Университет ИТМО
Физико-технический мегафакультет
Физический факультет

Группа М3213 К работе допущен___

Студент Алексеева Виктория Работа выполнена___

Преподаватель Громова Наира Отчет принят_____

Рабочий протокол и отчет по моделированию №1. Вариант 12

Цель работы:

Исследовать движение тела массой m, которое, разгоняясь на горизонтальной плоскости, попадает на вертикально расположенную дугу кольца с радиусом R и угловым размером a, с целью определения минимальной скорости тела, необходимой для прохождения всей дуги. Построить траекторию тела после отрыва от дуги с учётом коэффициента трения μ между телом и дугой.

Задачи, решаемые при выполнении работы:

1. Рассчитать силы, действующие на тело на дуге
2. Рассчитать минимальную скорость, необходимую для преодоления дуги, учитывая трение.
3. Построить траекторию тела после отрыва от дуги, основываясь на вычисленной скорости в точке отрыва.
4. Визуализировать и анимировать движение тела по дуге и его дальнейшую траекторию после отрыва.

Теория:

Начальные данные:

```
double m = 3.0;
double R = 2.0;
double frictionCoeff = 0.02;
double g = 9.81;
```

- Масса тела, m = 3кг
- Радиус дуги, R = 2м
- Коэффициент трения, μ (frictionCoeff)= 0,02
- Гравитационная постоянная, g=9.81м/с^2
- Угловой размер дуги, a = π+ π/3 рад

Формулы, используемые в программе:

1. Сила тяжести: F=mg
2. Сила трения: Fтр = μ*F
3. Длина дуги: L = R*(π+ π/3)
4. Работа силы трения: Атр = Fтр*L
5. Высота: h= R*(1-cos(pi+pi/3))
6. Работа силы тяжести: Атяж= mgh = F*h
7. Скорость, чтобы преодолеть дугу: V = $\sqrt{\frac{2*(Атяж-Атр+Ек)}{m}}$
8. Угол отрыва из дуги: theta = $\pi/6$
9. Ускорение, которое тело будет испытывать вдоль дуги кольца: a=g*cos(theta)
10. Скорость тела во время отрыва от дуги: V = $\sqrt{R * a}$
11. Кинетическая энергия: Ek = mv^2/2
12. Горизонтальная скорость: vx = V*cos(theta)
13. Вертикальная скорость: vy = V*sin(theta)
14. X = x0 + vx * t
15. Y = y0 + vy * t – ½*g*t^2

Ход работы:

1. Тело движется по дуге с радиусом R=2м, и для того, чтобы оно преодолело всю длину дуги, нужно вычислить начальную скорость V0, которая будет обеспечивать движение тела с учетом всех сил, действующих на него.

Найдем работу силы трения:
Атр = Fтр*L
Вычислим силу трения:
Fтр = μ*F = μmg = 0,02 * 3 * 9,81 = 0.5886
Вычислим длину дуги: L = R*(π+ π/3), где π+ π/3 – угловой размер дуги.
L = 2*4π/3 = 8,37758
Атр = 0,5886*8,37758 = 4,93 Дж

```
double F = m * g;
double frictionF = frictionCoeff * F;
double L = R * (Math.PI + Math.PI / 3);
double frictionA = frictionF * L;
```

Найдем работу силы тяжести:
Атяж = F*h = mgh
h= R - R* cos (π+ π/3) = R*(1-cos(π+ π/3))
h = 2*(1-cos(4π/3)) = 3м
Атяж = 3*9,81*3=88,29 Дж

```
double h = R * (1 - Math.Cos(Math.PI + Math.PI / 3));
double Amg = F * h;
```

Найдем ускорение, которое тело будет испытывать вдоль дуги кольца:
a=g*cos(theta) = 9,81*0,866025=8,49571 м/с^2

```
double theta = Math.PI/6;
double a = g * Math.Cos(theta);
```

Найдем нужную нам скорость во время отрыва от дуги:
V = $\sqrt{R * a}$ = 4,12207 м/с

```
double V = Math.Sqrt(R * a);
```

Найдем кинетическую энергию:
Ек = mv^2/2 = 3*4,12207 ^2 / 2 =25,48713

```
double Ek = 0.5 * m * V * V;
```

Найдем начальну скорость, необходимую для прохождения дуги:
V0 = $\sqrt{\frac{2*(Атяж+Атр+Ек)}{m}}$ = $\sqrt{\frac{2*(88,29-4,93+25,48713)}{3}}$ = **8,51849 м/с**

```
double V0 = Math.Sqrt(2 * (Ek + Amg - frictionA) / m);
```

2.Теперь найдем траекторию тела после отрыва от дуги.
Вычислим горизонтальную и вертикальную скорости по формулам:

- vx = V*cos(theta)
- vy = V*sin(theta)

```
double vx = V * Math.Cos(theta);
double vy = V * Math.Sin(theta);
```

После обрыва тело начинает полёт с точки, в которой оно покинуло дугу:

```
double x = arcPoints.Last().X;
double y = arcPoints.Last().Y;
```

Эти начальные координаты устанавливают исходную позицию для траектории свободного падения.

Зададим шаг времени (timestep), что позволит рассчитывать координаты x и y в каждый момент времени, а также время (time), которое увеличивается с каждым шагом расчёта и используется для вычисления координат.
Внутри цикла вычисляем новые координаты x и y:

- X = x0 + vx * t

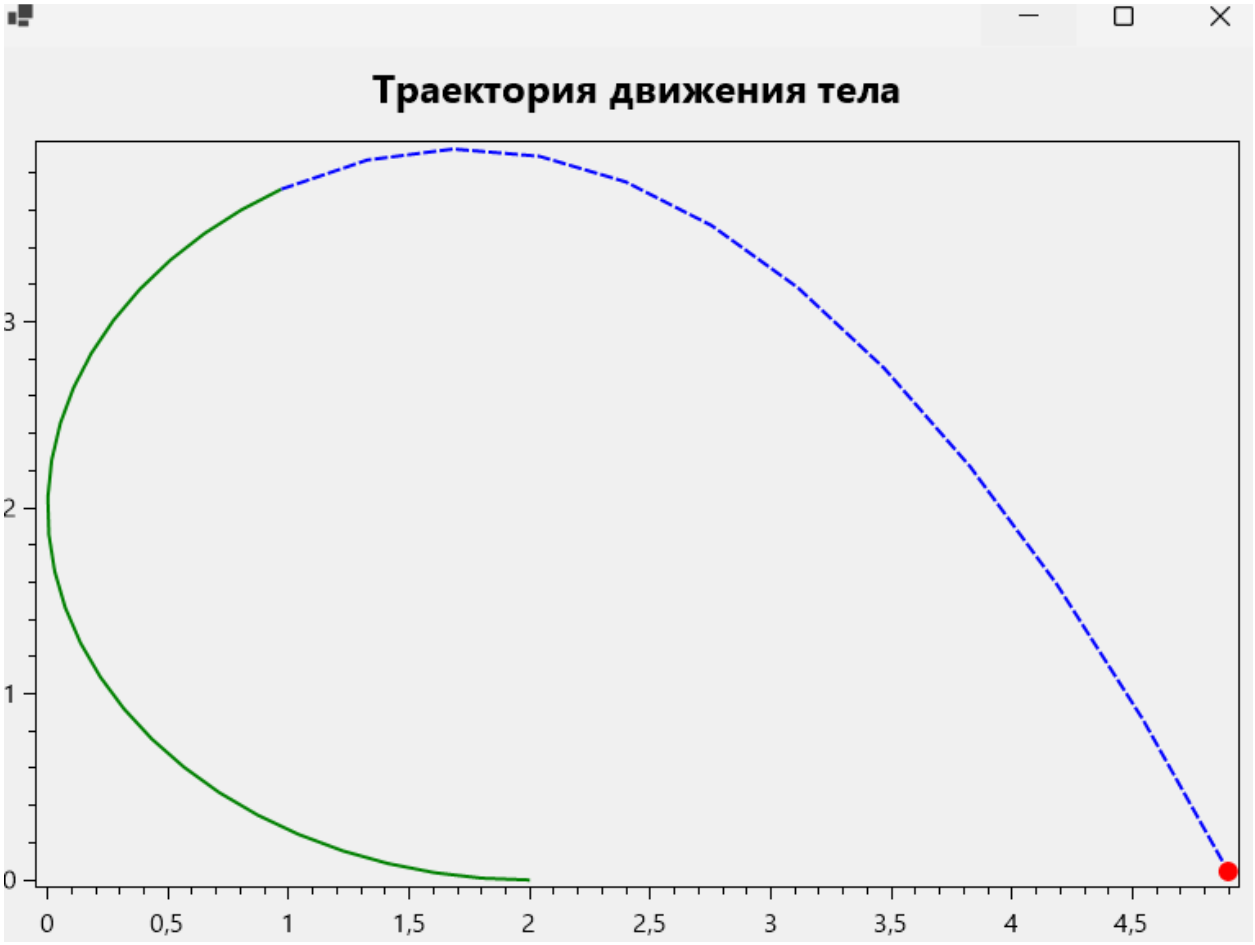
Поскольку на тело не действует горизонтальное ускорение, его горизонтальная скорость остаётся постоянной, и координата x линейно увеличивается со временем.

- Y = y0 + vy * t – ½*g*t^2

Здесь учитывается вертикальная составляющая скорости vy и g, которое замедляет вертикальную скорость и приводит к падению тела.

```
while (y >= 0)
{
    x = arcPoints.Last().X + vx * time;
    y = arcPoints.Last().Y + vy * time - 0.5 * g * time * time;
    if (y >= 0) points.Add(new DataPoint(x, y));
    time += timeStep;
}
```

Визуализация:



Зеленая линия – дуга, синяя пунктирная – траектория падения, красная точка – тело.

Код:

```
using OxyPlot;
using OxyPlot.Series;
using OxyPlot.WindowsForms;
using System.Timers;

namespace PhysicsModeling1
{
    1 usage
    public partial class Form1 : Form
    {
        private PlotView plotView;
        private ScatterSeries movingPointSeries;
        private List<DataPoint> trajectoryPoints;
        private List<DataPoint> arcPoints;
        private int currPoint;
        private System.Timers.Timer timer;

        1 usage
        public Form1()
        {
            Width = 800;
            Height = 600;

            plotView = new PlotView();
            plotView.Dock = DockStyle.Fill;

            arcPoints = CalculateArcPoints();
            trajectoryPoints = CalculateTrajectoryPoints();

            var plotModel = new PlotModel { Title = "Траектория движения тела" };
            plotModel.Series.Add(CreateLineSeries(arcPoints, title: "Дуга", color: OxyColors.Green));
    plotModel.Series.Add(CreateLineSeries(trajectoryPoints, title: "Траектория падения", color: OxyColors.Blue, LineStyle.Dash));

    movingPointSeries = new ScatterSeries
    {
        MarkerType = MarkerType.Circle,
        MarkerSize = 6,
        MarkerFill = OxyColors.Red,
        Title = "Тело"
    };
    plotModel.Series.Add(movingPointSeries);

    plotView.Model = plotModel;
    Controls.Add(plotView);

    timer = new System.Timers.Timer(interval: 100);
    timer.Elapsed += Update;
    timer.Start();
}

1 usage
private List<DataPoint> CalculateArcPoints()
{
    List<DataPoint> points = new List<DataPoint>();
    double R = 2.0;
    double centerX = 2.0;
    double centerY = 2.0;
```

```
    for (double theta = Math.PI / 2; theta <= Math.PI + Math.PI / 3; theta += 0.1)
    {
        double arcX = centerX + R * Math.Cos(theta);
        double arcY = centerY - R * Math.Sin(theta);
        points.Add(new DataPoint(arcX, arcY));
    }
    return points;
}

1 usage
private List<DataPoint> CalculateTrajectoryPoints()
{
    List<DataPoint> points = new List<DataPoint>();

    double m = 3.0;
    double R = 2.0;
    double frictionCoeff = 0.02;
    double g = 9.81;

    double theta = Math.PI/6;
    double a = g * Math.Cos(theta);

    double V = Math.Sqrt(R * a);
    double Ek = 0.5 * m * V * V;

    double F = m * g;
    double frictionF = frictionCoeff * F;
    double L = R * (Math.PI + Math.PI / 3);
```

```
        double h = R * (1 - Math.Cos(Math.PI + Math.PI / 3));
        double Amg = F * h;

        double V0 = Math.Sqrt(2 * (Ek + Amg - frictionA) / m);
        Console.WriteLine(V0);



        double vx = V * Math.Cos(theta);
        double vy = V * Math.Sin(theta);



        double timeStep = 0.1;
        double time = 0;

        double x = arcPoints.Last().X;
        double y = arcPoints.Last().Y;

        while (y >= 0)
        {
            x = arcPoints.Last().X + vx * time;
            y = arcPoints.Last().Y + vy * time - 0.5 * g * time * time;
            if (y >= 0) points.Add(new DataPoint(x, y));
            time += timeStep;
        }

        return points;
    }
```

```
private LineSeries CreateLineSeries(List<DataPoint> points, string title, OxyColor color, LineStyle lineStyle = LineStyle.Solid)
{
    var series = new LineSeries
    {
        Title = title,
        Color = color,
        StrokeThickness = 2,
        LineStyle = lineStyle
    };
    foreach (var point in points)
    {
        series.Points.Add(point);
    }
    return series;
}
1 usage
private void Update(object sender, ElapsedEventArgs e)
{
    if (currPoint < arcPoints.Count)
    {
        var currentArcPoint = arcPoints[currPoint];
        movingPointSeries.Points.Clear();
        movingPointSeries.Points.Add(new ScatterPoint(currentArcPoint.X, currentArcPoint.Y));
    }
    else if (currPoint < arcPoints.Count + trajectoryPoints.Count)
    {
        var currentTrajectoryPoint = trajectoryPoints[currPoint - arcPoints.Count];
        movingPointSeries.Points.Clear();
```

```
            movingPointSeries.Points.Add(new ScatterPoint(currentTrajectoryPoint.X, currentTrajectoryPoint.Y));
        }
        else
        {
            timer.Stop();
            return;
        }

        plotView.InvalidatePlot(updateData: true);
        currPoint++;
    }
```

Вывод:

В данном моделировании мы определили минимальную скорость, необходимую для прохождения тела по дуге с учётом сил трения и тяжести, и визуализировали траекторию движения по дуге и траекторию полёта тела после выхода с дуги. Эта работа показала, как силы и начальные условия влияют на движение тела на дуге, обеспечивая наглядное понимание кинематики и последующего свободного полёта.