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.

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).

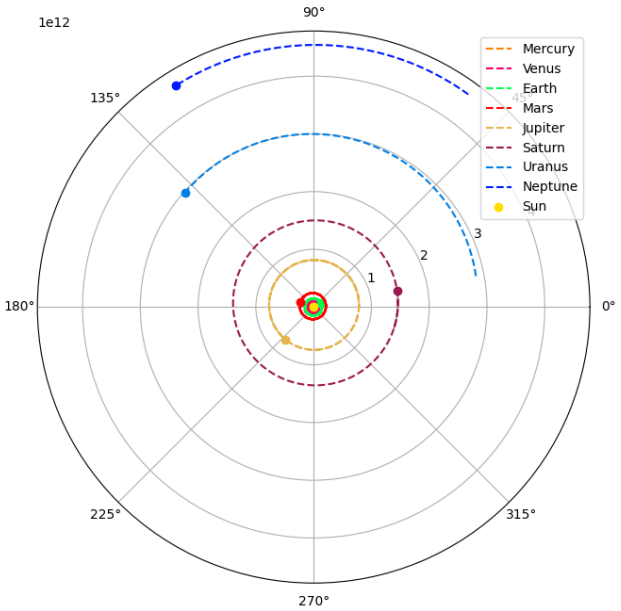


Рис. 1: Модель Солнечной системы

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. Параметры орбит планет считаются фиксированными и заданы предварительно на основе информации из открытых источников.

Планеты и законы их движения описываются классом CelestialBody в файле SolarSystem.py. Каждая планета - инстанция класса. Комета описывается отдельным классом Comet, в котором также присутствует метод класса

evaluate_model, который является основным в моделировании.

В качетсве параметров кометы выступают масса, начальные координаты и скорости. Также нужно задать время моделирования t_end - до какого момента времени в секундах моделировать движение. Также есть возможность выбрать метод интгрирования, задав соответствующее строковое значение переменной integration_method

```
TheComet = Comet("Comet",
                 8*10**22,    # Масса, кг
                 1.5*10**12,  # Начальный радиус, м
                 0.8*np.pi,   # Начальный угол, радианы
                 0,           # Начальная радиальная скорость, м/с
                 10000        # Начальная угловая скорость, мс/
             )
integration_method = 'Radau'
t_end = 20*10**8
```

Листинг 1: Блок параметров симуляции

3.1 Скорость и координата кометы в зависимости от начальных параметров

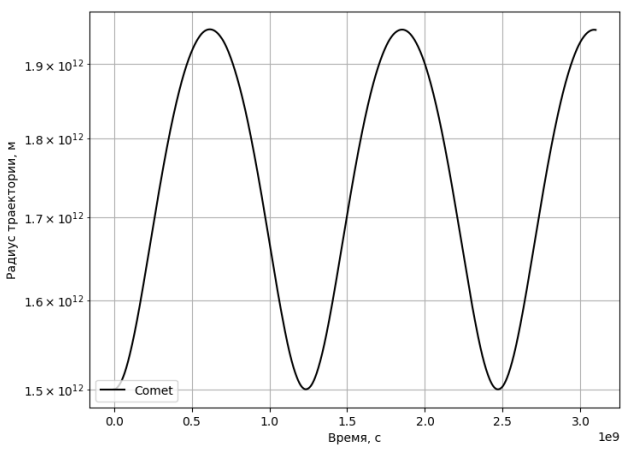
Начальные параметры в тексте будем указывать как m - масса кометы, R_0 - начальный радиус траектории кометы, θ_0 - начальный угол, v_r - начальная радиальная скорость, v_θ - начальная угловая скорость. В основном будет метод интегрирования "Radau метод Рунге-Кутты 5-го порядка. параметры моделирования будут указыаваться следующим образом $[m, R_0, \theta_0, v_r, v_\theta]$

```
"ConfigName": [8*10**22, 1.5*10**12, 0, 0, 10000]
```

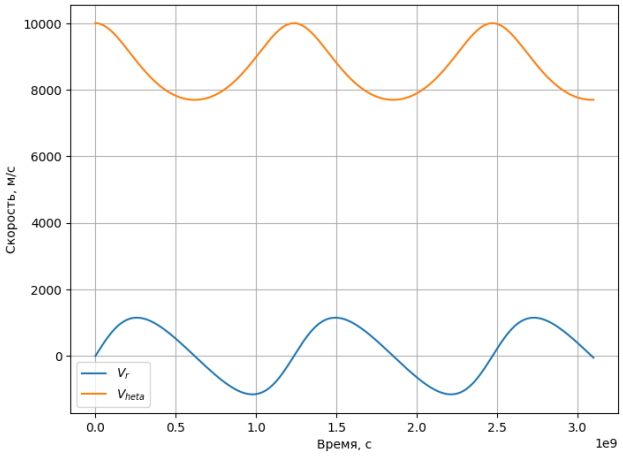
Модель комета - Солнце

Рассмотрим простой случай, когда в модели отсутствуют другие планеты. Начальные параметры кометы:

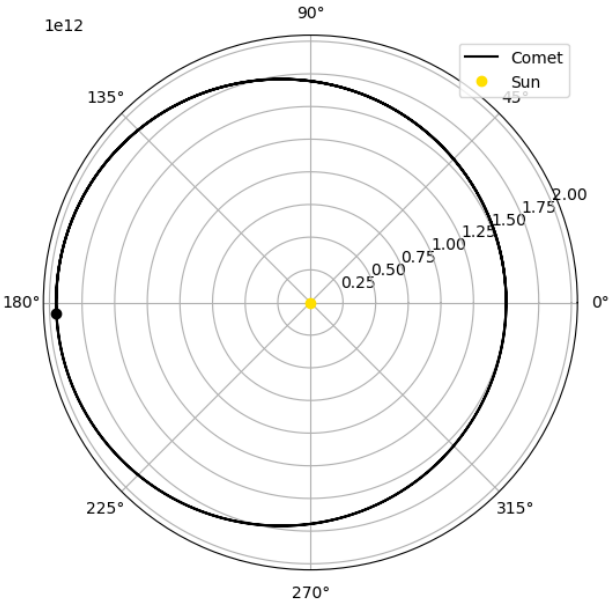
```
"SunComet": [8*10**22, 1.5*10**12, 0, 0, 10000]
```



(a) Координата $r(t)$



(b) Скорости $V_r(t), V_\theta(t)$



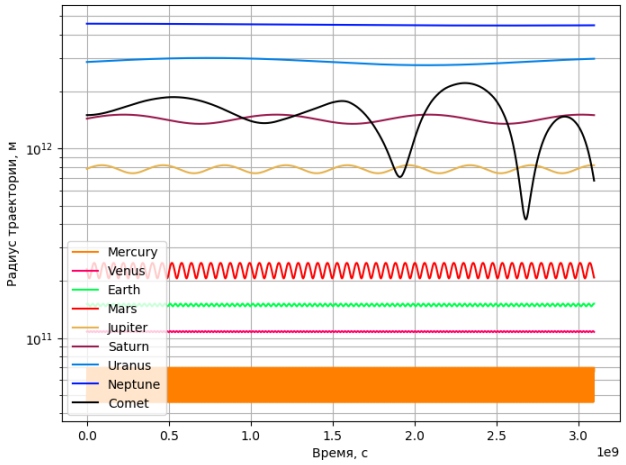
(c) Траектория

Рис. 2: Модель комета - Солнце.

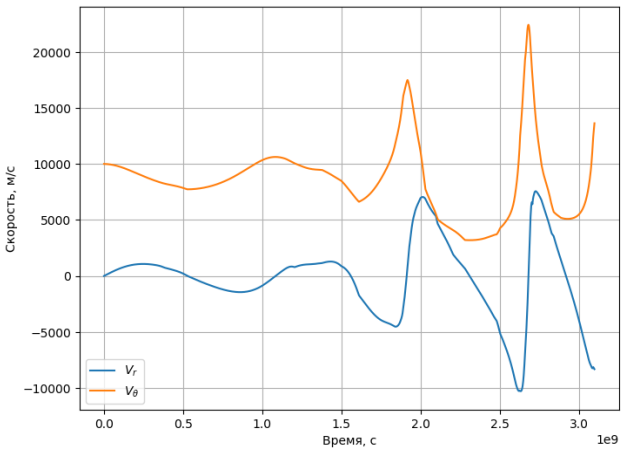
На рис. 2(c) представлена траектория кометы - в данном случае, т.к. других планет нету, то комета невозмущенно вращается вокруг Солнца по эллипсу.

Комета появляется внутри Солнечной системы

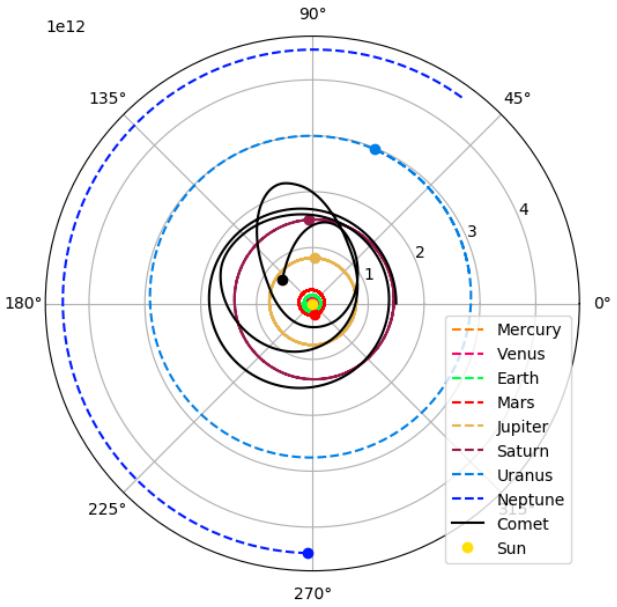
Если же мы, не меняя начальных параметров, "включим"влияние планет, картина сильно изменится:



(a) Координата $r(t)$, а также координаты планет $r_i(t)$



(b) Скорости $V_r(t), V_\theta(t)$



(c) Траектория кометы и планет

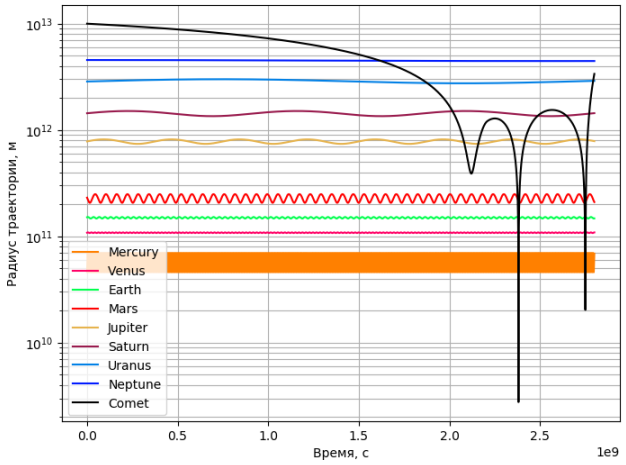
Рис. 3: Результаты моделирования при "включении"влияния планет

Траектория кометы становится сильно искаженной (см. рис. 3), т.к. она подвергается сильному влиянию Сатурна и Юпитера.

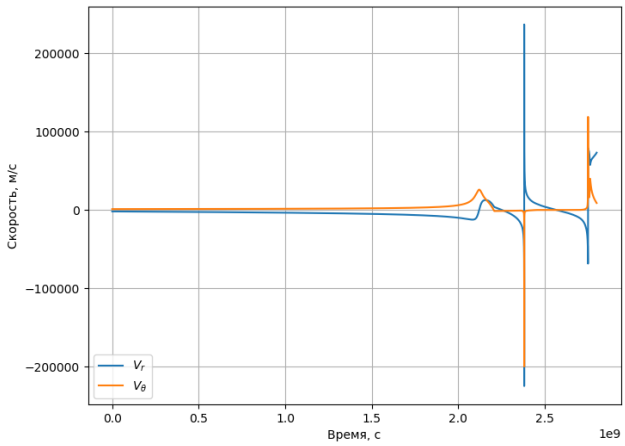
Комета прилетает извне Солнечной системы

Рассмотрим случай когда комета прилетает извне Солнечной системы, и проходит сквозь нее. Начальные параметры кометы:

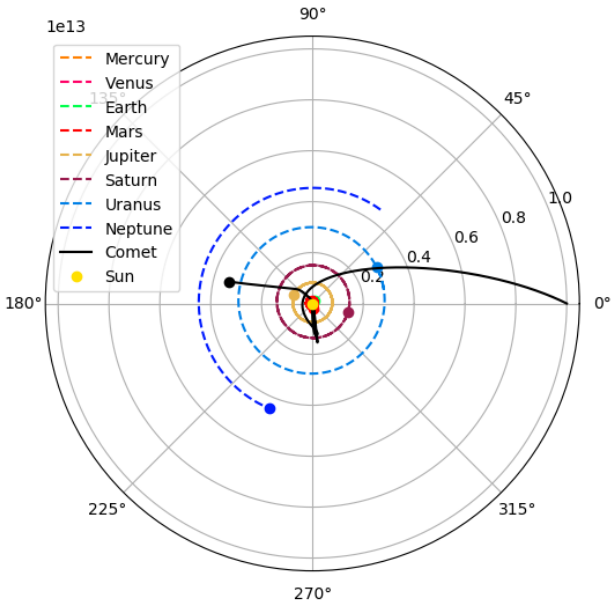
```
"Outside": [8*10**22, 10**13, 0, -2000, 1000]
```



(a) Координата $r(t)$, а также координаты планет $r_i(t)$



(b) Скорости $V_r(t), V_\theta(t)$



(c) Траектория

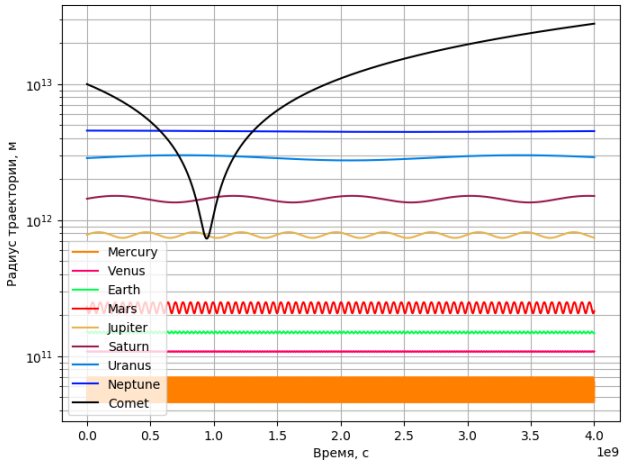
Рис. 4: Комета прилетает извне Солнечной системы

Как можно видеть (см. рис. 4), комета была "поймана"Юпитером, а затем отправлена за пределы системы.

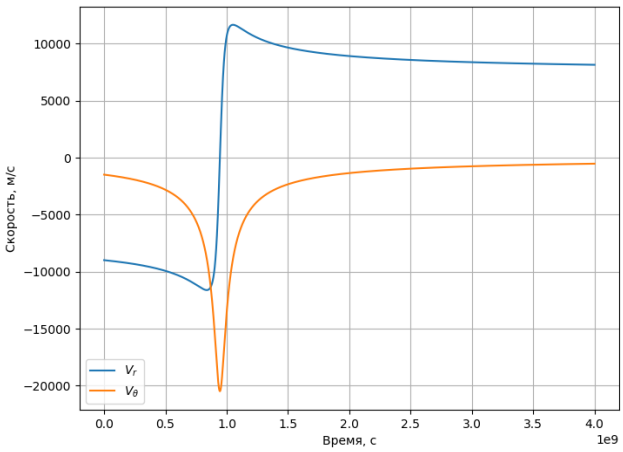
Комета прилетает извне Солнечной системы с малым воздействием

Похожий сценарий может иметь совершенно другой исход. Например, планеты практически не подействуют на комету. Начальные параметры кометы:

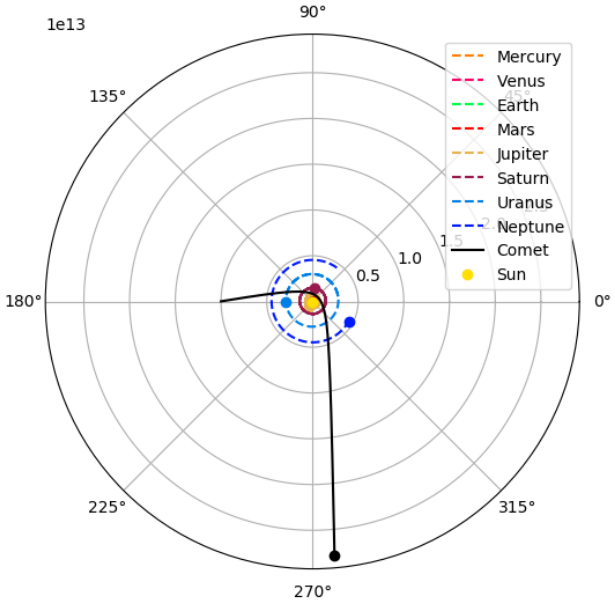
```
"RunBy":[8*10**22, 10**13, 3.14, -9000, -1500]
```



(a) Координата $r(t)$, а также координаты планет $r_i(t)$



(b) Скорости $V_r(t), V_\theta(t)$



(c) Траектория

Рис. 5: Комета прилетает извне Солнечной системы, и практически не взаимодуйствует с планетами

Здесь (см. рис. 5) можно наблюдать близкую к гиперболической траекторию кометы - двигаясь по такой траектории тело сможет удалиться на бесконечность.

3.2 Оценка точности интегрирования

4 Выводы

В работе была имплементирована модель Солнечной систсемы, смоделировано движение кометы и воздействие на ее траекторию гравитации планет и Солнца.

При малоразличных начальных параметрах можно получить совершенно разыне картины траекторий, т.е. система является хаотичной.

5 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):
        ''' Класс описывающий Солнце
        '''
        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):
        ''' Класс для описания движения кометы в поле
            тяготения тел Солнечной системы
        '''
        self.name = name # Название
        self.mass = mass # Масса, кг
        self.start_r = start_r  # Начальный радиус, м
        self.start_theta = start_theta  # Начальный угол, радианы
        self.r = start_r
        self.theta = start_theta
        self.v_r = v0_r  # Радиальная скорость
        self.v_theta = v0_theta # Угловая скорость
        # Пересчет в декартовы координаты, тк.. вычисления в них проще
        self.x, self.y = self.ccptd(start_r, start_theta)
        self.v_x, self.v_y = self.csptd(v0_r, v0_theta, start_theta)

    def ccptd(self, r, theta):
        ''' Конвертация координат из полярных в декартовы
        '''
```

```
46          x = r*np.cos(theta)
47          y = r*np.sin(theta)
48          return x, y
49
50      def csptd(self, v_r, v_theta, theta):
51          ''' Конвертация скоростей из полярных в декартовы
52          '''
53          v_x = v_r*np.cos(theta) - v_theta*np.sin(theta)
54          v_y = v_r*np.sin(theta) + v_theta*np.cos(theta)
55          return v_x, v_y
56
57      def ccdtp(self, x, y):
58          ''' Конвертация координат из декартовых в полярные
59          '''
60          r = np.sqrt(x**2 + y**2)
61          theta = np.arctan2(y, x)
62          return r, theta
63
64      def csdtp(self, x, y, v_x, v_y):
65          ''' Конвертация скоростей из декартовых в полярные
66          '''
67          v_r = (x*v_x + y*v_y)/np.sqrt(x**2 + y**2)
68          v_theta = (x*v_y - y*v_x)/np.sqrt(x**2 + y**2)
69          return v_r, v_theta
70
71      def calculate_distance_to_body(self, body):
72          ''' Расчет расстояния до тела
73          '''
74          distance = np.sqrt( body.r**2 + self.r**2  - 2*body.r*self.r*np.cos(body.theta-self.theta))
75          return distance
76
77      def calculate_force_to_body(self, body):
78          ''' Расчет силы притяжения от одного тела
79          '''
80          distance = self.calculate_distance_to_body(body)
81          F = G*self.mass*body.mass/distance**2
82          body_x, body_y = self.ccptd(body.r, body.theta)
83          F_x = F*(body_x - self.x)/distance
84          F_y = F*(body_y - self.y)/distance
85          return F_x, F_y
86
87      def calculate_summary_acceleration(self, bodies):
88          ''' Расчет суммарного ускорения от нескольких тел
89          '''
90          a_x_list, a_y_list = [], []
91          for body in bodies:
92              F_x, F_y = self.calculate_force_to_body(body)
93              a_x_list.append(F_x/self.mass)
```

```
            a_y_list.append(F_y/self.mass)
        a_x = np.sum(a_x_list)
        a_y = np.sum(a_y_list)
        return a_x, a_y

    def model_func(self, t, data_vec):
        ''' Функция для решения путем интегрирования в solve_ivp.
            На вход принимает data_vec вектор, имеющий следующие составляющие
                data_vec = [x, y, vx, vy]
            Возвращает правые части системы уравнений
                r' = v
                v' = a = G sum(m_i/Ri^2)
            в виде вектора [vx, vy, ax, ay]
        '''
        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])

        # расчет положения тел в момент времени t
        for body in self.bodies:
            body.calculate_position_by_time(t)
        ax, ay = self.calculate_summary_acceleration(self.bodies)
        dvdt = [ax, ay]
        return np.hstack((vel, dvdt))

    def evaluate_model(self, t_end, bodies, t_eval=None, method='RK23'):
        ''' Рассчет модели, с заданым временем окончания, методом интегрирования,
            а также набором тел.
        '''
        self.bodies = bodies
        print(method)
        res = solve_ivp(self.model_func, t_end,
                        y0=[self.x, self.y, self.v_x, self.v_y],
                        t_eval=t_eval, method=method)
        return res


class CelestialBody():
    def __init__(self, name, mass, a, eccentricity,
                 offset, start_theta, plot_color):
        ''' Основной класс для описания орбит планет вокруг Солнца.
            Включает методы для рассчета движения во времени по заданной траектории.
        '''
        self.name = name # Название тела
        self.mass = mass # Масса, кг
```

```
142         self.a = a        # Большая полуось, м
143         self.ecc = eccentricity  # Эксцентриситет
144         self.angle_offset = np.pi*offset/180  # Долгота восходящего узла
145         self.start_theta = start_theta        # Начальное угловое положение
146         self.p = a*(1-self.ecc**2)  # Фокальный параметр
147         self.mu = G*M0              # Гравитационный параметр
148         self.color = plot_color     # Цвет графика
149 
150         self.const1 = self.a*(1-self.ecc**2)
151         self.const2 = np.sqrt(self.mu*self.a*(1-self.ecc**2))
152         self.get_r(self.start_theta) # Рассчет расстояния по начальному углу theta
153         self.theta = start_theta
154 
155         # Запись координат - используется для отрисовки движения планет
156         self.recorded_theta = [self.start_theta]
157         self.recorded_r = [self.r]
158 
159     def calculate_position_by_time(self, t, points=100):
160         ''' Рассчет положения тела на орбите в определенный момент времени
161         '''
162         dt = t/points
163         for i in np.linspace(0, t, points):
164             self.calculate_next_posistion(dt)
165         r, theta = self.recorded_r[-1], self.recorded_theta[-1]
166         self.reset()
167         self.r, self.theta = r, theta
168         return r, theta
169 
170     def get_r(self, theta):
171         ''' Рассчет радиуса положения тела в зависимости
172             от угла в полярных координатах
173         '''
174         self.r = self.const1/(1+self.ecc*np.cos(theta + self.angle_offset))
175         return self.r
176 
177     def get_d_theta(self, dt):
178         '''  Рассчет инкремента угла d_theta за определенный промежуток времени,
179              при текущем значении радиуса r
180         '''
181         d_theta = 1/self.r**2*self.const2*dt
182         return d_theta
183 
184     def calculate_next_posistion(self, dt):
185         '''  Рассчет нового положения через dt секунд от текущего.
186              Сохраняет значений угла и радиуса в список записанных значений.
187         '''
188         d_theta = self.get_d_theta(dt)
189         next_theta = self.recorded_theta[-1] + d_theta
```

```
190         next_r = self.get_r(next_theta)
191         self.recorded_theta.append(next_theta)
192         self.recorded_r.append(self.r)
193         return next_r, next_theta
194 
195     def reset(self):
196         ''' Сброс записаных значений углов и радиусов в начальное положение.
197         '''
198         self.theta = self.start_theta
199         self.get_r(self.start_theta)
200         self.recorded_theta = [self.start_theta]
201         self.recorded_r = [self.r]
202 
203 TheSun = Sun()
204 
205 Mercury = CelestialBody("Mercury", 3.33022*10**23, 57909227000, 0.20563593, 48.33167, -0.2*np.pi,
        [1,0.5,0])
206 Venus = CelestialBody("Venus", 4.8675*10**24, 108208930000, 0.0068, 76.67069, 0.1*np.pi, [1,0,0.4])
207 Earth = CelestialBody("Earth", 5.9726*10**24, 149598261000, 0.01671123, 348.73936, 0.8*np.pi, [0,1,0.3])
208 Mars = CelestialBody("Mars", 6.4171*10**23, 2.2794382*10**8*1000, 0.0933941, 49.57854, 1.15*np.pi,
        [1,0,0])
209 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])
210 Saturn = CelestialBody("Saturn", 5.6846*10**26, 1429394069000, 0.055723219, 113.642, -0.08*np.pi, [0.6, 0.1,
         0.3])
211 Uranus = CelestialBody("Uranus", 8.6813*10**25, 2876679082000, 0.044, 73.9898, 0.06*np.pi, [0,0.5,0.9])
212 Neptune = CelestialBody("Neptune", 1.0243*10**26, 4503443661000, 0.011214269, 131.794, 0.3*np.pi,
        [0,0.1,1])
213 Planets = [Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune]
214 System = [TheSun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune]
```

Листинг 2: SolarSystem.py

```
1 
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from SolarSystem import System, Planets, Comet
5  from configs import configs
6 
7  use_config = "RunBy"
8  if use_config:
9      cfg = configs[use_config]
10     TheComet = Comet("Comet", cfg[0], cfg[1], cfg[2], cfg[3], cfg[4])
11 else:
12     TheComet = Comet("Comet",
13                      8*10**22,   # Масса, кг
14                      1.5*10**12,  # Начальный радиус, м
15                      0,  # Начальный угол, радианы
16                      0,          # Начальная радиальная скорость, м/с
```

```
                    10000       # Начальная угловая скорость, мс/
                    )


### Параметры симуляции ###

# Метод интегрирования в функцие np.solve_ivp()
integration_method = 'Radau'
# Время моделирования в секундах
t_end = 40*10**8
# t_end = 3.154*10**7  # Земной год
# Шаг времени в секундах
dt = 2*10**4
# Временные отсчеты в которых посчитать значение координат
t = np.arange(0, t_end, dt)

# Моделируем движение кометы в Солнечной системе
# res = TheComet.evaluate_model([0, t_end], [System[0]], t_eval=t, method=integration_method)
res = TheComet.evaluate_model([0, t_end], System, t_eval=t, method=integration_method)
res_time = res.t
comet_coords = res.y[0:2]
comet_vels = res.y[2:]
# Конвертируем координаты в полярные для дальнейшего построения графиков
comet_r, comet_theta = TheComet.ccdtp(comet_coords[0], comet_coords[1])
comet_vr, comet_vtheta = TheComet.csdtp(comet_coords[0], comet_coords[1], comet_vels[0], comet_vels
    [1])

### Далее код отвечает за отрисовку ###
fig, ax = plt.subplots(figsize=(8, 6), dpi=100, subplot_kw={'projection': 'polar'})
fig2, ax2 = plt.subplots(figsize=(8, 6), dpi=100)
fig3, ax3 = plt.subplots(figsize=(8, 6), dpi=100)

# 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)

    ax2.plot(t, Planet.recorded_r[0:-1], label=Planet.name, 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")
ax.legend()

ax2.plot(res_time, comet_r, color='k', label="Comet")
ax2.grid(which='both')
ax2.legend(loc=3)
ax2.set_yscale('log')
ax2.set_xlabel("Время, с")
ax2.set_ylabel("Радиус траектории, м")

ax3.plot(res_time, comet_vr, label='$V_r$')
ax3.plot(res_time, comet_vtheta, label='$V_{\\theta}$')
ax3.set_xlabel("Время, с")
ax3.set_ylabel("Скорость, м/с")
ax3.grid(which='both')
ax3.legend(loc=3)

if use_config:
    fig.savefig('imgs_8/trj{}.png'.format(use_config), bbox_inches='tight')
    fig2.savefig('imgs_8/r{}.png'.format(use_config), bbox_inches='tight')
    fig3.savefig('imgs_8/v{}.png'.format(use_config), bbox_inches='tight')
else:
    fig.savefig('imgs_8/trj_custom.png', bbox_inches='tight')
    fig2.savefig('imgs_8/r_custom.png', bbox_inches='tight')
    fig3.savefig('imgs_8/v_custom.png', bbox_inches='tight')
plt.show()
```

Листинг 3: simulation.py

```

configs = {
    "SunComet": [8*10**22, 1.5*10**12, 0, 0, 10000],
    "Inside": [8*10**22, 1.5*10**12, 0, 0, 10000],
    "Outside": [8*10**22, 10**13, 0, -2000, 1000],
    "RunBy":[8*10**22, 10**13, 3.14, -9000, -1500]
}
```

Листинг 4: configs.py