

School of Physics and Astronomy



Mphys project Log book

Sahl Rowther
September 28th 2017

Abstract

Supervisor: Prof. Ken Rice

22 Weeks

Contents

1	Week 1: September 11th–17th 2017	1
1.1	Storing planet data	1
1.2	Storing star system data	2
1.3	Replicating inclination output	4
2	Week 2: September 18th–24th	6
3	Week 3: September 25th–31st	7
3.1	Simulation of solar system	7
3.2	Tests of simulation	14
3.2.1	Jupiter and Saturn	14
3.2.2	Whole Solar System	15
3.2.3	Precession of mercury	18
4	Week 4: October 1st–8th	19
4.1	Keplerian to Cartesian coordinates	19
5	Bibliography	21

1 Week 1: September 11th–17th 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 ($^{\circ} \text{yr}^{-1}$).
- *e* - the eccentricity.
- *i* - the inclination (degrees).
- Ω - the longitude of ascending node (degrees).
- ϖ - the longitude of pericentre (degrees).

Code:

```

1 class planet():
2     def __init__(self, Name="", Period=None, e=None, a=None, i=None, Omega=None, omega_bar=
      ↳ None, Mass=None, n=None):
3         self.name = Name
4         self.period = Period
5         self.e = e
6         self.a = a
7         self.i = i
8         self.omega = Omega
9         self.omega_bar = omega_bar
10        self.mass = Mass
11        self.n = n
12        self.units = {'a': 'AU', 'mass': 'M.EARTH', 'period': 'days', 'i': 'degrees', 'omega': 'degrees', '
      ↳ omega_bar': 'degrees', 'n': 'degrees yr-1'}
13
14    def toString(self):
15        unit_keys = list(self.units.keys())
16        for attr in self.__dict__:
17            if attr is not 'units':
18                if self.__dict__[attr] is not None:
19                    if attr in unit_keys:
20                        print('{} : {}'.format(attr, self.__dict__[attr], self.units[attr]))
21                    else:
22                        print('{} : {}'.format(attr, self.__dict__[attr]))
23        print()

```

Code 1: Planet object

Test code:

```
1 import pandas as pd
2
3 planets = pd.read_csv('solar_system.csv')
4 planet_b = planet(**planets.ix[2])
5 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:

```
1 from planet import planet
2
3 class starSystem():
4
5     def __init__(self, starMass, starRadius, planet_data_file):
6         self.star_mass = starMass
7         self.star_radius = starRadius
8         self.planets = self.addPlanets(planet_data_file)
9
10    def addPlanets(self, planet_data_file):
11        planets = pd.read_csv(planet_data_file)
12        planet_list = []
13        for p in range(len(planets)):
14            planet_list.append(planet(**planets.ix[p]))
```

```

15         return planet_list
16
17     def print_planets(self):
18         print('Star mass =', self.star_mass, 'Msun')
19         print('Star radius = ', self.star_radius, 'Rsun\n')
20         for p in self.planets:
21             p.toString()
22

```

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:

```

1 import pandas as pd
2
3 star_system = starSystem(0.86, 0.86, 'Planets.csv')
4 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

```

1 def get_property_all_planets(self, property_name, data_type="float"):
2     property_list = np.zeros(len(self.planets), dtype=data_type)
3     for idx, p in enumerate(self.planets):
4         property_list[idx] = p.__dict__[property_name]
5
6     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, $z = i \exp(i\Omega)$, where i is the inclination and Ω is the ascending node, can be simplified to a linear eigenvalue problem:

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

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

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

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

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

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

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

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

```

1 import numpy as np
2 from scipy import integrate
3
4 M_SUN = 1.9885*10**30
5 R_SUN = 6.9551*10**8
6 M_EARTH = 5.9726*10**24
7 AU = 149597870700
8
9 def laplace_coefficient(self, alpha):
10     integral_func = lambda psi, alpha: np.cos(psi)/(1+alpha**2-(2*alpha*np.cos(psi)))**(3./2.)
11     return 1/np.pi*integrate.quad(integral_func, 0, 2*np.pi, args=(alpha,))[0]
12
13 def matrix_B.eigenmodes(self):
14     G_const = 6.6738*10**(-11)
15     a = AU*self.get_property_all_planets('a')
16     M_star_kg = M_SUN*self.star_mass
17     n = np.sqrt(G_const*M_star_kg/a**3)
18
19     m = M_EARTH*self.get_property_all_planets('mass')
20
21     n_planets = len(self.planets)
22     B = np.zeros([n_planets, n_planets])
23
24     for j in range(n_planets):
25         for k in range(n_planets):
26             if j != k:
27                 alpha_jk = a[j]/a[k]
28                 if alpha_jk > 1:
29                     alpha_jk = alpha_jk**(-1)
30                 laplace_coeff = self.laplace_coefficient(alpha_jk)
31                 alpha_jk_bar = np.where(a[k] < a[j], 1, alpha_jk)
32                 B[j, k] = (n[j]/4)*(m[k]/M_star_kg)*alpha_jk*alpha_jk_bar*laplace_coeff
33             else:
34                 for kk in range(n_planets):
35                     if kk != j:
36                         alpha_jj = a[j]/a[kk]
37                         if alpha_jj > 1:
38                             alpha_jj = alpha_jj**(-1)
39                         laplace_coeff = self.laplace_coefficient(alpha_jj)
40                         alpha_jj_bar = np.where(a[kk] < a[j], 1, alpha_jj)
41                         B[j, k] += (m[kk]/M_star_kg)*alpha_jj*alpha_jj_bar*laplace_coeff
42                 B[j, k] *= -(n[j]/4)
43     eigenvalues, eigenvectors = np.linalg.eig(B)
44     return B, eigenvalues, eigenvectors

```

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

2 Week 2: September 18th–24th

3 Week 3: September 25th–31st

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

```

1 import numpy as np
2 import numpy.ma as ma
3 from scipy import integrate
4 import scipy.linalg
5 from scipy.optimize import fsolve
6 from sympy import symbols, Matrix, linsolve, diag
7 import matplotlib.pyplot as plt
8 from planet import planet
9
10 class solar_System():
11
12     def __init__(self, starMass, starRadius, planet_data_file):
13         self.star_mass = starMass
14         self.star_radius = starRadius
15         self.planets = self.addPlanets(planet_data_file)
16
17     def addPlanets(self, planet_data_file):
18         planets = pd.read_csv(planet_data_file)
19         planet_list = []
20         for p in range(len(planets)):
21             planet_list.append(planet(**planets.ix[p]))
22         return planet_list
23

```

```

24 def get_property_all_planets(self, property_name, data_type="float"):
25     property_list = np.zeros(len(self.planets), dtype=data_type)
26     for idx, p in enumerate(self.planets):
27         property_list[idx] = p.__dict__[property_name]
28     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:

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

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:

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

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

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

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

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:

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

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

```

1 def calculate_laplace_coeff(alpha, j, s):
2     return integrate.quad(lambda psi, alpha, j, s: np.cos(j*psi)/(1-2*alpha*np.cos(psi)+alpha**2)**s,
3                           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:

$$h_j = e_j \cos \varpi_j \quad (8a)$$

$$k_j = e_j \sin \varpi_j \quad (8b)$$

$$p_j = i_j \cos \Omega_j \quad (8c)$$

$$q_j = i_j \sin \Omega_j \quad (8d)$$

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:

```

20 def frequency_matrix(self, matrix_id, J2=0, J4=0):
21     M_star_kg = M_SUN*self.star_mass
22     R = R_SUN*self.star_radius
23     m = M_EARTH*self.get_property_all_planets('mass')
24     n = self.get_property_all_planets('n')
25     a = AU*self.get_property_all_planets('a')
26     n_planets = len(self.planets)
27     f_mat = np.zeros([n_planets, n_planets])
28
29     if matrix_id == 'A':
30         j_laplace_coeff_jk, j_laplace_coeff_jj = 2, 1
31         front_factor = -1
32         J2_correction = (((3/2)*J2*(R/a)**2)-((9/8)*(J2**2)*(R/a)**4)-((15/4)*J4*(R/a)**4))
33
34     if matrix_id == 'B':
35         j_laplace_coeff_jk = j_laplace_coeff_jj = 1
36         front_factor = 1
37         J2_correction = (((3/2)*J2*(R/a)**2)-((27/8)*(J2**2)*(R/a)**4)-((15/4)*J4*(R/a)**4))
38
39     for j in range(n_planets):
40         for k in range(n_planets):
41             if j != k:
42                 alpha_jk = a[j]/a[k]
43                 if alpha_jk > 1:
44                     alpha_jk = alpha_jk**(-1)
45                 laplace_coeff = calculate_laplace_coeff(alpha_jk, j_laplace_coeff_jk, 3/2)
46                 alpha_jk_bar = np.where(a[k] < a[j], 1, alpha_jk)
47                 f_mat[j, k] = front_factor*(n[j]/4)*(m[k]/(M_star_kg+m[j]))*alpha_jk*alpha_jk_bar*
48                 ↪ laplace_coeff
49
50     else:
51         for kk in range(n_planets):
52             if kk != j:

```

```

52         alpha_jj = a[j]/a[kk]
53         if alpha_jj > 1:
54             alpha_jj = alpha_jj*(-1)
55         laplace_coeff = calculate_laplace_coeff(alpha_jj, j_laplace_coeff_jj, 3/2)
56         alpha_jj_bar = np.where(a[kk] < a[j], 1, alpha_jj)
57         f_mat[j, k] += (1/4)*(m[kk]/(M_star_kg+m[j]))*alpha_jj*alpha_jj_bar*
↪ laplace_coeff
58         f_mat[j, k] += J2_correction[j]
59         f_mat[j, k] *= -front_factor*(n[j])
60     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:

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

and

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

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:

$$S_i \bar{e}_{ji} = e_{ji} \quad \text{and} \quad T_i \bar{I}_{ji} = I_{ji}. \quad (10)$$

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.

```

61     def initial_conditions(self):
62         e = self.get_property_all_planets('e')
63         omega_bar = self.get_property_all_planets('omega_bar')*np.pi/180
64         i = self.get_property_all_planets('i')*np.pi/180
65         omega = self.get_property_all_planets('omega')*np.pi/180
66
67         h = e*np.sin(omega_bar)
68         k = e*np.cos(omega_bar)
69         p = i*np.sin(omega)
70         q = i*np.cos(omega)
71
72     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.

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

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

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

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 .

```

73 def solve_property(self, eigenvectors, initial_conditions):
74     n = len(self.planets)
75     aug = Matrix(np.zeros([n, n+1]))
76     aug[:, :n] = eigenvectors
77     aug[:, n] = initial_conditions
78
79     result = linsolve(aug, *symbols('x0:' + str(n)))
80     answers = np.zeros(n)
81     for a, answer in enumerate(ans):
82         answers[a] = answer
83     return answers
84
85
86 def find_all_scaling_factor_and_phase(self, eigenvectors_of_A, eigenvectors_of_B):
87     x, y = eigenvectors_of_A, eigenvectors_of_B
88
89     init_conditions = np.array(star_system.initial_conditions())
90     h_solved = self.solve_property(x, init_conditions[0, :])
91     k_solved = self.solve_property(x, init_conditions[1, :])
92     p_solved = self.solve_property(y, init_conditions[2, :])
93     q_solved = self.solve_property(y, init_conditions[3, :])
94
95     n = len(self.planets)
96     S, beta = np.zeros(n), np.zeros(n)
97     T, gamma = np.zeros(n), np.zeros(n)
98
99     for i in range(n):
100         S[i], beta[i] = fsolve(scaling_factor_and_phase, (1, -1), args=(h_solved[i], k_solved[i],))
101         T[i], gamma[i] = fsolve(scaling_factor_and_phase, (-1, 1), args=(p_solved[i], q_solved[i],))

```

```
102 return S, beta, T, gamma
```

Code 11: 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 .

```
103 def eq_of_motion(self, scaled_eigenvector, eigenvalue, phase, t, eq_id):
104     # eq_id = 'h', 'k', 'p', 'q'
105     kwargs = {'scaled_eigenvector': scaled_eigenvector, 'eigenvalue': eigenvalue, 'phase': phase, 't':
↪ t}
106     if eq_id == 'h' or eq_id == 'p':
107         return self.get_h_or_p(**kwargs)
108     if eq_id == 'k' or eq_id == 'q':
109         return self.get_k_or_q(**kwargs)
110
111 def get_h_or_p(self, scaled_eigenvector, eigenvalue, phase, t):
112     n = len(self.planets)
113     h_list = []
114     for j in range(n):
115         h = np.zeros_like(t)
116         for i in range(n):
117             h += scaled_eigenvector[j, i]*np.sin((eigenvalue[i]*t+phase[i])*np.pi/180)
118         h_list.append(h)
119     return np.array(h_list)
120
121 def get_k_or_q(self, scaled_eigenvector, eigenvalue, phase, t):
122     n = len(self.planets)
123     k_list = []
124     for j in range(n):
125         k = np.zeros_like(t)
126         for i in range(n):
127             k += scaled_eigenvector[j, i]*np.cos((eigenvalue[i]*t+phase[i])*np.pi/180)
128         k_list.append(k)
129     return np.array(k_list)
```

Code 12: Calculating the equations of motion

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

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

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

```
131 def get_eccentricity(self, scaled_eigenvector_of_A, eigenvalue_of_A, beta, t):
132     n = len(self.planets)
133     kwargs = {'scaled_eigenvector': scaled_eigenvector_of_A, 'eigenvalue': eigenvalue_of_A, 'phase':
↪ beta, 't': t}
134     eccentricities = []
135     h, k = self.eq_of_motion(**kwargs, eq_id='h'), self.eq_of_motion(**kwargs, eq_id='k')
136     for j in range(n):
137         eccentricities.append(np.sqrt(h[j]**2+k[j]**2))
```

```

138     return np.array(eccentricities)
139
140     def get_inclination(self, scaled_eigenvector_of_B, eigenvalue_of_B, gamma, t):
141         n = len(self.planets)
142         kwargs = {'scaled_eigenvector' : scaled_eigenvector_of_B, 'eigenvalue' : eigenvalue_of_B, 'phase' :
143         ↪ gamma, 't' : t}
144         inclinations = []
145         p, q = self.eq_of_motion(**kwargs, eq_id='p'), self.eq_of_motion(**kwargs, eq_id='q')
146         for j in range(n):
147             inclinations.append(np.sqrt(p[j]**2+q[j]**2))
148     return np.array(inclinations)

```

Code 13: 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,

$$\tan \varpi = \frac{h_j}{k_j}. \quad (14)$$

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

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

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:

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

Which then become:

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

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

```

156     def get_perihelion_precession_rates(self, A, eccentricities, h_list, k_list):
157         n = len(self.planets)
158         d_pidot_dt_list = []
159         masks = []
160
161         for j in range(n):
162             h_dot_j, k_dot_j = 0, 0
163             for i in range(n):

```

```

164         h_dot_j += A[j, i]*k_list[i]
165         k_dot_j -= A[j, i]*h_list[i]
166         pidot_j = 3600*(k_list[j]*h_dot_j - h_list[j]*k_dot_j)/(eccentricities[j])**2
167         d_pidot_dt_list.append(pidot_j)
168     return d_pidot_dt_list

```

Code 14: Calculating precession rate, $\dot{\varpi}$ of Mercury

3.2 Tests of simulation

The following code was used to test the simulation.

```

1  def simulate(self, t, plot=False, separate=True):
2      A, B = [star_system.frequency_matrix(matrix_id=mat_id, J2=-6.84*10**(-7), J4
3      ↪ =2.8*10**(-12)) for mat_id in ['A', 'B']]
4      g, x, f, y = *np.linalg.eig(A), *np.linalg.eig(B)
5      S, beta, T, gamma = self.find_all_scaling_factor_and_phase(x, y)
6
7      eccentricities = self.get_eccentricity(S*x, g, beta, t)
8      inclinations = self.get_inclination(T*y, f, gamma, t)*180/np.pi
9      names = [self.planets[p].name for p in range(len(self.planets))]
10     if plot:
11         if separate:
12             plot_simulation_separate(t/10**6, eccentricities, 'Time (Myr)', 'Eccentricity', names)
13             plot_simulation_separate(t/10**6, inclinations, 'Time (Myr)', 'Inclination', names)
14         else:
15             plot_simulation_all(t/10**6, eccentricities, 'Time (Myr)', 'Eccentricity', names)
16             plot_simulation_all(t/10**6, inclinations, 'Time (Myr)', 'Inclination', names)
17
18     kwargs = {'scaled_eigenvector' : S*x, 'eigenvalue' : g, 'phase' : beta,
19             't' : t}
20     h_list = self.eq_of_motion(**kwargs, eq_id='h')
21     k_list = self.eq_of_motion(**kwargs, eq_id='k')
22     kwargs = {'scaled_eigenvector' : S*x, 'eigenvalue' : f, 'phase' : gamma,
23             't' : t}
24     p_list = self.eq_of_motion(**kwargs, eq_id='p')
25     q_list = self.eq_of_motion(**kwargs, eq_id='q')
26
27     precession_rates = self.get_perihelion_precession_rates(A, eccentricities, h_list, k_list)
28
29     idx = 0
30     plot_precession_rate(t, precession_rates[idx], 'Mercury')
31     plot_eccentricity(t, eccentricities[idx], 'Mercury')

```

Code 15: 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].

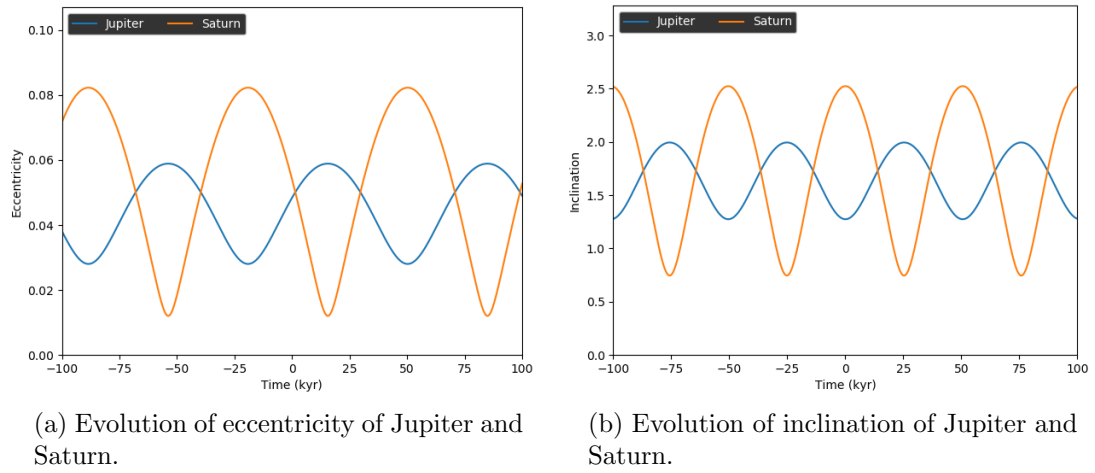


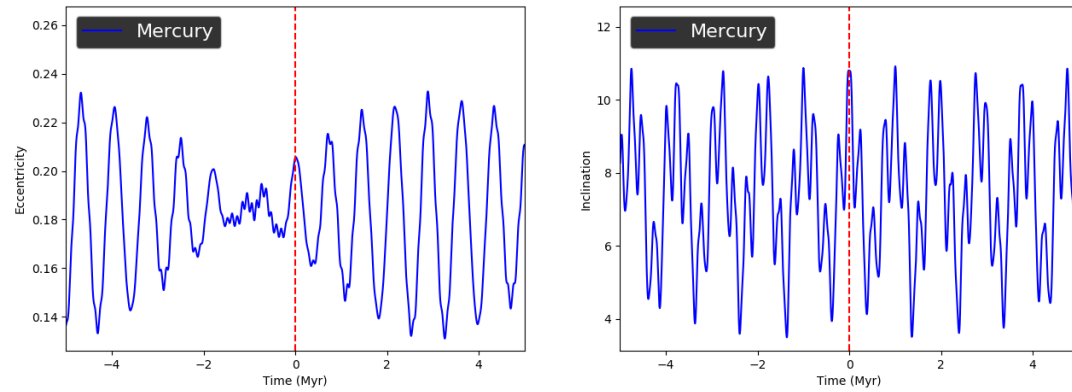
Figure 1

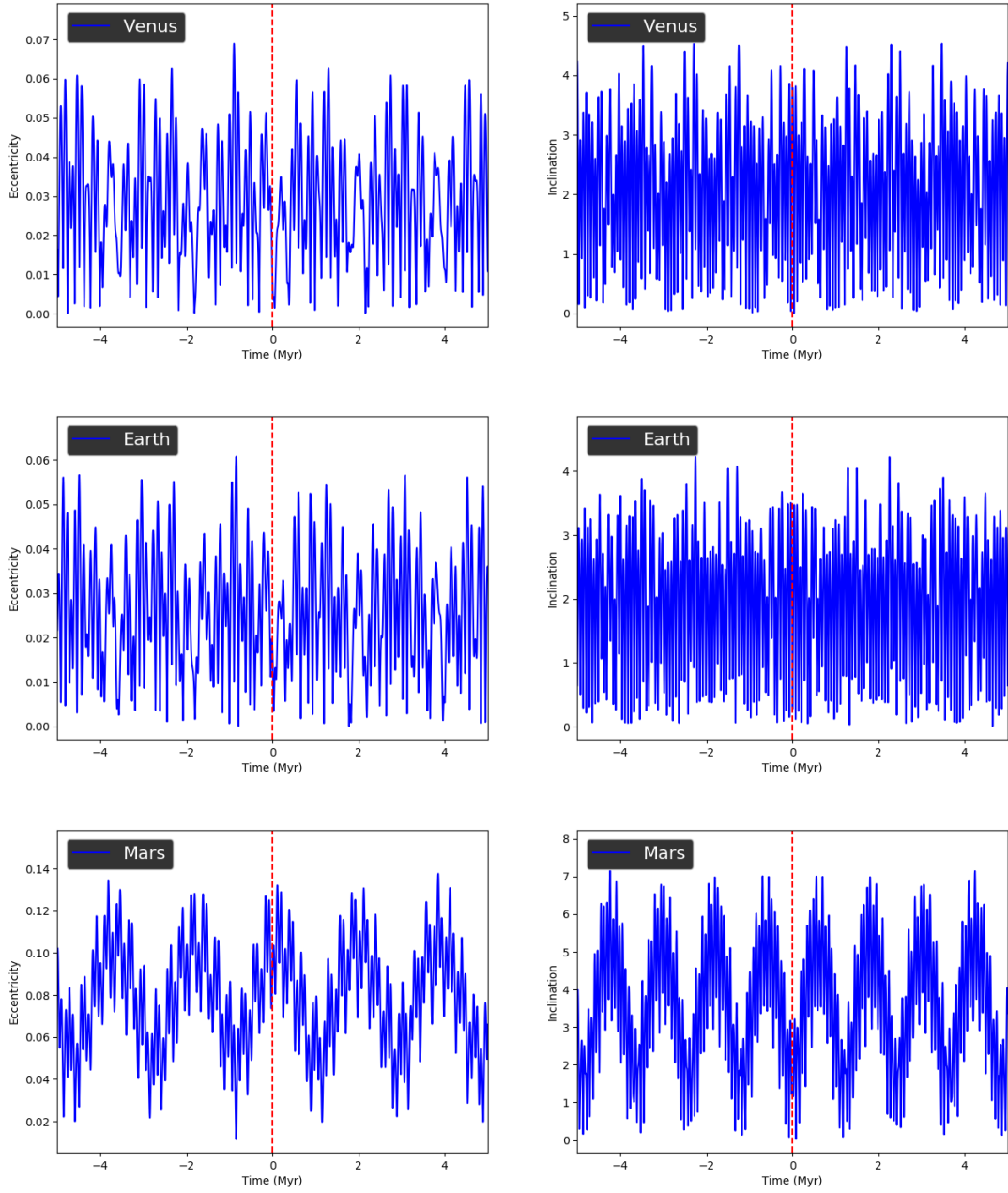
Verdict

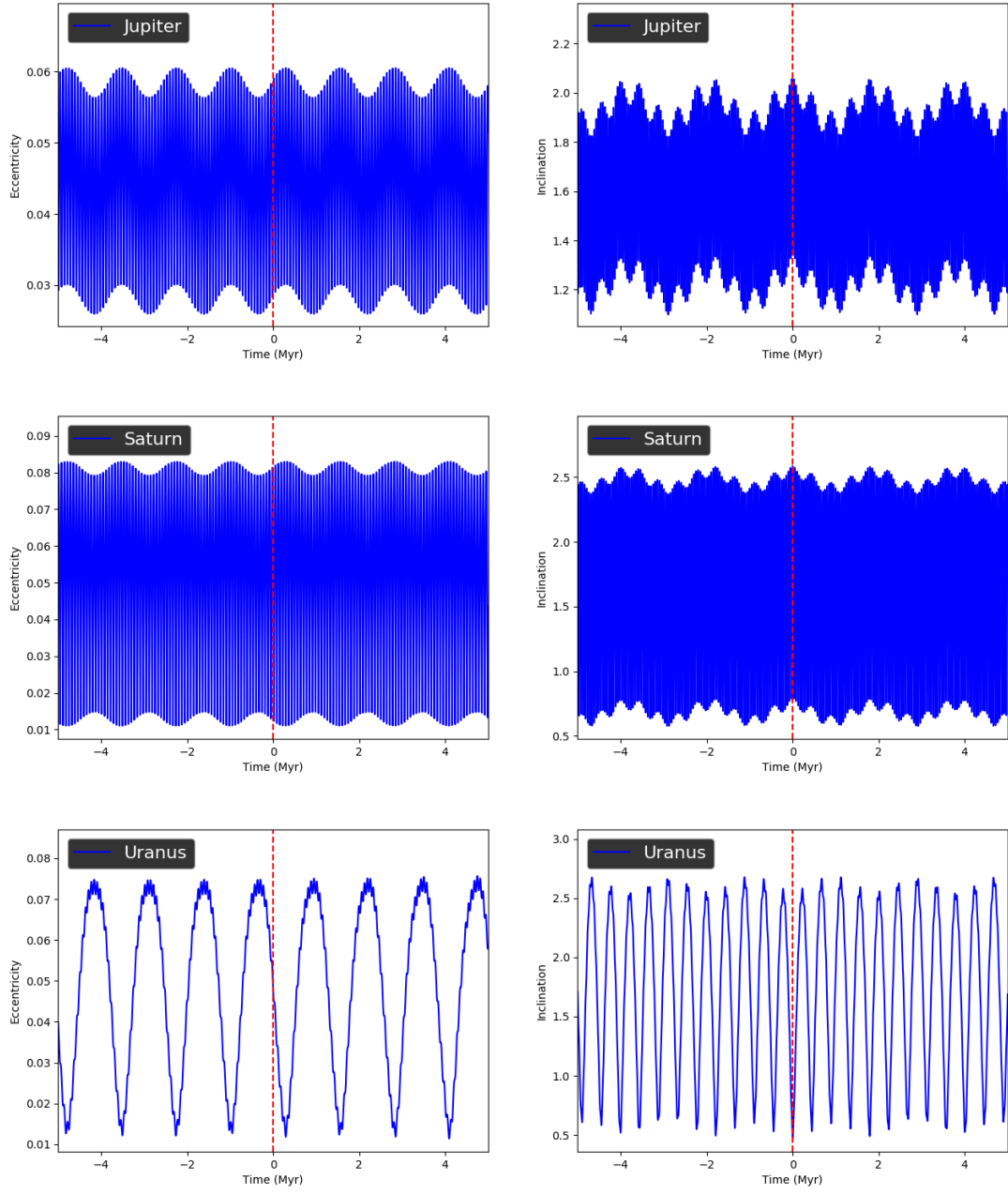
Figure 1 is a very good match to that of the output in Murray & Dermott (1999) ^[1].

3.2.2 Whole Solar System

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







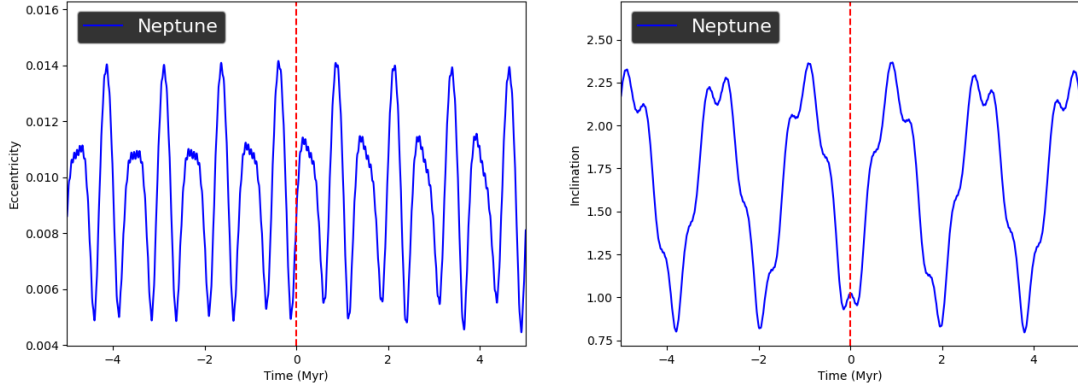


Figure 2: Eccentricity and inclination (in degrees) of each planet in the Solar System using J2000 data.

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 3b, 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].

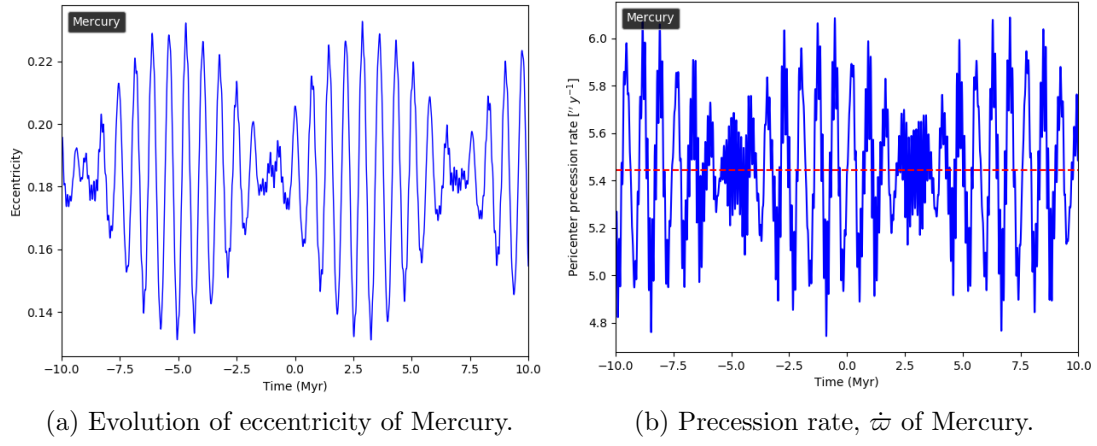


Figure 3

4 Week 4: October 1st–8th

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.

Code:

```

1  def kep2cart(self, ecc, inc, h_list, k_list, p_list, q_list, time, t0, idx):
2      O_list = self.get_pi_or.omega(p_list, q_list)
3      w_list = O_list - self.get_pi_or.omega(h_list, k_list)
4
5      n = self.get_property_all_planets('n')
6      a = self.get_property_all_planets('a')
7      l = self.get_property_all_planets('l') * np.pi / 180
8      pi = self.get_property_all_planets('omega_bar') * np.pi / 180
9
10     M0 = l[idx] - pi[idx]
11     Mt = n[idx] * np.pi / 180 * (time - t0)
12
13     EA = []
14     e, w, O, i = ecc[idx], w_list[idx], O_list[idx], inc[idx]
15     for t in range(len(time)):
16         E = Mt[t]
17         f_by_dfdE = (E - e[t] * np.sin(E) - Mt[t]) / (1 - e[t] * np.cos(E))
18         j, maxIter, delta = 0, 30, 0.0000000001
19         while (j < maxIter) * (np.abs(f_by_dfdE) > delta):
20             E = E - f_by_dfdE
21             f_by_dfdE = (E - e[t] * np.sin(E) - Mt[t]) / (1 - e[t] * np.cos(E))
22             j += 1
23         EA.append(E)
24     EA = np.array(EA)
25     nu = 2 * np.arctan2(np.sqrt(1 + e) * np.sin(EA / 2), np.sqrt(1 - e) * np.cos(EA / 2))
26
27     rc = a[idx] * (1 - e * np.cos(EA))
28     o_vec = np.array([rc * np.cos(nu), rc * np.sin(nu), 0])
29
30     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.
31     ↪ cos(O) + np.cos(w) * np.cos(i) * np.sin(O)))
32     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.
33     ↪ .cos(i) * np.cos(O) - np.sin(w) * np.sin(O)))
34     rz = (o_vec[0] * (np.sin(w) * np.sin(i)) + o_vec[1] * (np.cos(w) * np.sin(i)))
35
36     return rx, ry, rz

```

Code 16: Converting from Keplerian to Cartesian coordinates

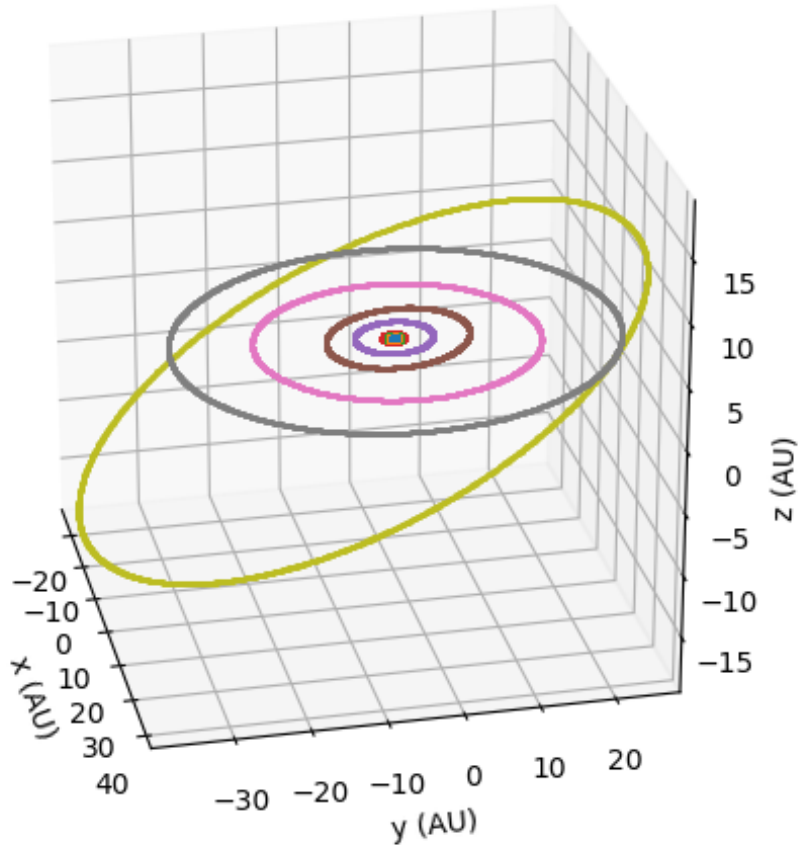


Figure 4: Plot of the Solar System simulated for a 1000 years from J2000 data.

5 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.