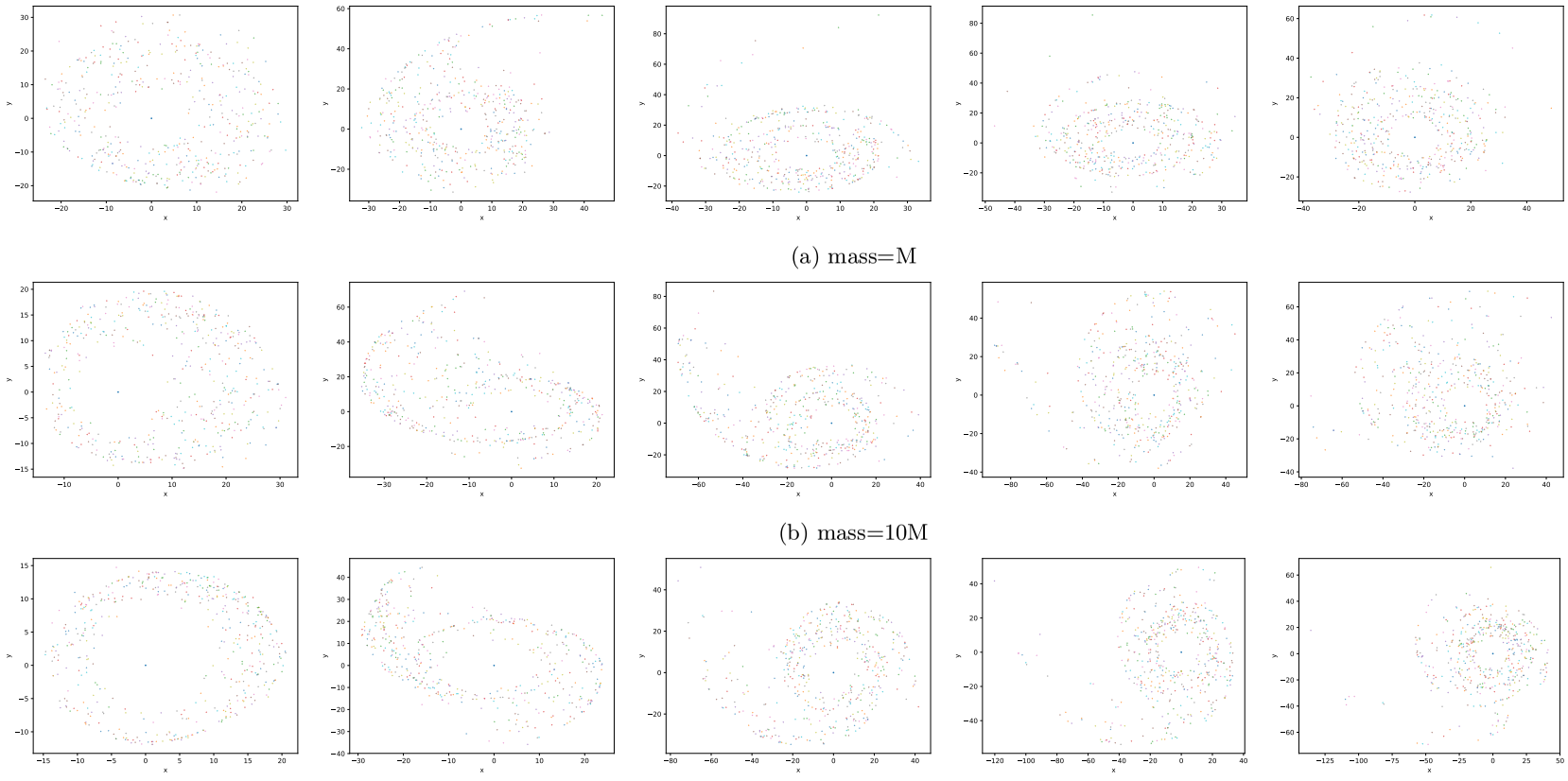


(a) mass=1M

(b) mass=15M

(c) mass=30M

Figure 7: The formation of galaxy tails for a prograde approach of 1.1r. Time is increasing from left to right, in increments of 200T.

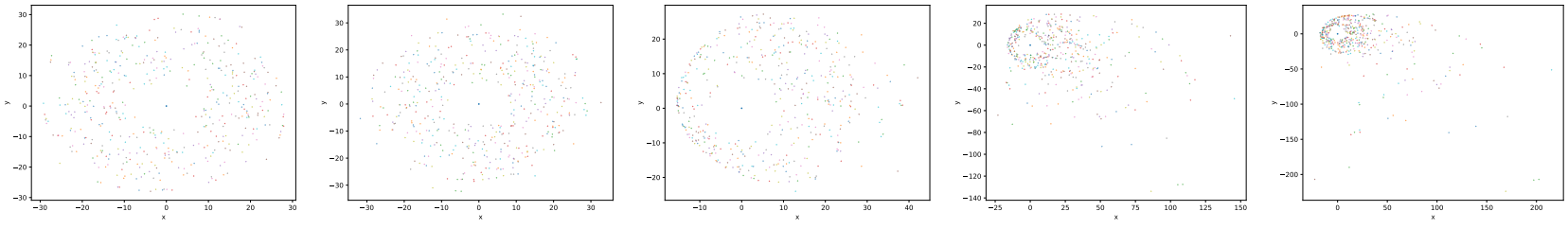


(a) mass=M

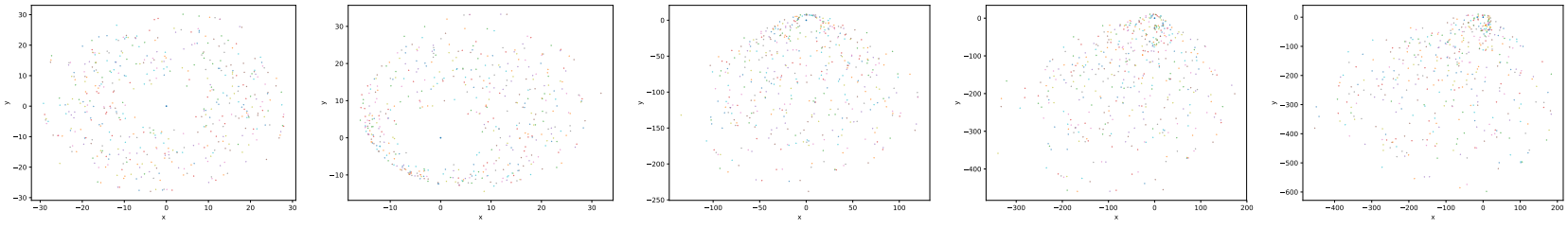
(b) mass=10M

(c) mass=30M

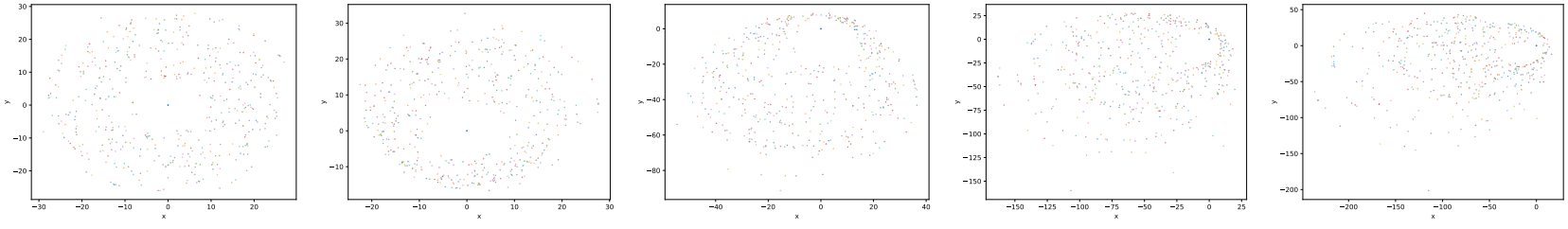
Figure 8: The formation of galaxy tails for a prograde approach of 1.75r. Time is increasing from left to right, in increments of 200T.



(a) mass=M



(b) mass=10M



(c) mass=30M

Figure 9: The formation of galaxy tails for a retrograde approach of 1.75r. Time is increasing from left to right, in increments of 200T. The galactic structures formed for different closest approaches distances were not found to be qualitatively different.

Appendix

Implementation of algorithm in python. Note, the presence of '...' below denotes that there shouldn't be a new line in the code, but in order to format the code for this document a new line has had to be taken.

```
import numpy as np
import matplotlib.pyplot as plt


class Particle:

    def __init__(self, mass):
        self.R = np.zeros((int(N), 3))
        self.V = np.zeros((int(N), 3))
        self.M = mass
        self.time = np.zeros((int(N)))

    def initialstate(self, x, y, z, vx, vy, vz):
        self.R[0, 0] = x
        self.R[0, 1] = y
        self.R[0, 2] = z
        self.V[0, 0] = vx
        self.V[0, 1] = vy
        self.V[0, 2] = vz
        self.time[0] = 0


class Gravity:
    def __init__(self, centres, orbits):
        self.centres = centres
        self.orbs = orbits

    ''' Function defined to compute the acceleration and velocity due to
    the gravitational interaction between a test-particle in the presence
    of a massive object 'gravitating_object'
    using a Runge-Kutta fourth order method. The function returns two
    coefficients, KTv and KTp which can then be used to propagate the
    velocity and position a step forward
    dT in time by multiplying theses coefficients by dT.  '''

    def RK4_acceleration(self, gravitating_object, testparticle, h, i):

        dis = ((testparticle.R[i, 0] - gravitating_object.R[i, 0])**2 + (testparticle.R[i, 1] ...
        ...- gravitating_object.R[i, 1])**2 + (testparticle.R[i, 2] -...
        ...gravitating_object.R[i, 2])**2)**0.5
        K1v = np.zeros(3)
        K1p = np.zeros(3)
        K2v = np.zeros(3)
        K2p = np.zeros(3)
        K3v = np.zeros(3)
        K3p = np.zeros(3)
        K4v = np.zeros(3)
        K4p = np.zeros(3)
        KTv = np.zeros(3)
        KTp = np.zeros(3)
        a = np.zeros(3)
        if gravitating_object.M != 0:
```

```
        for k in range(3):
            a[k] = - gravitating_object.M * G * testparticle.R[i, k] / dis**3
            K1v[k] = a[k] * h
            K1p[k] = testparticle.V[i, k] * h

        dis2 = ((testparticle.R[i, 0] - gravitating_object.R[i, 0] + K1p[0] * h / 2)**2 +...
        ...
        (testparticle.R[i, 1] - gravitating_object.R[i, 1] + K1p[1] * h / 2)**2 +...
        ...(testparticle.R[i, 2] - gravitating_object.R[i, 2] + K1p[2] * h / 2)**2)**0.5
        for k in range(3):
            K2v[k] = -(gravitating_object.M * G * (testparticle.R[i, k] + (K1p[k] * h / 2)))..
            .../ dis2**3
            K2p[k] = testparticle.V[i, k] + (K1v[k] * h / 2)

        dis3 = ((testparticle.R[i, 0] - gravitating_object.R[i, 0] + (K2p[0] * h / 2))**2...
        +...(testparticle.R[i, 1] - gravitating_object.R[i, 1] + (K2p[1] * h / 2))**2 +...
        ...(testparticle.R[i, 2] - gravitating_object.R[i, 2] + (K2p[2] * h / 2))**2)**0.5
        for k in range(3):
            K3v[k] = -(gravitating_object.M * G * (testparticle.R[i, k] +...
            ...(K2p[k] * h / 2))) / dis3**3
            K3p[k] = testparticle.V[i, k] + (K2v[k] * h / 2)

        dis4 = ((testparticle.R[i, 0] - gravitating_object.R[i, 0] + (K3p[0] * h))**2 ...
        ...+ (testparticle.R[i, 1] - gravitating_object.R[i, 1] + (K3p[1] * h))**2 ...
        ...+ (testparticle.R[i, 2] - gravitating_object.R[i, 2] + (K3p[2] * h))**2)**0.5
        for k in range(3):
            K4v[k] = -(gravitating_object.M * G * (testparticle.R[i, k] + ...
            ...(K3p[k] * h))) / dis4**3
            K4p[k] = testparticle.V[i, k] + (K3v[k] * h)

        KTv = ((1 / 6) * (K1v + (2 * K2v) + (2 * K3v) + K4v))
        KTp = ((1 / 6) * (K1p + (2 * K2p) + (2 * K3p) + K4p))
    return KTv, KTp

''' Function defined to return two coefficients to propagate the
velocity and position using a first order Euler method. '''

def Euler_acceleration(self, gravitating_object, testparticle, h, i):
    dis = ((testparticle.R[i, 0] - gravitating_object.R[i, 0])**2 +...
    ...(testparticle.R[i, 1] - gravitating_object.R[i, 1])**2 + (testparticle.R[i, 2] -...
    ...gravitating_object.R[i, 2])**2)**0.5
    K1v = np.zeros(3)
    K1p = np.zeros(3)
    a = np.zeros(3)
    for k in range(3):
        a[k] = - gravitating_object.M * G * testparticle.R[i, k] / dis**3
        K1v[k] = a[k]
        K1p[k] = testparticle.V[i, k]
    return K1v, K1p

''' Function defined to propagate the velocity and position of
massive objects defined by the array 'objects' which are all massive
enough so that they all produce a
significant gravitational influence on each other. '''

def iterate_mutualattract_RK4(self, test_particle, target_index, h, i):
```

```
        positioniterator = np.zeros(3)
        velocityiterator = np.zeros(3)
        iteratorv = np.zeros((len(centres), 3))
        iteratorx = np.zeros((len(centres), 3))
        for index in range(len(self.centres)):
            if index != target_index:
                velocityiterator, positioniterator = self.RK4_acceleration(...
                ...self.centres[index], test_particle, h, i)
                iteratorv[index, :] = velocityiterator
                iteratorx[index, :] = positioniterator
        test_particle.V[i + 1, :] = h * np.sum(iteratorv, axis=0) + test_particle.V[i, :]
        test_particle.R[i + 1, :] = h * np.sum(iteratorx, axis=0) + test_particle.R[i, :]
        return test_particle.V[i + 1, :], test_particle.R[i + 1, :]

    ''' Function defined which propagates the velocity and position of a
    set of test-particles which have small masses so that they do not
    produce a significant
    gravitational influence on one another. Because of this, to propagate
    their coordinates only the gravitational influence of the objects in
    the array 'orbits' is
    accounted for in the calculation of the KTv and KTp coefficients. '''

    def iterate_centralfield_RK4(self, testparticle, h, i):
        positioniterator = np.zeros(3)
        velocityiterator = np.zeros(3)
        iteratorv = np.zeros((len(centres), 3))
        iteratorx = np.zeros((len(centres), 3))
        for j in range(len(centres)):
            velocityiterator, positioniterator = self.RK4_acceleration(...
            ...self.centres[j], testparticle, h, i)
            iteratorv[j, :] = velocityiterator
            iteratorx[j, :] = positioniterator
        testparticle.V[i + 1, :] = h * np.sum(iteratorv, axis=0) + testparticle.V[i, :]
        testparticle.R[i + 1, :] = h * iteratorx[0] + testparticle.R[i, :]
        return testparticle.V[i + 1, :], testparticle.R[i + 1, :]


if __name__ == '__main__':
    ''' The total time for which the simulation is to be run is defined
    in addition to the base time interval dT. The adapted time,
    'dT_adapted'
    is used to propagate the velocity and position of the bodies in the
    simulation, and it is a fraction of dT determined by the adaptive
    step code.
    The physical constant G is defined also, in addition to the masses of
    the bodies in the simulation: the mass of the massive bodies is
    taken as
    10^12, as the program is currently set-up to simulate galaxy
    interaction, and the mass of a galaxy is approximately 12 orders of
    magnitude
    greater than a star. The letters R and r hold the distances used in
    computations, in units of kiloparsecs, and thus the time is in units
    of 500000 years (the same units as used in Toomre & Toomre 'Galactic
    Bridges and tails' 1972, The Astrophysical journal, 178, pg 623-666).  '''

    T = 2000
```

```
dT = 5
dT_adapted = dT
time = 0
N = 1000000
G = 1e-11
M = 1e12
m = 1
r = 25
R = 10 * r
V = (G * M / R)**0.5
v = (G * M / r)**0.5
centre1 = Particle(M)
centre1.initialstate(0, 0, 0, 0, 0, 0)
centre2 = Particle(1 * M)
centre2.initialstate(-0.5 * R, R, 0, 0.4 * V, -2 * V, 0)
centre3 = Particle(M)
centre3.initialstate(-1.4 * R, 2 * R, 0, 0.45 * V, -1.5 * V, 0)

centres = [centre1, centre2] # , centre3]

''' Code which initialises a set of test particles with masses between 0.2 and 1.2
solar masses, in circular orbits between 0.4 and 1.4 times r, around a
central galaxy mass at (0,0,0). '''

orbits = []
difference_centres = []
for i in range(500):
    new_particle = Particle((np.random.rand() + 0.2) * m)
    rx = (2 * np.random.rand() - 1) * r
    ry = (2 * np.random.rand() - 1) * r
    r0 = (rx**2 + ry**2)**0.5
    vy = (G * M / r0)**0.5 * rx / r0
    vx = -(G * M / r0)**0.5 * ry / r0
    new_particle.initialstate(rx, ry, 0, vx, vy, 0)
    if (rx**2 + ry**2)**0.5 >= 0.3 * r:
        orbits.append(new_particle)
gravity = Gravity(centres, orbits)
i = 0

''' Loop defined to simulate the propagation of the test-particles
and galaxy centres defined above. An adaptive step size algorithm is
implemented to make the simulation more efficient. '''

while time < T:
    solution1 = []
    solution2 = []
    difference = []
    vector = np.zeros(3)
    ''' adaptive step size determination'''
    for k in range(len(centres)): # iteration with initial step size
        solution1.append(gravity.iterate_mutualattract_RK4(centres[k], k, dT, i))
    for k in range(len(orbits)):
        solution1.append(gravity.iterate_centralfield_RK4(orbits[k], dT, i))

    for k in range(len(centres)): # iteration with half initial step size
        centres[k].V[i + 1, :], centres[k].R[i + 1, :] =...
```

```
        ...gravity.iterate_mutualattract_RK4(centres[k], k, dT / 2, i)
    for k in range(len(orbits)):
        orbits[k].V[i + 1, :], orbits[k].R[i + 1, :] =...
        ...gravity.iterate_centralfield_RK4(orbits[k], dT / 2, i)

    for k in range(len(centres)): # iterate with half step size to get to time + dT
        solution2.append(gravity.iterate_mutualattract_RK4(centres[k], k, dT / 2, i + 1))
    for k in range(len(orbits)):
        solution2.append(gravity.iterate_centralfield_RK4(orbits[k], dT / 2, i + 1))

    for k in range(len(centres)):
        centres[k].V[i + 1, :] = np.zeros(3)
        centres[k].R[i + 1, :] = np.zeros(3)
    for k in range(len(orbits)):
        orbits[k].V[i + 1, :] = np.zeros(3)
        orbits[k].R[i + 1, :] = np.zeros(3)
    for k in range(len(solution1)):
    # difference in vectors when i + 1 is determined with dT and 2 * dT/2
        difference.append(np.linalg.norm(solution1[k]) - np.linalg.norm(solution2[k]))

    if np.linalg.norm(difference) != 0:
    # adjustment of step size so that the error is below a given bound
        dT_adapted = dT * (0.008008522 / np.linalg.norm(difference))**0.2
    time += dT_adapted

    for k in range(len(centres)): # determination of i + 1 using the adatped step size
        centres[k].V[i + 1, :], centres[k].R[i + 1, :] =...
        ...gravity.iterate_mutualattract_RK4(centres[k], k, dT_adapted, i)
        centres[k].time[i + 1] = time
    for k in range(len(orbits)):
        orbits[k].V[i + 1, :], orbits[k].R[i + 1, :] =...
        ...gravity.iterate_centralfield_RK4(orbits[k], dT_adapted, i)
        orbits[k].time[i + 1] = time
    vector = centres[0].R[i, :] - centres[1].R[i, :]
    a = np.linalg.norm(vector)
    difference_centres.append(a)
    i += 1

plotting_lim = i
print(time, i, 'average step size=', time / i)
closest_approach = np.amin(difference_centres)

'''fractional energy change calculator, for analysis of accuracy of
circular orbits after given amount of simulated time.'''

fchange = []
for i in range(len(orbits)):
   U0 = (0.5 * G * centres[0].M * orbits[0].M) / (np.linalg.norm(orbits[0].R[0]))
   Uf = (0.5 * G * centres[0].M * orbits[0].M) / (np.linalg.norm(orbits[0].R[plotting_lim]))
   fractionalchange = (Uf - U0) / U0
   fchange.append(fractionalchange)
print('fractional change in energy=', np.mean(fchange))
print('closest approach distance=', closest_approach / r)
print('orbits:', print(len(orbits)))
print('fractional change in energy=', fractionalchange)
print('closest approach distance=', closest_approach / r)
```

```
# plotting
fig1 = plt.figure()
ax = fig1.add_subplot(111)
for i in range(len(orbits)):
    if np.linalg.norm(orbits[i].R[plotting_lim, :]) <= 60000:
        plt.plot(orbits[i].R[0: plotting_lim, 0], orbits[i].R[0: plotting_lim, 1], ...
        ...linewidth=0.01)
plt.plot(centres[0].R[0: plotting_lim, 0], centres[0].R[0: plotting_lim, 1], 'b', ...
...linewidth=0.07)
plt.plot(centres[1].R[0:plotting_lim, 0], centres[1].R[0: plotting_lim, 1], 'm', ...
...linewidth=0.01)
#plt.plot(centres[2].R[0:plotting_lim, 0], centres[2].R[0: plotting_lim, 1], 'm', ...
...linewidth=0.01)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.tight_layout()
plt.savefig('orbits.pdf', bbox="tight")

fig1 = plt.figure()
ax = fig1.add_subplot(111)
for i in range(len(orbits)):
    if np.linalg.norm(orbits[i].R[int(plotting_lim * 0.1), :]) <= 60000:
        plt.plot(orbits[i].R[0: int(plotting_lim * 0.1), 0], orbits[i].R[0: ...
        ...int(plotting_lim * 0.1), 1], linewidth=0.01)
plt.plot(centres[0].R[0: int(plotting_lim * 0.1), 0], centres[0].R[0: ...
...int(plotting_lim * 0.1), 1], 'b', linewidth=0.07)
plt.plot(centres[1].R[0:int(plotting_lim * 0.1), 0], centres[1].R[0: ...
...int(plotting_lim * 0.1), 1], 'm', linewidth=0.01)
#plt.plot(centres[2].R[0:plotting_lim, 0], centres[2].R[0: plotting_lim, 1], 'm', ...
...linewidth=0.01)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.tight_layout()
plt.savefig('orbits0.1.pdf', bbox="tight")

fig = plt.figure()
ax = fig.add_subplot(111)
plt.scatter(centre1.R[plotting_lim, 0], centre1.R[plotting_lim, 1], s=1)
for i in range(len(orbits)):
    if np.linalg.norm(orbits[i].R[plotting_lim, :]) <= 600:
        plt.scatter(orbits[i].R[plotting_lim, 0], orbits[i].R[plotting_lim, 1], s=0.4)
    else:
        unplotted.append(orbits[i].R[plotting_lim, :])
plt.plot(centres[1].R[0:plotting_lim, 0], centres[1].R[0: plotting_lim, 1], 'm', ...
...linewidth=0.1)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.tight_layout()
plt.savefig('galaxy_finalwpath.pdf', bbox="tight")
```

References

[1] Peter S. Conti and William D. Vacca. The Distribution of Massive Stars in the Galaxy. I. The Wolf-Rayet Stars. , 100:431, Aug 1990.

[2] A. Toomre and J. Toomre. Galactic Bridges and Tails. , 178:623–666, December 1972.

[3] Maosheng Xiang, Jianrong Shi, Xiaowei Liu, Haibo Yuan, Bingqiu Chen, Yang Huang, Chun Wang, Yaqian Wu, Zhijia Tian, Zhiying Huo, Huawei Zhang, and Meng Zhang. Stellar Mass Distribution and Star Formation History of the Galactic Disk Revealed by Mono-age Stellar Populations from LAMOST. *The Astrophysical Journal Supplement Series*, 237(2):33, Aug 2018.