Взрыв сверхновой

Выполнили: А.А. Обыночный, А.Д. Ле

Научный руководитель: А.С. Байгашов

Аннотация

В работе представлены результаты моделирования процесса взрыва сверхновой звезды. В частности, рассмотрено падение оболочки звезды на ее ядро и последующее обратное движение этой оболочки вследствие взрыва. Рассмотрены два гипотетических сценария взрыва звёзд, размерами и массой соответствующих Солнцу и Сириусу.

Введение

Взрыв сверхновой – редкое, но впечатляющее событие. Процесс взрыва сверхновой чрезвычайно сложен для моделирования, однако отдельные его элементы вполне поддаются описанию с точки зрения ньютоновской механики. В частности, сам процесс взрыва можно представить как последовательное схлопывание (коллапс) оболочки звезды на её ядро, а затем – разлёт остатков оболочки в сторону от ядра в результате взрыва. Подобный процесс можно смоделировать, выделив в структуре рассматриваемой звезды две области – статичное ядро и окружающую его оболочку, состоящую из счётного числа элементов. Тогда, применяя к элементам оболочки законы механики, можно смоделировать её движение при взрыве. Именно это и является целью настоящей работы. Для её достижения требуется создать набор скриптов, позволяющих численно решать необходимые уравнения, а также анимировать результат решения для большей наглядности. В качестве инструмента для решения этих задач был выбран язык программирования Python 3, вместе с открытыми библиотеками позволяющий выполнять весь комплекс поставленных задач.

Постановка задачи

Для моделирования этого события необходимо решить дифференциальное уравнение, базирующееся на втором законе Ньютона:

$$\frac{dx}{dt} = v_x$$

$$\frac{dv_x}{dt} = -\frac{g * x}{R * \sqrt{x^2 + y^2}}$$

$$\frac{dy}{dt} = v_y$$

$$\frac{dv_y}{dt} = -\frac{g * y}{R * \sqrt{x^2 + y^2}}$$

Таким образом, задача сводится к разбиению оболочки звезды на набор элементов, к каждому из которых необходимо применить указанные уравнения.

Начальные условия

Для решения поставленной задачи необходимо определить следующие начальные условия:

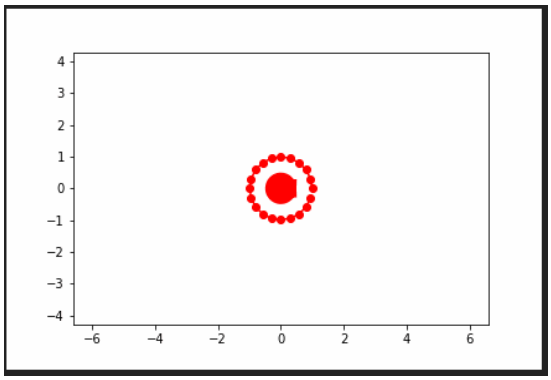
- Радиус Солнца = 696340 км
- Радиус ядра Солнца = 173 000 км
- Масса Солнца = 1.989*10**30 кг

Сириус – ближайшая к нам звезда после Солнца, поэтому именно ее взрыва мы должны больше всего опасаться. Рассмотрим начальные данные для звезды Сириус.

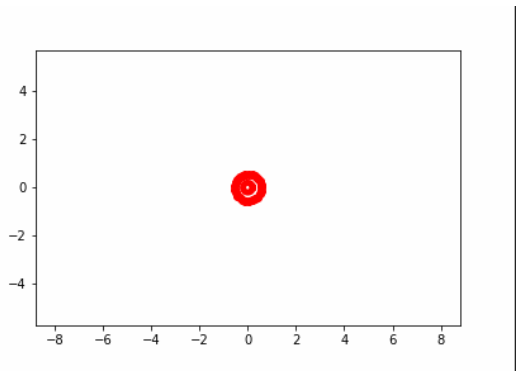
- Радиус оболочки Сириуса примерно равен 1.7 солнечным радиусам
- Радиус ядра Сириуса примерно равен 1.35 солнечным радиусам
- Масса Сириуса равна 2 солнечным массам

Результаты моделирования

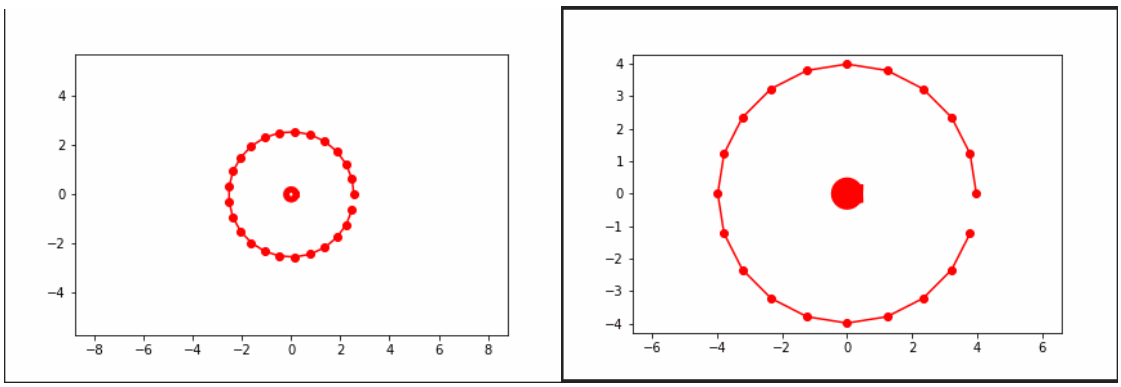
В результате численного моделирования была получена анимация движения набора элементов, симулирующих оболочку взрывающейся звезды. Были рассмотрены два варианта – с Солнцем (рис. 1) и Сириусом (рис. 2).



Солнце в начале

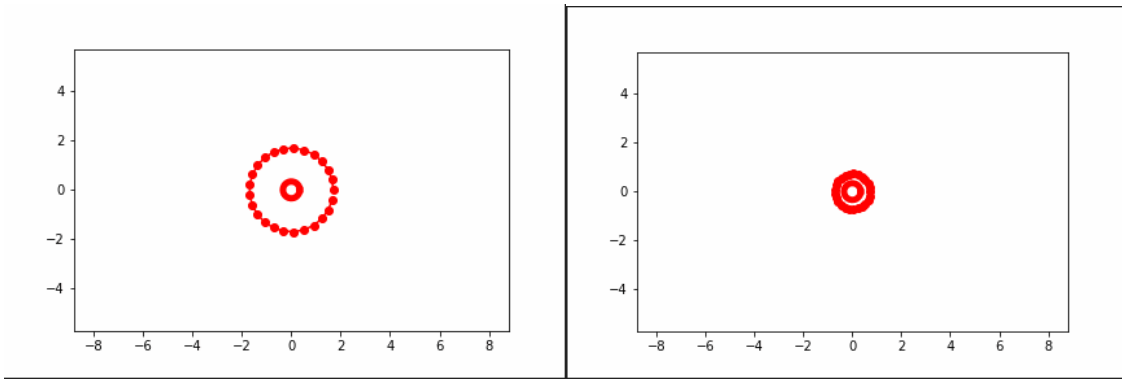


Момент перед взрывом

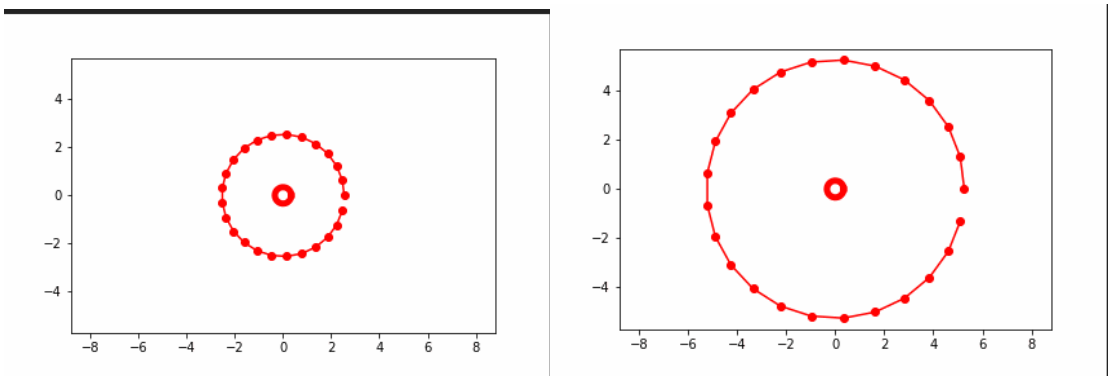


Середина взрыва Конец взрыва

Рис. 1. Взрыв Солнца



Сириус в начале Момент перед взрывом



Середина взрыва Конец взрыва

Рис. 2. Взрыв Сириуса

Заключение

Проведённое исследование продемонстрировало принципиальную возможность моделирования отдельных аспектов даже таких сложных процессов, как взрыв сверхновых. Использование функционала Python и его открытых библиотек позволило не только численно решить систему дифференциальных уравнений, но и наглядно проиллюстрировать результат решения с помощью анимации движения частиц оболочки.

Как показал опыт выполнения этой работы, готовые скрипты легко модифицируются под различные задачи, например, моделирование аналогичного процесса с иными начальными условиями. В качестве дальнейшего развития нашей работы, для большего реализма, можно рассмотреть взрыв звезды в окружении других космических объектов для того, чтобы понять, как оболочка взорвавшейся звезды будет взаимодействовать с другими телами.

Листинг кода для решения задачи:

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

M = 1 # Масса ядра в Солнечных массах
R = 0.2  # Радиус ядра в Солнечных радиусах
G = 6.67430 * 10**(-11)
m_sun = 1.989*10**30
R_sun = 696340*10**3
R_shel = 1 # Радиус оболочки в солнечных радиусах

T = 20000 # Время анимации в секундах (соответствует примерно реальному
времени коллапса оболочки)
n = 2000  # Число шагов / кадров

tau = np.linspace(0,T,n) # Массив для одного временного шага

N = 25 # Число частиц оболочки
p = np.zeros((N,4)) # Массив для координат и скоростей всех точек

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

for i in range(N): # Цикл для расстановки точек по кругу
    p[i,0], p[i,1], p[i,2], p[i,3] = R_shel*np.cos(2*np.pi*i/N), 0,
R_shel*np.sin(2*np.pi*i/N), 0
    x[i,0], y[i,0] = p[i,0], p[i,2]

def move_func(s, t):
    x, v_x, y, v_y = s

    # Система диф. уравнений на базе второго закона Ньютона
    g = G * m_sun * M / (R_sun**2*(x**2 + y**2))

    dxdt = v_x
    dv_xdt = - g * x / (R_sun * np.sqrt(x**2 + y**2))

    dydt = v_y
    dv_ydt = - g * y / (R_sun * np.sqrt(x**2 + y**2))

    return dxdt, dv_xdt, dydt, dv_ydt

def collision(x1, vx1, y1, vy1, x2, vx2, y2, vy2, radius1, radius2, mass1,
mass2, K):
    """Аргументы функции:
    x1,y1,vx1,vy1 - координаты и компоненты скорости 1-ой частицы
    x2,y2,vx2,vy2 - координаты и компоненты скорости 2-ой частицы
```

```
    radius, mass1, mass2 - радиус частиц и их массы (массы разные можно
задавать, радиус для простоты взят одинаковый)
    K - коэффициент восстановления (K=1 для абсолютного упругого удара, K=0
    для абсолютно неупругого удара, 0<K<1 для реального удара).
    В данном случае коэффициент ВАЖНО положить больше 1, чтобы учесть
дополнительную кинетическую энергию, возникающую в результате взрыва.
    Функция возвращает компоненты скоростей частиц, рассчитанные по формулам
для реального удара, если столкновение произошло. Если удара нет, то
возвращаются те же значения скоростей, что и заданные в качестве аргументов.
    """
    r12 = np.sqrt((x1-x2)**2 + (y1-y2)**2) #расчет расстояния между центрами
частиц
    # расчет модулей скоростей частиц
    v1 = np.sqrt(vx1**2 + vy1**2)
    v2 = np.sqrt(vx2**2 + vy2**2)

    #проверка условия на столкновение: расстояние должно быть меньше 2-х
радиусов
    if r12 <= radius1 + radius2:
        '''вычисление углов движения частиц theta1(2), т.е. углов между
        направлением скорости частицы и положительным направлением оси X.
        Если частица  покоится, то угол считается равным нулю. Т.к. функция
        arccos имеет область значений от 0 до pi, то в случае отрицательных
        y-компонент скорости для вычисления угла theta1(2) надо из 2*pi
        вычесть значение arccos(vx/v)
        '''
        if v1!=0:
            theta1 = np.arccos(vx1 / v1)
        else:
            theta1 = 0
        if v2!=0:
            theta2 = np.arccos(vx2 / v2)
        else:
            theta2 = 0
        if vy1<0:
            theta1 = - theta1 + 2 * np.pi
        if vy2<0:
            theta2 = - theta2 + 2 * np.pi

        #вычисление угла соприкосновения.
        if (y1-y2)<0:
            phi = - np.arccos((x1-x2) / r12) + 2 * np.pi
        else:
            phi = np.arccos((x1-x2) / r12)

        # 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 столкновения не выполнено, то скорости частиц не
пересчитываются
        VX1, VY1, VX2, VY2 = vx1,vy1,vx2,vy2

    return VX1, VY1, VX2, VY2

for k in range(n-1):
    t=[tau[k],tau[k+1]]
    for m in range(N):
        s0 = p[m,0], p[m,1], p[m,2], p[m,3]
        sol = odeint(move_func, s0, t)

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

        x[m,k+1], y[m,k+1] = p[m,0], p[m,2]

        res = collision(p[m,0],p[m,1],p[m,2],p[m,3], 0, 0, 0, 0, 0.01, 0.5,
0, 1, 2)

        p[m,1], p[m,3] = res[0], res[1]

# Графика
def circle(radius, x0, y0): #Функция, генерирующая координаты ядра звезды
    phi = np.linspace(0, 2*np.pi, 100)
    x = x0 + radius * np.cos(phi)
    y = y0 + radius * np.sin(phi)
    return x, y

fig, ax = plt.subplots() #Создание пространства для анимации
nucleus, = plt.plot([], [],  color='r', lw=5) #Анимируемый объект
particles, = plt.plot([], [], marker='o', color='r', label='circle')

# Определение области
edge = 8
ax.set_xlim(-edge, edge)
ax.set_ylim(-edge, edge)

def animate(i): #Функция подстановки координат в анимируемый объект
    nucleus.set_data(circle(radius=R, x0=0, y0=0)) # Ядро звезды
    particles.set_data(x[:,i], y[:,i])

ani = animation.FuncAnimation(fig,
                              animate,
                              frames=1000,
                              interval=0.1)

plt.axis('equal')
ani.save('Supernova.gif')

plt.show()
```