Astrophysical systems using Runge-Kutta

24511

Abstract

This is a computational investigation into astrophysical systems. A variety of Runge-Kutta methods were used to model trajectories of comets and N-body systems.

1. Introduction

Computational modelling has become a critical tool within theoretical astrophysics, providing rapid analysis that would otherwise not be possible. In this study variations of the Runge-Kutta (RK) algorithm were applied to second-order ordinary differential equations (ODEs) to model the trajectory of Halley's comet and N-body systems.

The RK methods are an integration technique that estimates slopes at different points and takes a weighted average. This builds upon Euler's method, which is known as the first order RK method. Higher orders of the RK method give better accuracy and do not require any specific derivatives.

1.1a

A programme using the fourth-order RK method to model the orbit of Halley's comet around the Sun that obeys the equation,

$$m\frac{d^2\vec{\boldsymbol{r}}}{dt^2} = -\left(\frac{GMm}{r^2}\right)\frac{\vec{\boldsymbol{r}}}{r}. \tag{1}$$

Assuming the Sun of mass M is stationary in and the comet of mass m orbits in the z=0 plane.

1.1b

Developed 1a by implementing an adaptive time step, in the form of the Runge-Kutta Cash-Karp[1] (RK CK) method. This method uses 6 functions and calculates the difference in fifth and fourth-order solutions; this embedded method gives the error of the fourth-order solution, which changes step size appropriately.

$$\begin{array}{c|cccccc}
0 & 0 \\
\frac{1}{5} & \frac{1}{5} & 0 \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & 0 \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & 0 \\
1 & -\frac{11}{54} & \frac{5}{2} & -\frac{70}{27} & \frac{35}{27} & 0 \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & 0
\end{array}$$

FIG. 1. The Butcher tableau of the CK method.

Astronomical values for Halley's comet were obtained from online sources[2].

Halley's Comet	Aphelion (10^6 km)
	87.66

TAB. 1. Table of relevant astronomical constants.

1.2a

A programme using the fourth-order RK method to model a 3-body problem, consisting of two bodies of mass:

$$m1/_M = 10^{-3}, \qquad m2/_M = 4\times 10^{-2}$$

and circular orbits:

$$a1 = 2.52 \text{ AU}, \qquad a2 = 5.24 \text{ AU}$$

respectively, orbiting a fixed third body of mass M. Orbital velocity and period were calculated using:

$$v = \sqrt{\frac{GM}{R}} \tag{2}$$

and Kepler's 3rd law:

$$P^2 = \frac{4\pi^2 a^3}{G(m1+m2)}. \tag{3}$$

This programme assumed circular orbits to calculate initial values and orbited in the z=0 plane.

1.2b

Developed 2a by transforming the coordinate system to a frame based upon the CoM, where the third body is no longer fixed. Modelling the orbits of Jupiter, Saturn and the Sun around the Jupiter-Sun barycentre. Position and velocity vectors for the CoM:

$$\overrightarrow{r_{cm}}(t) = \sum_i \frac{m_i}{M_{total}}\vec{r_i}(t) \tag{4}$$

and

$$\overrightarrow{v_{cm}}(t) = \sum_i \frac{m_i}{M_{total}}\vec{v_i}(t). \tag{5}$$

Astronomical values for Jupiter, Saturn and the Sun. were obtained from online sources[3,4].

	Min. orbital velocity (km/s)	Aphelion (10^6 km)	Perihelion (10^6 km)
Jupiter	12.44	816.4	740.6
Saturn	9.09	1,506.53	1,357.55
Sun	-12	-	-

TAB. 2. Table of relevant astronomical constants.

2. Method

1.1a

Flowchart and code are detailed in A1.1a and B1.1a respectively. This programme used 3 black boxes: orbits, RK4step and RK4intergrate. The orbits function takes the initial conditions and returns an array of the 4 first-order ODEs, initially derived from the second-order ODE equation, equation 1. The RK4step function performs the RK4 algorithm and returns an array of ODE solutions. The RK4intergration function then iterates through RK4step over the total time span and stores the solutions in a list. Providing information on position and velocity of the system.

1.1b

Flowchart and code are detailed in A1.1b and B1.1b respectively. This programme used 3 black boxes: orbits, RKCK and RKCKintergrate, where orbits act the same as in 1.1a. The RKCK algorithm is an embedded function where the step size changes based on the error. This function returns step size, time and an array of ODE solutions. The RKCKintergration function iterates through RKCK whilst the time is less than the end time (total period). The solutions are appended to a list until this condition is broken. The list provides information on position and velocity.

1.2a

Flowchart and code are detailed in A1.2a and B1.2a respectively. Initial values were calculated using equations 2 and 3. This programme used 3 black boxes: orbits, RK4step and RK4intergrate. The functions behaved as previously mentioned, the only difference being an additional object was accounted for.

1.2b

Flowchart and code are detailed in A1.2b and B1.2b respectively. Initial values for Jupiter and Saturn's position was calculated using equation 4, the other initial conditions are in table 2. This programme used 3 black boxes the same as in 1.2a. Although this model now dealt with 3 orbiting bodies around a common CoM.

3. Results and Discussion

The sun is represented by a black dot.

1.1a

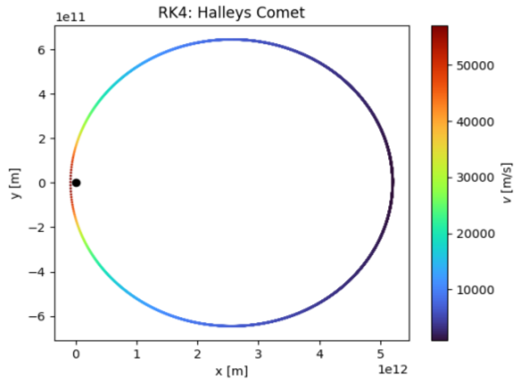


FIG. 2. Orbit of Halley's comet using RK4.
N = 10, 000 and P = 79 years.

The orbit began at the aphelion with a period close to Halley's period. This produced an elliptical orbit. Where the plot is fastest the points appear most spread out, this is a drawback from the static RK4 method.

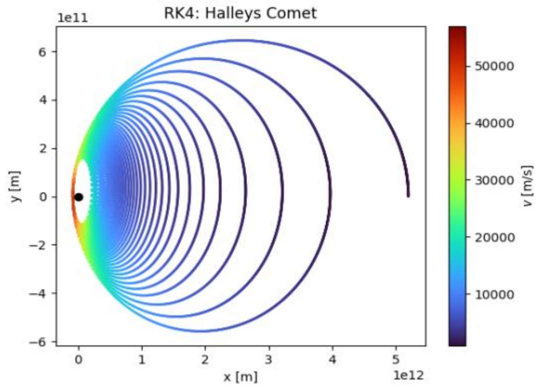


FIG. 3. Orbit of Halley's comet using RK4.
N = 10, 000 and P = 359 years.

In figure 3 multiplying the period by a factor of 5 depicts the instability of the static RK4 model. The static model requires a large sampling rate, or the system decays. Although the large sampling rate effects computational speed.

1.1b

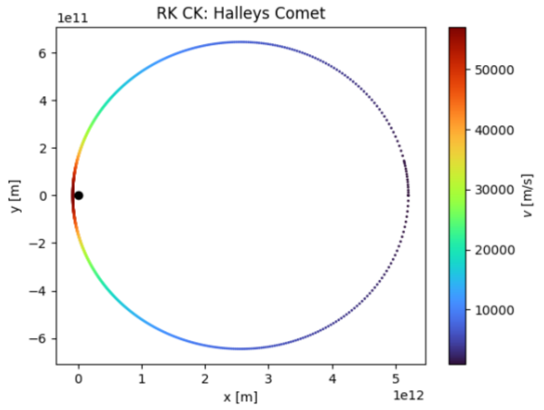


FIG. 4. Orbit of Halley's comet using RK CK.
N = 10, 000 and P = 79 years.

In figure 4 the largest velocity appears at the perihelion, as the scatter plot is most dense here. The aphelion appears denser than expected due to a larger value than the actual period being used. The adaptive model can produce accurate plots with less sampling points, making it a more computationally efficient tool.

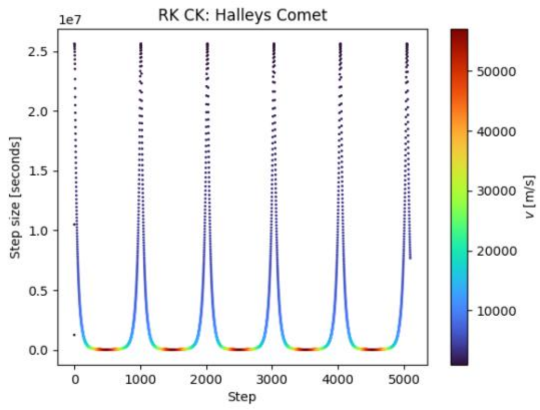


FIG. 5. Timestep variation of Halley's comet using RK CK. N = 10, 000 and P = 359 years.

Figure 5 depicts the change in time between each timestep. At large velocities (red) the timestep is smallest, suggesting the success of the adaptive time step. When compared to the RK4 algorithm, which has a constant time step.

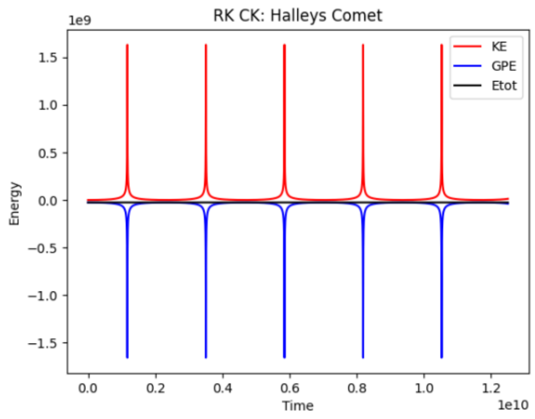


FIG. 6. Conservation of energy using RK CK. N = 10, 000 and P = 359 years.

In figure 6 the total energy of the system is conserved over 5 periods, suggesting the long-term stability of the RK CK algorithm.

Halley's Comet	Aphelion (10^6 km)	Uncertainty (%)
RK4	80.07	8.7
RK CK	80.11	8.6

TAB. 3. Table of calculated astronomical values.

The two models are inaccurate, this could be due to the initial values used. The adaptive step however is a slightly better estimation.

1.2a

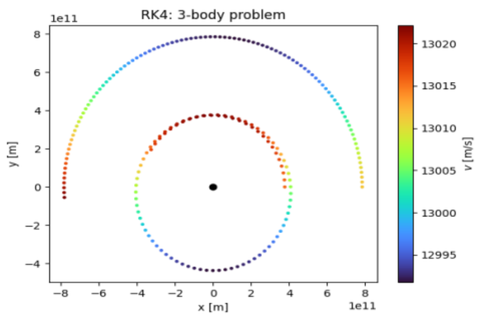


FIG. 7. 3 body problem around stationary Sun. N = 100 and P = 6.2 years.

Figure 7 details the interactions of the two bodies, the difference in period suggests planet 1 (inner) surpasses planet 2 (outer) roughly 1.5 times during planet 2's period. The increases in velocity correspond to the planets interacting with each other's orbits, supported by figure 8.

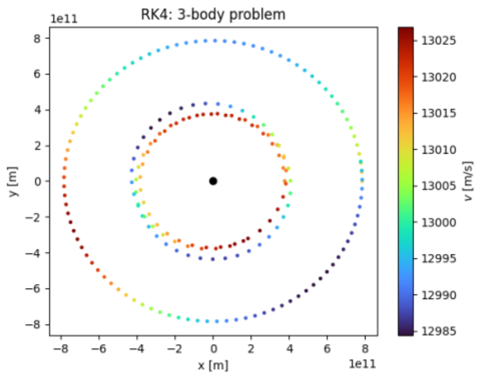


FIG. 8. 3 body problem around stationary Sun. N = 10, 000 and P = 12.4 years.

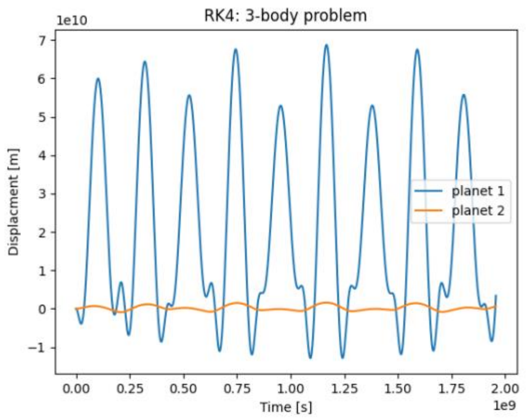


FIG. 9. Displacement of bodies orbiting Sun. N = 10000 and P = 62 years.

Figure 9 shows perturbations of planets 1 and 2. Planet 1 shows much larger displacement than 2, on inspection slight orbital resonance can be seen between the two bodies.

1.2b

These simulations started at the aphelion as the minimum velocities were used. This system was modelled around the Jupiter-Sun barycentre.

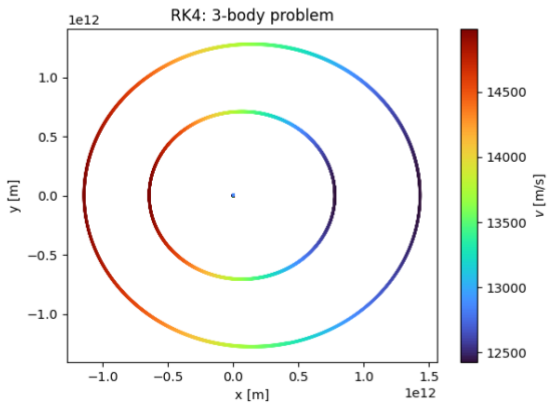


FIG. 10. 3 Displacement of bodies orbiting Jupiter-Sun barycentre. N = 10000 and P = 95 years.

Figure 10 suggests an elliptical orbit, velocities are largest at the perihelion as expected.

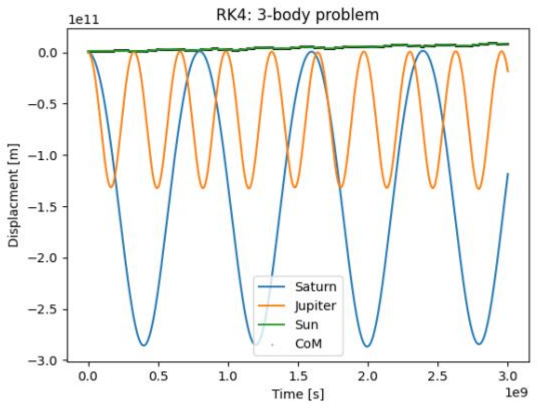


FIG. 11. Displacement of bodies orbiting common CoM. N = 10000 and P = 95 years.

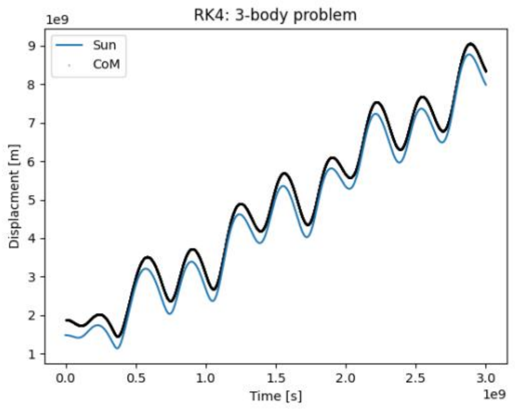


FIG. 12. Displacement of Sun orbiting CoM. N = 10000 and P = 95 years.

Figures 11 and 12 show the perturbations over multiple periods of the systems as expected the 3 bodies move with the CoM. The resonance between the orbits effects the Suns trajectory, this is most visible from the Suns wobbles. Values calculated from simulation are as follows.

	Aphelion (10^6 km)	Perihelion (10^6 km)	
Jupiter	778.3	645.8	
Saturn	1,430.82	1,145.08	

TAB. 4. Table of calculated astronomical constants.

Compared to the actual values there is an uncertainty of 4.7% and 12.7% for the perihelion and aphelion of Jupiter respectively. Similarly, an uncertainty of 5.0% and 15.6% for the perihelion and aphelion of Saturn. Suggesting the inaccuracy of this model. Improved initial values could help model this more accurately. The simulation provides a fairly accurate model of N-body systems. Future improvements to the simulation would be to develop this framework for N body systems, analyse the energy of these N-body system and experiment with other algorithms, such as the Verlet method.

4. Conclusions

It has been shown that the motion of a comet can be modelled using the RK4 method, further investigation into RK algorithms showed the RK CK provided long term accuracy to this model. It has also been possible to simulate an N-body system using the RK4 method. Simulations supported theory, however implementation of more suitable algorithm such as the Verlet model and generalising the framework to N-bodies would provide more accurate analysis.

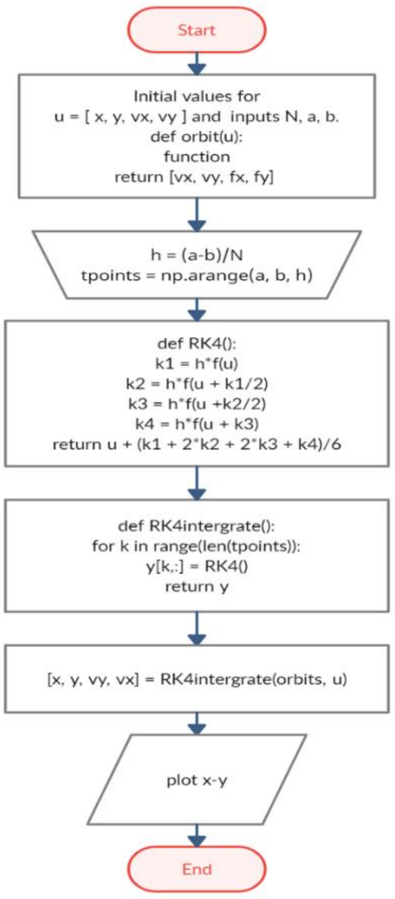
References

1. Cash J, Karp A. ACM Transactions on Mathematical Software. 201-222 (1990)
2. Conservation of momentum. www.sciencedirect.com . (n.d.). Halley's Comet – an overview Science Direct Topics.[Internet]. cited 9 March 2022]. Available from: https://www.sciencedirect.com/topics/physics-and-astronomy/halleys-comet
3. Wright J, Kanodia S. Barycentric Corrections for Precise Radial Velocity Measurements of Sunlight. The Planetary Science Journal. 1(2):38. (2020)
4. Planetary Fact Sheet [Internet]. Nssdc.gsfc.nasa.gov. 2022 [cited 9 March 2022]. Available from: https://nssdc.gsfc.nasa.gov/planetary/factsheet/?fbclid=IwAR2bwpqtM2Ij-uKBAfHka4QzYxXxQfR7KGBPs2FJ3VKfQwsFraxNR-R41v0

Appendix

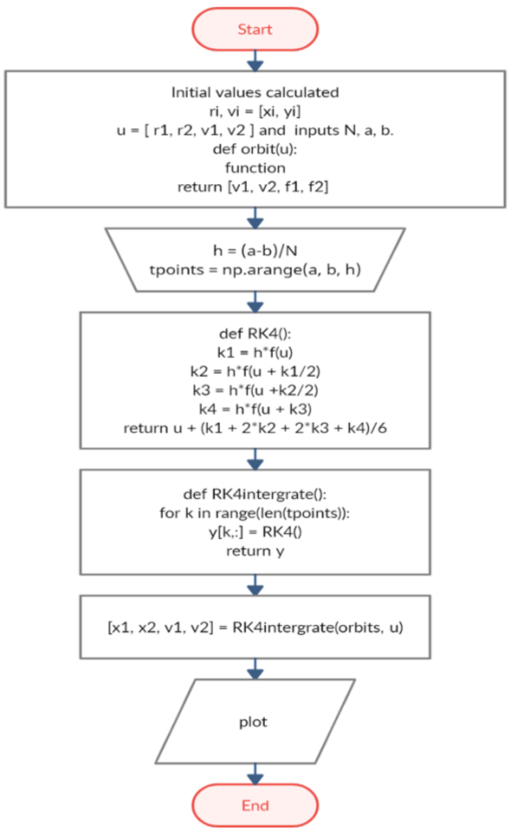
Appendix A

A1.1a



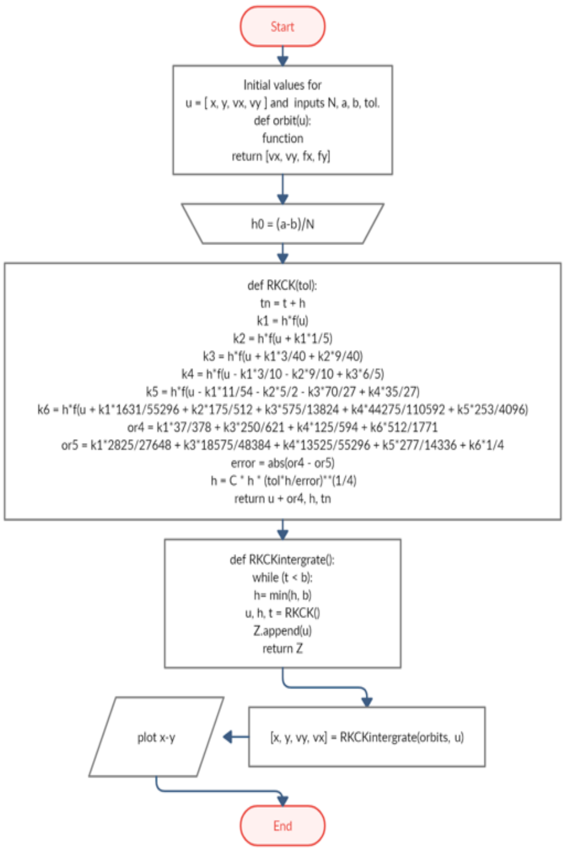
```
Start

Initial values for
u = [ x, y, vx, vy ] and  inputs N, a, b.
def orbit(u):
function
return [vx, vy, fx, fy]

h = (a-b)/N
tpoints = np.arange(a, b, h)

def RK4():
k1 = h*f(u)
k2 = h*f(u + k1/2)
k3 = h*f(u +k2/2)
k4 = h*f(u + k3)
return u + (k1 + 2*k2 + 2*k3 + k4)/6

def RK4intergrate():
for k in range(len(tpoints)):
y[k,:] = RK4()
return y

[x, y, vy, vx] = RK4intergrate(orbits, u)

plot x-y

End
```

A1.2a



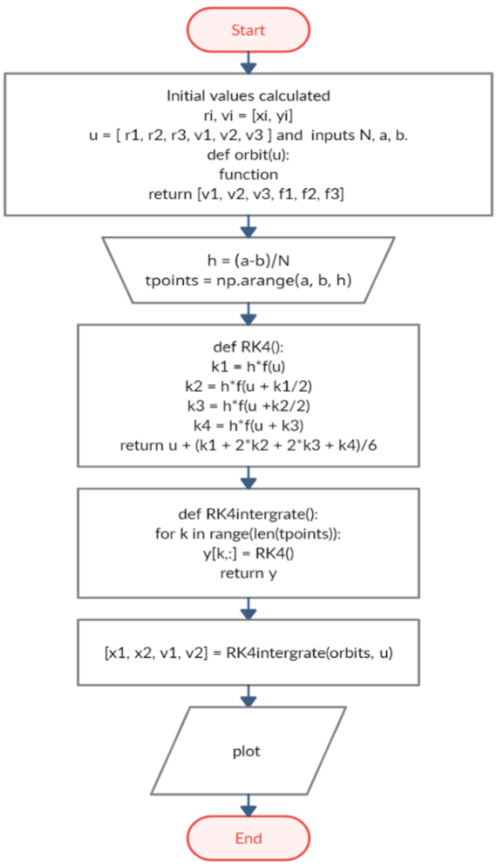
```
Start

Initial values calculated
ri, vi = [xi, yi]
u = [ r1, r2, v1, v2 ] and  inputs N, a, b.
def orbit(u):
function
return [v1, v2, f1, f2]

h = (a-b)/N
tpoints = np.arange(a, b, h)

def RK4():
k1 = h*f(u)
k2 = h*f(u + k1/2)
k3 = h*f(u +k2/2)
k4 = h*f(u + k3)
return u + (k1 + 2*k2 + 2*k3 + k4)/6

def RK4intergrate():
for k in range(len(tpoints)):
y[k,:] = RK4()
return y

[x1, x2, v1, v2] = RK4intergrate(orbits, u)

plot

End
```

A1.1b



```
Start

Initial values for
u = [ x, y, vx, vy ] and  inputs N, a, b, tol.
def orbit(u):
function
return [vx, vy, fx, fy]

h0 = (a-b)/N

def RKCK(tol):
tn = t + h
k1 = h*f(u)
k2 = h*f(u + k1*1/5)
k3 = h*f(u + k1*3/40 + k2*9/40)
k4 = h*f(u - k1*3/10 - k2*9/10 + k3*6/5)
k5 = h*f(u - k1*11/54 - k2*5/2 - k3*70/27 + k4*35/27)
k6 = h*f(u + k1*1631/55296 + k2*175/512 + k3*575/13824 + k4*44275/110592 + k5*253/4096)
or4 = k1*37/378 + k3*250/621 + k4*125/594 + k6*512/1771
or5 = k1*2825/27648 + k3*18575/48384 + k4*13525/55296 + k5*277/14336 + k6*1/4
error = abs(or4 - or5)
h = C * h * (tol*h/error)**(1/4)
return u + or4, h, tn

def RKCKintergrate():
while (t < b):
h= min(h, b)
u, h, t = RKCK()
Z.append(u)
return Z

[x, y, vy, vx] = RKCKintergrate(orbits, u)

plot x-y

End
```

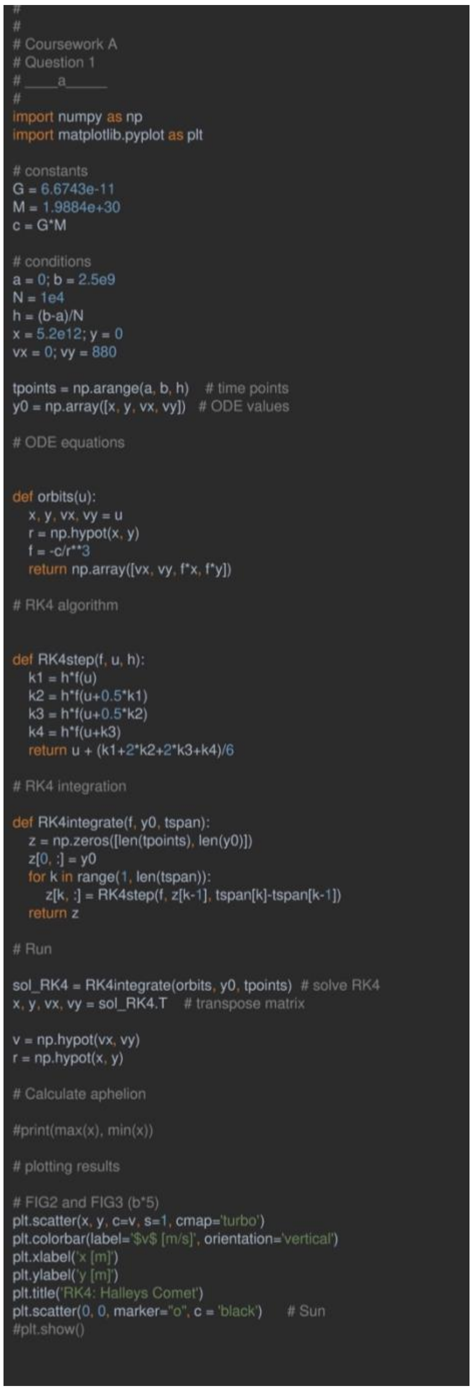
A1.2b



```
Start

Initial values calculated
ri, vi = [xi, yi]
u = [ r1, r2, r3, v1, v2, v3 ] and  inputs N, a, b.
def orbit(u):
function
return [v1, v2, v3, f1, f2, f3]

h = (a-b)/N
tpoints = np.arange(a, b, h)

def RK4():
k1 = h*f(u)
k2 = h*f(u + k1/2)
k3 = h*f(u +k2/2)
k4 = h*f(u + k3)
return u + (k1 + 2*k2 + 2*k3 + k4)/6

def RK4intergrate():
for k in range(len(tpoints)):
y[k,:] = RK4()
return y

[x1, x2, v1, v2] = RK4intergrate(orbits, u)

plot

End
```

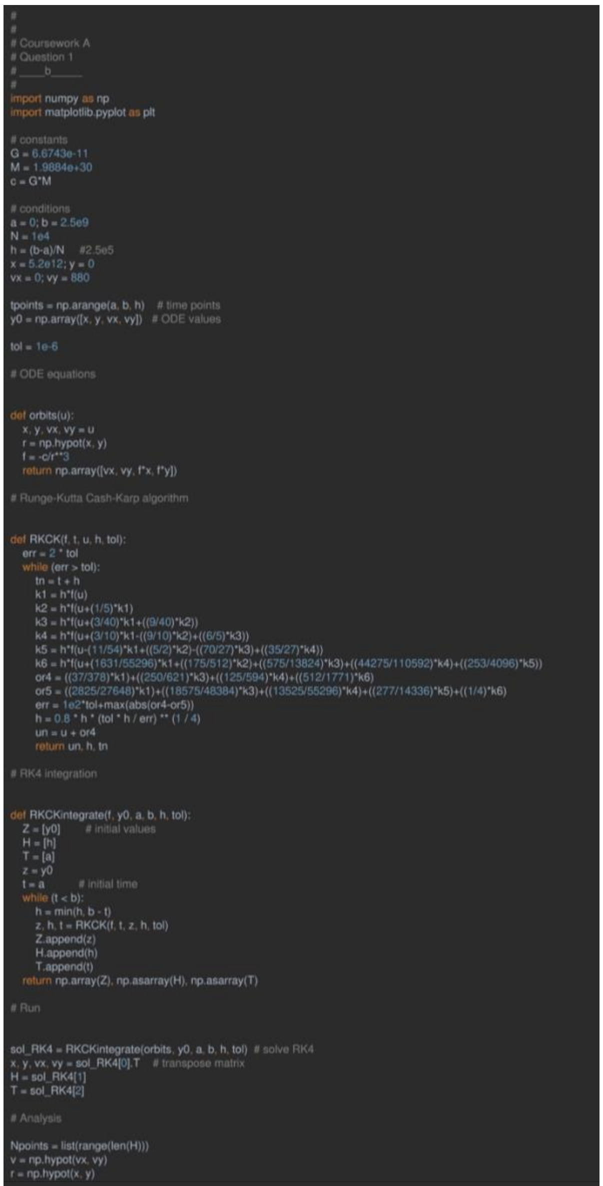
Appendix B

B1.1a

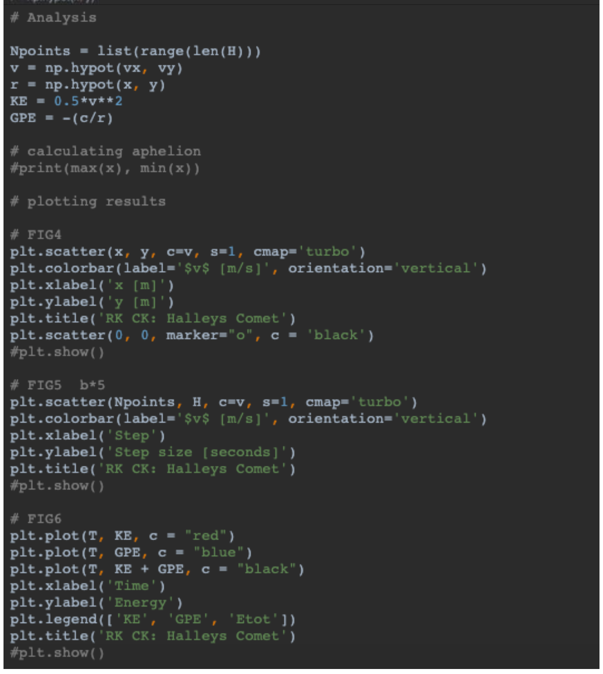


```
#
#
# Coursework A
# Question 1
# ____a_____
#
import numpy as np
import matplotlib.pyplot as plt

# constants
G = 6.6743e-11
M = 1.9884e+30
c = G*M

# conditions
a = 0; b = 2.5e9
N = 1e4
h = (b-a)/N
x = 5.2e12; y = 0
vx = 0; vy = 880

tpoints = np.arange(a, b, h)   # time points
y0 = np.array([x, y, vx, vy])   # ODE values

# ODE equations


def orbits(u):
    x, y, vx, vy = u
    r = np.hypot(x, y)
    f = -c/r**3
    return np.array([vx, vy, f*x, f*y])

# RK4 algorithm


def RK4step(f, u, h):
    k1 = h*f(u)
    k2 = h*f(u+0.5*k1)
    k3 = h*f(u+0.5*k2)
    k4 = h*f(u+k3)
    return u + (k1+2*k2+2*k3+k4)/6

# RK4 integration

def RK4integrate(f, y0, tspan):
    z = np.zeros([len(tpoints), len(y0)])
    z[0, :] = y0
    for k in range(1, len(tspan)):
        z[k, :] = RK4step(f, z[k-1], tspan[k]-tspan[k-1])
    return z

# Run

sol_RK4 = RK4integrate(orbits, y0, tpoints)  # solve RK4
x, y, vx, vy = sol_RK4.T    # transpose matrix

v = np.hypot(vx, vy)
r = np.hypot(x, y)

# Calculate aphelion

#print(max(x), min(x))

# plotting results

# FIG2 and FIG3 (b*5)
plt.scatter(x, y, c=v, s=1, cmap='turbo')
plt.colorbar(label='$v$ [m/s]', orientation='vertical')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.title('RK4: Halleys Comet')
plt.scatter(0, 0, marker="o", c = 'black')      # Sun
#plt.show()
```

B1.1b

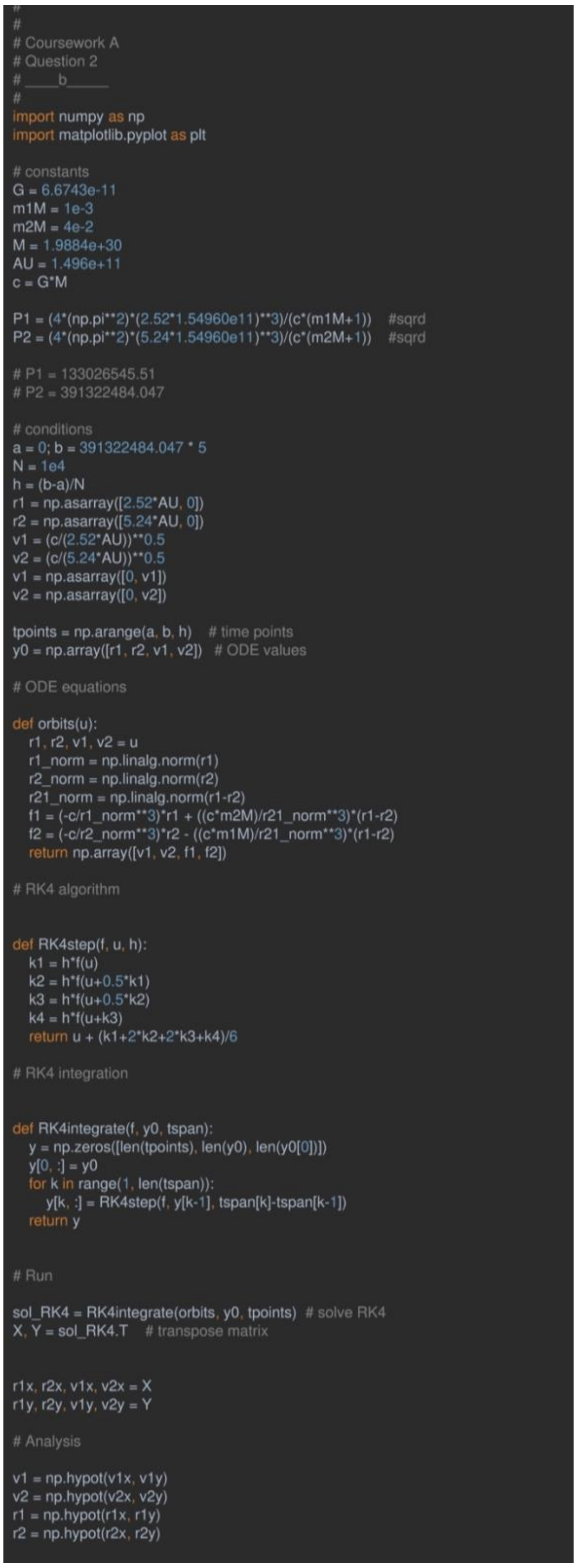


```
#
#
# Coursework A
# Question 1
# ____b_____
#
import numpy as np
import matplotlib.pyplot as plt

# constants
G = 6.6743e-11
M = 1.9884e+30
c = G*M

# conditions
a = 0; b = 2.5e9
N = 1e4
h = (b-a)/N    #2.5e5
x = 5.2e12; y = 0
vx = 0; vy = 880

tpoints = np.arange(a, b, h)    # time points
y0 = np.array([x, y, vx, vy])   # ODE values

tol = 1e-6

# ODE equations


def orbits(u):
    x, y, vx, vy = u
    r = np.hypot(x, y)
    f = -c/r**3
    return np.array([vx, vy, f*x, f*y])

# Runge-Kutta Cash-Karp algorithm


def RKCK(f, t, u, h, tol):
    err = 2 * tol
    while (err > tol):
        tn = t + h
        k1 = h*f(u)
        k2 = h*f(u+(1/5)*k1)
        k3 = h*f(u+(3/40)*k1+((9/40)*k2))
        k4 = h*f(u+(3/10)*k1-((9/10)*k2)+((6/5)*k3))
        k5 = h*f(u-(11/54)*k1+((5/2)*k2)-((70/27)*k3)+((35/27)*k4))
        k6 = h*f(u+(1631/55296)*k1+((175/512)*k2)+((575/13824)*k3)+((44275/110592)*k4)+((253/4096)*k5))
        or4 = ((37/378)*k1)+((250/621)*k3)+((125/594)*k4)+((512/1771)*k6)
        or5 = ((2825/27648)*k1)+((18575/48384)*k3)+((13525/55296)*k4)+((277/14336)*k5)+((1/4)*k6)
        err = 1e2*tol+max(abs(or4-or5))
        h = 0.8 * h * (tol * h / err) ** (1 / 4)
        un = u + or4
        return un, h, tn

# RK4 integration


def RKCKintegrate(f, y0, a, b, h, tol):
    Z = [y0]       # initial values
    H = [h]
    T = [a]
    z = y0
    t = a          # initial time
    while (t < b):
        h = min(h, b - t)
        z, h, t = RKCK(f, t, z, h, tol)
        Z.append(z)
        H.append(h)
        T.append(t)
    return np.array(Z), np.asarray(H), np.asarray(T)

# Run


sol_RK4 = RKCKintegrate(orbits, y0, a, b, h, tol)  # solve RK4
x, y, vx, vy = sol_RK4[0].T    # transpose matrix
H = sol_RK4[1]
T = sol_RK4[2]

# Analysis

Npoints = list(range(len(H)))
v = np.hypot(vx, vy)
r = np.hypot(x, y)
```

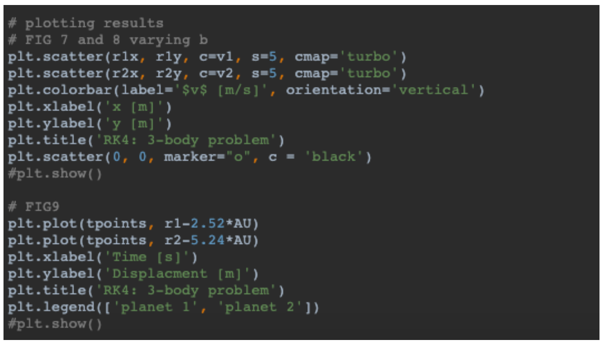


```
# Analysis

Npoints = list(range(len(H)))
v = np.hypot(vx, vy)
r = np.hypot(x, y)
KE = 0.5*v**2
GPE = -(c/r)

# calculating aphelion
#print(max(x), min(x))

# plotting results

# FIG4
plt.scatter(x, y, c=v, s=1, cmap='turbo')
plt.colorbar(label='$v$ [m/s]', orientation='vertical')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.title('RK CK: Halleys Comet')
plt.scatter(0, 0, marker="o", c = 'black')
#plt.show()

# FIG5  b*5
plt.scatter(Npoints, H, c=v, s=1, cmap='turbo')
plt.colorbar(label='$v$ [m/s]', orientation='vertical')
plt.xlabel('Step')
plt.ylabel('Step size [seconds]')
plt.title('RK CK: Halleys Comet')
#plt.show()

# FIG6
plt.plot(T, KE, c = "red")
plt.plot(T, GPE, c = "blue")
plt.plot(T, KE + GPE, c = "black")
plt.xlabel('Time')
plt.ylabel('Energy')
plt.legend(['KE', 'GPE', 'Etot'])
plt.title('RK CK: Halleys Comet')
#plt.show()
```

B1.2a

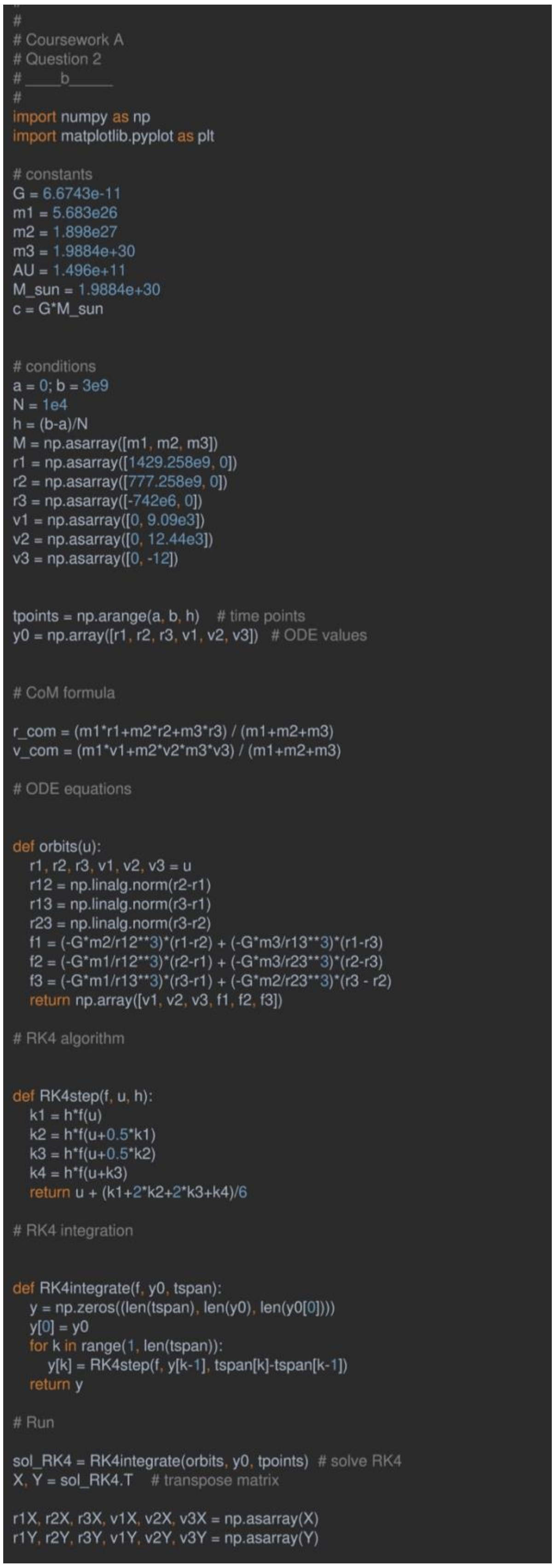


```python
#
#
# Coursework A
# Question 2
# ____b_____
#
import numpy as np
import matplotlib.pyplot as plt

# constants
G = 6.6743e-11
m1M = 1e-3
m2M = 4e-2
M = 1.9884e+30
AU = 1.496e+11
c = G*M

P1 = (4*(np.pi**2)*(2.52*1.54960e11)**3)/(c*(m1M+1))    #sqrd
P2 = (4*(np.pi**2)*(5.24*1.54960e11)**3)/(c*(m2M+1))    #sqrd

# P1 = 133026545.51
# P2 = 391322484.047

# conditions
a = 0; b = 391322484.047 * 5
N = 1e4
h = (b-a)/N
r1 = np.asarray([2.52*AU, 0])
r2 = np.asarray([5.24*AU, 0])
v1 = (c/(2.52*AU))**0.5
v2 = (c/(5.24*AU))**0.5
v1 = np.asarray([0, v1])
v2 = np.asarray([0, v2])

tpoints = np.arange(a, b, h)    # time points
y0 = np.array([r1, r2, v1, v2])   # ODE values

# ODE equations

def orbits(u):
   r1, r2, v1, v2 = u
   r1_norm = np.linalg.norm(r1)
   r2_norm = np.linalg.norm(r2)
   r21_norm = np.linalg.norm(r1-r2)
   f1 = (-c/r1_norm**3)*r1 + ((c*m2M)/r21_norm**3)*(r1-r2)
   f2 = (-c/r2_norm**3)*r2 - ((c*m1M)/r21_norm**3)*(r1-r2)
   return np.array([v1, v2, f1, f2])

# RK4 algorithm


def RK4step(f, u, h):
   k1 = h*f(u)
   k2 = h*f(u+0.5*k1)
   k3 = h*f(u+0.5*k2)
   k4 = h*f(u+k3)
   return u + (k1+2*k2+2*k3+k4)/6

# RK4 integration


def RK4integrate(f, y0, tspan):
   y = np.zeros([len(tpoints), len(y0), len(y0[0])])
   y[0, :] = y0
   for k in range(1, len(tspan)):
      y[k, :] = RK4step(f, y[k-1], tspan[k]-tspan[k-1])
   return y


# Run

sol_RK4 = RK4integrate(orbits, y0, tpoints)  # solve RK4
X, Y = sol_RK4.T    # transpose matrix


r1x, r2x, v1x, v2x = X
r1y, r2y, v1y, v2y = Y

# Analysis

v1 = np.hypot(v1x, v1y)
v2 = np.hypot(v2x, v2y)
r1 = np.hypot(r1x, r1y)
r2 = np.hypot(r2x, r2y)
```

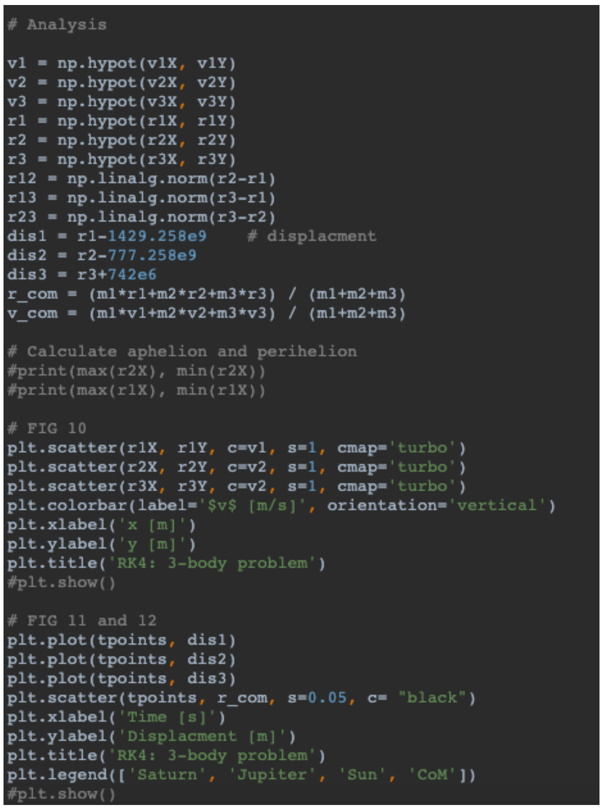


```python
# plotting results
# FIG 7 and 8 varying b
plt.scatter(r1x, r1y, c=v1, s=5, cmap='turbo')
plt.scatter(r2x, r2y, c=v2, s=5, cmap='turbo')
plt.colorbar(label='$v$ [m/s]', orientation='vertical')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.title('RK4: 3-body problem')
plt.scatter(0, 0, marker="o", c = 'black')
#plt.show()

# FIG9
plt.plot(tpoints, r1-2.52*AU)
plt.plot(tpoints, r2-5.24*AU)
plt.xlabel('Time [s]')
plt.ylabel('Displacment [m]')
plt.title('RK4: 3-body problem')
plt.legend(['planet 1', 'planet 2'])
#plt.show()
```

B1.2b



```
#
# Coursework A
# Question 2
# ____b_____
#
import numpy as np
import matplotlib.pyplot as plt

# constants
G = 6.6743e-11
m1 = 5.683e26
m2 = 1.898e27
m3 = 1.9884e+30
AU = 1.496e+11
M_sun = 1.9884e+30
c = G*M_sun


# conditions
a = 0; b = 3e9
N = 1e4
h = (b-a)/N
M = np.asarray([m1, m2, m3])
r1 = np.asarray([1429.258e9, 0])
r2 = np.asarray([777.258e9, 0])
r3 = np.asarray([-742e6, 0])
v1 = np.asarray([0, 9.09e3])
v2 = np.asarray([0, 12.44e3])
v3 = np.asarray([0, -12])


tpoints = np.arange(a, b, h)    # time points
y0 = np.array([r1, r2, r3, v1, v2, v3])   # ODE values


# CoM formula

r_com = (m1*r1+m2*r2+m3*r3) / (m1+m2+m3)
v_com = (m1*v1+m2*v2*m3*v3) / (m1+m2+m3)

# ODE equations


def orbits(u):
    r1, r2, r3, v1, v2, v3 = u
    r12 = np.linalg.norm(r2-r1)
    r13 = np.linalg.norm(r3-r1)
    r23 = np.linalg.norm(r3-r2)
    f1 = (-G*m2/r12**3)*(r1-r2) + (-G*m3/r13**3)*(r1-r3)
    f2 = (-G*m1/r12**3)*(r2-r1) + (-G*m3/r23**3)*(r2-r3)
    f3 = (-G*m1/r13**3)*(r3-r1) + (-G*m2/r23**3)*(r3 - r2)
    return np.array([v1, v2, v3, f1, f2, f3])

# RK4 algorithm


def RK4step(f, u, h):
    k1 = h*f(u)
    k2 = h*f(u+0.5*k1)
    k3 = h*f(u+0.5*k2)
    k4 = h*f(u+k3)
    return u + (k1+2*k2+2*k3+k4)/6

# RK4 integration


def RK4integrate(f, y0, tspan):
    y = np.zeros((len(tspan), len(y0), len(y0[0])))
    y[0] = y0
    for k in range(1, len(tspan)):
        y[k] = RK4step(f, y[k-1], tspan[k]-tspan[k-1])
    return y

# Run

sol_RK4 = RK4integrate(orbits, y0, tpoints)  # solve RK4
X, Y = sol_RK4.T    # transpose matrix

r1X, r2X, r3X, v1X, v2X, v3X = np.asarray(X)
r1Y, r2Y, r3Y, v1Y, v2Y, v3Y = np.asarray(Y)
```



```
# Analysis

v1 = np.hypot(v1X, v1Y)
v2 = np.hypot(v2X, v2Y)
v3 = np.hypot(v3X, v3Y)
r1 = np.hypot(r1X, r1Y)
r2 = np.hypot(r2X, r2Y)
r3 = np.hypot(r3X, r3Y)
r12 = np.linalg.norm(r2-r1)
r13 = np.linalg.norm(r3-r1)
r23 = np.linalg.norm(r3-r2)
dis1 = r1-1429.258e9    # displacment
dis2 = r2-777.258e9
dis3 = r3+742e6
r_com = (m1*r1+m2*r2+m3*r3) / (m1+m2+m3)
v_com = (m1*v1+m2*v2+m3*v3) / (m1+m2+m3)

# Calculate aphelion and perihelion
#print(max(r2X), min(r2X))
#print(max(r1X), min(r1X))

# FIG 10
plt.scatter(r1X, r1Y, c=v1, s=1, cmap='turbo')
plt.scatter(r2X, r2Y, c=v2, s=1, cmap='turbo')
plt.scatter(r3X, r3Y, c=v2, s=1, cmap='turbo')
plt.colorbar(label='$v$ [m/s]', orientation='vertical')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.title('RK4: 3-body problem')
#plt.show()

# FIG 11 and 12
plt.plot(tpoints, dis1)
plt.plot(tpoints, dis2)
plt.plot(tpoints, dis3)
plt.scatter(tpoints, r_com, s=0.05, c= "black")
plt.xlabel('Time [s]')
plt.ylabel('Displacment [m]')
plt.title('RK4: 3-body problem')
plt.legend(['Saturn', 'Jupiter', 'Sun', 'CoM'])
#plt.show()
```