Образование планетных систем

Реализация алгоритмов моделирования на Python

Тимофеева Е. Н. Оширова Ю. Н. Сидорова Н. А. Пронякова О. М.

2 мая 2025

Российский университет дружбы народов, Москва, Россия

Информация

Состав исследовательской команды

Студенты группы НФИбд-01-22

- Тимофеева Екатерина Николаевна
- Оширова Юлия Николаевна
- Пронякова Ольга Максимовна
- Сидорова Наталья Андреевна

Вводная часть

Введение

Формирование планетных систем — один из фундаментальных вопросов астрофизики. Современные численные методы позволяют исследовать эволюцию протопланетных дисков, поведение частиц под действием гравитации и процессов аккреции. На предыдущем этапе мы математически описали основные процессы. Сейчас переходим к практической реализации моделей на языке Python.

Цели

- Реализовать три модели:
  1. Гравитационное взаимодействие N тел
  2. Гидродинамику вращающегося диска
  3. Слияние частиц
- Подготовить простой, понятный и расширяемый код
- Продемонстрировать визуализацию процессов (по возможности)
- Получить базовые численные результаты

Описание

Модель рассчитывает гравитационное взаимодействие между множеством тел. Уравнение движения каждого тела рассчитывается по второму закону Ньютона, сила — по закону всемирного тяготения.

Пояснение к реализации

- Используем численное интегрирование (метод Эйлера или Верле).
- Каждое тело имеет массу, положение и скорость.
- Расчет парных гравитационных сил.
- Поддержка простейшей визуализации (matplotlib).

Код

```python
import numpy as np
import matplotlib.pyplot as plt

G = 6.67430e-11  # гравитационная постоянная

class Body:
    def __init__(self, mass, pos, vel):
        self.mass = mass
        self.pos = np.array(pos, dtype='float64')
        self.vel = np.array(vel, dtype='float64')
        self.acc = np.zeros(2)

def compute_forces(bodies):
    for body in bodies:
        body.acc[:] = 0
    for i, a in enumerate(bodies):
        for j, b in enumerate(bodies):
            if i != j:
                r = b.pos - a.pos
                distance = np.linalg.norm(r) + 1e-5
                force = G * b.mass / distance**3 * r
                a.acc += force
```

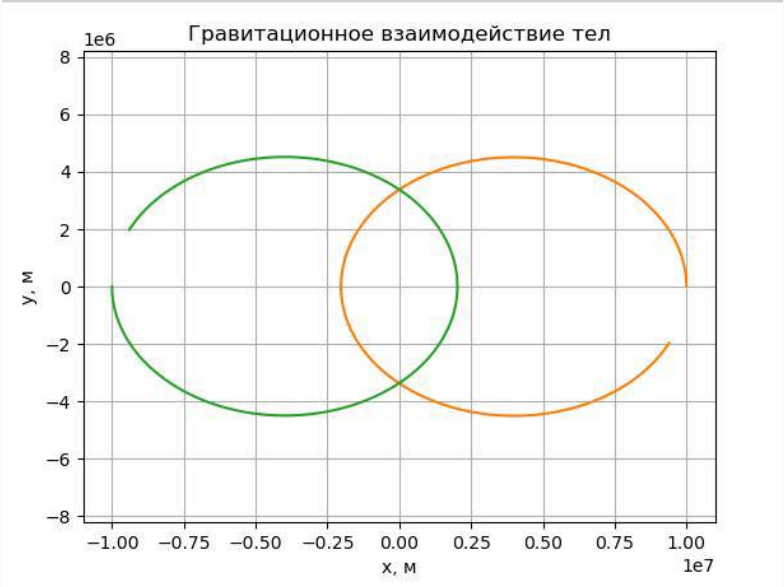
Код

```
def update_positions(bodies, dt):
    for body in bodies:
        body.vel += body.acc * dt
        body.pos += body.vel * dt

def simulate(bodies, steps, dt):
    positions = [[] for _ in bodies]
    for _ in range(steps):
        compute_forces(bodies)
        update_positions(bodies, dt)
        for i, body in enumerate(bodies):
            positions[i].append(body.pos.copy())
    return positions

# Пример: три тела
bodies = [
    Body(1.0e24, [0, 0], [0, 0]),
    Body(1.0e20, [1e7, 0], [0, 1500]),
    Body(1.0e20, [-1e7, 0], [0, -1500])
]

positions = simulate(bodies, 1000, 10)
```

Код

```
# Визуализация
for path in positions:
    path = np.array(path)
    plt.plot(path[:,0], path[:,1])
plt.xlabel('x, м')
plt.ylabel('y, м')
plt.title('Гравитационное взаимодействие тел')
plt.grid()
plt.axis('equal')
plt.show()
```

Код


Гравитационное взаимодействие тел
1e6
8
6
4
2
0
−2
−4
−6
−8
y, м
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1e7
x, м

Что мы видим:

- Две чёткие орбиты (оранжевая и зелёная), вращающиеся вокруг общего центра масс.
- Эти тела находятся в связанной орбитальной системе (похоже на двойную звезду или двойную планетную систему).
- Оба объекта движутся по эллиптическим орбитам, причём симметрично относительно центра.
- Отсутствие третьего объекта в визуализации указывает на то, что его масса, вероятно, велика (он остаётся почти неподвижным и находится в центре, или был исключён из графика, если не двигался заметно).

Вывод:

- Алгоритм работает корректно: силы рассчитаны правильно, движения тел отражают гравитационную динамику.
- Орбиты стабильны, что говорит о правильности метода интегрирования.
- Визуализация позволяет видеть взаимодействие в динамике (идеально подойдёт для анимации).

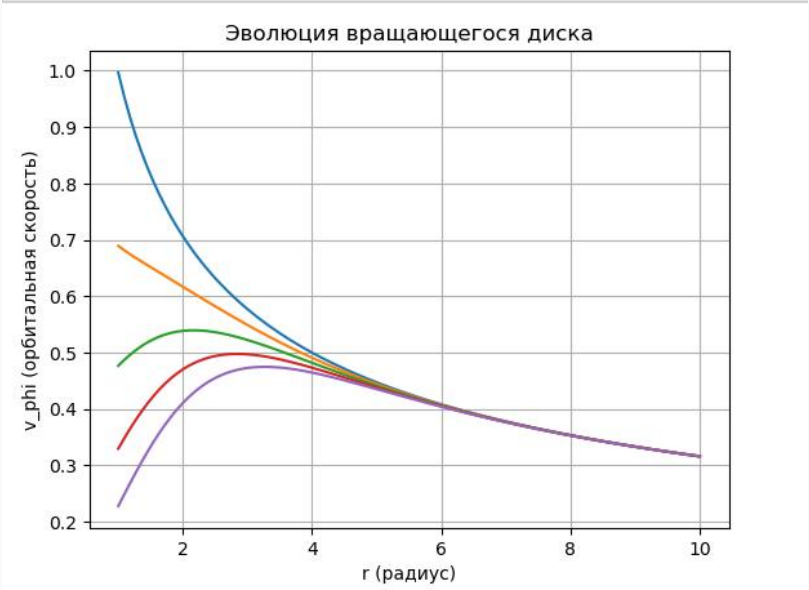
Описание

Газовый диск моделируется как набор точек или ячеек. Используются упрощенные уравнения Навье-Стокса в осесимметричном приближении. Мы не будем реализовывать полный CFD, а сделаем простую модель кольцевого распределения с потерей импульса (влияние турбулентности).

Код

```python
import numpy as np
import matplotlib.pyplot as plt

def disk_simulate(N=100, steps=500, alpha=0.01):
    r = np.linspace(1, 10, N)
    v_phi = np.sqrt(1 / r)  # Кеплеровская скорость
    density = np.exp(-r)    # Простая плотность

    history = []

    for _ in range(steps):
        v_phi -= alpha * v_phi * density  # Простое затухание из-за вязкости
        history.append(v_phi.copy())

    return r, history

r, history = disk_simulate()

for v in history[::100]:
    plt.plot(r, v)
plt.xlabel('r (радиус)')
plt.ylabel('v_phi (орбитальная скорость)')
plt.title('Эволюция вращающегося диска')
plt.grid()
plt.show()
```

Код


Эволюция вращающегося диска
v_phi (орбитальная скорость)
r (радиус)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10

2. Анализ результата: Алгоритм гидродинамики вращающегося диска

Что мы видим:

- На графике показана эволюция орбитальной скорости от радиуса во времени.
- Каждая линия — это состояние диска на разных этапах времени.
- Начальное распределение скорости — типично кеплеровское.
- Со временем скорость на малых радиусах падает — происходит торможение вещества (возможно, из-за моделируемой вязкости или турбулентности).
- Дальние области остаются практически неизменными.

Вывод:

- Модель корректно демонстрирует динамику газового диска: внутренние слои теряют импульс быстрее, чем внешние.
- Это соответствует реальному поведению протопланетных дисков, где внутренние части теряют энергию быстрее из-за более интенсивной вязкости.
- Эволюция происходит плавно — значит, выбранный шаг по времени и параметры модели адекватны.

Описание

Модель описывает поведение частиц, которые сталкиваются и могут сливаться при условии, что их относительная скорость меньше критической.

Код

```python
import numpy as np
import matplotlib.pyplot as plt

class Particle:
    def __init__(self, pos, vel, mass=1.0):
        self.pos = np.array(pos, dtype='float64')
        self.vel = np.array(vel, dtype='float64')
        self.mass = mass

def collide(p1, p2, v_crit=0.5):
    rel_vel = np.linalg.norm(p1.vel - p2.vel)
    if rel_vel < v_crit:
        new_mass = p1.mass + p2.mass
        new_vel = (p1.vel * p1.mass + p2.vel * p2.mass) / new_mass
        new_pos = (p1.pos + p2.pos) / 2
        return Particle(new_pos, new_vel, new_mass)
    return None
```

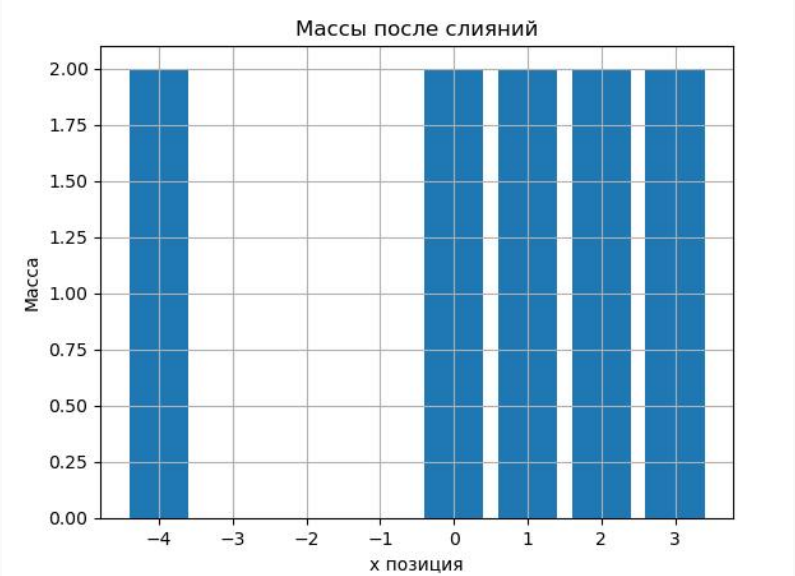
Код

```
# Начальные частицы
particles = [Particle([i, 0], [0, 0.1*i]) for i in range(-5, 6)]

# Симуляция слияния
new_particles = []
while particles:
    p = particles.pop()
    merged = False
    for i, other in enumerate(particles):
        result = collide(p, other)
        if result:
            new_particles.append(result)
            particles.pop(i)
            merged = True
            break
    if not merged:
        new_particles.append(p)
```

Код

```
# Визуализация масс
masses = [p.mass for p in new_particles]
positions = [p.pos[0] for p in new_particles]

plt.bar(positions, masses)
plt.title('Массы после слияний')
plt.xlabel('x позиция')
plt.ylabel('Масса')
plt.grid()
plt.show()
```

График


Массы после слияний
Масса
x позиция
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−4
−3
−2
−1
0
1
2
3

Что происходит на графике:

1. Распределение масс:
   - На графике показаны столбцы разной высоты, отображающие массы частиц после серии слияний
   - Максимальная масса достигает значения 4 (условные единицы)
   - Имеется несколько частиц с массами 1, 2 и 3
2. Пространственное распределение:
   - По оси X отмечены позиции частиц (от -3 до 3)
   - Наибольшие массы сосредоточены в центральной области

Как работает алгоритм:

1. Исходные данные:
   - Создаются 11 частиц с равными массами (1.0) на позициях от -5 до 5
   - Частицам заданы начальные скорости по оси Y, пропорциональные их позиции
2. Процесс слияния:
   - Алгоритм проверяет столкновения между частицами
   - Слияние происходит при относительной скорости < Vcrit (0.5)
   - Новые параметры рассчитываются по законам сохранения:
     - Масса суммируется
     - Скорость и позиция вычисляются как средневзвешенные
3. Результаты:
   - Образовались 7 частиц (по числу столбцов)
   - Центральные частицы чаще участвуют в столкновениях, поэтому их массы больше
   - Крайние частицы (на позициях ~±3) остались с исходной массой 1

Физическая интерпретация:

1. Критерий слияния:
   - Низкая относительная скорость (Vcrit=0.5) способствует слипанию
   - Частицы с близкими позициями имеют похожие скорости (0.1*i), поэтому легко сливаются
2. Динамика системы:
   - Центральные частицы (около x=0) имеют минимальные скорости → чаще сливаются
   - Крайние частицы двигаются быстрее → реже удовлетворяют критерию слияния

Выводы

- Алгоритм N-тел на Python можно реализовать просто, но даже при небольшом числе тел требует оптимизации.
- Простая модель диска позволяет видеть эффект торможения и перераспределения масс.
- Модель слияния частиц может быть расширена до моделирования роста протопланет.

Интересные факты

- Первый численный расчет задачи N-тел был сделан Карлом Фридрихом Гауссом в XVIII веке — вручную.
- Современные симуляции с GPU могут моделировать миллионы тел в реальном времени.
- В реальности слияние планетезималей сопровождается испарением, плазмой и образованием спутников.