Жесткий астероид

Выполнили: Подулько В.А., Кудрявцев Н.А., Герасимов Г.И., Соколов Т.О., Молчанов И.В., Грасмик А.А.

Научный руководитель: Байгашов А.С.

Аннотация

В работе проведено исследование столкновений космических тел. Рассмотрен случай столкновения астероида с телом Солнечной системы. Проведено моделирование такого столкновения, созданы анимированные иллюстрации процесса выхода астероида на орбиту небесного тела и столкновения с ним.

Введение

Тема столкновения космических тел является актуальным вопросом современной астрофизики. Она играет важную роль в понимании процессов, происходящих как в открытом космосе, так и на отдельно взятых космических телах. Больше того, изучение процессов столкновений позволяет создавать модели событий, происходивших в далёком прошлом. Так, именно это позволил создать теорию о происхождении Луны в результате столкновения объекта массой 2/3 земных с поверхностью Земли.

Современные технологии позволяют во многом автоматизировать процесс моделирования, используя специализированные среды программирования и готовые библиотеки для них. В рамках данной работы была создана упрощенная модель столкновения астероида с планетой. С этой целью средствами языка Python была создана численная модель, описывающая законы всемирного тяготения применительно к небесным телам.

Постановка задачи

Для моделирования движения небесных тел запишем закон всемирного тяготения в дифференциальной форме:

$$\begin{cases} \frac{dx}{dt} = v_x \\ \frac{dv_x}{dt} = -\frac{GM}{(x^2+y^2)^{\frac{3}{2}}} \cdot x \\ \frac{dy}{dt} = v_y \\ \frac{dv_y}{dt} = -\frac{GM}{(x^2+y^2)^{\frac{3}{2}}} \cdot y \end{cases}$$

Для расчёта столкновений используем формулу для нецентральных столкновений:

$$v'_{2x} = \frac{v_2 \cos(\theta_2 - \varphi)(m_2 - m_1) + 2m_1 v_1 \cos(\theta_1 - \varphi)}{m_1 + m_2} \cos(\varphi) + \tag{13}$$
$$+ v_2 \sin(\theta_2 - \varphi) \cos(\varphi + \frac{\pi}{2})$$

$$v'_{2y} = \frac{v_2 \cos(\theta_2 - \varphi)(m_2 - m_1) + 2m_2 v_1 \cos(\theta_1 - \varphi)}{m_1 + m_2} \sin(\varphi) + \tag{14}$$
$$+ v_2 \sin(\theta_2 - \varphi) \sin(\varphi + \frac{\pi}{2})$$

Начальные условия

Массы моделируемых объектов:

Масса Солнца: 1,9891*10^{30}

Масса астероида: 2.22*10^{23}

Масса Земли:5.94*10^{25}

Начальные координаты:

Астероид: a.e.; 0; 0; 29000.

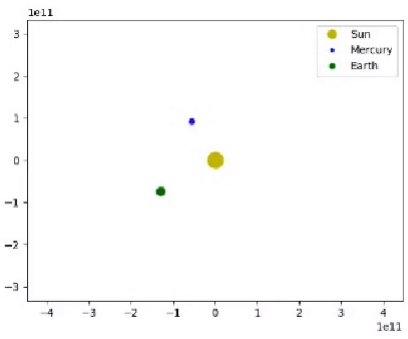
Земля: a.e.; 0; 0; 29000.

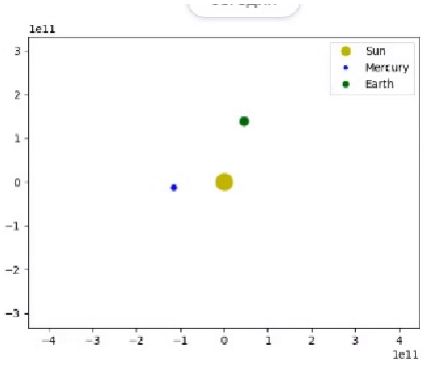
Солнце: 0;0; 0; 0.

Меркурий: a.e.; 2, 0, 0, 40000

Результаты моделирования

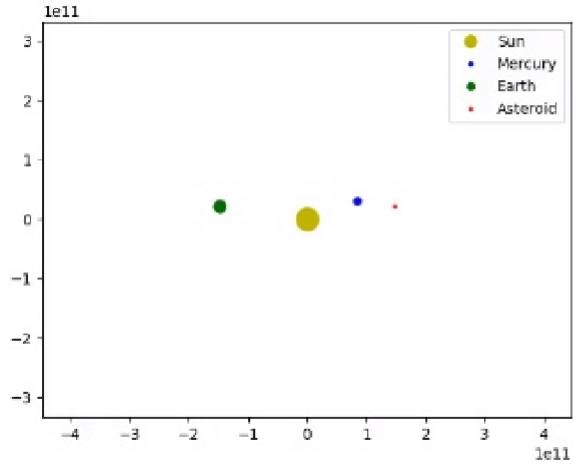
В результате проведённой работы была создана модель, рассчитывающая процесс столкновения различных тел Солнечной системы с астероидами.



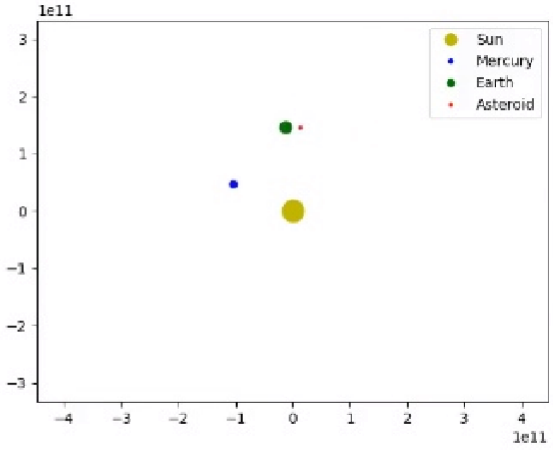


1.Движение Меркурия и Земли вокруг Солнца

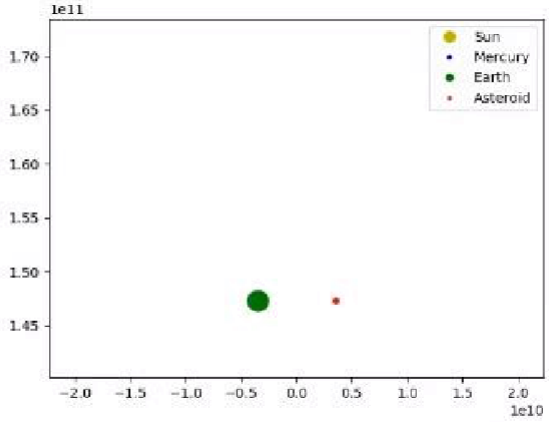
2. .Глобальный вид столкновения астероида с Землей, находящегося на ее орбите

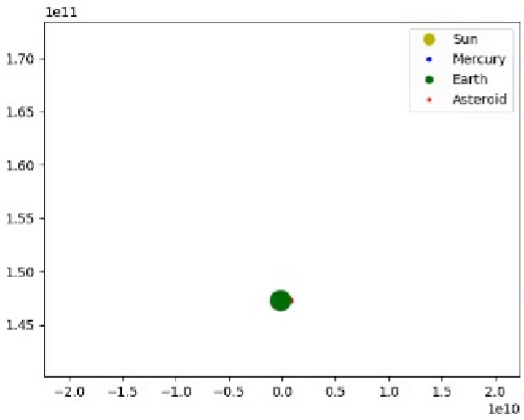


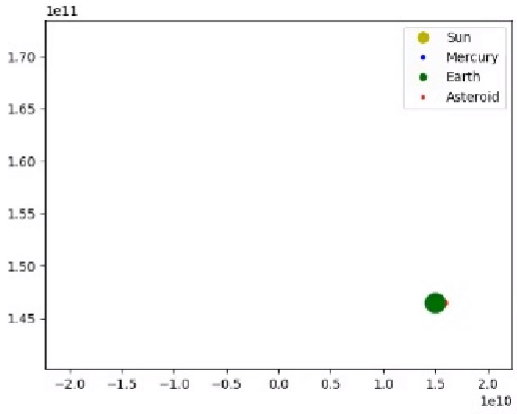
2.



3.Локальный вид столкновения Земли с астероидом





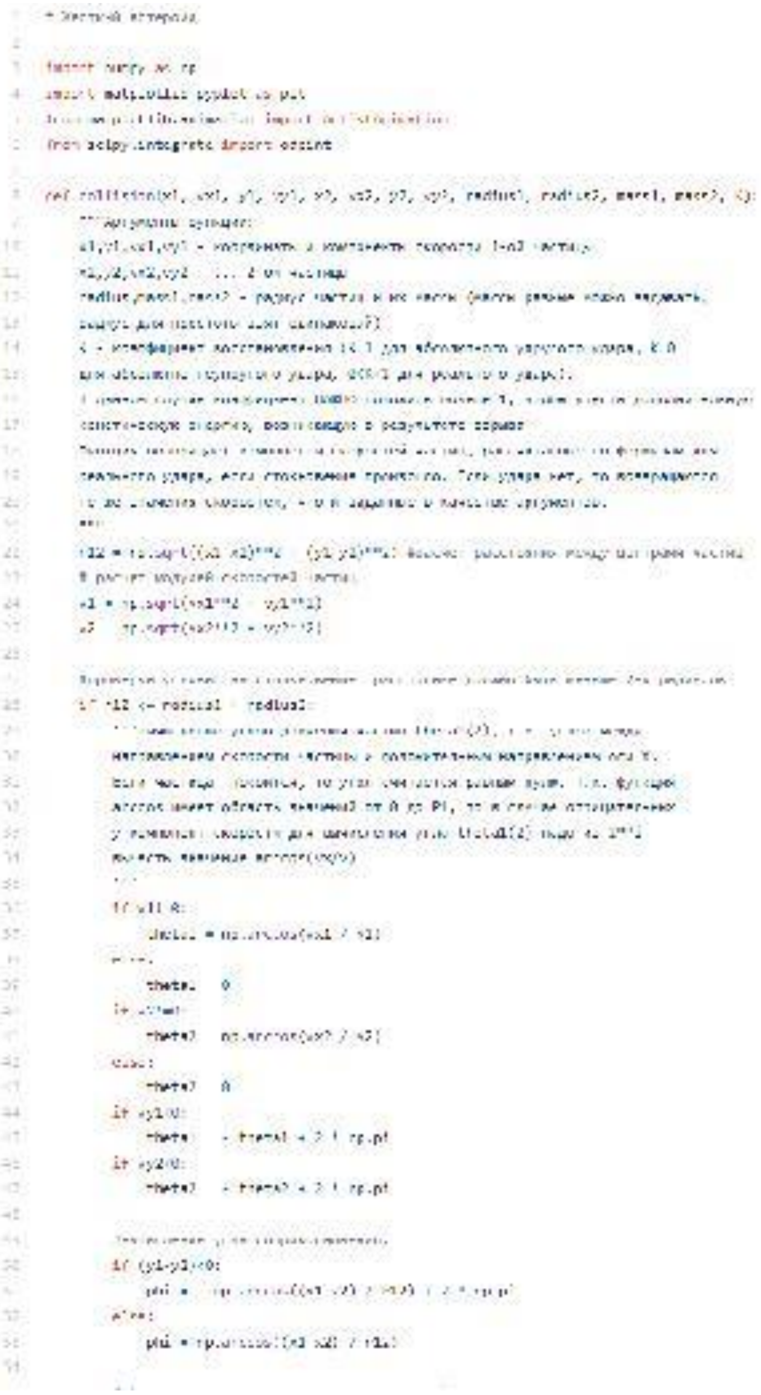


Как видно из полученных изображений, астероид, оказавшись на орбите одной из планет, сталкивается с ней. Процесс столкновений продемонстрирован в глобальном и локальном виде.

Заключение

В результате работы была получена корректно действующая модель столкновений тел Солнечной системы с малыми объектами. Дальнейшим развитием этой работы может стать полноценная модель столкновений целых систем или даже галактик, что может быть полезным при прогнозировании и наблюдениях.

Листинг кода



```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

```
        # Пересчет  x-компоненты скорости первой частицы
        VX1 = v1 * np.cos(theta1 - phi) * (mass1 - K * mass2) \
        * np.cos(phi) / (mass1 + mass2)\
        + ((1 + K) * mass2 * v2 * np.cos(theta2 - phi))\
        * np.cos(phi) / (mass1 + mass2)\
        + K * v1 * np.sin(theta1 - phi) * np.cos(phi + np.pi / 2)

        # Пересчет y-компоненты скорости первой частицы
        VY1 = v1 * np.cos(theta1 - phi) * (mass1 - K * mass2) \
        * np.sin(phi) / (mass1 + mass2) \
        + ((1 + K) * mass2 * v2 * np.cos(theta2 - phi)) \
        * np.sin(phi) / (mass1 + mass2) \
        + K * v1 * np.sin(theta1 - phi) * np.sin(phi + np.pi / 2)

        # Пересчет x-компоненты скорости второй частицы
        VX2 = v2 * np.cos(theta2 - phi) * (mass2 - K * mass1) \
        * np.cos(phi) / (mass1 + mass2)\
        + ((1 + K) * mass1 * v1 * np.cos(theta1 - phi)) \
        * np.cos(phi) / (mass1 + mass2)\
        + K * v2 * np.sin(theta2 - phi) * np.cos(phi + np.pi / 2)

        # Пересчет y-компоненты скорости второй частицы
        VY2 = v2 * np.cos(theta2 - phi) * (mass2 - K * mass1) \
        * np.sin(phi) / (mass1 + mass2) \
        + ((1 + K) * mass1 * v1 * np.cos(theta1 - phi)) \
        * np.sin(phi) / (mass1 + mass2)\
        + K * v2 * np.sin(theta2 - phi) * np.sin(phi + np.pi / 2)

    else:
        #если условие столкновнеия не выполнено, то скорости частиц не пересчитываются
        VX1, VY1, VX2, VY2 = vx1,vy1,vx2,vy2

    return VX1, VY1, VX2, VY2

# Определяем переменную величину
T = 365 * 24 * 60 * 60 # Общее время анимации
years = 0.03
n = 100
dT = T / n # Время одного шага итерации
# radius1 = 6400000
# radius2 = 6400000
radius1 = 130*6400000
radius2 = 30*6400000

ae = 149*10**9

tau = np.linspace(0, years*T, n)

# Определяем функцию для системы диф. уравнений
def move_func(s, t):
    x, v_x, y, v_y = s

    dxdt = v_x
    dv_xdt = - G * mc * (x - xc) / ((x - xc)**2 + (y - yc)**2)**1.5
    dydt = v_y
    dv_ydt = - G * mc * (y - yc) / ((x - xc)**2 + (y - yc)**2)**1.5

```

```
    return dxdt, dv_xdt, dydt, dv_ydt

# Определяем начальные значения и параметры, входящие в систему диф. уравнений
N = 3
p = np.zeros((N,4)) # Массив для координат и скоростей всех точек

# Массивы для записи итоговых координат на каждой итерации для итоговой анимации
x = np.zeros((N,n))
y = np.zeros((N,n))

# # --------- Начальные параметры для глобальной области ---------
# p[0,0], p[0,1], p[0,2], p[0,3] = ae/3, 0, 0, 29000
# p[1,0], p[1,1], p[1,2], p[1,3] = -ae/3, 0, 0, 29000
# p[2,0], p[2,1], p[2,2], p[2,3] = ae/5, 0, 0, 40000

# --------- Начальные параметры для локальной области ---------
p[0,0], p[0,1], p[0,2], p[0,3] = 3543373941.396241, 147325824201.57532, -29320.37157763709, 376.20900700518484
p[1,0], p[1,1], p[1,2], p[1,3] = -3543373941.396241, 147325824201.57532, 29320.37157763709, 376.20900700518484
p[2,0], p[2,1], p[2,2], p[2,3] = ae/2, 0, 0, 40000

x[0,0], y[0,0] = p[0,0], p[0,1]
x[1,0], y[1,0] = p[1,0], p[1,1]
x[2,0], y[2,0] = p[2,0], p[2,1]

mc = 1.9 * 10**(30)
xc = 0
yc = 0

mass1 = 0.94 * 10**24
mass2 = 5.94 * 10**25
K = 0

G = 6.67 * 10**(-11)


# Решение задачи и проверка условий столковения
for k in range(n-1):  # Цикл перебора шагов временеи анимации
    t = [tau[k], tau[k+1]]

    for m in range(N):  # Цикл перебора частиц для столкновений со стенками
        s0 = p[m,0], p[m,2], p[m,1], p[m,3]
        sol = odeint(move_func, s0, t)

        # Перезаписываем положения частиц
        p[m,0] = sol[1,0]
        p[m,2] = sol[1,1]
        p[m,1] = sol[1,2]
```

```
        p[m,0] = sol[1,0]
        p[m,2] = sol[1,1]
        p[m,1] = sol[1,2]
        p[m,3] = sol[1,3]

        # Заноим новые положения в итоговый массив для анимации
        x[m,k+1], y[m,k+1] = p[m,0], p[m,1]

    # Циклы перебора частиц для столкновений друг с другом
    for i in range(N): # Базовая частица
        x1, y1, vx1, vy1 = p[i,0], p[i,1], p[i,2], p[i,3] # Запись текущих координат базовой частицы
        x10, y10 = x[i,k], y[i,k] # Запись координат предыдущего шага базовой частицы

        for j in range(i+1,N): # Запись текущих координат остальных частиц
            x2, y2, vx2, vy2 = p[j,0], p[j,1], p[j,2], p[j,3] # Запись текущих
            x20, y20 = x[j,k], y[j,k] # Запись координат предыдущего шага

            # Проверка условий столкновения
            r1 = np.sqrt((x1-x2)**2 + (y1-y2)**2)
            r0 = np.sqrt((x10-x20)**2 + (y10-y20)**2)

            if  r1 <= radius1 + radius2 and r0 > radius1 + radius2:
                res = collision(x1, vx1, y1, vy1, x2, vx2, y2, vy2, radius1, radius2, mass1, mass2, K)

                # Перезаписывание условий, в случае столкновения
                p[i,2], p[i,3] = res[0], res[1]
                p[j,2], p[j,3] = res[2], res[3]

# # Определение координат перед столкновением для первого объекта
# print(p[1,0])
# print(p[1,1])
# print(p[1,2])
# print(p[1,3])

# # Определение координат перед столкновением для второго объекта
# print(p[0,0])
# print(p[0,1])
# print(p[0,2])
# print(p[0,3])

# Графический вывод
fig = plt.figure()

bodys = []
plt.plot([0], [0], 'o', color='y', ms=8,  label='Sun')
plt.plot([0], [0], 'o',color='b', ms=3, label='Mercury')
plt.plot([0], [0], 'o',color='g', ms=5, label='Earth')
plt.plot([0], [0], 'o',color='r', ms=2, label='Asteroid')

for i in range(n):
    bodyc, = plt.plot([0], [0], 'o', color='y', ms=15)
    body1, = plt.plot(x[0,i], y[0,i], 'o', color='r', ms=4)
    body2, = plt.plot(x[1,i], y[1,i], 'o', color='g', ms=15)
    body3, = plt.plot(x[2,i], y[2,i], 'o', color='b', ms=8)
    bodys.append([bodyc, body1, body2, body3])

ani = ArtistAnimation(fig, bodys, interval=25)

plt.axis('equal')

# # --------- Прежделы глобальной области ---------
# plt.xlim(-ae, ae)
# plt.ylim(-ae, ae)
plt.legend()

# --------- Прежделы локальной области ---------
plt.xlim(-0.15*ae, 0.15*ae)
plt.ylim(1.05*ae, 1.055*ae)

ani.save('results/asteroid_lokal.gif')
# plt.show()
```