Компьютерные технологии

Гвоздков Е.

6 июня 2021 г.

Задание 8 Постройте модель Солнечной системы. Рассчитайте параметры траектории кометы, попавшей в Солнечную систему извне. Постройте зависимости скорости и координаты кометы от времени при различных начальных параметрах, а также оцените точность интегрирования в зависимости от схемы интегрирования и величины шага интегрирования.

1 Описание модели Солнечной системы

Для описания движения планет и кометы в поле тяготения Солнца примем несколько приближений:

1. Планеты не влияют гравитацией друг на друга
2. Описание движения будет происходить в плоскости, т.е. не учитывается координата z
3. Комета не влияет на орбиты планет Солнечной системы
4. Солнце неподвижно в начале координат

Поскольку влиянием планет друг на друга принебрегается, их орбиты описываются определенным образом. Траектория орбиты представляет из себя эллипс, в фокусе которого расположено тяготеющее тело, в данном случае Солнце.

1.1 Описание движения планет

Уравнение эллипса орбиты в полярных координатах задается следующим образом:

$$r = \frac{a(1-e^2)}{1+e\cos(\theta+\alpha)},$$

где a - большая полуось эллипса, e - эксцентриситет, α - угловой сдвиг эллипса относительно $\theta = 0$.

Закон невозмущенного движения тела по эллиптической орбите из второго закона Кеплера имеет вид

$$r^2\frac{d\theta}{dt} = \text{const} = \sqrt{\mu a(1-e^2)},$$

где $\mu = GM$ - гравитационный параметр (G - гравитационная постоянная, M - масса Солнца).

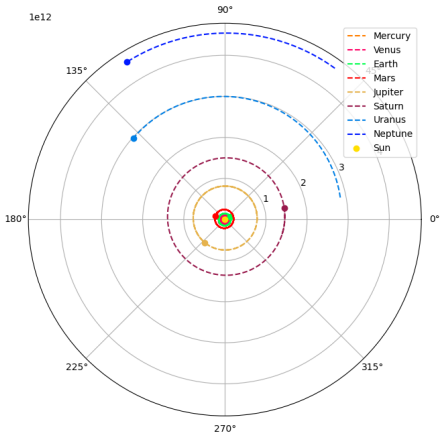

Рис. 1: Модель Солнечной системы

2 Описание движения кометы в Солнечной системе

Движение тела в поле тяготения описываются законом всемирного тяготения

$$F_g = G\frac{m_1 m_2}{R^2},$$

где m_1, m_2 - массы тел, R - расстояние между телами. Сила при этом направлена от кометы к планете. В случае нескольких тел N, действующих гравитацией на конкретное тело (комету), силы суммируются, и закон примет виде

$$F_g = Gm \sum_{i=0}^{N-1} \frac{m_i}{R_i^2},$$

где i - индекс, m - масса кометы, m_i - масса i-ой планеты, R_i - расстояние между кометой и i-той планетой.

Для моделирования влияния нескольких тел на движение кометы, воспользуемся вторым законом Ньютона:

$$\vec{F} = m\vec{a} = \vec{F_g} = Gm \sum_{i=0}^{N-1} \frac{m_i}{R_i^2} \vec{e_i},$$

где $\vec{e_i}$ - единичный вектор, направленный от кометы к планете с индексом i. Приведем выражение выше в другом виде

$$\vec{r''} = G \sum_{i=0}^{N-1} m_i \frac{\vec{r_i} - \vec{r}}{R_i^3},$$

где $\vec{r_i}$ - радиус вектор положения планеты с индексом i. Введем $\vec{v} = \vec{r'}$, тогда получим следующую систему уравнений

$$\vec{v'} = G \sum_{i=0}^{N-1} m_i \frac{\vec{r_i} - \vec{r}}{R_i^3}, \quad \vec{r'} = \vec{v}$$

3 Результаты моделирования

Описание и моделирование системы производится на языке Python. Начальными параметрами моделирования выступают первоначальные положения планет Солнечной системы, а также начальные координаты и скорость кометы.

Планеты и их движение описываются классом CelestialBody в файле SolarSystem.py. Каждая планета - инстанция класса. Комета описывается отдельным клас-

сом Comet, в котором также присутствует метод evaluate_model, который является основным в моделировании.

4 Исходный код

```
# Задание 8. Постройте модель Солнечной системы. Рассчитайте параметры траектории кометы,
# попавшей в Солнечную систему извне. Постройте зависимости скорости и координаты кометы
# от времени при различных начальных параметрах, а также оцените точность интегрирования в
# зависимости от схемы интегрирования и величины шага интегрирования.
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

G = 6.67430*10**(-11)
M0 = 1.9885*10**30

class Sun():
    def __init__(self):
        self.mass = M0
        self.r = 0
        self.theta = 0

    def calculate_position_by_time(self, t):
        self.r = 0
        self.theta = 0
        return self.r, self.theta


class Comet():
    def __init__(self, name, mass, start_r, start_theta, v0_r, v0_theta):
        self.name = name
        self.mass = mass
        self.start_r = start_r
        self.start_theta = start_theta
        self.r = start_r
        self.theta = start_theta
        self.v_r = v0_r
        self.v_theta = v0_theta
        self.x, self.y = self.convert_coord_polar_to_decart(start_r, start_theta)
        self.v_x, self.v_y = self.convert_speed_polar_to_decart(v0_r, v0_theta, start_theta)
        self.recorded_theta = [start_theta]
        self.recorded_r = [start_r]

    def convert_coord_polar_to_decart(self, r, theta):
        x = r*np.cos(theta)
        y = r*np.sin(theta)
        return x, y

    def convert_speed_polar_to_decart(self, v_r, v_theta, theta):
        v_x = v_r*np.cos(theta) - v_theta*np.sin(theta)
```

```
        v_y = v_r*np.sin(theta) + v_theta*np.cos(theta)
        return v_x, v_y

    def convert_coord_decart_to_polar(self, x, y):
        r = np.sqrt(x**2 + y**2)
        theta = np.arctan2(y, x)
        return r, theta

    def calculate_distance_to_body(self, body):
        distance = np.sqrt( body.r**2 + self.r**2  - 2*body.r*self.r*np.cos(body.theta-self.theta))
        return distance

    def calculate_force_to_body(self, body):
        distance = self.calculate_distance_to_body(body)
        F = G*self.mass*body.mass/distance**2
        body_x, body_y = self.convert_coord_polar_to_decart(body.r, body.theta)
        F_x = F*(body_x - self.x)/distance
        F_y = F*(body_y - self.y)/distance
        return F_x, F_y

    def calculate_summary_acceleration(self, bodies):
        a_x_list, a_y_list = [], []
        for body in bodies:
            F_x, F_y = self.calculate_force_to_body(body)
            a_x_list.append(F_x/self.mass)
            a_y_list.append(F_y/self.mass)
        a_x = np.mean(a_x_list)
        a_y = np.mean(a_y_list)
        return a_x, a_y

    def model_func(self, t, data_vec):
        #  data_vec = [x,y, vx,vy]
        coords = data_vec[0:2]
        vel = data_vec[2:]

        self.x, self.y = coords[0], coords[1]
        self.r = np.sqrt(coords[0]**2 + coords[1]**2)
        self.theta = np.arctan2(coords[1], coords[0])

        for body in self.bodies:
            body.calculate_position_by_time(t)
        ax, ay = self.calculate_summary_acceleration(self.bodies)
        dvdt = [ax, ay]
        # returns r', v'
        return np.hstack((vel, dvdt))

    def evaluate_model(self, t_end, bodies, t_eval=None, method='RK45'):
        self.bodies = bodies
```

```
94          res = solve_ivp(self.model_func, t_end,
95                          y0=[self.x, self.y, self.v_x, self.v_y],
96                          t_eval=t_eval, method=method)
97          return res
98
99
100 class CelestialBody():
101     def __init__(self, name, mass, a, eccentricity,
102                  offset, start_theta, plot_color):
103         ''' Основной класс для описания орбит планет вокруг Солнца.
104             Включает методы для рассчета движения во времени по заданной траектории.
105         '''
106         self.name = name
107         self.mass = mass
108         self.a = a  # Большая полуось
109         self.ecc = eccentricity  # Эксцентриситет
110         self.angle_offset = np.pi*offset/180 # Долгота восходящего узла
111         self.start_theta = start_theta # Начальное положение
112         self.p = a*(1-self.ecc**2)
113         self.mu = G*M0
114         self.color = plot_color
115
116         self.const1 = self.a*(1-self.ecc**2)
117         self.const2 = np.sqrt(self.mu*self.a*(1-self.ecc**2))
118         self.get_r(self.start_theta)
119         self.theta = start_theta
120
121         self.recorded_theta = [self.start_theta]
122         self.recorded_r = [self.r]
123
124     def calculate_position_by_time(self, t, points=100):
125         dt = t/points
126         for i in np.linspace(0, t, points):
127             self.calculate_next_posistion(dt)
128         r, theta = self.recorded_r[-1], self.recorded_theta[-1]
129         self.reset
130         self.r, self.theta = r, theta
131         return r, theta
132
133     def get_r(self, theta):
134         ''' Get radius value for current angle theta.
135         '''
136         self.r = self.const1/(1+self.ecc*np.cos(theta + self.angle_offset))
137         return self.r
138
139     def get_d_theta(self, dt):
140         '''  Get angle increment d_theta for current radius value
141              and time increment dt.
```

```
        '''
        d_theta = 1/self.r**2*self.const2*dt
        return d_theta

    def calculate_next_posistion(self, dt):
        d_theta = self.get_d_theta(dt)
        next_theta = self.recorded_theta[-1] + d_theta
        next_r = self.get_r(next_theta)
        self.recorded_theta.append(next_theta)
        self.recorded_r.append(self.r)
        return next_r, next_theta

    def reset(self):
        self.get_r(self.start_theta)
        self.recorded_theta = [self.start_theta]
        self.recorded_r = [self.r]

TheSun = Sun()

Mercury = CelestialBody("Mercury", 3.33022*10**23, 57909227000, 0.20563593, 48.33167, -0.2*np.pi,
    [1,0.5,0])
Venus = CelestialBody("Venus", 4.8675*10**24, 108208930000, 0.0068, 76.67069, 0.1*np.pi, [1,0,0.4])
Earth = CelestialBody("Earth", 5.9726*10**24, 149598261000, 0.01671123, 348.73936, 0.8*np.pi, [0,1,0.3])
Mars = CelestialBody("Mars", 6.4171*10**23, 2.2794382*10**8*1000, 0.0933941, 49.57854, 1.15*np.pi,
    [1,0,0])
Jupiter = CelestialBody("Jupiter", 1.8986*10**27, 7.785472*10**8*1000, 0.048775, 100.55615, -0.02*np.pi,
    [0.9,0.7,0.3])
Saturn = CelestialBody("Saturn", 5.6846*10**26, 1429394069000, 0.055723219, 113.642, -0.08*np.pi, [0.6, 0.1,
     0.3])
Uranus = CelestialBody("Uranus", 8.6813*10**25, 2876679082000, 0.044, 73.9898, 0.06*np.pi, [0,0.5,0.9])
Neptune = CelestialBody("Neptune", 1.0243*10**26, 4503443661000, 0.011214269, 131.794, 0.3*np.pi,
    [0,0.1,1])
Planets = [Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune]
System = [TheSun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune]
```

Листинг 1: Исходный код задания

```

import matplotlib.pyplot as plt
import numpy as np
from SolarSystem import System, Planets, Comet
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})

# Начальные параметры кометы
TheComet = Comet("Comet",
                 1*10**11,   # Масса
                 50*10**11,  # Начальный радиус
                 3*np.pi/4,  # Начальный угол
                 0,          # Начальная радиальная скорость
```

```
                1500        # Начальная угловая скорость
                )

# Параметры симуляции
integration_method = 'RK23'
# Время моделирования
t_end = 1*10**9
# Шаг моделирования
dt = 2*10**6
t = np.arange(0, t_end, dt)

# Моделируем модель кометы в Солнечной системе
res = TheComet.evaluate_model([0, t_end], System, t_eval=t, method=integration_method)
time = res.t
coords = res.y[0:2]
vels = res.y[2:]
# Конвертируем координаты в полярные для дальнейшего построения графиков
comet_r, comet_theta = TheComet.convert_coord_decart_to_polar(
    coords[0], coords[1])

### Далее код отвечает за отрисовку

# Ресетим планеты в их изначальное положение
for Planet in Planets:
    Planet.reset()

# Расчет движения планет во времени Нужно( для для построения траекторий планет)
for i, val in enumerate(t):
    for Planet in Planets:
        r_next, theta_next = Planet.calculate_next_posistion(dt)

# Строим траектории планет и кометы в полярных координатах
for Planet in Planets:
    ax.plot(Planet.recorded_theta, Planet.recorded_r,
            '--', color=Planet.color, label=Planet.name)
    ax.plot(Planet.recorded_theta[-1],
            Planet.recorded_r[-1], 'o', color=Planet.color)
ax.plot(comet_theta, comet_r, color='k', label="Comet")
ax.plot(comet_theta[-1], comet_r[-1], 'o', color='k')
ax.plot(0, 0, 'o', color='#FFDF00', label="Sun")

plt.legend()
plt.show()
```

Листинг 2: Исходный код задания