ГБОУ «Президентский ФМЛ №239»

Визуализация гравитационных маневров
Годовой проект по информатике

Булыгин Игорь, 10-2 класс

Постановка задачи.

Задачей проекта является визуализация (то есть рисование на экране) движения материальных точек под действием закона всемирного тяготения.

Таким образом, условием задачи является написание программы, рассчитывающей положения N тел на плоскости. Тела двигаются под действием гравитационного притяжения. Программа осуществляет вычисления точек, в которых находятся тела, их скоростей и ускорений. Также движение тел рисуется в отдельном окне, выводимом программой в реальном времени. Тела изображаются в виде кружков, вектора — в виде отрезков с началом в точке, в которой находится тело. Кроме этого, рисуется траектория тела. При закрытии окна программа прекращает работу.

Исходные и выходные данные.

Исходными данными являются начальные условия системы из N тел и константы в используемых формулах.

Вычисления происходят с учетом того, что все движение происходит только на плоскости, значит начальные условия включают в себя (для N тел соответственно): количество тел, 2N координат, то есть N точек на плоскости, в которых тела находятся в начальный момент времени; N векторов начальных скоростей (т. е. 2N чисел); N масс материальных точек, гравитационная постоянная в системе единиц программы (см. пункт «Математическая модель»); размеры выводимого окна (2 числа); шаг вычислений dt (см. пункт «Математическая модель»). Итого это 5N+2 вещественных значений, для которых был использован примитивный тип данных double, и еще 3 целых значения (integer).

Замечу, что объем входных данных зависит от конкретной задачи (с конкретным числом тел), из этого следует, что теоретического ограничения на объем данных не существует, ограничение устанавливает мощность используемой вычислительной машины.

Ограничение на диапазон возможных значений получается из двух ограничений: необходимо получить наглядный результат и минимизировать ошибку вычислений (см. пункт «Математическая модель»). Говоря простым языком, такие параметры как скорость*шаг и изменение координаты должны быть несравнимо меньше размеров окна, чтобы препятствовать исчезновению изображения, которое происходит при большом значении координат. Также скорость каждого из тел не должна превышать вторую космическую для данной системы (энергия взаимодействия тела со всей остальной системой должна быть <0, чтобы система была стабильной). Также при подлете друг к другу на близкое расстояние, ошибка в определении координат тел увеличивается, что дает ограничение снизу на возможные скорости (нельзя ставить маленькие скорости, это повысит неадекватность результата из-за большой ошибки при сближении).

Выходными данными являются все последующие значения координат и скоростей тел, вычисленные за все шаги программы. Также выходными данными является множество всех изображений, возникающих на экране при визуализации вычисленного.

Математическая модель.

Закон всемирного тяготения:

$$Ax(M, R, R0) = \frac{GM}{(R-R0)^3}(Rx-R0x) \qquad Ay(M, R, R0) = \frac{GM}{(R-R0)^3}(Ry-R0y)$$

Ax и Ay — составляющие вектора ускорения притягивающегося тела по осям x и y соответственно. M – масса притягивающего тела. R и R0 — радиус вектора из начала координат для двух тел; Rx, R0x, Ry, R0y – их проекции на оси. G – гравитационная постоянная, определяемая специально для адекватной работы программы. При возведении в степень в знаменателе считается, что берется модуль полученного вектора.

Модель равноускоренного движения материальной точки (МТ) на плоскости:

$$A=\frac{dV}{dt} \qquad V=\frac{dR}{dt} \qquad R=(Rx, Ry) \qquad V=(Vx, Vy) \qquad A=(Ax, Ay)$$

A — ускорение MT, V – скорость MT, R – радиус-вектор MT, t – время.

Решение такого количества дифференциальных уравнений в явном виде невозможно, однако можно это сделать численно, тем самым определив траекторию с минимально возможной ошибкой. Метод Рунге-Кутты 4 порядка был выбран для решения этой системы. Описание метода:

$$a=a(R1, R2, M2)$$

$$p1=dt* a(R1, R2, M2); q1=dt* v$$

$$p2=dt* a\left(R1+\frac{q1}{2}, R2+\frac{q1'}{2}, M2\right); q2=dt* \left(v+\frac{p1}{2}\right)$$

$$p3=dt* a\left(R1+\frac{q2}{2}, R2+\frac{q2'}{2}, M2\right); q3=dt* \left(v+\frac{p2}{2}\right)$$

$$p4=dt* a(R1+q3, R2+q3', M2); q4=dt* (v+p3)$$

$$v=v+\frac{1}{6}* (p1+2* p2+2* p3+p4)$$

$$R1=R1+\frac{1}{6}* (q1+2* q2+2* q3+q4)$$

Здесь a, v – ускорение и скорость тела. R1 и R2 - радиус-векторы тел. Величина шага вычислений — dt. Остальные величины не несут физического смысла и являются частью метода, они позволяют уменьшить порядок ошибки вычислений.

Анализ используемой структуры данных.

Все параметры тел хранятся в двумерном массиве double[N][5], где N – количество тел, 5 — количество параметров (масса, положение и скорость). Начальные данные не сохраняются, так как это не нужно. С каждым шагом параметры тел изменяются в соответствии с методом решения (см. пункт «Математическая модель») и эти изменения вносятся в массив, что означает, что уже на первом шаге значения начальных положений и скоростей будут потеряны. Массы тел не изменяются, их надо хранить, потому что они необходимы для вычислений. Константы (разрешение экрана, гравитационная постоянная) тоже необходимо хранить.

Траектория каждого тела состоит из прямых линий, соединяющих его положения через каждый шаг. Для того, чтобы хранить все положения, нужен «массив» double, в который можно будет добавлять новые элементы каждый шаг и брать их. Для этого был использован ArrayList, в котором хранятся все координаты тела после каждого шага вычислений.

Описание алгоритма.

Программа состоит из 2 основных этапов: вычисление следующего положения всех тел и рисования изменений на экране в выводимом окне. Сначала происходит создание окна и настройка его параметров. После этого создается специальный класс GraphicsPanel, в котором происходят вычисления и отрисовка. При создании с помощью конструктора класс запрашивает начальные условия, которые вводятся в консоль, а после этого сразу же начинается вычисление следующего положения, далее включается секундомер, который позволяет подождать некоторое время, потому что иначе бы вычисления происходили слишком быстро и человек бы не успевал наблюдать за происходящим. Далее положения всех тел, все их траектории и все вектора рисуются в окне. Причем используется команда repaint();, чтобы не было наложений предыдущей картинки на следующую (экран полностью очищается, а потом все рисуется заново). При закрытии окна работа программы прекращается.

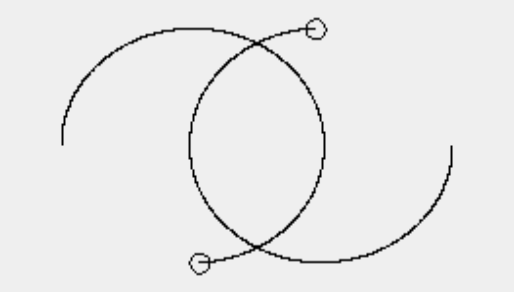
Примеры работы программы.

- Два тела одинаковой массы

Входные данные:

```
arr[0][0]=100;
arr[1][0]=100;
arr[0][1]=200;
arr[0][2]=200;
arr[1][1]=400;
arr[1][2]=200;
arr[0][3]=0;
arr[0][4]=-8;
arr[1][3]=0;
arr[1][4]=8;
```

Результат:

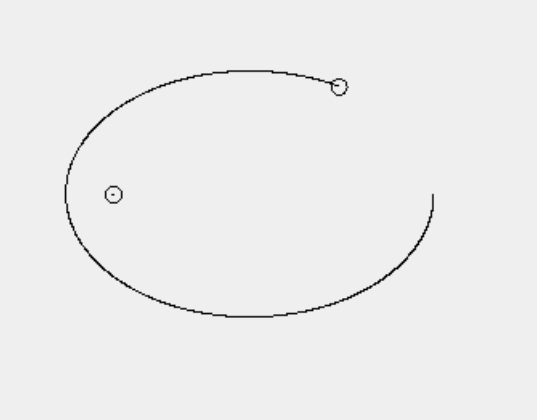


- Два тела, масса одного много больше массы другого (Земля и Солнце)

Входные данные:

```
arr[0][0]=100;
arr[1][0]=0.01;
arr[0][1]=200;
arr[0][2]=200;
arr[1][1]=400;
arr[1][2]=200;
arr[0][3]=-0.0005;
arr[0][4]=0;
arr[1][3]=0;
arr[1][4]=8;
```

Результат:

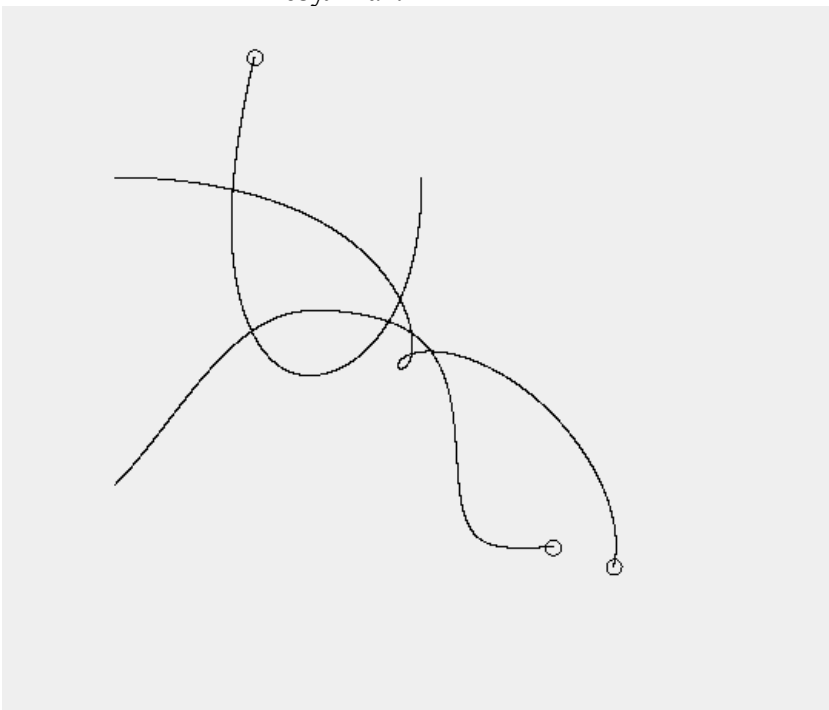


- Три тела сравнимых масс.

Входные данные:

```
arr[0][0]=10;
arr[1][0]=10;
arr[2][0]=10;
arr[0][1]=200;
arr[0][2]=200;
arr[1][1]=400;
arr[1][2]=200;
arr[2][1]=200;
arr[2][2]=400;
arr[0][3]=5;
arr[0][4]=0;
arr[1][3]=0;
arr[1][4]=5;
arr[2][3]=2;
arr[2][4]=-2;
```

Результат:

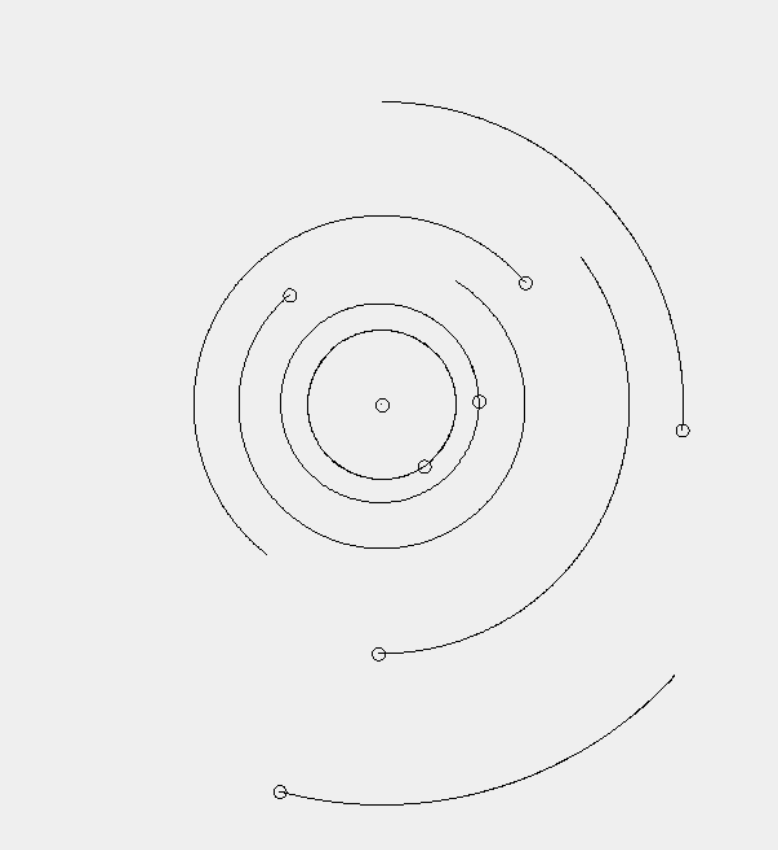


-
-
- 8 тел, масса одного много много больше массы всех остальных. Моделирование системы TRAPPIST-1. Значения скоростей вычисляются далее и выбраны так, чтобы скорость была круговой для каждого тела малой массы.

Входные данные:

```
//МАССЫ
arr[0][0]=29600;
arr[1][0]=1.02;
arr[2][0]=1.16;
arr[3][0]=0.297;
arr[4][0]=0.772;
arr[5][0]=0.934;
arr[6][0]=1.148;
arr[7][0]=0.331;
//X
arr[0][1]=500*(1);
arr[1][1]=500*(1+0.1150);
arr[2][1]=500*(1-0.12589);
arr[3][1]=500*(1+0.1123);
arr[4][1]=500*(1-0.1771);
arr[5][1]=500*(1+0.3083);
arr[6][1]=500*(1);
arr[7][1]=500*(1+0.4549);
//Y
arr[0][2]=500*(1);
arr[1][2]=500*(1);
arr[2][2]=500*(1-0.09485);
arr[3][2]=500*(1-0.1914);
arr[4][2]=500*(1+0.23165);
arr[5][2]=500*(1-0.2282);
arr[6][2]=500*(1-0.467);
arr[7][2]=500*(1+0.4168);
```

Результат:

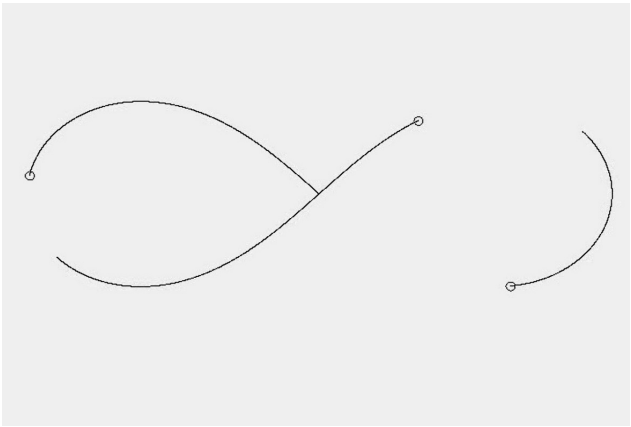


- Три тела одинаковых масс движутся по лемнискате Бернулли (восьмерка)

Входные данные:

```
arr[0][0]=1;
arr[1][0]=1;
arr[2][0]=1;
arr[0][1]=500+300*0.97000436;
arr[0][2]=500-300*0.24308753;
arr[1][1]=500+300*(-0.97000436);
arr[1][2]=500+300*0.24308753;
arr[2][1]=500;
arr[2][2]=500;
arr[0][3]=0.5*300*(0.93240737);
arr[0][4]=0.5*300*(0.86473146);
arr[1][3]=0.5*300*(0.93240737);
arr[1][4]=0.5*300*(0.86473146);
arr[2][3]=-300*(0.93240737);
arr[2][4]=-300*(0.86473146);
```

Результат:



Примечание 1:

В данных примерах не рисуются вектора скорости и ускорения, потому что тогда рисунки были бы слишком громоздкими и ненаглядными.

Примечание 2:
arr[m][0] — масса m-ого тела;
arr[m][1] – координата по оси X;
arr[m][2] – координата по оси Y;
arr[m][3] – скорость по оси X;
arr[m][4] – скорость по оси Y.

Комментированный листинг

- **Class Main**

```
import javax.swing.*;
import java.awt.*;
public class Main {
    public static void main(String[] args) {
        JFrame frame = new Jframe();
//создание нового фрейма
        frame.setTitle("Gravity project");
//название фрейма
        frame.setSize(new Dimension(1800, 1000));
//размеры фрейма
        frame.setLocationRelativeTo(null);
        frame.setLayout(new GridBagLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//окончание работы при закрытии фрейма
        GraphicsPanel graphicsPanel = new GraphicsPanel();
//создание вспомогательного объекта отрисовки
        frame.add(graphicsPanel, new GridBagConstraints(
           0, 0, 1, 1, 1, 1,
           GridBagConstraints.NORTH, GridBagConstraints.BOTH,
           new Insets(2, 2,2, 2), 0, 0
        ));
//связь объекта отрисовки с фреймом
        frame.setVisible(true);
//вывод фрейма на экран
    }
}
```

- **Class GraphicsPanel**

```
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.util.ArrayList;
import javax.swing.*;
public class GraphicsPanel extends JPanel implements Runnable{
    //массив для праметров объектов
    private double [][]arr = new double [3][5];
    //массив, куда запихиваются координаты тела для отрисовки
кривой
    private ArrayList <Double> curve1 = new ArrayList<>();
    private ArrayList <Double> curve2 = new ArrayList<>();
    private ArrayList <Double> curve3 = new ArrayList<>();
    private double g = arr.length-1;
    //конструктор панели
```

```
    public GraphicsPanel(){
        //Вот тут вводятся параметры.
        ...
        //создание секундомера
        (new Thread(this)).start();
    }
    //метод рисовалки
    @Override
    public void paint(Graphics g){
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        Paint paint = g2.getPaint();
        //эта штука позволяет их двигать, то есть обновлять экран

((Graphics2D)g).setComposite(AlphaComposite.getInstance(AlphaCompo
site.SRC));
        //рисовалка всех объектов
        for(int i=0; i<arr.length; i++){
            g2.draw(new Ellipse2D.Double(arr[i][1]-5, arr[i][2]-5,
10, 10));
        }
        //отрисовка траектории
        for(int i=0; i<(curve1.size()-2)/2; i++){
            g2.draw(new Line2D.Double(curve1.get(2*i),
curve1.get(2*i+1), curve1.get(2*(i+1)), curve1.get(2*(i+1)+1)));
        }
        for(int i=0; i<(curve2.size()-2)/2; i++){
            g2.draw(new Line2D.Double(curve2.get(2*i),
curve2.get(2*i+1), curve2.get(2*(i+1)), curve2.get(2*(i+1)+1)));
        }
        for(int i=0; i<(curve3.size()-2)/2; i++){
            g2.draw(new Line2D.Double(curve3.get(2*i),
curve3.get(2*i+1), curve3.get(2*(i+1)), curve3.get(2*(i+1)+1)));
        }
        //отрисовка вектора скорости 1 тела, для примера
        g2.draw(new Line2D.Double(arr[0][1], arr[0][2], arr[0]
[1]+arr[0][3], arr[0][2]+arr[0][4]));
    }
    //функция расстояния между двумя объектами
    public double mod(double x1, double x2, double y1, double y2){
        return Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
    }
    //метод, рассчитывающий силы притяжения объектов
    public double[] Force( double m2, double x1, double x2, double
y1, double y2 ){
        double []test = new double[2];
        //double g =9000;
        test[0]=g*m2*(x2-x1)/Math.pow(mod(x1, x2, y1, y2)/300, 3);
        test[1]=g*m2*(y2-y1)/Math.pow(mod(x1, x2, y1, y2)/300, 3);
        return test;
    }
    //метод, двигающий объекты
    public void move( double dt){
```

```
        double [][] px = new double[arr.length][5];
        double [][] py = new double[arr.length][5];
        double [][] qx = new double[arr.length][5];
        double [][] qy = new double[arr.length][5];
        // вот тут Рунге-Кутта
        for(int o=0; o<arr.length; o++) {
            for (int w=0; w<arr.length; w++) {
                if (w!=o){
                    px[o][1] += dt * Force( arr[w][0], arr[o][1],
arr[w][1], arr[o][2], arr[w][2])[0];
                    qx[o][1] += dt * arr[o][3];
                    py[o][1] += dt * Force( arr[w][0], arr[o][1],
arr[w][1], arr[o][2], arr[w][2])[1];
                    qy[o][1] += dt * arr[o][4];
                }
            }
        }
        for(int o=0; o<arr.length; o++) {
            for (int w=0; w<arr.length; w++) {
                if (w!=o) {
                    px[o][2] += dt * Force( arr[w][0], arr[o][1] +
qx[o][1] / 2, arr[w][1] + qx[w][1] / 2, arr[o][2] + qy[o][1] / 2,
arr[w][2] + qx[w][1] / 2)[0];
                    qx[o][2] += dt * (arr[o][3] + px[o][1] / 2);
                    py[o][2] += dt * Force( arr[w][0], arr[o][1] +
qx[o][1] / 2, arr[w][1] + qx[w][1] / 2, arr[o][2] + qy[o][1] / 2,
arr[w][2] + qx[w][1] / 2)[1];
                    qy[o][2] += dt * (arr[o][4] + py[o][1] / 2);
                }
            }
        }
        for(int o=0; o<arr.length; o++) {
            for (int w = 0; w < arr.length; w++) {
                if (w != o) {
                    px[o][3] += dt * Force( arr[w][0], arr[o][1] +
qx[o][2] / 2, arr[w][1] + qx[w][2] / 2, arr[o][2] + qy[o][2] / 2,
arr[w][2] + qx[w][2] / 2)[0];
                    qx[o][3] += dt * (arr[o][3] + px[o][2] / 2);
                    py[o][3] += dt * Force( arr[w][0], arr[o][1] +
qx[o][2] / 2, arr[w][1] + qx[w][2] / 2, arr[o][2] + qy[o][2] / 2,
arr[w][2] + qx[w][2] / 2)[1];
                    qy[o][3] += dt * (arr[o][4] + py[o][2] / 2);
                }
            }
        }
        for(int o=0; o<arr.length; o++) {
            for (int w=0; w<arr.length; w++) {
                if (w != o) {
                    px[o][4] += dt * Force( arr[w][0], arr[o][1] +
qx[o][3], arr[w][1] + qx[w][3], arr[o][2] + qy[o][3], arr[w][2] +
qy[w][3])[0];
                    qx[o][4] += dt * (arr[o][3] + px[o][3]);
                    py[o][4] += dt * Force( arr[w][0], arr[o][1] +
```

```
qx[o][3], arr[w][1] + qx[w][3], arr[o][2] + qy[o][3], arr[w][2] +
qy[w][3])[1];
                    qy[o][4] += dt * (arr[o][4] + py[o][3]);
                }
            }
        }
        for(int o=0; o<arr.length; o++) {
            arr[o][3] += (px[o][1] + 2 * px[o][2] + 2 * px[o][3] +
px[o][4]) / 6;
            arr[o][1] += (qx[o][1] + 2 * qx[o][2] + 2 * qx[o][3] +
qx[o][4]) / 6;
            arr[o][4] += (py[o][1] + 2 * py[o][2] + 2 * py[o][3] +
py[o][4]) / 6;
            arr[o][2] += (qy[o][1] + 2 * qy[o][2] + 2 * qy[o][3] +
qy[o][4]) / 6;
        }
        //новая точка кривой кладется в массив
        curve1.add(arr[0][1]);
        curve1.add(arr[0][2]);
        curve2.add(arr[1][1]);
        curve2.add(arr[1][2]);
        curve3.add(arr[2][1]);
        curve3.add(arr[2][2]);
    }
    //здесь запускается программа
    @Override
    public void run(){
        while(true){
            try {
                move(0.0003);
                super.repaint();
                //перерисовка каждый шаг
                Thread.sleep(1);
                //ждем одну миллисекунду
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Анализ правильности решения.

На основе проведенных тестов можно сказать, что программа работает правильно, потому что при ее тестировании на двух телах можно увидеть, что они двигаются по эллипсам, что соответствует одному из решений задачи двух тел в механике. То есть численно был получен тот же самый результат, что и в теории, что подтверждает правильность вычислений и работы программы.