Что будет, если ещё одна звезда войдёт в Солнечную систему?

Л.В. Галкин, А.С. Поляков ,О.Е. Трофимова, Л.К. Шевкунов
Научный руководитель: М.В. Царьков

Цель работы и перспективы

Цель: Основной задачей проекта было сделать модель солнечной системы наглядно демонстрирующую силы взаимного притяжения планет друг к другу и к солнцу.
Видео представленные в результатах работы учителя могут использовать на своих уроках.
Перспективы: В дальнейшем можно сделать удобный GUI, который позволит моментально добавлять объекты и изменять их параметры.

Аннотация

В работе приведен пример действия гравитационных сил планет солнечной системы. В конечном итоге у нас получилась программа, с помощью которой можно спроецировать модель вращения планет вокруг Солнца, а также посмотреть что произойдет если в Солнечную систему войдет один или несколько посторонних объектов.

Введение

Мы поставили перед собой цель сделать модель Солнечной системы, а также посмотреть
что произойдет если один или несколько посторонних объектов войдут в нее.
Для достижения поставленной цели необходимо решить следующие задачи:

1. *Изучить язык программирования Python.*
2. *Изучить такие библиотеки Python как matplotlib, numpy, math и др.*
3. *Смоделировать поведение Солнечной системы при появлении в ней новой звезды.*

Представленное решение задачи является актуальным, поскольку для ее решения мы использовали передовые методы численного моделирования и вычисления, базирующиеся на открытых библиотеках языка программирования Python.

Моделирование Солнечной системы

Для получения анимированного графика модели Солнечной системы мы должны в результате вычислений для каждой планеты получить массив точек.

1 Создание 9 экземпляров класса Planet с характеристиками планет и Солнца, установка начальных значений. (рис 1, 2)

2 Для вычисления силы притяжения между всеми планетами и солнцем, каждая из планет должна пройти через метод acceleration() (рис 3), который возвращает ускорение для осей X и Y.
Ускорение показывает сумму сил притяжения одной планеты ко всем остальным и Солнцу, этот метод применяется ко всем планетам.

Вычисление радиуса:

$$r = \sqrt{x^2 + y^2} = \left(x^2 + y^2\right)^{\frac{1}{2}}$$

Формулы для расчета силы гравитации:

$$F = G\frac{m_1 m_2}{r^2}$$

3 Анимированный график рисуется с помощью библиотеки pygame из массива точек. Массив точек возвращается методом updatePlanet() - для каждой планеты выполняется дифференциальные уравнения, здесь также вычисляется мгновенная скорость планет (рис 4,5).

Формулы для расчета дифференциальных уравнений:

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

Заключение

Проведенное исследование наглядно продемонстрировало возможности языка Python и его библиотек pygame, numpy и math,
принципы ООП. С помощью написанной программы можно спроецировать Солнечную систему и тела с разной массой и радиусом входящие в нее. Также можно создать собственную планетарную систему и посмотреть за действием гравитационных сил.

Листинг кода

```
58
59  class Planet:
60      def __init__(self):
61              #Earth          #width height speed acceleration
62              self._st = State(500,   605,   2.28,     0.06)
63              self._r = 1.5
64              self.setMassFromRadius(5.5153)
65              self._merged = False
66
67
```

рис. 1

```
    sun = Planet()              # center
    sun._st._x, sun._st._y = WIDTHD2, HEIGHTD2
    sun._st._vx = sun._st._vy = 0.
    sun._r *= 109
    sun.setMassFromRadius(1.409)
    sun._rv = 23
    g_listOfPlanets.append(sun)

    Jupiter = Planet()
    Jupiter._st = State(500, 700, 1.7, 0.34)
    Jupiter._r *= 11.2
    Jupiter.setRadiusFromMass(1.326)
    g_listOfPlanets.append(Jupiter)
```

рис. 2

...

```
def acceleration(self, state, unused_t):
    ax = 0.0
    ay = 0.0
    for p in g_listOfPlanets:
        if p is self or p._merged:
            continue  # ignore ourselves and merged planets
        dx = p._st._x - state._x
        dy = p._st._y - state._y
        dsq = dx*dx + dy*dy  # distance squared
        dr = math.sqrt(dsq)  # distance
        force = GRAVITYSTRENGTH*self._m*p._m/dsq if dsq>1e-10 else 0.
        # Accumulate acceleration...
        ax += force*dx/dr
        ay += force*dy/dr
    return (ax, ay)
```

рис. 3

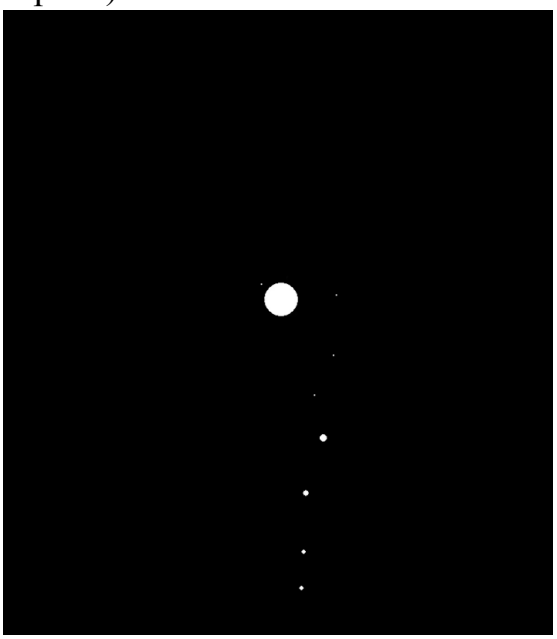
```
def initialDerivative(self, state, t):
    ax, ay = self.acceleration(state, t)
    return Derivative(state._vx, state._vy, ax, ay)

def nextDerivative(self, initialState, derivative, t, dt):
    state = State(0., 0., 0., 0.)
    state._x = initialState._x + derivative._dx*dt
    state._y = initialState._y + derivative._dy*dt
    state._vx = initialState._vx + derivative._dvx*dt
    state._vy = initialState._vy + derivative._dvy*dt
    ax, ay = self.acceleration(state, t+dt)
    return Derivative(state._vx, state._vy, ax, ay)
```

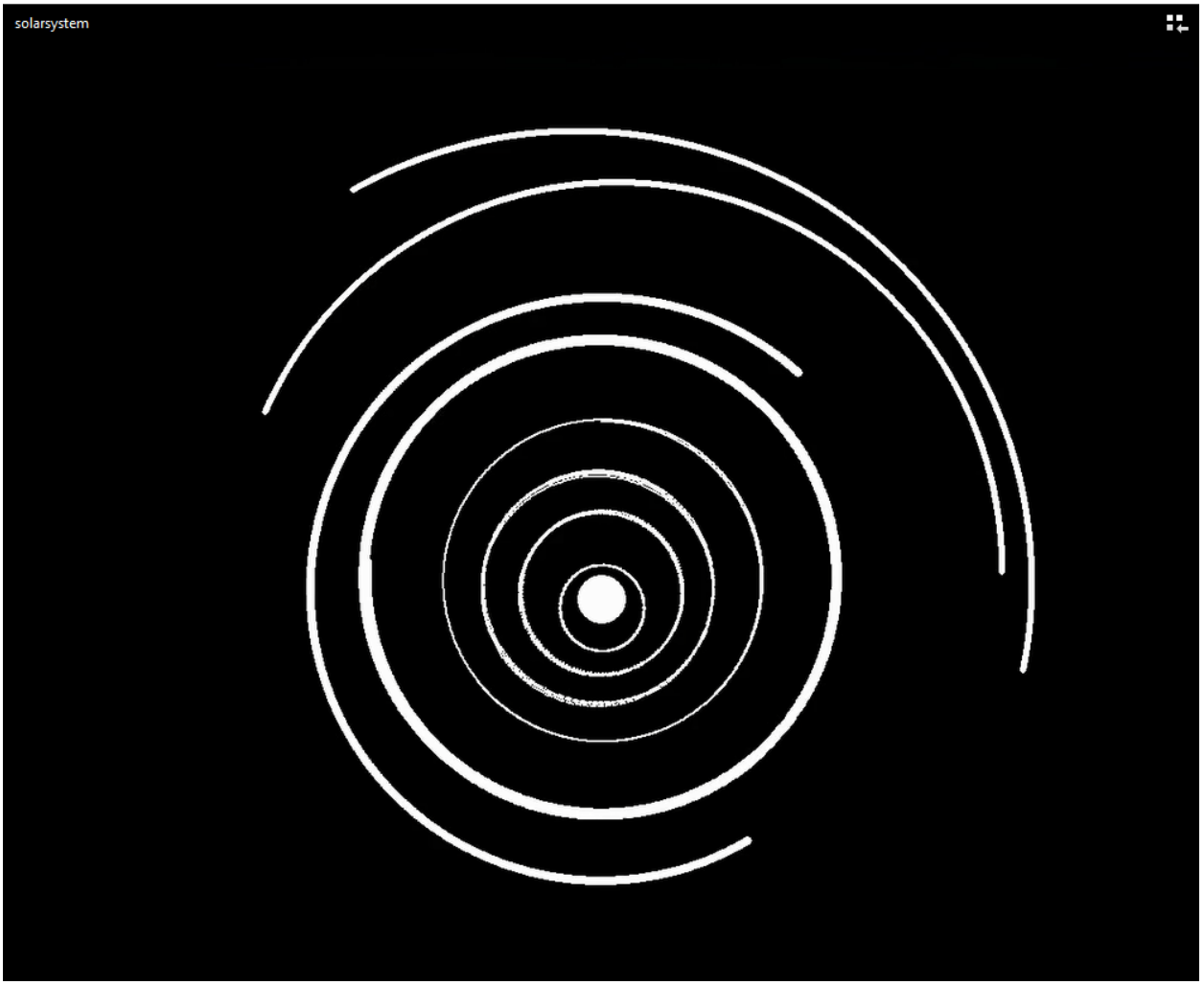
рис.4

```
def updatePlanet(self, t, dt):
    a = self.initialDerivative(self._st, t)
    b = self.nextDerivative(self._st, a, t, dt*0.5)
    c = self.nextDerivative(self._st, b, t, dt*0.5)
    d = self.nextDerivative(self._st, c, t, dt)
    dxdt = (a._dx + (b._dx + c._dx) + d._dx)
    dydt = (a._dy + (b._dy + c._dy) + d._dy)
    dvxdt = (a._dvx + (b._dvx + c._dvx) + d._dvx)
    dvydt = (a._dvy + (b._dvy + c._dvy) + d._dvy)
    self._st._x += dxdt*dt
    self._st._y += dydt*dt
    self._st._vx += dvxdt*dt
    self._st._vy += dvydt*dt
```

рис 5

Результаты работы представлены в виде анимированных графиков (.gif изображения вышли слишком большими, поэтому в докладе снимки экрана)





для примера с одним (массой и размерами схожим с юпитером) и несколькими посторонними объектами мы добавили еще несколько экземпляров класс Planet.