БАЛТИЙСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Астрономическое сообщество БФУ им. И. Канта

Коллапс сверхновой

Выполнил:

Вайда Максим

Научный руководитель:

А.С. Байгашов

Калининград, 2020

**Содержание**

|  | стр. |
| --- | --- |
| Введение…………………………………………………………………… | 3 |
| Постановка дифференциальной задачи…………………………………. | 4 |
| Начальные условия………………………………………………………... | 6 |
| Численное решение………………………………………………………... | 6 |
| Заключение ………………………………………………………………... | 7 |
| Приложение |  |

**Аннотация**

Данная работа посвящена исследованию движения оболочки сверхновой звезды, а также падению поверхностных слоев звезды на ядро в результате гравитационного притяжения. Приводятся результаты моделирования траектории движения оболочки в виде частиц, расположенных вокруг центра на большом расстоянии. Построена зависимость времени от координат точек траектории.

**Введение**

Одним из важных и актуальных вопросов современной астрофизики является изучение сверхновых звезд и их вспышек, которые играют важную роль в эволюции Вселенной. Они извергают колоссальное количество вещества в виде газопылевых облаков далеко в межзвездное пространство, ускоряют космический круговорот рождения и распада материи, сжимают межзвездный газ в гигантские нити и дают импульс к образованию новых планет и светил.

В рамках настоящей работы рассматривается ситуация движения оболочки сверхновой в результате вспышки звезды. Подробные наблюдения с вычислениями описываются в работах группы ученных из Калифорнийского технического института, возглавляемой Фионой Харрисон, после астрономической сенсации в 1987 году – открытия сверхновой под кодом SN1987A. Наиболее корректный имитирующий процесс коллапса сверхновой звезды рассматривается в фильме «NOVAE» французского режиссера, художника и дизайнера Томаса Ванса.

В данном исследовании для моделирования коллапса используются знания, полученные из лекционного материала математического моделирования, из книги Дэвида Бранча и J. Craig Wheeler «Supernova Explosions» и открытых Интернет-ресурсов.

Таким образом, **целью** работы является применение изученных методов численного моделирования для решения задачи о движении оболочки сверхновой звезды. Для достижения поставленной цели необходимо решить следующие задачи:

* определить системы дифференциальных уравнений, описывающих динамику частиц оболочки сверхновой звезды;
* определить начальные условия для решения системы дифференциальных уравнений;
* исследовать явление коллапса сверхновой звезды;
* написать алгоритм решения поставленной задачи.

**Постановка дифференциальной задачи**

В процессе вспышки сверхновой звезды происходит падение поверхностных слоёв звезды на ядро (центральную массивную область) в результате гравитационного притяжения. После этого происходит “отскок” вещества (это выглядит как резкое увеличение звезды в размерах). В данной работе реализовывается модель расчета движения оболочки: оболочка представляет собой совокупность частиц, которые падают на центральную часть звезды, ядро, под действием сил тяготения. Отскок происходит подобно упругому удару (рис. 1).

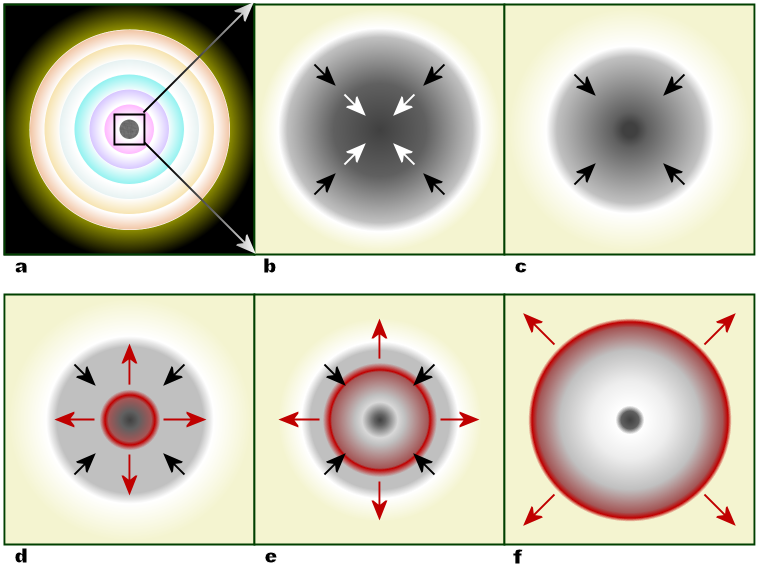


Рис. 1. Схема коллапса сверхновой

Ориентируясь на цель и формулировку задачи нашего исследования, мы составили систему дифференциальных уравнений, которая выражает изменение положения частиц оболочки относительно ядра со временем:

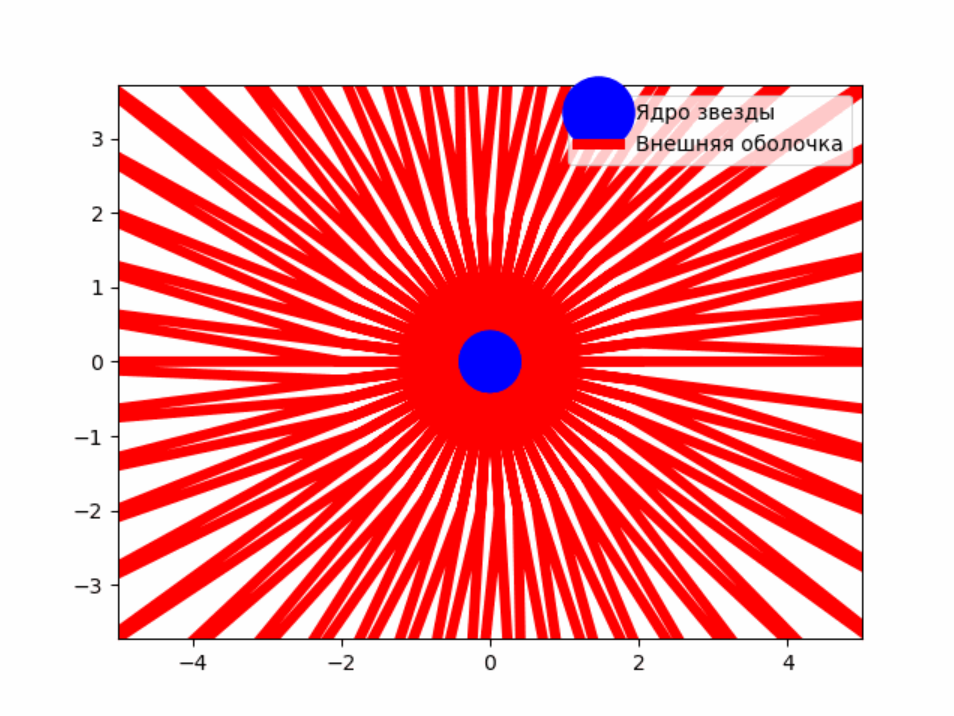
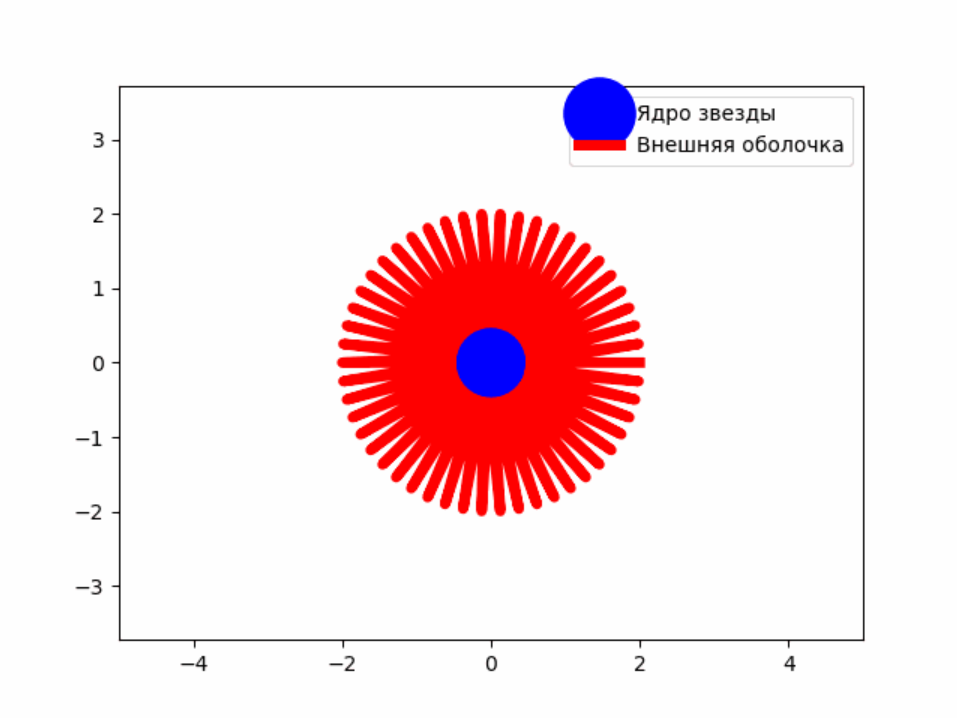
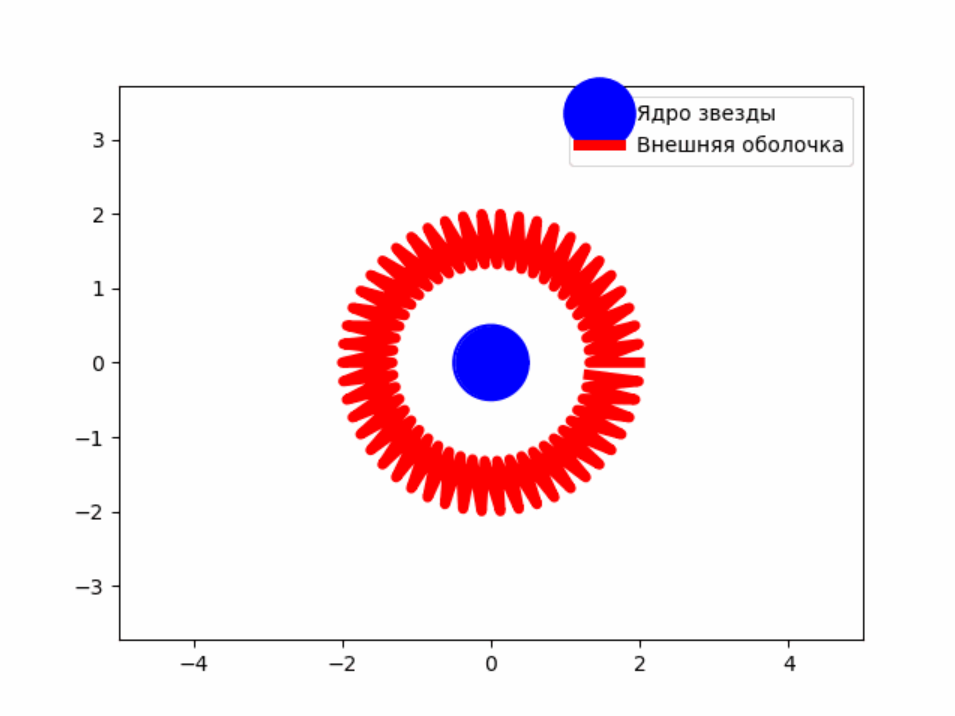
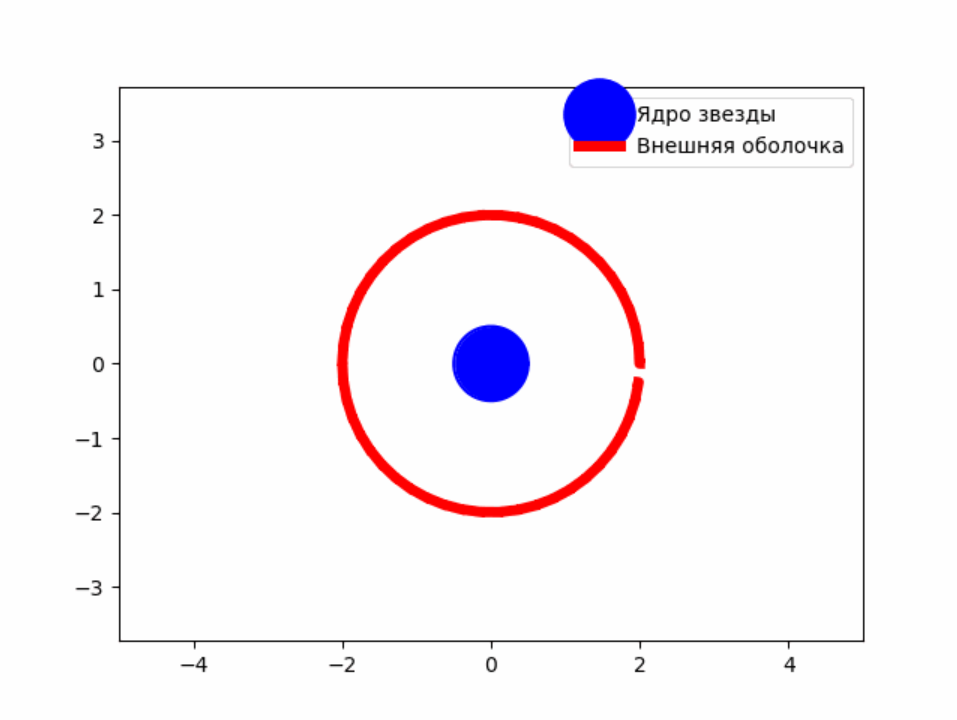
Изменяющейся величиной в системе является положение частиц в системе координат относительно ядра, а переменной – время.

**Начальные условия**

Для решения системы дифференциальных уравнений и построения математических моделей, определим начальные параметры:

* размер ядра звезды – 1-1,5 солнечных масс;
* радиус ядра – 10-15 км (фактически – нейтронная звезда);
* число частиц оболочки – 50.

**Численное решение**

Результатом исследования движения оболочки сверхновой звезды во время коллапса стала модель оболочки из нескольких частиц (рис. 2-5). 

**Заключение**

Выполненное исследование продемонстрировало возможность применения средств численного моделирования для расчёта процесса коллапса сверхновой звезды и падения её оболочки на ядро. Посредством проведенного исследования мы:

* определили системы дифференциальных уравнений, описывающих движение оболочки относительно ядра сверхновой звезды во время коллапса;
* определили начальные условия для решения системы дифференциальных уравнений;
* написали алгоритм решения поставленной задачи.

В качестве дальнейшего развития настоящей работы можно рассмотреть более подробное моделирования процесса коллапса, включая систему из большего числа точек, в том числе, расположенных в несколько слоёв, отражающих пространственную структуру оболочки взрывающейся звезды. Кроме того, варьируя численные значения параметров звезды, можно рассмотреть несколько случаев коллапса, протекающего в различных условиях.

**Список литературы и интернет-ресурсов**

1. Лутц М. Изучаем Python 4-е издание. – Пер. с англ.
2. Supernova Explosions – David Branch
3. https://www.youtube.com/watch?v=jfN3lXblv9Y&t=114s
4. https://ru.wikipedia.org/wiki/Сверхновая\_звезда
5. http://galspace.spb.ru/indvop.file/72.html
6. https://www.youtube.com/watch?v=HE-A6bqnZZc

**Приложение**

Листинг кода решения задачи:

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **from** matplotlib.animation **import** FuncAnimation
4. **import** matplotlib.animation as animation
5. **from** scipy.integrate **import** odeint
7. M **=** 1 # Масса ядра в Солнечных массах
8. R **=** 0.5  # Радиус ядра в Солнечных радиусах
9. G **=** 6.67430 **\*** 10**\*\***(**-**11)
10. m\_sun **=** 1.989**\***10**\*\***30
11. R\_sun **=** 696340**\***10**\*\***3
12. R\_shel **=** 2 # Радиус оболочки в солнечных радиусах
14. T **=** 20000 # Время анимации в секундах (соответствует примерно реальному времени коллапса оболочки)
15. n **=** 500  # Число шагов / кадров
17. tau **=** np.linspace(0,T,n) # Массив для одного временного шага
19. N **=** 50 # Число частиц оболочки
20. p **=** np.zeros((N,4)) # Массив для координат и скоростей всех точек
22. # Массивы для записи координат для итоговой анимации
23. x **=** np.zeros((N,n))
24. y **=** np.zeros((N,n))
26. **for** i **in** range(N): # Цикл для расстоновки точек по кругу
27. 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
28. x[i,0], y[i,0] **=** p[i,0], p[i,2]
30. **def** move\_func(s, t):
31. x, v\_x, y, v\_y **=** s
33. # Система диф. уравнений на базе второго закона Ньютона
34. g **=** G **\*** m\_sun **\*** M **/** (R\_sun**\*\***2**\***(x**\*\***2 **+** y**\*\***2))
36. dxdt **=** v\_x
37. dv\_xdt **=** **-** g **\*** x **/** (R\_sun **\*** np.sqrt(x**\*\***2 **+** y**\*\***2))
39. dydt **=** v\_y
40. dv\_ydt **=** **-** g **\*** y **/** (R\_sun **\*** np.sqrt(x**\*\***2 **+** y**\*\***2))
42. **return** dxdt, dv\_xdt, dydt, dv\_ydt
44. **def** collision(x1, vx1, y1, vy1, x2, vx2, y2, vy2, radius1, radius2, mass1, mass2, K):
45. """Аргументы функции:
46. x1,y1,vx1,vy1 - координаты и компоненты скорости 1-ой частицы
47. x2,y2,vx2,vy2 - ... 2-ой частицы
48. radius,mass1,mass2 - радиус частиц и их массы (массы разные можно задавать,
49. радиус для простоты взят одинаковый)
50. K - коэффициент восстановления (K=1 для абсолютного упругого удара, K=0
51. для абсолютно неупругого удара, 0<K<1 для реального удара).
52. В данном случае коэффициент ВАЖНО положить больше 1, чтобы учесть дополнительную
53. кенетическую энергию, возникающую в результате взрыва.
54. Функция возвращает компоненты скоростей частиц, рассчитанные по формулам для
55. реального удара, если стокновение произошло. Если удара нет, то возвращаются
56. те же значения скоростей, что и заданные в качестве аргументов.
57. """
58. r12 **=** np.sqrt((x1**-**x2)**\*\***2 **+** (y1**-**y2)**\*\***2) #расчет расстояния между центрами частиц
59. # расчет модулей скоростей частиц
60. v1 **=** np.sqrt(vx1**\*\***2 **+** vy1**\*\***2)
61. v2 **=** np.sqrt(vx2**\*\***2 **+** vy2**\*\***2)
63. #проверка условия на столкновение: расстояние должно быть меньше 2-х радиусов
64. **if** r12 <**=** radius1 **+** radius2:
65. '''вычисление углов движения частиц theta1(2), т.е. углов между
66. направлением скорости частицы и положительным направлением оси X.
67. Если частица  покоится, то угол считается равным нулю. Т.к. функция
68. arccos имеет область значений от 0 до Pi, то в случае отрицательных
69. y-компонент скорости для вычисления угла theta1(2) надо из 2\*Pi
70. вычесть значение arccos(vx/v)
71. '''
72. **if** v1!**=**0:
73. theta1 **=** np.arccos(vx1 **/** v1)
74. **else**:
75. theta1 **=** 0
76. **if** v2!**=**0:
77. theta2 **=** np.arccos(vx2 **/** v2)
78. **else**:
79. theta2 **=** 0
80. **if** vy1<0:
81. theta1 **=** **-** theta1 **+** 2 **\*** np.pi
82. **if** vy2<0:
83. theta2 **=** **-** theta2 **+** 2 **\*** np.pi
85. #вычисление угла соприкосновения.
86. **if** (y1**-**y2)<0:
87. phi **=** **-** np.arccos((x1**-**x2) **/** r12) **+** 2 **\*** np.pi
88. **else**:
89. phi **=** np.arccos((x1**-**x2) **/** r12)
91. # Пересчет  x-компоненты скорости первой частицы
92. VX1 **=** v1 **\*** np.cos(theta1 **-** phi) **\*** (mass1 **-** K **\*** mass2) \
93. **\*** np.cos(phi) **/** (mass1 **+** mass2)\
94. **+** ((1 **+** K) **\*** mass2 **\*** v2 **\*** np.cos(theta2 **-** phi))\
95. **\*** np.cos(phi) **/** (mass1 **+** mass2)\
96. **+** K **\*** v1 **\*** np.sin(theta1 **-** phi) **\*** np.cos(phi **+** np.pi **/** 2)
98. # Пересчет y-компоненты скорости первой частицы
99. VY1 **=** v1 **\*** np.cos(theta1 **-** phi) **\*** (mass1 **-** K **\*** mass2) \
100. **\*** np.sin(phi) **/** (mass1 **+** mass2) \
101. **+** ((1 **+** K) **\*** mass2 **\*** v2 **\*** np.cos(theta2 **-** phi)) \
102. **\*** np.sin(phi) **/** (mass1 **+** mass2) \
103. **+** K **\*** v1 **\*** np.sin(theta1 **-** phi) **\*** np.sin(phi **+** np.pi **/** 2)
105. # Пересчет x-компоненты скорости второй частицы
106. VX2 **=** v2 **\*** np.cos(theta2 **-** phi) **\*** (mass2 **-** K **\*** mass1) \
107. **\*** np.cos(phi) **/** (mass1 **+** mass2)\
108. **+** ((1 **+** K) **\*** mass1 **\*** v1 **\*** np.cos(theta1 **-** phi)) \
109. **\*** np.cos(phi) **/** (mass1 **+** mass2)\
110. **+** K **\*** v2 **\*** np.sin(theta2 **-** phi) **\*** np.cos(phi **+** np.pi **/** 2)
112. # Пересчет y-компоненты скорости второй частицы
113. VY2 **=** v2 **\*** np.cos(theta2 **-** phi) **\*** (mass2 **-** K **\*** mass1) \
114. **\*** np.sin(phi) **/** (mass1 **+** mass2) \
115. **+** ((1 **+** K) **\*** mass1 **\*** v1 **\*** np.cos(theta1 **-** phi)) \
116. **\*** np.sin(phi) **/** (mass1 **+** mass2)\
117. **+** K **\*** v2 **\*** np.sin(theta2 **-** phi) **\*** np.sin(phi **+** np.pi **/** 2)
119. **else**:
120. #если условие столкновнеия не выполнено, то скорости частиц не пересчитываются
121. VX1, VY1, VX2, VY2 **=** vx1,vy1,vx2,vy2
123. **return** VX1, VY1, VX2, VY2
125. # Решение задачи и проверка условий столковения
126. **for** k **in** range(n**-**1):
127. t**=**[tau[k],tau[k**+**1]]
128. **for** m **in** range(N):
129. s0 **=** p[m,0], p[m,1], p[m,2], p[m,3]
130. sol **=** odeint(move\_func, s0, t)
132. p[m,0] **=** sol[1,0]
133. p[m,1] **=** sol[1,1]
134. p[m,2] **=** sol[1,2]
135. p[m,3] **=** sol[1,3]
137. x[m,k**+**1], y[m,k**+**1] **=** p[m,0], p[m,2]
139. 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)
141. p[m,1], p[m,3] **=** res[0], res[1]
143. # Графика
144. **def** circle(radius, x0, y0): #Функция, генерирующая координаты ядра звезды
145. phi **=** np.linspace(0, 2**\***np.pi, 100)
146. x **=** x0 **+** radius **\*** np.cos(phi)
147. y **=** y0 **+** radius **\*** np.sin(phi)
148. **return** x, y
150. fig, ax **=** plt.subplots() #Создание пространства для анимации
151. plt.plot([0], [0], 'o', ms**=**34, color**=**'b', label**=**'Ядро звезды')
152. nucleus, **=** plt.plot([], [], '-', color**=**'b') #Анимируемый объект
153. particles, **=** plt.plot([], [], '-', color**=**'r', lw**=**5, label**=**'Внешняя оболочка')
155. # Определение области
156. edge **=** 8
157. ax.set\_xlim(**-**edge, edge)
158. ax.set\_ylim(**-**edge, edge)
160. **def** animate(i): #Функция подстановки координат в анимируемый объект
161. nucleus.set\_data(circle(radius**=**R, x0**=**0, y0**=**0)) # Ядро звезды
162. particles.set\_data([x[:,:i]],  y[:,:i])
164. ani **=** animation.FuncAnimation(fig,
165. animate,
166. frames**=**n,
167. interval**=**0.1)
169. plt.axis('equal')
170. plt.xlim(**-**5, 5)
171. plt.ylim(**-**5, 5)
172. plt.legend()
173. ani.save('Supernova.gif')
175. # plt.show()