Tidal Tails and Interacting Galaxies

University of Cambridge

April 7, 2020

Abstract

A tidal tail is a long, thin region of stars and interstellar dust extending from a galaxy, and is a result of gravitational forces between interacting galaxies. A simple N-body simulation of two interacting galaxies was performed where specifically, a star-less perturbing galaxy with initial conditions of a parabolic orbit was introduced to a central galaxy with circularly orbiting stars. The simulation showed that the tidal tail was created from the outer orbiting stars of the central galaxy. Quantitative results, such as the fraction of stars disturbed from their initial orbits, were obtained for different characteristics and initial conditions of the perturbing galaxy. Varying the distance of closest approach of the two galaxies showed that a larger fraction of stars were disturbed when the distance of closest approach is smaller. Changing the mass of the perturbing galaxy demonstrated that the heavier the perturbing galaxy, the more stars that were stripped away from the central galaxy. [Word count: 2408]

1. Introduction

Tidal tails are long, thin, and often curved regions of stars and interstellar debris extending from a galaxy, and are a result of the strong tidal gravitational forces of interacting galaxies. Tidal tails are often a useful feature in the study and characterisation of the evolution of galactic phenomena, which includes galaxy mergers and the formation of tidal dwarf galaxies [1]. A well-known example of tidal tails are the Antennae Galaxies – a pair of galaxies currently undergoing galactic collision. The system's pair of prominent tidal tails extend from the top of the two galaxies and is what gives its characteristic insect-like outline, as seen in Figure 1. The Tadpole Galaxy provides another good example of tidal tails, where its long tidal tail of approximately 86 kpc has been attributed to a merger with a smaller galaxy in the past [2]. An image of the Tadpole Galaxy is provided in Figure 1.

Figure 1: *Top image: The Antennae Galaxies. A pair of prominent tidal tails can be seen extending from both sides of the system. Bottom image: Tadpole Galaxy. Its characteristic long tidal tail can be seen extending from the core of the galaxy.*

The formation of tidal tails is a difficult phenomena to describe analytically, thus computer simulations have come in handy to provide possible explanations for its formation. In one of the first simulations of the interaction between two galaxies by Toomre and Toomre in 1972 [3], they found that tidal tails were primarily formed through prograde encounters between the 'central' and perturbing galaxy, where both the stars in the central galaxy and the perturbing galaxy are orbiting in the same direction. Toomre and Toomre used the restricted three-body equations of motion, alongside the fourth-order Runge-Kutta method in order to simulate the interaction. A few years later, Keenan and Innanen investigated the same effects using numerical methods, and came to the same conclusion as Toomre and Toomre [4]. Their studies were able to accurately

reconstruct the appearance and outer outlines of a number of galactic systems, including Arp 295, M51 + NGC 5195, NGC 4676 and the Antennae Galaxies.

Just like in the simulations mentioned above, the aim of this investigation is to run a simple N-body simulation of two interacting galaxies, which shall be called the *central* and *perturbing* galaxies. The central galaxy will contain test particles, which represent the stars and interstellar dust, orbiting around a central heavy mass at multiple radii. Given that the masses of stars are negligible compared to the central mass of a galaxy, the test particles' masses can and will be assumed to be zero in this investigation. The perturbing galaxy will be introduced with the initial conditions of a parabolic orbit, and the effects of this perturbing galaxy will be observed.

2. Analysis

The primary concern of this investigation is the choice of the N-body simulation method. As stated by Toomre and Toomre in their paper form 1972, there are two simple methods that can be used for such simulations – the timestep method, and the integration of the restricted three-body equations of motion using the fourth-order Runge-Kutta method. There are also more modern and sophisticated techniques, such as tree methods (Barnes-Hut) and particle mesh methods, which reduces the runtime of the such simulations. These methods were not considered here due to their complexity, as well as the small scale of this investigation not requiring high efficiency. The timestep and restricted three-body methods for a single test particle are outlined below:

Timestep method

1. Provide the initial position (x_0, y_0) and velocity $(v_{x,0}, v_{y,0})$ of the test particle.

2. Calculate the forces acting on the test particle, resolving them in each axis direction. Working in two dimensions, the gravitational acceleration acting in each axis direction as a result of a massive object is

$$a_x = \frac{GM(x-X)}{r^3}, \quad a_y = \frac{GM(y-Y)}{r^3} \tag{1}$$

where G is the gravitational constant, M is the mass of the massive object, (x, y) is the position of the test particle, (X, Y) is the position of the massive object and $r = \sqrt{(x-X)^2 + (y-Y)^2}$.

3. Let δt be a small timestep. The velocity, as a result of the gravitational acceleration, is

$$v_x = v_{x,0} + a_x \delta t, \quad v_y = v_{y,0} + a_y \delta t \tag{2}$$

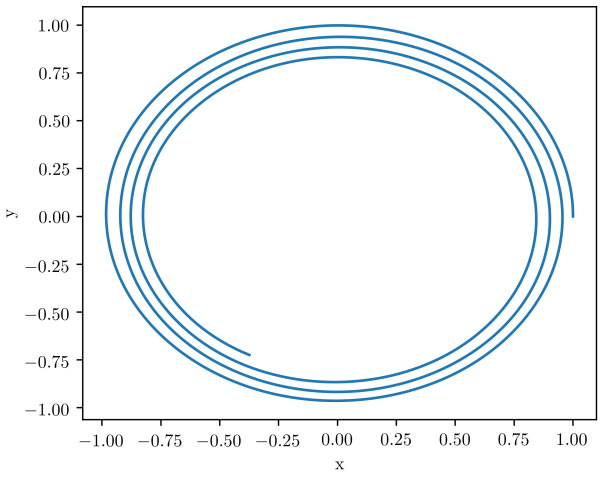


Figure 2: *With the central mass positioned at $(0, 0)$ and the test particle having an initial velocity $v = \sqrt{GM/r}$, the path of the test particle spirals inward when the fourth-order Runge-Kutta method is used to integrate its equation of motion.*

and the position is

$$x = x_0 + v_x \delta t, \quad y = y_0 + v_y \delta t \tag{3}$$

4. Repeat these steps until the end time is reached.

Restricted three-body method

1. For the three-body system, the equation of motion is given as [5]

$$\begin{aligned} a_x &= -\frac{GM_1(x-X_1)}{r_1^3} - \frac{GM_2(x-X_2)}{r_2^3} \\ a_y &= -\frac{GM_1(y-Y_1)}{r_1^3} - \frac{GM_2(y-Y_2)}{r_2^3}, \end{aligned} \tag{4}$$

where $M_{1,2}$ are the masses of the massive objects, $(X_{1,2}, Y_{1,2})$ are the positions of the massive objects and $r_{1,2}$ are the distances between the test particle and the massive objects.

2. Two first order ODEs in each axis direction are obtained from the equation of motion. The fourth-order Runge-Kutta method is then used here to integrate the first order ODEs for each test particle.

3. The motion of the test particle is then known based off the motion of the two massive interacting bodies.

In order to determine the preferred technique, these two methods were tested on a simple two-body system consisting of a large mass and a circularly orbiting test particle. This small experiment showed that the Runge-Kutta method lost its accuracy quickly for orbiting systems such as this one, where the path of the test

particle was found to spiral inward, as shown in Figure 2. The timestep method, despite being more inefficient, proved to be more accurate for such a system. Thus, the timestep method was the method of choice for this investigation into the effects of interacting galaxies.

Another key computational aspect in this investigation was the use and storage of the calculated particle trajectories from the simulation. Knowing that the simulation will contain many test particles, potentially > 1000 particles, it was important to keep in mind that a good system to store and read the calculated data was required. Such a system would most likely require to be able to create and read multiple large `.csv` files for the experiments that will be performed.

3. Implementation

This N-body simulation contained 1200 test particles of zero mass orbiting circularly around the central galaxy. To be more detailed, there were 120, 180, 240, 300 and 360 particles orbiting at a radius of 2, 3, 4, 5 and 6 units respectively. The perturbing galaxy contained no test particles (to simplify the problem), and it had the initial conditions of a parabolic orbit about the central galaxy. The test particles experienced the gravitational forces of both massive galaxies, whilst both galaxies experienced the gravitational force from each another. Note that the gravitational constant $G = 1$ was used in this simulation, and units were arbitrary and generally disregarded here.

A few experiments were performed with this simulation, and they are given below:

- comparing prograde and retrograde encounters,
- varying the mass of the perturbing galaxy,
- and varying the distance of closest approach of the perturbing galaxy.

Two quantitative metrics were used here to quantify the two latter experiments – the fraction of test particles disturbed from their initial orbits, and the median distance of the test particles from the central galaxy. Here, the term "disturbed" was arbitrarily defined as when the test particle deviates from its original orbiting radius by > 0.5. The choice of using the median instead of the average was due to the possibility of stray test particles being 'shot' very far out. More details of this will be given in §4.3.

3.1. Physics

In order for test particles to orbit around the central galaxy in a stable circular fashion, suitable initial velocities were required. Equating the centripetal force and the gravitational force, a test particle required its velocity $\boldsymbol{v}$ to have a magnitude $|\boldsymbol{v}| = \sqrt{GM/r}$, travelling perpendicular to the radial direction. The velocities in the x and y direction could then be written as $v_x = -|\boldsymbol{v}|\sin\theta$ and $v_y = |\boldsymbol{v}|\cos\theta$, where θ is the angle between the positive x-axis and the position vector of the test particle relative to the central galaxy. Note that the test particles were initialised such that they orbit the central galaxy in the anti-clockwise direction.

As mentioned above, the perturbing galaxy is initially in a parabolic orbit. A parabolic orbit has the characteristic of the system having zero net energy, where the kinetic energy is balanced with the gravitational potential energy. This can be shown as

$$E = 0 = \frac{mv^2}{2} - \frac{GMm}{r} \tag{5}$$

where M is the mass of the central galaxy, m is the mass of the perturbing galaxy, v is the velocity of the orbiting mass and r is the distance between the central and orbiting masses. Solving this leads to a velocity with magnitude $v = \sqrt{2GM/r}$.

The trajectory of a parabolic orbit is given by [6]

$$y^2 = 4r_{min}^2 - 4r_{min}x, \tag{6}$$

where r_{min} is the minimum distance of approach of the orbiting mass. The initial velocities of the perturbing galaxy in the x and y direction can then be determined using $v^2 = v_x^2 + v_y^2$ and the derivative of the trajectory $y' = dy/dx = v_x/v_y$. Combining these equations give

$$v_y = \pm\frac{y'|\boldsymbol{v}|}{\sqrt{1+y'^2}},\ v_x = \frac{v_y}{y'}. \tag{7}$$

It was important to keep track of the signs here since the perturbing galaxy could be introduced either prograde or retrograde with respect to the test particles orbiting in the anti-clockwise direction.

3.2. Code

A `Body` class was created to represent both the massive galaxies and test particles. This class contains two important functions, `set_gforce` and `update_speed_position`, which obtains the resultant forces acting on the body and updates the body's speed and position respectively. This class is housed in the `core.py` file. This file also contains two other important functions – `initialise_moving_body` which initialises the perturbing galaxy in its parabolic orbit with the velocities from §3.1, and `initialise_test_particles` which initialises all the test particles at random angles around the central galaxy.

The `simulation.py` contains the script which runs the simulation and writes the position data from the

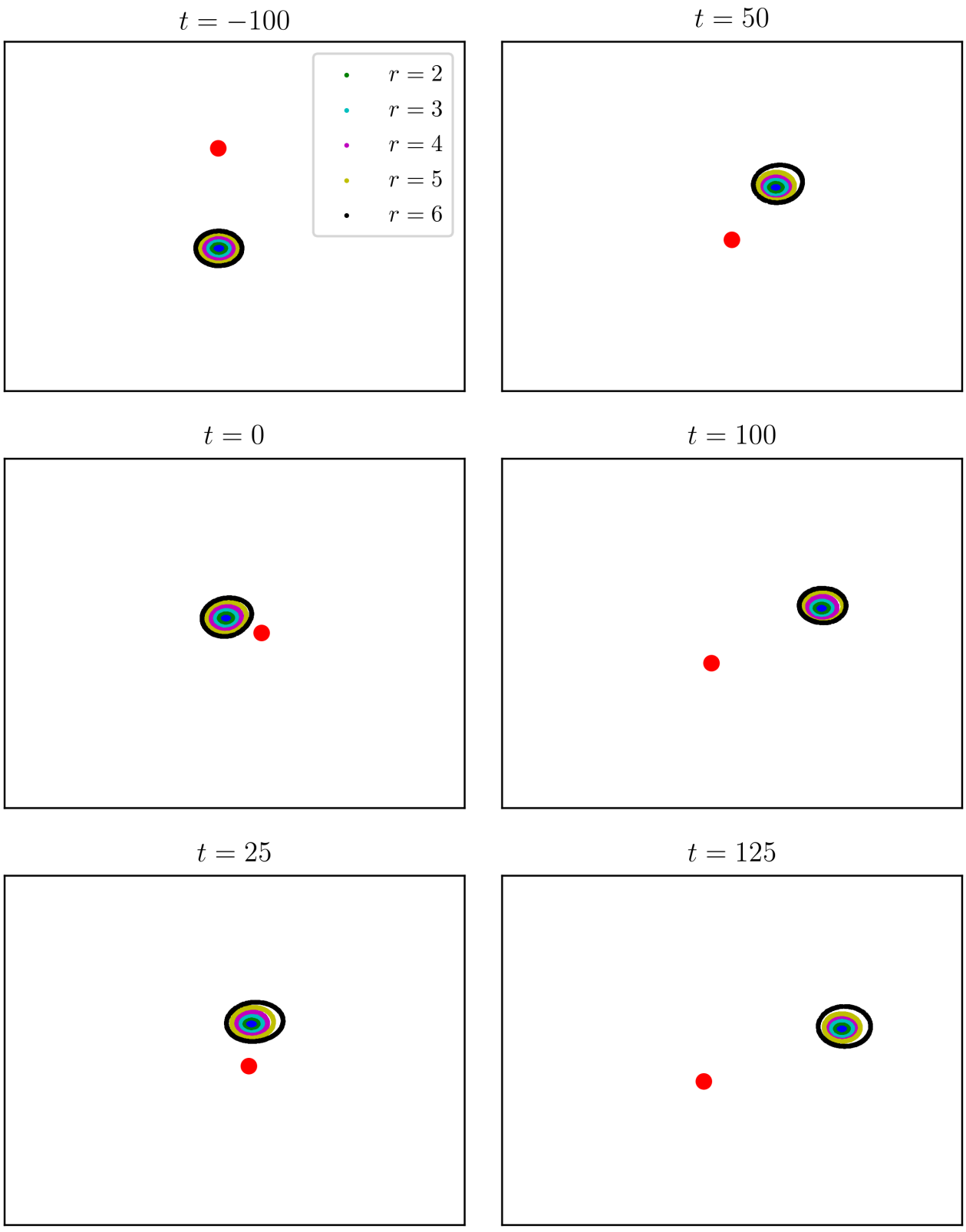


Figure 3: *A time evolution of the retrograde encounter between the two galaxies for variables* `r_min = 20` *and* `mass = 1`. *Notice that none of the stars are thrown out of orbit, however the outer orbiting stars seem to have its orbit destabilised.*

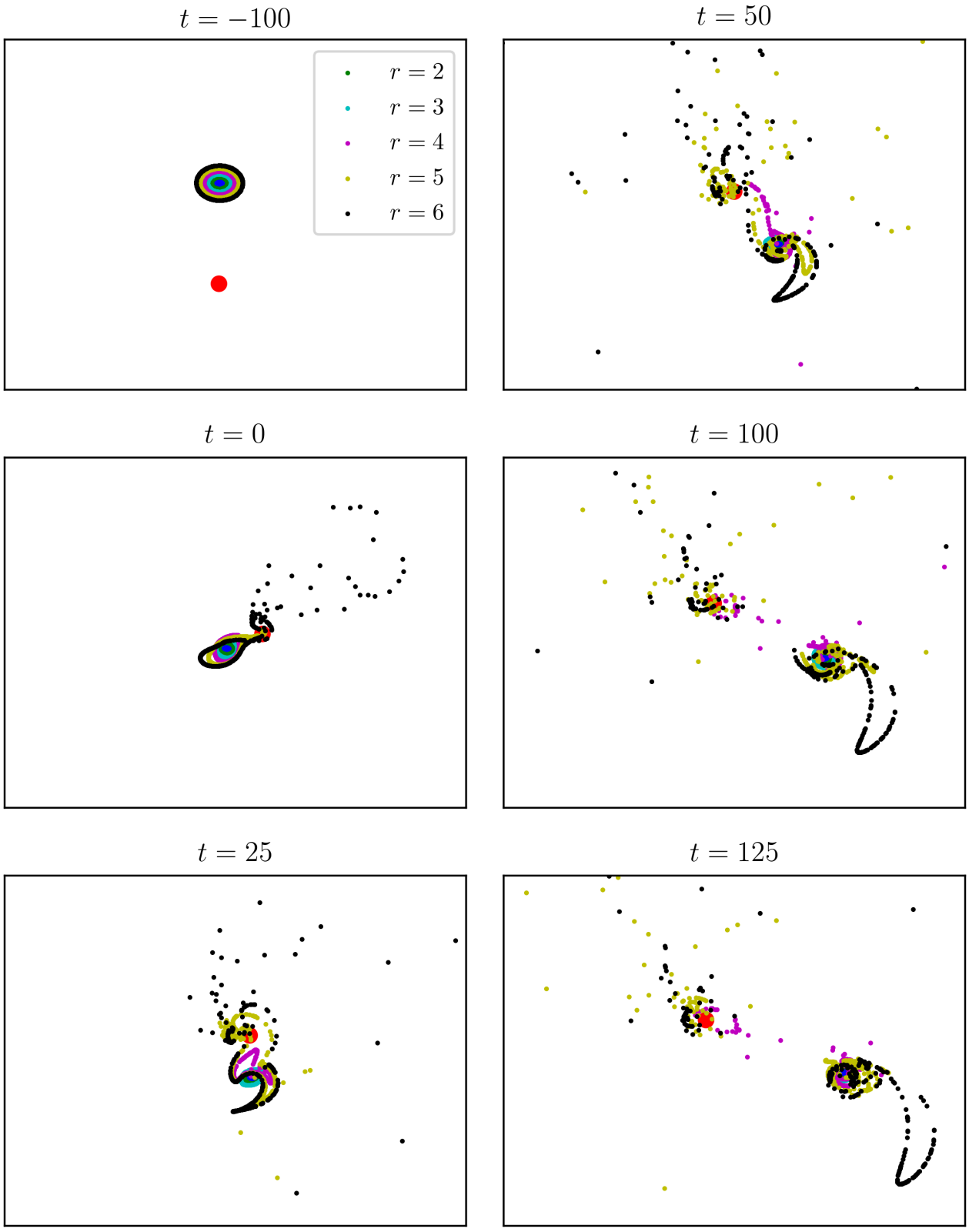


Figure 4: *A time evolution of the prograde encounter between the two galaxies for variables* `r_min = 20` *and* `mass = 1`*. The final two snapshots of the encounter show the development of a tidal tail made up of stars from the outer orbits of the central galaxy.*

simulation into suitably named files. A small timestep of `dt = 0.1` was used to iterate through from `t = 0` to `t = 700`, where the positions and velocities of the galaxies and test particles were updated at each iteration. This file, alongside the `core.py` file, allows multiple runs of the simulation for different variables (distance of closest approach `r_min`, mass of perturbing galaxy `mass`) to be performed consecutively.

The `animation.py` and `get_quantitative.py` files were used to read the `.csv` files, albeit with different functions. The former was used to create a simple animation of the interaction with `matplotlib`, and the latter was used to extract the quantitative metrics from the data produced from the simulation.

The code in these files can be found in the Appendix.

3.3. Performance

Given the large number of test particles, the simulation was expected to take some time to run, especially given the limitations of the hardware of a laptop. Particular attention was given to using Python's list comprehension rather than using `for` loops in order to improve the runtime of the program. One run of the simulation took on average around 90 s on a laptop with an i5 two core processor.

When it came to reading the large `.csv` files, the main bottleneck was loading the position data from the file into the laptop's memory. Two known built-in methods that does this job are `numpy`'s `loadtxt` and `pandas`'s `read_csv` methods. It was found that `pandas.read_csv` method was significantly faster than `numpy.loadtxt`, where `pandas.read_csv` took around 9 s whereas `numpy.loadtxt` took around 22 s. Thus, the `pandas` method was chosen to be used here.

4. Results and Discussion

4.1. Prograde and Retrograde Encounters

Figure 3 and 4 show the retrograde and prograde encounters of the two galaxies respectively. Here, the distance of closest approach `r_min` was set to 20 units and the mass of the perturbing galaxy `mass` was set to 1 unit. The `t = 0` snapshot was set to the frame where the distance between the two galaxies was at a minimum. Multiple runs of the same simulation were performed, and very similar results were obtained from them. The figures show a clear agreement between the results of this simulation and those performed by Toomre and Toomre, where the prograde encounter was shown to have a significantly more disruptive effect than the retrograde encounter.

In the retrograde encounter, there was a noticeable destabilisation of the outermost circular orbit in the $t = 25, 50, 100, 125$ snapshots. However, the outermost stars remained in orbit, albeit not circular, around the central galaxy. This mild effect can possibly be attributed to weak tidal effects from the perturbing galaxy, where the test particles nearest to the perturbing galaxy moved in a relatively antiparallel direction to the velocity of the perturbing galaxy.

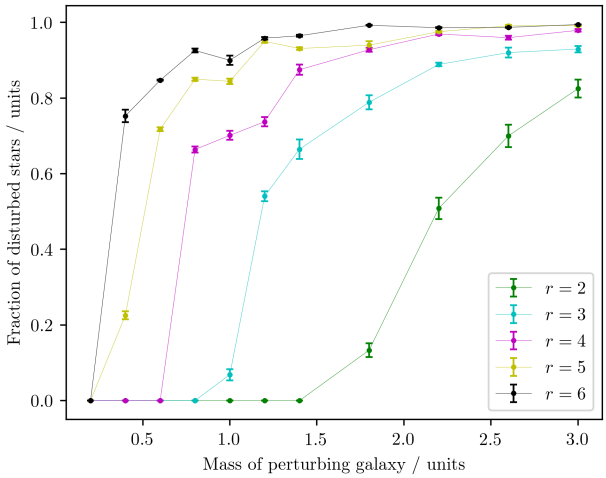
In the prograde encounter, strong tidal effects, arising from the similarity in the direction of the velocities of the test particles and perturbing galaxy, tore apart the outer stars with an orbiting radius of 4, 5 and 6 units. The formation of the tidal tail, which was made up of the outermost stars (radius of 6 units), seen in the bottom right corner of the $t = 100, 125$ snapshots was the most noticeable effect of such strong tidal forces. A large proportion of the outer stars were taken away by the perturbing galaxy, where these stars form new orbits around the perturbing galaxy. A considerable amount of stars were also seem to have been flung away from both galaxies in a seemingly random distribution.

4.2. Varying Initial Conditions of Perturbing Galaxy

As mentioned in §3, two quantitative metrics were used here – the fraction of disturbed stars and the median distance between test particles and the central galaxy. These quantitative metrics were calculated at the snapshot $t = 100$ units after the frame where the distance between the two galaxies was at a minimum. The results of varying the mass of the perturbing galaxy and the distance of closest approach are shown in Figure 5 and Figure 6. Three independent runs of the simulation were performed for each data point, and the errors shown in the plots are the standard error of the mean from the three runs.

Both figures show general trends that are expected from intuition, where a more massive perturbing galaxy and/or a smaller distance of closest approach leads to a larger disruption of the orbiting stars. Both figures also show that stars with a larger orbiting radius are more easily disturbed, and this is most clearly seen in both plots in Figure 5, where the graph for $r = 5, 6$ has a high number of disturbed stars even at low perturbing mass and a large distance of closest approach.

Figure 6 shows how the median distance varies in the experiments. The top plot generally shows that stars at larger orbiting radius get flung out further as the mass of the perturbing galaxy increases. It also shows an interesting characteristic, where the median distance for different orbiting radii appear to converge at larger masses of the perturbing galaxy. However, further experimentation is required to provide a more concrete result. The bottom plot shows less conclusive



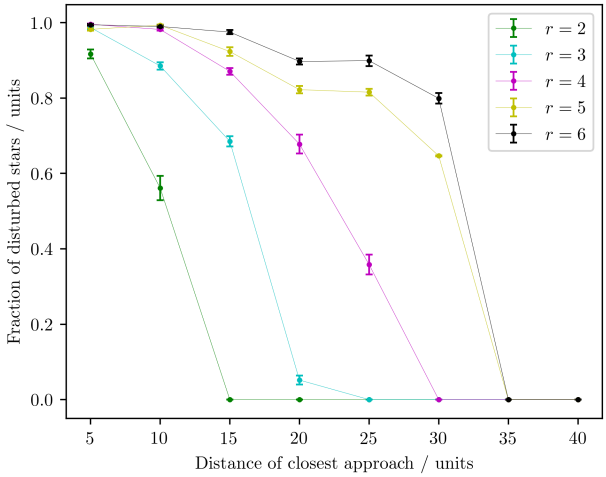
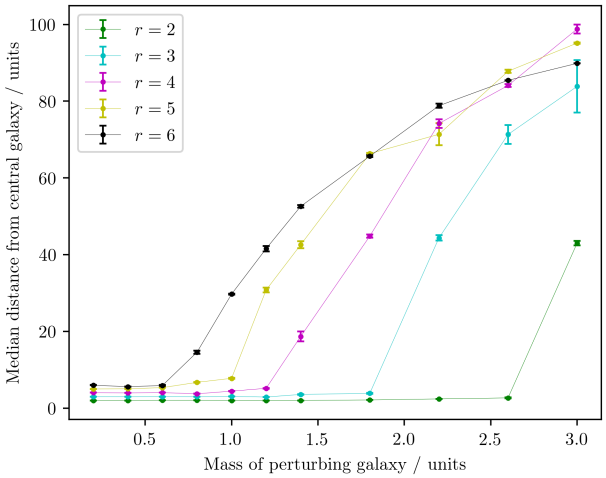


Figure 5: *Plots of the fraction of stars disturbed from their initial orbit against the mass of the perturbing galaxy (top plot) and the distance of closest approach (bottom plot) for each initial orbiting radius.*



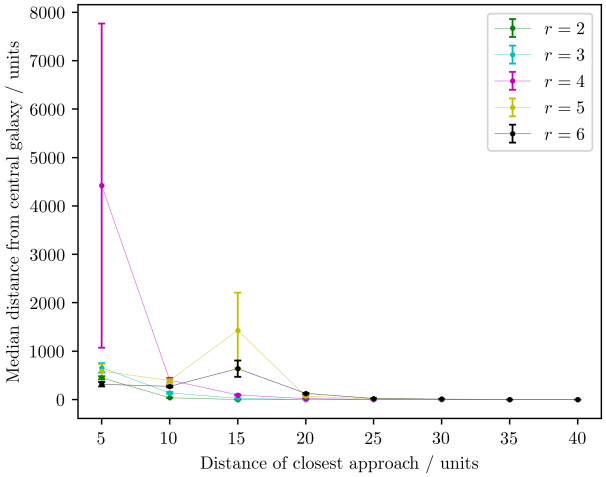


Figure 6: *Plots of the median distance between the test particles and the central galaxy mass of the perturbing galaxy (top plot) and the distance of closest approach (bottom plot) for each initial orbiting radius.*

results, where a smaller distance of approach does not necessarily mean stars with larger orbits can flung out further. However, the inconclusive results, large errors and absurdly large values here might be a product of the simulation setup, and this will be discussed in the following section §4.3.

4.3. Sources of Errors

From the results above, in particular for the median distance between the test particles and the central galaxy, there appears to be a large source of error. Looking into the setup of the simulation, this large error can be traced to the fact that the galaxies are treated as point masses. When a test particle gets very close ($r \to 0$) to the galaxies represented as point masses, the acceleration of the test particle, given as $a_x = GM(x - X)/r^3$, will become extremely large. This results in the test particle being 'shot' extremely far outwards, and this effect can actually be observed in the absurdly large error and distance in the bottom plot of Figure 6. In order to mitigate this effect, the massive galaxies should be modelled to either have a finite radius or tweaking the potential such that the galaxies have a repulsive force at small radii. This was also the primary reason that the median rather than the average distance was used as the quantitative metric.

Another potential source of error in this simulation is the use of a finite timestep. It is impossible to achieve an infinitesimally small timestep due to the restrictions of a classical computer, however a sufficiently small finite timestep would create a reliable and accurate simulation. The choice of picking a timestep of `dt = 0.1` was a balance between the accuracy and the efficiency of the simulation. A larger timestep can lead to inaccuracies and can potentailly lead to a divergence of the positions and velocities of the simulated particles. A smaller timestep would mean a slower program, where given the timestep method is of order $\mathcal{O}(N^2)$, a decrease of the timestep by a factor of 10 would result in a runtime of 100 times longer, which is not ideal!

A final, albeit minor, factor for the error of this simulation is the assumption that the test particles are collisionless. Collisions between stars can change the trajectories of the stars and potentially result in a different outcome to the simulation. The concept of viscosity in instellar medium arises from such collisions, and this can lead to a different and/or more interesting fluid dynamical outcome. However, the scale of a galaxy is much bigger than that of a star, and such effects are often negligible in scenarios like this in the simulation. Thus, the assumption of collisionless test particles is a sound assumption in this simulation.

5. Conclusion

Using the simple timestep method, this simulation was able to successfully simulate the formation of tidal tails through the close encounter between two galaxies. This simulation was able to recreate parts of the results of the simulations performed by Toomre and Toomre in 1972, where prograde encounters were found to be significantly more disruptive than retrograde encounters. The strong tidal effects in prograde encounters were found to be the cause of the formation of tidal tails.

In addition, experiments varying the mass of the perturbing galaxy and the distance of closest approach between the two galaxies showed intuitive results. These experiments showed that a heavier perturbing galaxy and/or a smaller distance of closest approach would result in a larger disruption caused by the perturbing galaxy. A larger proportion of stars were disturbed and displaced from their initial orbits, and the median distance between the test particles and the central galaxy was larger. Test particles with larger initial radius was also found to be more easily disturbed than those with a smaller initial radius.

References

[1] M. Alavi and H. Razmi. On the tidal evolution and tails formation of disc galaxies. *Astrophysics and Space Science*, 360(26), 2015.

[2] B. Dunbar. Tadpole's tidal tail. `https://www.nasa.gov/multimedia/imagegallery/image_feature_573.html`. Accessed: 21 March 2020.

[3] A. Toomre, J. Toomre. Galactic bridges and tails. *Astrophysical Journal*, 178:623–666, 1972.

[4] D. W. Keenan and K. A. Innanen. Numerical investigation of galactic tidal effects on spherical stellar systems. *Astronomical Journal*, 80:290–302, 1975.

[5] R. Fitzpatrick. Circular restricted three-body problem. `http://farside.ph.utexas.edu/teaching/336k/Newtonhtml/node120.html`, 2011. Accessed: 24 March 2020.

[6] M. C. Payne. NST 1B Physics B (2019/2020) Dynamics: Handout II, January 2019.

6. Appendix

core.py

```
import numpy as np

radii_units_colors = [
    (2, 120, 'g'),
    (3, 180, 'c'),
    (4, 240, 'm'),
    (5, 300, 'y'),
    (6, 360, 'k')
]

G = 1
clockwise = False
mass_arr = [0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.8, 2.2, 2.6, 3.0]
r_min_arr = [5, 10, 15, 20, 25, 30, 35, 40]
turns = [1, 2, 3]



class Body:
    def __init__(self, position, velocity, mass=1, G=G):
        self.mass = mass
        self.G = G
        self.x = position[0]
        self.y = position[1]
        self.v_x = velocity[0]
        self.v_y = velocity[1]
        self.f_x = 0
        self.f_y = 0

    def set_gforce(self, bodies):
        self.f_x = 0
        self.f_y = 0

        for body in bodies:
            d_x = body.x - self.x
            d_y = body.y - self.y
            distance = np.sqrt(d_x**2 + d_y**2)
            force_mag = self.G * body.mass * self.mass / distance**2
            self.f_x += force_mag * d_x / distance
            self.f_y += force_mag * d_y / distance

        return self

    def update_speed_position(self, dt):
        self.v_x += dt * self.f_x / self.mass
        self.v_y += dt * self.f_y / self.mass
        self.x += dt * self.v_x
        self.y += dt * self.v_y

        return self


def initialise_moving_body(initial_x, r_min, mass, central_body, clockwise=False):
    # parabolic orbit: y^2 = 4 * r_min^2 - 4 * r_min * x
    moving_x = -50
```

```
    moving_y = (1 if clockwise else -1) * 2 * np.sqrt(r_min**2 - r_min * moving_x)

    dy_dx = (-1 if clockwise else 1) * r_min / np.sqrt(r_min**2 - r_min * moving_x)
    moving_v_mag = np.sqrt(2 * G * central_body.mass / np.sqrt(moving_x**2 + moving_y**2))
    moving_v_y = dy_dx * moving_v_mag / np.sqrt(1 + dy_dx**2)
    moving_v_x = moving_v_y / dy_dx

    return Body([moving_x, moving_y], [moving_v_x, moving_v_y], mass=mass)

def initialise_test_particles(central_body):
    all_particles = []

    for (radius, num_of_units, color) in radii_units_colors:
        thetas = np.random.rand(num_of_units) * 2 * np.pi
        v_mag = np.sqrt(G * central_body.mass / radius)

        init_x = lambda t : radius * np.cos(t)
        init_y = lambda t : radius * np.sin(t)
        init_v_x = lambda t : -v_mag * np.sin(t)
        init_v_y = lambda t : v_mag * np.cos(t)

        particles = [Body(
            [init_x(theta), init_y(theta)],
            [init_v_x(theta), init_v_y(theta)],
            mass = 0.01
        ) for theta in thetas]

        all_particles.append(particles)

    all_particles = [part for radius_particles in all_particles for part in
        radius_particles]
    return all_particles


def get_positions_from_row(row):
    time = row[0]
    central_x = row[1]
    central_y = row[2]
    moving_x = row[3]
    moving_y = row[4]
    particles_positions = row[5:]
    all_particles_x = [coord for idx, coord in enumerate(particles_positions) if idx % 2
        == 0]
    all_particles_y = [coord for idx, coord in enumerate(particles_positions) if idx % 2
        == 1]

    particles_xs = []
    particles_ys = []
    index = 0
    for radius, num, color in radii_units_colors:
        particles_xs.append(np.array(all_particles_x[index:index + num]))
        particles_ys.append(np.array(all_particles_y[index:index + num]))
        index += num

    return (time, central_x, central_y, moving_x, moving_y, particles_xs, particles_ys)
```

simulation.py

```
import os
import time
import sys
import matplotlib.pyplot as plt
import numpy as np
from core import Body, initialise_moving_body, initialise_test_particles
from core import G, turns, r_min_arr, mass_arr, radii_units_colors, clockwise


try:
    if len(sys.argv) < 2 or sys.argv[1] not in ["rmin", "mass", "single"]:
        raise OSError("An argument of 'single', 'rmin' or 'mass' needs to be provided in
            order to read the data")

    variable_type = sys.argv[1]
    variable_arr = []
    if variable_type == "single":
        variable_arr = [0]
        turns = [0]
    else:
        variable_arr = r_min_arr if variable_type == "rmin" else mass_arr

    if not os.path.isdir('data/'):
        os.mkdir('data')

    for val in variable_arr:
        for turn in turns:
            print(f"Simulating {variable_type}: {val}, turn: {turn}")
            start = time.time()

            # initialising the central galaxy
            central_body = Body([0, 0], [0, 0], mass=1)

            # initialising the perturbing galaxy
            r_min = 20
            moving_mass = 1
            if variable_type != "single":
                r_min = val if variable_type == "rmin" else 20
                moving_mass = val if variable_type == "mass" else 1

            moving_body = initialise_moving_body(-40, r_min, moving_mass, central_body,
                clockwise=clockwise)

            # initialising the test particles
            all_particles = initialise_test_particles(central_body)

            dt = 0.1
            t = 0
            t_max = 700

            with open(f"data/positions_{'clockwise_' if clockwise else ''}{variable_type}_
                {val}_{turn}.csv", "w+") as file:
                header = "time,central_x,central_y,moving_x,moving_y"
                for radius, num_of_units, color in radii_units_colors:
                    header_str = ",".join([f"particle_{radius}_{i}_x,particle_{radius}_{i}
                        _y" for i in range(num_of_units)])
                    header += "," + header_str
```

```
                file.write(header + "\n")

                count = 0
                while (t < t_max):
                    row = str(t) + ","

                    # update the force, position and velocity of the central galaxy
                    central_body.set_gforce([moving_body])
                    central_body.update_speed_position(dt)
                    row += str(central_body.x) + "," + str(central_body.y)

                    # update the force, position and velocity of the perturbing galaxy
                    moving_body.set_gforce([central_body])
                    moving_body.update_speed_position(dt)
                    row += "," + str(moving_body.x) + "," + str(moving_body.y)

                    # update the force, position and velocity of all the test particles
                    all_particles = [part.set_gforce([central_body, moving_body]) for part
                        in all_particles]
                    all_particles = [part.update_speed_position(dt) for part in
                        all_particles]
                    particle_positions = [str(part.x) + "," + str(part.y) for part in
                        all_particles]
                    row += "," + ",".join(particle_positions)

                    file.write(row + "\n")

                    t += dt
                    if count % 1000 == 0:
                        print(f"Simulation at t = {int(t)} / {t_max} units")

                    count += 1

            end = time.time()
            print(f"Time taken for simulation: {end - start} \n")

except OSError as e:
    print(repr(e))
```

animation.py

```
import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from core import radii_units_colors
from core import get_positions_from_row


try:
    if len(sys.argv) < 2:
        raise OSError("Provide an existing data file name from the data/ folder")

    filename = sys.argv[1]
    if not os.path.isfile(f"data/{filename}"):
        raise OSError(f"The file {filename} does not exist. Please provide an existing
            data file in the data/ folder")

    print("Reading file...")
    positions = pd.read_csv(f"data/{filename}", delimiter=",")
    print("File read.")

    animation_rows = np.linspace(0, len(positions)- 1, 100, dtype=int)
    animation_positions = positions.iloc[animation_rows,:]

    for idx, row in animation_positions.iterrows():
        (time, central_x, central_y, moving_x, moving_y, particles_xs, particles_ys) =
            get_positions_from_row(row)

        plt.clf()
        plt.scatter(central_x, central_y, color='b')
        plt.scatter(moving_x, moving_y, color='r')
        for idx, (radius, num, color) in enumerate(radii_units_colors):
            plt.scatter(particles_xs[idx], particles_ys[idx], s=1, color=color, label=rf"
                $r = {radius}$")

        plt.xlim([-100, 100])
        plt.ylim([-100, 100])
        plt.title(time)
        plt.pause(0.01)

except OSError as e:
    print(repr(e))
```

get_quantitative.py

```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from core import radii_units_colors, r_min_arr, mass_arr, turns
from core import get_positions_from_row


try:
    if len(sys.argv) < 2 or sys.argv[1] not in ["rmin", "mass"]:
        raise OSError("An argument of 'rmin' or 'mass' needs to be provided in order to
            read the data")

    variable_type = sys.argv[1]
    variable_arr = r_min_arr if variable_type == "rmin" else mass_arr

    with open(f"data/quantitative_{variable_type}.csv", "w+") as file:

        for val in variable_arr:
            all_radii = np.zeros([len(radii_units_colors), 3])
            all_averages = np.zeros([len(radii_units_colors), 3])

            for turn in turns:
                print(f"Reading file ... (type: {variable_type}, value: {val}, Turn: {turn
                    })")
                positions = pd.read_csv(f"data/positions_{variable_type}_{val}_{turn}.csv"
                    , delimiter=",")
                print("File read.")

                positions["pos_difference"] = np.sqrt(np.square(positions.central_x -
                    positions.moving_x) + np.square(positions.central_y - positions.
                    moving_y))
                closest_approach = positions[positions.pos_difference == positions.
                    pos_difference.min()].iloc[0]

                # this snapshot is at approximately t = 50 units
                snapshot_row = closest_approach.name + 1000
                if snapshot_row > len(positions):
                    continue
                row = positions.iloc[snapshot_row]
                (time, central_x, central_y, moving_x, moving_y, particles_xs,
                    particles_ys) = get_positions_from_row(row)

                for idx, (radius, num, color) in enumerate(radii_units_colors):
                    distance_to_central = np.sqrt(np.square(particles_xs[idx] - central_x)
                        + np.square(particles_ys[idx] - central_y))
                    average = np.median(distance_to_central)
                    all_averages[idx][turn - 1] = average

                    disturbed_number = len([dist for dist in distance_to_central if dist >
                        (radius + 0.5) or dist < (radius - 0.5)])
                    disturbed_frac = disturbed_number / num
                    all_radii[idx][turn - 1] = disturbed_frac

                    print(f"RADIUS: {radius} - No. of disturbed particles: {
                        disturbed_number}/{num}, Average distance: {average}")
```

```
            av_of_averages = np.average(all_averages, axis=1)
            err_of_averages = np.std(all_averages, axis=1) / np.sqrt(len(turns))
            av_of_averages = [str(av) for av in av_of_averages]
            err_of_averages = [str(std) for std in err_of_averages]

            av_of_radii = np.average(all_radii, axis=1)
            err_of_radii = np.std(all_radii, axis=1) / np.sqrt(len(turns))
            av_of_radii = [str(av) for av in av_of_radii]
            err_of_radii = [str(std) for std in err_of_radii]

            row = f"{val}"
            for av_std_arr in [av_of_averages, err_of_averages, av_of_radii, err_of_radii
                ]:
                row += "," + ','.join(av_std_arr)
            row += "\n"
            file.write(row)

    print(f"Created file data/quantitative_{variable_type}.csv")

except OSError as e:
    print(repr(e))
```