School of Physics and Astronomy

Mphys project
Log book

Sahl Rowther

September 28^{th} 2017

Abstract

Supervisor: Prof. Ken Rice 22 Weeks

Contents

1 Week 1: September 11^{th}–17^{th} 2017

1.1 Storing planet data

Aim: To write a class that can store various properties of a planet. The class can take in the following properties:

- *Name* - the planets name.
- *mass* - the mass of the planet ($M_\oplus$).
- a - the orbital radius (AU).
- n - the orbital frequency ($^\circ\, yr^{-1}$).
- e - the eccentricity.
- i - the inclination (degrees).
- Ω - the longitude of ascending node (degrees).
- ϖ - the longitude of pericentre (degrees).

Code:

```
class planet():
    def __init__(self, Name="", Period=None, e=None, a=None, i=None, Omega=None, omega_bar=None, Mass=None, n=None):
        self.name = Name
        self.period = Period
        self.e = e
        self.a = a
        self.i = i
        self.omega = Omega
        self.omega_bar = omega_bar
        self.mass = Mass
        self.n = n
        self.units = {'a' : 'AU', 'mass' : 'M_EARTH', 'period' : 'days', 'i' : 'degrees', 'omega' : 'degrees', 'omega_bar' : 'degrees', 'n' : 'degrees yr^(-1)'}

    def toString(self):
        unit_keys = list(self.units.keys())
        for attr in self.__dict__:
            if attr is not 'units':
                if self.__dict__[attr] is not None:
                    if attr in unit_keys:
                        print('{} : {} {}'.format(attr, self.__dict__[attr], self.units[attr]))
                    else:
                        print('{} : {}'.format(attr, self.__dict__[attr]))
        print()
```

Code 1: Planet object

Test code:

```
import pandas as pd

planets = pd.read_csv('solar_system.csv')
planet_b = planet(**planets.ix[2])
planet_b.toString()
```

Code 2: Test of planet object

Output:

```
name : Earth
e : 0.01671022
a : 1.00000011 AU
i : 5e-05 degrees
omega : 348.73936000000003 degrees
omega_bar : 102.94719 degrees
mass : 1.000167431 M_EARTH
n : 359.7480668 degrees yr^(-1)
```

Vedict: Test successful

1.2 Storing star system data

Aim: Create a class that stores the mass and radius of the central body. And also stores all the planets as a list. The class takes the following arguments:

- starMass - the mass of the star.
- starRadius - the radius of the star.
- planet_data_file - a file containing a list of planets with properties described in Section 1.1.

Code:

```
from planet import planet

class starSystem():

    def __init__(self, starMass, starRadius, planet_data_file):
        self.star_mass = starMass
        self.star_radius = starRadius
        self.planets = self.addPlanets(planet_data_file)

    def addPlanets(self, planet_data_file):
        planets = pd.read_csv(planet_data_file)
        planet_list = []
        for p in range(len(planets)):
            planet_list.append(planet(**planets.ix[p]))
```

```

        return planet_list

    def print_planets(self):
        print('Star mass =', self.star_mass, 'Msun')
        print('Star radius = ', self.star_radius, 'Rsun\n')
        for p in self.planets:
            p.toString()
```

Code 3: Star system object

Data: For testing, data from the HD3167 system were used.

Table 1: HD3167 planet data. Period is in days, a is in AU, Mass is in $M_{\oplus}$, i and Ω are in degrees.

Name	Period	a	Mass	i	e	Ω
b	0.959641	0.01815	5.02	0	0	0
c	29.8454	0.1795	9.8	0	0.267	0
d	8.509	0.07757	6.9	20	0.36	0

The mass and radius of the star is $0.86\,M_{\odot}$ and $0.86\,R_{\odot}$.

Test code:

```
import pandas as pd

star_system = starSystem(0.86, 0.86, 'Planets.csv')
star_system.print_planets()
```

Code 4: Test of star system object

Output:

```
Star mass = 0.86 Msun
Star radius =  0.86 Rsun

name : b
period : 0.959641 days
e : 0.0
a : 0.01815 AU
i : 0 degrees
omega : 0 degrees
mass : 5.02 M_EARTH

name : c
period : 29.8454 days
```

```
e : 0.267
a : 0.1795 AU
i : 0 degrees
omega : 0 degrees
mass : 9.8 M_EARTH

name : d
period : 8.509 days
e : 0.36
a : 0.07757 AU
i : 20 degrees
omega : 0 degrees
mass : 6.9 M_EARTH
```

Verdict: Test successful. All planetary data and star data stored successful.

1.3 Replicating inclination output

```python
def get_property_all_planets(self, property_name, data_type="float"):
    property_list = np.zeros(len(self.planets), dtype=data_type)
    for idx, p in enumerate(self.planets):
        property_list[idx] = p.__dict__[property_name]

    return property_list
```

Code 5: Helper function to get a property value of all planets

Using Laplace-Lagrange secular theory, the equations of motion for the complex inclination vector, $z = i\exp(\imath\Omega)$, where i is the inclination and Ω is the ascending node, can be simplified to a linear eigenvalue problem:

$$\frac{dz_j}{dt} = i\sum_{k=1}^{N-1} B_{jk} z_k. \tag{1}$$

The frequency matrix $\mathbf{B}$ is only dependent on the mass and semi-major axis ratios of the planets, and is given by

$$B_{jj} = -\frac{n_j}{4}\sum_{k=0,\,k\neq j}^{N-1} \frac{m_k}{M_\star}\alpha_{jk}\bar{\alpha}_{jk} b_{3/2}^{(1)}(\alpha_{jk}), \tag{2a}$$

$$B_{jk} = -\frac{n_j}{4}\frac{m_k}{M_\star}\alpha_{jk}\bar{\alpha}_{jk} b_{3/2}^{(1)}(\alpha_{jk}). \tag{2b}$$

Where $n = \sqrt{GM_\star/a^3}$ is the mean orbital frequency, α_{jk} is the semi-major axis ratio given by

$$\alpha_{jk} = \begin{cases} a_j/a_k; & \text{if } a_j < a_k \\ a_k/a_j; & \text{if } a_k < a_j \end{cases} \tag{3}$$

and $b^{(1)}_{3/2}(\alpha)$ is the Laplace coefficient given by

$$b^{(1)}_{3/2}(\alpha) = \frac{1}{\pi}\int_0^{2\pi}\left[\frac{\cos\psi}{(1+\alpha^2-2\alpha\cos\,\psi)^{3/2}}\right]d\psi \tag{4}$$

```python
import numpy as np
from scipy import integrate

M_SUN = 1.9885*10**30
R_SUN = 6.9551*10**8
M_EARTH = 5.9726*10**24
AU = 149597870700

    def laplace_coefficient(self, alpha):
        integral_func = lambda psi, alpha: np.cos(psi)/(1+alpha**2-(2*alpha*np.cos(psi)))**(3./2.)
        return 1/np.pi*integrate.quad(integral_func, 0, 2*np.pi, args=(alpha,))[0]

    def matrix_B_eigenmodes(self):
        G_const = 6.6738*10**(-11)
        a = AU*self.get_property_all_planets('a')
        M_star_kg = M_SUN*self.star_mass
        n = np.sqrt(G_const*M_star_kg/a**3)

        m = M_EARTH*self.get_property_all_planets('mass')

        n_planets = len(self.planets)
        B = np.zeros([n_planets, n_planets])

        for j in range(n_planets):
            for k in range(n_planets):
                if j != k:
                    alpha_jk = a[j]/a[k]
                    if alpha_jk > 1:
                        alpha_jk = alpha_jk**(-1)
                    laplace_coeff = self.laplace_coefficient(alpha_jk)
                    alpha_jk_bar = np.where(a[k] < a[j], 1, alpha_jk)
                    B[j, k] = (n[j]/4)*(m[k]/M_star_kg)*alpha_jk*alpha_jk_bar*laplace_coeff
                else:
                    for kk in range(n_planets):
                        if kk != j:
                            alpha_jj = a[j]/a[kk]
                            if alpha_jj > 1:
                                alpha_jj = alpha_jj**(-1)
                            laplace_coeff = self.laplace_coefficient(alpha_jj)
                            alpha_jj_bar = np.where(a[kk] < a[j], 1, alpha_jj)
                            B[j, k] += (m[kk]/M_star_kg)*alpha_jj*alpha_jj_bar*laplace_coeff
                    B[j, k] *= -(n[j]/4)
        eigenvalues, eigenvectors = np.linalg.eig(B)
        return B, eigenvalues, eigenvectors
```

Code 6: Calculate the frequency matrix, **B**

2 Week 2: September 18^{th}–24^{th} 2017

3 Week 3: September 25^{th}–31^{st} 2017

3.1 Simulation of solar system

Storing the data

The planetary data for simulating the Solar System is given below.

Table 2: Solar System data. The mass in terms of $M_\oplus$ is given by m. The mean orbital frequency in degrees per year is given by n. The value of the semi-major axis in AU is given by a. The eccentricity of the orbit is given by e. The inclination of the orbit in degrees is given by i. The longitudes of pericentre and ascending node are given in degrees by ϖ and Ω respectively.

Name	m	n	a	e	i	ϖ	Ω
Mercury	0.055	1493.708	0.387	0.206	7.005	77.456	48.332
Venus	0.815	584.779	0.723	0.007	3.395	131.533	76.681
Earth	1.000	359.748	1.000	0.017	0.000	102.947	348.739
Mars	0.107	191.278	1.524	0.093	1.851	336.041	49.579
Jupiter	317.885	30.309	5.203	0.048	1.305	14.754	100.556
Saturn	95.178	12.215	9.537	0.054	2.484	92.432	113.715
Uranus	14.538	4.279	19.191	0.047	0.770	170.964	74.230
Neptune	17.150	2.182	30.069	0.009	1.769	44.971	131.722
Pluto	0.002	1.450	39.482	0.249	17.142	224.067	110.303

```
import numpy as np
import numpy.ma as ma
from scipy import integrate
import scipy.linalg
from scipy.optimize import fsolve
from sympy import symbols, Matrix, linsolve, diag
import matplotlib.pyplot as plt
from planet import planet

class solar_System():

    def __init__(self, starMass, starRadius, planet_data_file):
        self.star_mass = starMass
        self.star_radius = starRadius
        self.planets = self.addPlanets(planet_data_file)

    def addPlanets(self, planet_data_file):
        planets = pd.read_csv(planet_data_file)
        planet_list = []
        for p in range(len(planets)):
            planet_list.append(planet(**planets.ix[p]))
        return planet_list

```

```python
    def get_property_all_planets(self, property_name, data_type="float"):
        property_list = np.zeros(len(self.planets), dtype=data_type)
        for idx, p in enumerate(self.planets):
            property_list[idx] = p.__dict__[property_name]
        return property_list
```

Code 7: Object for storing the data

Solving the equations of motion

The expression for the disturbing function, $\mathcal{R}_j$ is given by:

$$\begin{aligned}\mathcal{R}_j = n_j a_j^2 \Bigg[& \frac{1}{2} A_{jj} \left(h_j^2 + k_j^2\right) + \frac{1}{2} B_{jj} \left(p_j^2 + q_j^2\right) \\ & + \sum_{i \neq j} A_{ji} \left(h_j h_i + k_j k_i\right) + \sum_{i \neq j} B_{ji} \left(p_j p_i + q_j q_i\right) \Bigg]\end{aligned} \tag{5}$$

Where n_j is the mean orbital frequency, a_j is the semi-major axis, and $\mathbf{A}$ and $\mathbf{B}$ are the frequency matrices defined as:

$$\begin{aligned}A_{jj} = n_j \Bigg[& \frac{3}{2} J_2 \left(\frac{R_\star}{a_j}\right)^2 - \frac{9}{8} J_2^2 \left(\frac{R_\star}{a_j}\right)^4 - \frac{15}{4} J_4^2 \left(\frac{R_\star}{a_j}\right)^4 \\ & + \frac{1}{4} \sum_{k \neq i} \frac{m_k}{m_\star + m_j} \alpha_{jk} \bar{\alpha}_{jk} b_{3/2}^{(1)}(\alpha_{jk}) \Bigg]\end{aligned} \tag{6a}$$

$$A_{jk} = -\frac{n_j}{4} \frac{m_k}{m_\star + m_j} \alpha_{jk} \bar{\alpha}_{jk} b_{3/2}^{(2)}(\alpha_{jk}) \qquad (j \neq k) \tag{6b}$$

$$\begin{aligned}B_{jj} = n_j \Bigg[& \frac{3}{2} J_2 \left(\frac{R_\star}{a_j}\right)^2 - \frac{27}{8} J_2^2 \left(\frac{R_\star}{a_j}\right)^4 - \frac{15}{4} J_4^2 \left(\frac{R_\star}{a_j}\right)^4 \\ & + \frac{1}{4} \sum_{k \neq i} \frac{m_k}{m_\star + m_j} \alpha_{jk} \bar{\alpha}_{jk} b_{3/2}^{(1)}(\alpha_{jk}) \Bigg]\end{aligned} \tag{6c}$$

$$B_{jk} = -\frac{n_j}{4} \frac{m_k}{m_\star + m_j} \alpha_{jk} \bar{\alpha}_{jk} b_{3/2}^{(1)}(\alpha_{jk}) \qquad (j \neq k). \tag{6d}$$

Where m is the mass, $\alpha < 1$ is the semi-major axis ratio, $\bar{\alpha} = 1$ if $a_k < a_j$, $\bar{\alpha} = \alpha$ if $a_j < a_k$, J_2 and J_4 are the first two zonal gravity coefficients, and the laplace coefficients are defined by:

$$b_s^{(j)}(\alpha) = \frac{1}{\pi} \int_0^{2\pi} \left[\frac{cos(j\psi)}{(1 + \alpha^2 - 2\alpha \cos\,\psi)^s} \right] d\psi. \tag{7}$$

Where s is a positive half integer, and j is an integer.

```
def calculate_laplace_coeff(alpha, j, s):
    return integrate.quad(lambda psi, alpha, j, s: np.cos(j*psi)/(1-2*alpha*np.cos(psi)+alpha**2)**s,
                         0, 2*np.pi, args=(alpha, j, s,))[0]/np.pi
```

Code 8: Calculating the laplace coefficient

And the equations of motion are given by:

$$h_j = e_j \cos \varpi_j \tag{8a}$$
$$k_j = e_j \sin \varpi_j \tag{8b}$$
$$p_j = i_j \cos \Omega j \tag{8c}$$
$$q_j = i_j \sin \Omega j \tag{8d}$$

Where e_j is the eccentricity, i_j is the inclination, and ϖ_j and Ω_j are the longitude of pericentre and ascending node respectively.

Code:

```
    def frequency_matrix(self, matrix_id, J2=0, J4=0):
        M_star_kg = M_SUN*self.star_mass
        R = R_SUN*self.star_radius
        m = M_EARTH*self.get_property_all_planets('mass')
        n = self.get_property_all_planets('n')
        a = AU*self.get_property_all_planets('a')
        n_planets = len(self.planets)
        f_mat = np.zeros([n_planets, n_planets])

        if matrix_id == 'A':
            j_laplace_coeff_jk, j_laplace_coeff_jj = 2, 1
            front_factor = -1
            J2_correction = (((3/2)*J2*(R/a)**2)-((9/8)*(J2**2)*(R/a)**4)-((15/4)*J4*(R/a)**4))

        if matrix_id == 'B':
            j_laplace_coeff_jk = j_laplace_coeff_jj = 1
            front_factor = 1
            J2_correction = (((3/2)*J2*(R/a)**2)-((27/8)*(J2**2)*(R/a)**4)-((15/4)*J4*(R/a)**4))

        for j in range(n_planets):
            for k in range(n_planets):
                if j != k:
                    alpha_jk = a[j]/a[k]
                    if alpha_jk > 1:
                        alpha_jk = alpha_jk**(-1)
                    laplace_coeff = calculate_laplace_coeff(alpha_jk, j_laplace_coeff_jk, 3/2)
                    alpha_jk_bar = np.where(a[k] < a[j], 1, alpha_jk)
                    f_mat[j, k] = front_factor*(n[j]/4)*(m[k]/(M_star_kg+m[j]))*alpha_jk*alpha_jk_bar*laplace_coeff

                else:
                    for kk in range(n_planets):
                        if kk != j:
```

```python
                    alpha_jj = a[j]/a[kk]
                    if alpha_jj > 1:
                        alpha_jj = alpha_jj**(-1)
                    laplace_coeff = calculate_laplace_coeff(alpha_jj, j_laplace_coeff_jj, 3/2)
                    alpha_jj_bar = np.where(a[kk] < a[j], 1, alpha_jj)
                    f_mat[j, k] += (1/4)*(m[kk]/(M_star_kg+m[j]))*alpha_jj*alpha_jj_bar*laplace_coeff
            f_mat[j, k] += J2_correction[j]
            f_mat[j, k] *= -front_factor*(n[j])
    return f_mat
```

Code 9: Calculating **A** and **B**

Using **A** and **B**, the equations of motion in equations 8a to 8d can be reduced to two sets of eigenvalue problems, whose solutions are given by:

$$h_j = \sum_{i=0}^{N-1} e_{ji} \sin(g_i t + \beta_i), \qquad k_j = \sum_{i=0}^{N-1} e_{ji} \cos(g_i t + \beta_i) \tag{9a}$$

and

$$p_j = \sum_{i=0}^{N-1} I_{ji} \sin(f_i t + \gamma_i), \qquad q_j = \sum_{i=0}^{N-1} I_{ji} \cos(f_i t + \gamma_i). \tag{9b}$$

Where e_{ji} and I_{ji} are the scaled components of the eigenvectors of **A** and **B**. The frequencies g_i and f_i are the eigenvalues of **A** and **B**. The scaled eigenvectors can be expressed as:

$$S_i \bar{e}_{ji} = e_{ji} \qquad \text{and} \qquad T_i \bar{I}_{ji} = I_{ji}. \tag{10}$$

Where $\bar{e}_{ji}$ and $\bar{I}_{ji}$ are the normalised eigenvectors of **A** and **B**. The phases β_i and γ_i, as well as the scaling factors of the eigenvectors S_i and T_i are determined by the initial conditions.

Using the data in Table 2 and equations 8a to 8d, the initial conditions can be calculated.

```python
def initial_conditions(self):
    e = self.get_property_all_planets('e')
    omega_bar = self.get_property_all_planets('omega_bar')*np.pi/180
    i = self.get_property_all_planets('i')*np.pi/180
    omega = self.get_property_all_planets('omega')*np.pi/180

    h = e*np.sin(omega_bar)
    k = e*np.cos(omega_bar)
    p = i*np.sin(omega)
    q = i*np.cos(omega)

    return h, k, p, q
```

Code 10: Calculating initial conditions

Using the calculated values of $\bar{e}_{ji}$ and by evaluating h_j in equation 9a at $t = 0$ and equating it to h_j from equation 8a, an augmented matrix can be created to solve for $S_i \sin\beta_i$, as shown below.

$$\left[\begin{array}{cccc|c} S_0\sin(\beta_0)\,\bar{e}_{00} & S_1\sin(\beta_1)\,\bar{e}_{01} & \cdots & S_{N-1}\sin(\beta_{N-1})\,\bar{e}_{0,N-1} & h_0 \\ S_1\sin(\beta_0)\,\bar{e}_{10} & S_1\sin(\beta_1)\,\bar{e}_{11} & \cdots & S_{N-1}\sin(\beta_{N-1})\,\bar{e}_{1,N-1} & h_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{N-1}\sin(\beta_0)\,\bar{e}_{N-1,0} & S_{N-1}\sin(\beta_1)\,\bar{e}_{N-1,1} & \cdots & S_{N-1}\sin(\beta_{N-1})\,\bar{e}_{N-1,N-1} & h_{N-1} \end{array}\right] \tag{11}$$

A similar process can be done with k_j to solve for $S_i \cos\beta_i$:

$$\left[\begin{array}{cccc|c} S_0\cos(\beta_0)\,\bar{e}_{00} & S_1\cos(\beta_1)\,\bar{e}_{01} & \cdots & S_{N-1}\cos(\beta_{N-1})\,\bar{e}_{0,N-1} & h_0 \\ S_1\cos(\beta_0)\,\bar{e}_{10} & S_1\cos(\beta_1)\,\bar{e}_{11} & \cdots & S_{N-1}\cos(\beta_{N-1})\,\bar{e}_{1,N-1} & h_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{N-1}\cos(\beta_0)\,\bar{e}_{N-1,0} & S_{N-1}\cos(\beta_1)\,\bar{e}_{N-1,1} & \cdots & S_{N-1}\cos(\beta_{N-1})\,\bar{e}_{N-1,N-1} & h_{N-1} \end{array}\right] \tag{12}$$

Solving the above two matrices gives one set of equations in terms of $S_i \sin\beta_i$ and another set of equations in terms of $S_i \cos\beta_i$. Solving them simultaneously results in values for S_i and β_i. A similar process can be done to solve for T_i and γ_i.

```
def scaling_factor_and_phase(p, *boundaries):
    s, phase = p
    return (s*np.sin(phase)-boundaries[0], s*np.cos(phase)-boundaries[1])
```

Code 11: Equations for simultaneously solving for the scale factor and phase in Code 12

```
    def solve_property(self, eigenvectors, initial_conditions):
        n = len(self.planets)
        aug = Matrix(np.zeros([n, n+1]))
        aug[:, :n] = eigenvectors
        aug[:, n] = initial_conditions

        result = linsolve(aug, *symbols('x0:'+str(n)))
        answers = np.zeros(n)
        for ans in result:
            for a, answer in enumerate(ans):
                answers[a] = answer
        return answers

    def find_all_scaling_factor_and_phase(self, eigenvectors_of_A, eigenvectors_of_B):
        x, y = eigenvectors_of_A, eigenvectors_of_B

        init_conditions = np.array(star_system.initial_conditions())
        h_solved = self.solve_property(x, init_conditions[0, :])
        k_solved = self.solve_property(x, init_conditions[1, :])
        p_solved = self.solve_property(y, init_conditions[2, :])
        q_solved = self.solve_property(y, init_conditions[3, :])

        n = len(self.planets)
```

```
        S, beta = np.zeros(n), np.zeros(n)
        T, gamma = np.zeros(n), np.zeros(n)

        for i in range(n):
            S[i], beta[i] = fsolve(scaling_factor_and_phase, (1, -1), args=(h_solved[i], k_solved[i],))
            T[i], gamma[i] = fsolve(scaling_factor_and_phase, (-1, 1), args=(p_solved[i], q_solved[i],))
        return S, beta, T, gamma
```

Code 12: Calculating the scale factors and phases

Once the scale factors and phases have been found, equations 9a and 9b can now be solved at any time t.

```
    def eq_of_motion(self, scaled_eigenvector, eigenvalue, phase, t, eq_id):
        # eq_id = 'h', 'k', 'p', 'q'
        kwargs = {'scaled_eigenvector' : scaled_eigenvector, 'eigenvalue' : eigenvalue, 'phase' : phase, 't' : t}
        if eq_id == 'h' or eq_id == 'p':
            return self.get_h_or_p(**kwargs)
        if eq_id == 'k' or eq_id == 'q':
            return self.get_k_or_q(**kwargs)

    def get_h_or_p(self, scaled_eigenvector, eigenvalue, phase, t):
        n = len(self.planets)
        h_list = []
        for j in range(n):
            h = np.zeros_like(t)
            for i in range(n):
                h += scaled_eigenvector[j, i]*np.sin((eigenvalue[i]*t+phase[i])*np.pi/180)
            h_list.append(h)
        return np.array(h_list)

    def get_k_or_q(self, scaled_eigenvector, eigenvalue, phase, t):
        n = len(self.planets)
        k_list = []
        for j in range(n):
            k = np.zeros_like(t)
            for i in range(n):
                k += scaled_eigenvector[j, i]*np.cos((eigenvalue[i]*t+phase[i])*np.pi/180)
            k_list.append(k)
        return np.array(k_list)
```

Code 13: Calculating the equations of motion

Finally, the eccentricity and inclination at any time t can be calculated using:

$$e_j(t) = \left(h_j^2 + k_j^2\right)^{1/2} \tag{13a}$$

$$i_j(t) = \left(p_j^2 + q_j^2\right)^{1/2} \tag{13b}$$

```
    def get_eccentricity(self, scaled_eigenvector_of_A, eigenvalue_of_A, beta, t):
        n = len(self.planets)
```

```
        kwargs = {'scaled_eigenvector' : scaled_eigenvector_of_A, 'eigenvalue' : eigenvalue_of_A, 'phase' : beta, 't' : t}
        eccentricities = []
        h, k = self.eq_of_motion(**kwargs, eq_id='h'), self.eq_of_motion(**kwargs, eq_id='k')
        for j in range(n):
            eccentricities.append(np.sqrt(h[j]**2+k[j]**2))
        return np.array(eccentricities)

    def get_inclination(self, scaled_eigenvector_of_B, eigenvalue_of_B, gamma, t):
        n = len(self.planets)
        kwargs = {'scaled_eigenvector' : scaled_eigenvector_of_B, 'eigenvalue' : eigenvalue_of_B, 'phase' : gamma, 't' : t}
        inclinations = []
        p, q = self.eq_of_motion(**kwargs, eq_id='p'), self.eq_of_motion(**kwargs, eq_id='q')
        for j in range(n):
            inclinations.append(np.sqrt(p[j]**2+q[j]**2))
        return np.array(inclinations)
```

Code 14: Calculating the eccentricity and inclination

The perihelion precession rate, $\dot{\varpi}$ can be found as follows. First equations 8a and 8b can be rearranged for ϖ as,

$$\tan\varpi = \frac{h_j}{k_j}. \tag{14}$$

Differentiating, using the chain rule, with respect to time gives,

$$\begin{aligned} \frac{1}{\cos^2\varpi}\frac{d\varpi}{dt} &= \frac{\frac{dh_j}{dt}k_j - \frac{dk_j}{dt}h_j}{k_j^2} \\ \frac{k_j^2}{\cos^2\varpi}\dot{\varpi} &= \dot{h}_j k_j - \dot{k}_j h_j \\ \dot{\varpi} &= \frac{\dot{h}_j k_j - \dot{k}_j h_j}{e_j^2}. \end{aligned} \tag{15}$$

Where in the last step, the substitution $e_j = h_j/\cos\varpi$ (from equation 8b) was used. The time derivatives of h_j and k_j can be found using the disturbing function:

$$\dot{h}_j = \frac{1}{n_j a_j^2}\frac{\partial\mathcal{R}_j}{\partial k_j}, \qquad \dot{k}_j = -\frac{1}{n_j a_j^2}\frac{\partial\mathcal{R}_j}{\partial h_j}. \tag{16}$$

Which then become:

$$\dot{h}_j = \sum_{i=0}^{N-1} A_{ji}k_i, \qquad \dot{k}_j = -\sum_{i=0}^{N-1} A_{ji}h_i. \tag{17}$$

Where the components of A_{ji} are described in equations 6a and 6b.

```
    def get_perihelion_precession_rates(self, A, eccentricities, h_list, k_list):
        n = len(self.planets)
```

```
        d_pidot_dt_list = []
        masks = []

        for j in range(n):
            h_dot_j, k_dot_j = 0, 0
            for i in range(n):
                h_dot_j += A[j, i]*k_list[i]
                k_dot_j -= A[j, i]*h_list[i]
            pidot_j = 3600*(k_list[j]*h_dot_j - h_list[j]*k_dot_j)/(eccentricities[j])**2
            d_pidot_dt_list.append(pidot_j)
        return d_pidot_dt_list
```

Code 15: Calculating precession rate, $\dot{\varpi}$ of Mercury

3.2 Tests of simulation

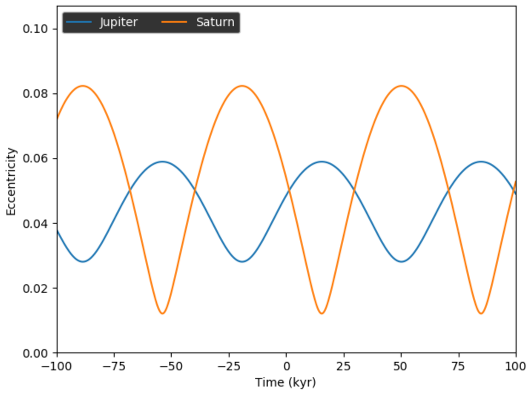
The following code was used to test the simulation.

```
    def simulate(self, t, plot=False, separate=True):
        A, B = [star_system.frequency_matrix(matrix_id=mat_id, J2=-6.84*10**(-7), J4=2.8*10**(-12)) for mat_id in ['A', 'B']]
        g, x, f, y = *np.linalg.eig(A), *np.linalg.eig(B)
        S, beta, T, gamma = self.find_all_scaling_factor_and_phase(x, y)

        eccentricities = self.get_eccentricity(S*x, g, beta, t)
        inclinations = self.get_inclination(T*y, f, gamma, t)*180/np.pi
        names = [self.planets[p].name for p in range(len(self.planets))]
        if plot:
            if separate:
                plot_simulation_separate(t/10**6, eccentricities, 'Time (Myr)', 'Eccentricity', names)
                plot_simulation_separate(t/10**6, inclinations, 'Time (Myr)', 'Inclination', names)
            else:
                plot_simulation_all(t/10**6, eccentricities, 'Time (Myr)', 'Eccentricity', names)
                plot_simulation_all(t/10**6, inclinations, 'Time (Myr)', 'Inclination', names)

        kwargs = {'scaled_eigenvector' : S*x, 'eigenvalue' : g, 'phase' : beta,
                  't' : t}
        h_list = self.eq_of_motion(**kwargs, eq_id='h')
        k_list = self.eq_of_motion(**kwargs, eq_id='k')
        kwargs = {'scaled_eigenvector' : S*x, 'eigenvalue' : f, 'phase' : gamma,
                    't' : t}
        p_list = self.eq_of_motion(**kwargs, eq_id='p')
        q_list = self.eq_of_motion(**kwargs, eq_id='q')

        precession_rates = self.get_perihelion_precession_rates(A, eccentricities, h_list, k_list)

        idx = 0
        plot_precession_rate(t, precession_rates[idx], 'Mercury')
        plot_eccentricity(t, eccentricities[idx], 'Mercury')
```

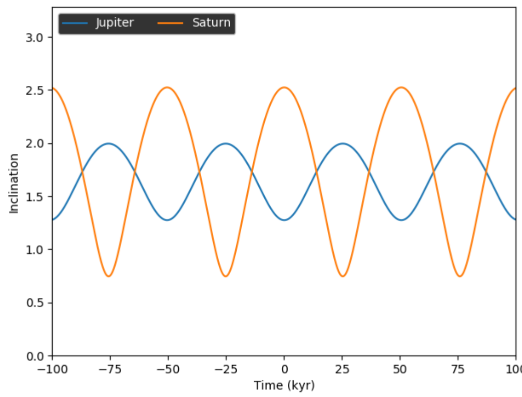
Code 16: Test code for simulation

3.2.1 Jupiter and Saturn

The first test is to replicate the eccentricity and inclination outputs in Figure 7.1 of Murray & Dermott (1999) [1].



(a) Evolution of eccentricity of Jupiter and Saturn.



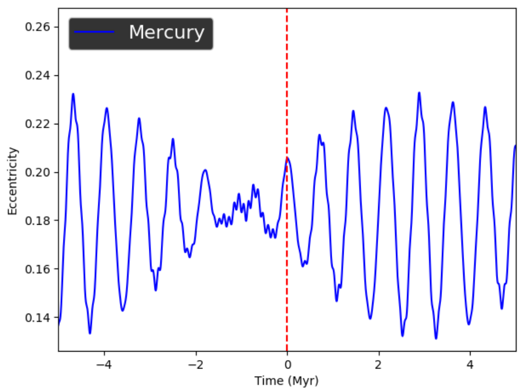
(b) Evolution of inclination of Jupiter and Saturn.

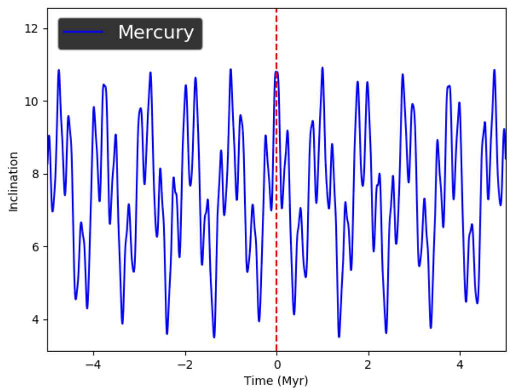
Figure 1

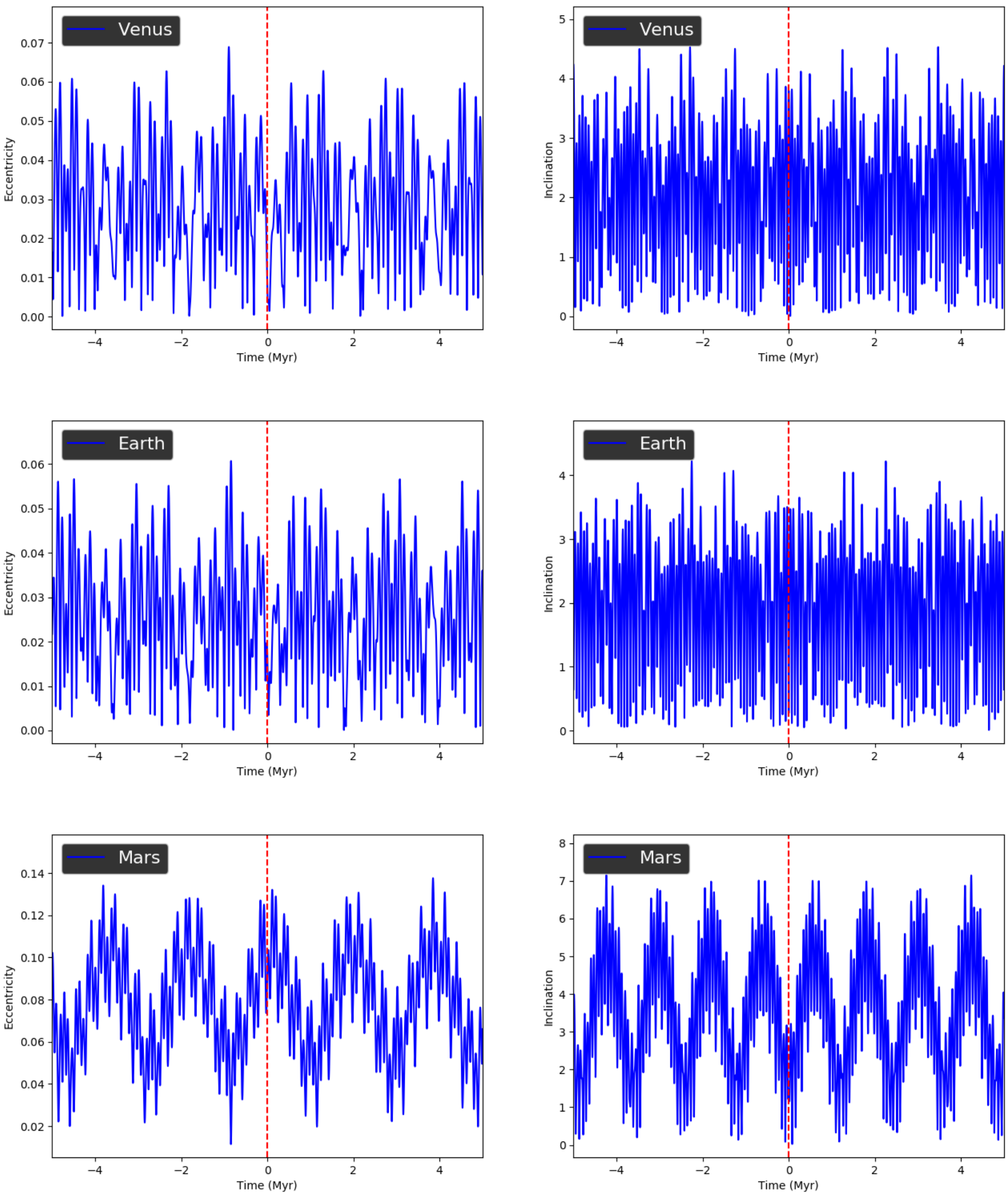
Verdict: Figure 1 is a very good match to that of the output in Murray & Dermott (1999) [1].

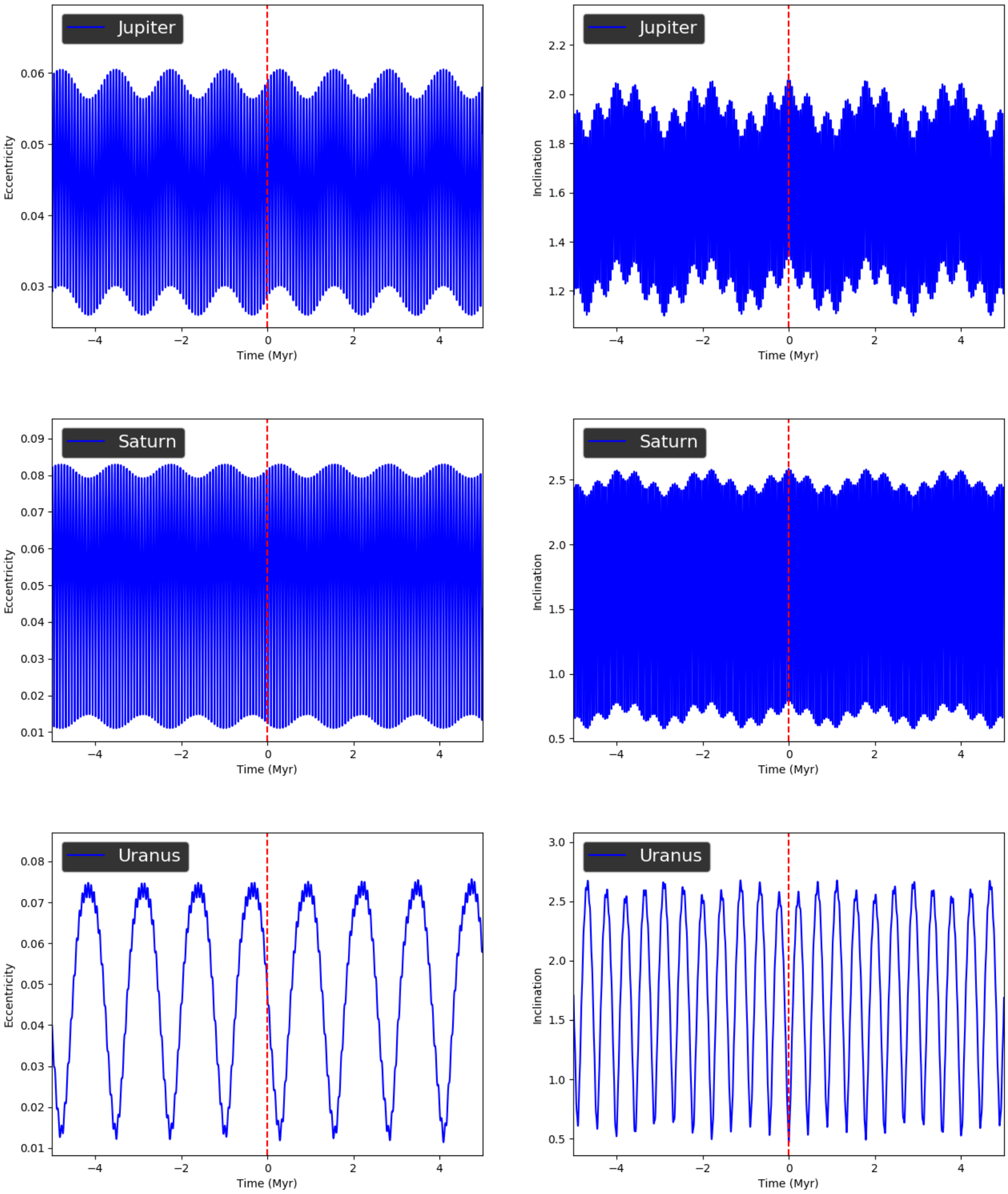
3.2.2 Whole Solar System

The plots for the eccentricity and inclination of each planet are as shown:






Venus
Eccentricity
Time (Myr)
Venus
Inclination
Time (Myr)
Earth
Eccentricity
Time (Myr)
Earth
Inclination
Time (Myr)
Mars
Eccentricity
Time (Myr)
Mars
Inclination
Time (Myr)


Jupiter
Eccentricity
Time (Myr)
Jupiter
Inclination
Time (Myr)
Saturn
Eccentricity
Time (Myr)
Saturn
Inclination
Time (Myr)
Uranus
Eccentricity
Time (Myr)
Uranus
Inclination
Time (Myr)

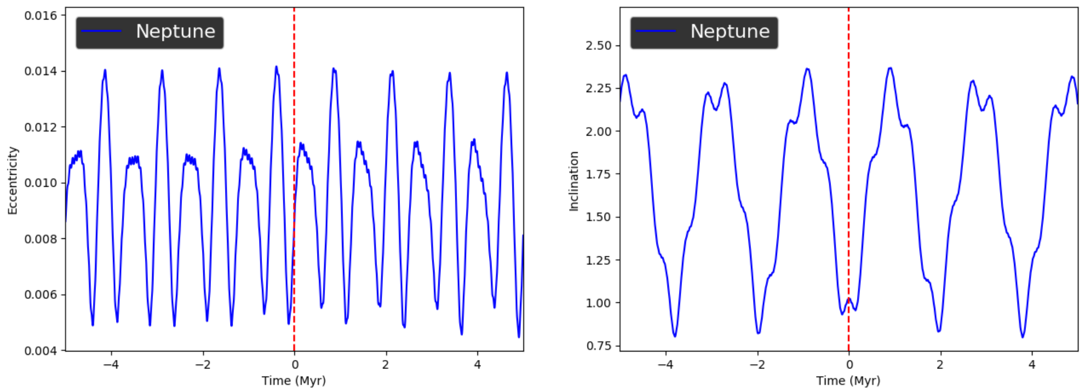


Figure 2: Eccentricity and inclination (in degrees) of each planet in the Solar System using J2000 data.

Verdict

By comparing these plots with existing results[1,2], it can be seen that the results are consistent with what is expected. It should be noted that the eccentricity plots do not match with Murray & Dermott as well as the inclinations. However when comparing the max and min eccentricity (especially Mercury) to the other data[2], the eccentricities do match well. Oddly, the inclinations in the other data[2] do not match as well; has lower I_{min} and higher I_{max}.

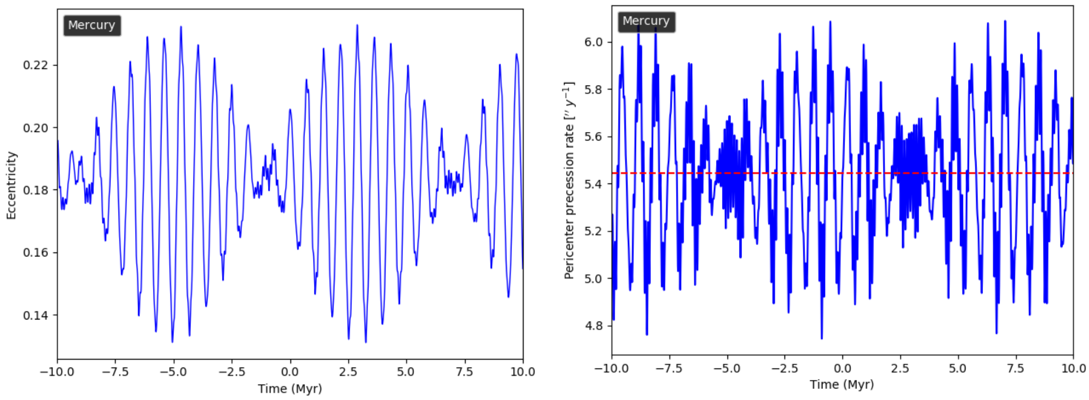
3.2.3 Precession of mercury

Another test that serves as a good indicator of the accuracy of the simulation is determining the precession of Mercury. Applying Laplace-Lagrange secular theory is expected to yield a precession rate, $\dot{\varpi}$ of $544''yr^{-1}$[2,3].

Verdict

From Figure 5b, it can be seen the mean precession, the red dashed line is equal to $544.86''$ per century; consistent with expectations and without the addition of General Relativity. The plot of the precession rate also matched with expected results[2].

The effect of the oblateness of the Sun on the precession rate of Mercury was also found to be $\sim 0.08''$ per century. The small magnitude of the change is as expected.



(a) Evolution of eccentricity of Mercury.

(b) Precession rate, $\dot{\varpi}$ of Mercury.

Figure 3

4 Week 4: October 1^{st}–8^{th} 2017

4.1 Keplerian to Cartesian coordinates

The next test to check accuracy is to convert the output of the equations of motion to cartesian coordinates in order to plot the Solar System for visual inspection. For this we use:

- h, k, p, q - the equations of motion.
- e - the eccentricity.
- i - the inclination (in radians).
- a - the semi-major axis (in AU).
- n - the orbital frequency (in radians per year)

First the mean anomaly, $M(t)$ is found using

$$M(t) = n(t - t_0). \tag{18}$$

Then the eccentric anomaly $E \equiv E(t)$ is calculated by solving

$$M(t) = E(t) - e \sin E(t) \tag{19}$$

using the Newton-Raphson method:

$$E_{j+1} = E_j - \frac{f(E_j)}{\frac{d}{dE_j} f(E_j)} = E_j - \frac{E_j - e \sin E_j - M}{1 - e \cos E_j}, \qquad E_0 = M \tag{20}$$

The above equation was iterated until $E_{j+1} = E_j$. Then the trua anomaly $\nu(t)$ was calculated using

$$\nu(t) = 2 \operatorname{arctan2}\left(\sqrt{1+e} \sin \frac{E(t)}{2}, \sqrt{1-e} \cos \frac{E(t)}{2}\right). \tag{21}$$

Where arctan2 is the two argument arctangent function. The distance r_c from the central body was then calculated using

$$r_c = a(1 - e \cos E(t)). \tag{22}$$

Then the position vector in the orbital frame was found using

$$\mathbf{o}(t) = \begin{pmatrix} o_x(t) \\ o_y(t) \\ o_z(t) \end{pmatrix} = r_c(t) \begin{pmatrix} \cos \nu(t) \\ \sin \nu(t) \\ 0 \end{pmatrix} \tag{23}$$

Then Ω, the longitude of the ascending node, and ω, the argument of the periapsis was found using

$$\Omega = \arctan2(p, q), \tag{24a}$$

$$\omega = \Omega - \arctan2(h, k). \tag{24b}$$

Finally the cartesian coordinates could be found using

$$\mathbf{r}(t) = \begin{pmatrix} o_x(t)(\cos(\omega)\cos(\Omega) - \sin(\omega)\cos(i)\sin(\Omega)) - o_y(t)(\sin(\omega)\cos(\Omega) + \cos(\omega)\cos(i)\sin(\Omega)) \\ o_x(t)(\cos(\omega)\sin(\Omega) + \sin(\omega)\cos(i)\cos(\Omega)) + o_y(t)(\cos(\omega)\cos(i)\cos(\Omega) - \sin(\omega)\sin(\Omega)) \\ o_x(t)(\sin(\omega)\sin(i)) + o_y(t)(\cos(\omega)\sin(i)) \end{pmatrix} \tag{25}$$

Code:

```python
    def get_pi_or_omega(self, hp, kq):
        pi_om = []
        for i in range(len(self.planets)):
            pi_om.append(np.arctan2(hp[i], kq[i]))
        return np.array(pi_om)

    def kep2cart(self, ecc, inc, h_list, k_list, p_list, q_list, time, t0, idx):
        O_list = self.get_pi_or_omega(p_list, q_list)
        w_list = O_list-self.get_pi_or_omega(h_list, k_list)
        n = self.get_property_all_planets('n')
        a = self.get_property_all_planets('a')

        Mt = n[idx]*np.pi/180*(time-t0)
        EA = []
        e, w, O, i = ecc[idx], w_list[idx], O_list[idx], inc[idx]
        for t in range(len(time)):
            E = Mt[t]
            f_by_dfdE = (E-e[t]*np.sin(E)-Mt[t])/(1-e[t]*np.cos(E))
            j, maxIter, delta = 0, 30, 0.0000000001
            while (j < maxIter)*(np.abs(f_by_dfdE) > delta):
                E = E-f_by_dfdE
                f_by_dfdE = (E-e[t]*np.sin(E)-Mt[t])/(1-e[t]*np.cos(E))
                j += 1
            EA.append(E)
        EA = np.array(EA)
        nu = 2*np.arctan2(np.sqrt(1+e)*np.sin(EA/2), np.sqrt(1-e)*np.cos(EA/2))

        rc = a[idx]*(1-e*np.cos(EA))
        o_vec = np.array([rc*np.cos(nu), rc*np.sin(nu), 0])

        rx = (o_vec[0]*(np.cos(w)*np.cos(O) - np.sin(w)*np.cos(i)*np.sin(O)) - o_vec[1]*(np.sin(w)*np.cos(O) + np.cos(w)*np.cos(i)*np.sin(O)))
        ry = (o_vec[0]*(np.cos(w)*np.sin(O) + np.sin(w)*np.cos(i)*np.cos(O)) + o_vec[1]*(np.cos(w)*np.cos(i)*np.cos(O) - np.sin(w)*np.sin(O)))
        rz = (o_vec[0]*(np.sin(w)*np.sin(i)) + o_vec[1]*(np.cos(w)*np.sin(i)))

        return rx, ry, rz
```

Code 17: Converting from Keplerian to Cartesian coordinates

4.2 Test of conversion

To test the conversion and the accuracy of the simulation, the Solar System was simulated for 1000 years and plotted below in Cartesian coordinates.

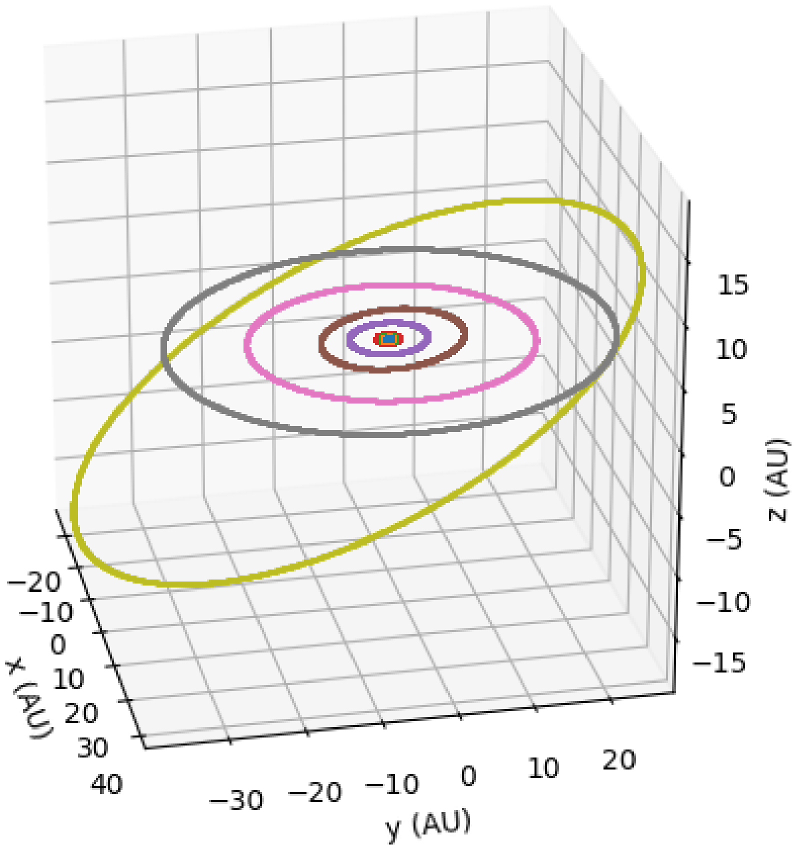


Figure 4: Plot of the Solar System simulated for a 1000 years from J2000 data.

Verdict: The plot is as expected. The plot was also used to ensure the time period of orbits were as expected, which they were.

4.3 General Relativity corrections

General Relativity (GR) corrections add an additional term to $\dot{\varpi}_j$. This additional term is given by [4]

$$\dot{\varpi}_j^{GR} = 3\frac{a_j^2 n_j^3}{c^2}. \tag{26}$$

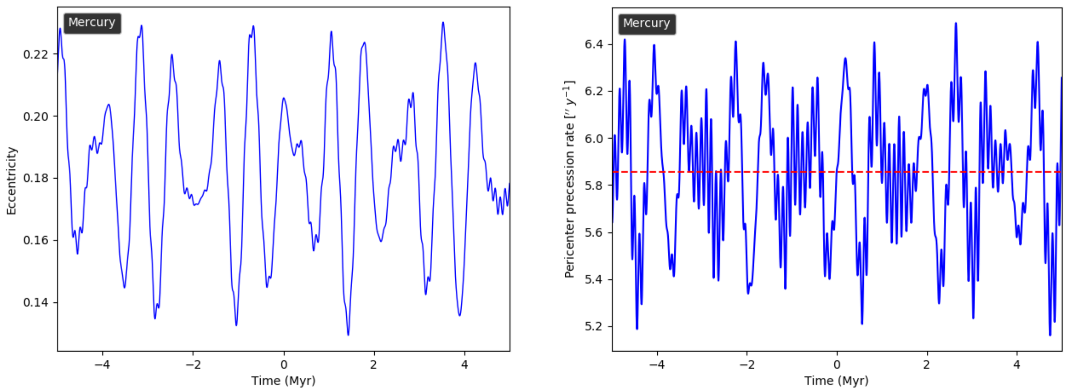
This term is added to the diagonal elements of $\mathbf{A}$ to accounts for the effects of GR.

4.4 Test of GR correction

To test the effects of GR on the precession rates $\dot{\varpi}$, each planet was simulated by itself and its precession rate was calculated. The effect of GR on $\dot{\varpi}$, in arc seconds per century, of each planet is shown below

```
Mercury : 42.8928
Venus : 8.6069
Earth : 3.8309
Mars : 1.3483
Jupiter : 0.0621
Saturn : 0.0137
Uranus : 0.0024
Neptune : 0.0008
Pluto : 0.0004
```

The inclusion of GR had the largest effect on Mercury, as expected. This can be seen in the difference of the eccentricity plots in Figure 5 and previously in Figure 3.



(a) Evolution of eccentricity of Mercury. (b) Precession rate, $\dot{\varpi}$ of Mercury.

Figure 5: Effect of GR on the orbit of Mercury.

Verdict: These values match very well to calculated values of GR's affect [5]. Additionally, including the effect of GR has also resulted in the eccentricity plot of Mercury becoming a closer match to Murray & Dermott (1999) [1].

5 Week 5: October 9^{th}–15^{th} 2017

5.1 Eccentricity damping corrections

We assume that the tides raised by planets are more dominant relative to tides raised by the star. Thus the eccentricity damping rate is given by [1,6]

$$\lambda = -\frac{\dot{e}}{e} = \frac{63}{4}\frac{1}{Q'_p}\frac{m_\star}{m_p}\left(\frac{R_p}{a_p}\right)^5 n_p. \tag{27}$$

Where R is the radius of the planet, $Q' \equiv 1.5Q/k_2$ is the modified tidal quality factor. Q is the tidal quality factor and k_2 is the Love number of degree 2 of the planet. All other terms are the same as before. The values of Q and k_2 are taken from Goldreich & Sotter (1966) [7], Zhang (1992) [8], and Gavrilov & Kharkov (1977) [9]. Similarly to the GR correction, $\sqrt{-1}\lambda$ is added to diagonal elements of $\mathbf{A}$ to account for the effect of tides.

5.2 Test of eccentricity damping

As before with GR, the effect of eccentricity damping is tested on the precession rate of the planets. It was found that this additional had no effect, at least on timescales of a few million years. However, for the Solar System, only a small change is expected, on timescales of billions of years.

5.3 Simulating other 2 planet systems

6 Bibliography

[1] C. D. Murray and S. F. Dermott, *Solar System Dynamics*. Feb. 2000.

[2] "Secular evolution of planetary orbits." http://farside.ph.utexas.edu/teaching/celestial/Celestialhtml/node91.html#t11.4. (Accessed on 10/04/2017).

[3] V. Godoi, "The precession of the perihelion of mercury explained by celestial mechanics of laplace," vol. 3, pp. 11–18, 12 2014.

[4] F. C. Adams and G. Laughlin, "Effects of secular interactions in extrasolar planetary systems," *Astrophysical Journal*, vol. 649, pp. 992–1003, Oct. 2006.

[5] "Effect of general relativity on the solar system." http://www.mathpages.com/rr/s6-02/6-02.htm. (Accessed on 10/11/2017).

[6] K. Zhang, D. P. Hamilton, and S. Matsumura, "Secular Orbital Evolution of Compact Planet Systems," *Astrophysical Journal*, vol. 778, p. 6, Nov. 2013.

[7] P. Goldreich and S. Soter, "Q in the Solar System," *Icarus*, vol. 5, pp. 375–389, 1966.

[8] C. Z. Zhang, "Love numbers of the moon and of the terrestrial planets," *Earth Moon and Planets*, vol. 56, pp. 193–207, Mar. 1992.

[9] S. V. Gavrilov and V. N. Zharkov, "Love numbers of the giant planets," *Icarus*, vol. 32, pp. 443–449, Dec. 1977.