

Лабораторная работа №2

Задача о погоне

Шубнякова Дарья НКНбд-01-22

1. Вводная часть

2. Основная часть

3. Результаты

1. Вводная часть

1.1 Цели и задачи

Ознакомиться с задачей о погоне. Реализовать задачу о погоне на языке Julia и в среде OpenModelica. На море в тумане катер береговой охраны преследует лодку браконьеров. Через определенный промежуток времени туман рассеивается, и лодка обнаруживается на расстоянии 7,1 км от катера. Затем лодка снова скрывается в тумане и уходит прямолинейно в неизвестном направлении. Известно, что скорость катера в 2,4 раза больше скорости браконьерской лодки. 1. Запишите уравнение, описывающее движение катера, с начальными условиями для двух случаев (в зависимости от расположения катера относительно лодки в начальный момент времени). 2. Постройте траекторию движения катера и лодки для двух случаев. 3. Найдите точку пересечения траектории катера и лодки

2. Основная часть

2.1 Выполнение лабораторной работы

2.1.1 Прописываем код в оболочке Julia 1.11.

[14]:

```
using Plots

# Параметры задачи
v = 1.0      # Скорость лодки
n = 2.4      # Отношение скоростей (катер/лодка)
k = 7.1      # Начальное расстояние (км)
boat_angle = 60 * 3.14159/180 # Угол движения лодки (60 градусов в радианах)

# Простое численное решение дифференциального уравнения методом Эйлера
function solve_pursuit_simple(initial_angle, initial_radius, n, num_steps=2000)
    angles = Float64[]
    radii = Float64[]

    push!(angles, initial_angle)
    push!(radii, initial_radius)

    # Шаг интегрирования
    d_angle = 0.01

    for i in 2:num_steps
        new_angle = angles[end] + d_angle
        # dr/dangle = r/sqrt(n^2-1)
        new_radius = radii[end] + (radii[end] / sqrt(n^2 - 1)) * d_angle

        push!(angles, new_angle)
        push!(radii, new_radius)

        # Останавливаемся, если радиус стал слишком большим
        if new_radius > 30.0 || length(angles) >= num_steps
            break
        end
    end
end
```

2.2 Прописываем нужные нам ф-ии.

```
# Функция для построения графика одного случая
function create_pursuit_plot(initial_angle, initial_radius, case_name, color, max_a
# Решаем уравнение с увеличенным диапазоном углов
angles, radii = solve_pursuit_simple(initial_angle, initial_radius, n, 3000)

# Координаты катера
x_cathet = radii .* cos.(angles)
y_cathet = radii .* sin.(angles)

# Координаты лодки (прямолинейное движение)
t_max = 25.0 # Увеличиваем время движения лодки
time_points = range(0, stop=t_max, length=200)
x_boat = v .* time_points .* cos.(boat_angle)
y_boat = v .* time_points .* sin.(boat_angle)

# Создаем график
plt = plot(size=(600, 500), dpi=300)

# Добавляем траектории
plot!(plt, x_boat, y_boat, label="Траектория лодки", linewidth=2,
      linestyle=:dash, color=:red)
plot!(plt, x_cathet, y_cathet, label="Траектория катера",
      linewidth=2, color=color)
scatter!(plt, [0], [0], label="Точка обнаружения", markersize=6, color=:black)
scatter!(plt, [x_cathet[1]], [y_cathet[1]], label="Начало катера",
      markersize=5, color=color)

# Находим приближительную точку пересечения
min_distance = Inf
intersection_point = (0.0, 0.0)

for i in 1:length(x_cathet)
    for j in 1:length(x_boat)
        dx = x_cathet[i] - x_boat[j]
        dy = y_cathet[i] - y_boat[j]
        distance = sqrt(dx^2 + dy^2)

        if distance < min_distance
            min_distance = distance
            intersection_point = (x_cathet[i], y_cathet[i])
        end
    end
end
end
```

2.3 Строим графики для первого и второго случаев.

```
# Отмечаем точку пересечения только если она достаточно близка
if min_distance < 1.0 # только если расстояние меньше 1 км
    scatter!(plt, [intersection_point[1]], [intersection_point[2]],
             label="Точка пересечения", markersize=8, color=:orange, marker=:star)
    println("$case_name: Точка пересечения ≈ ($(round(intersection_point[1], digits=2), round(intersection_point[2], digits=2))) км"
else
    println("$case_name: Точка пересечения не найдена (минимальное расстояние = $min_distance км)"
end

# Настройки графика
plot!(plt, xlim=(-5, 25), ylim=(-5, 25), aspect_ratio=:equal,
      xlabel="Расстояние, X (км)", ylabel="Расстояние, Y (км)",
      title="Случай: $case_name\n(n=$n, k=$k км)",
      legend=:bottomright)

return plt, intersection_point
end

# Основная программа
println("Решение задачи о погоне")
println("=="^50)

# Случай 1: начальный угол = 0, начальный радиус = k/(n+1)
println("Случай 1: Катер начинает с расстояния $(round(k/(n+1), digits=3)) км")
plt1, point1 = create_pursuit_plot(0.0, k/(n+1), "1", :blue, 4.0)
display(plt1)

# Случай 2: начальный угол = -π, начальный радиус = k
println("\nСлучай 2: Катер начинает с расстояния $k км")
plt2, point2 = create_pursuit_plot(-3.14159, k, "2", :green, 10.0) # Увеличиваем t
display(plt2)
```

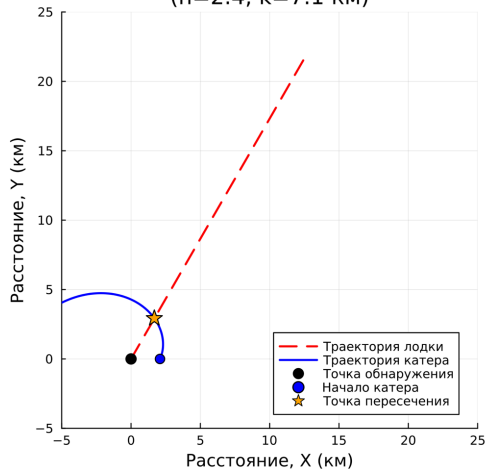

2.4 Получаем такой график для первого случая.

Решение задачи о погоне

Случай 1: Катер начинает с расстояния 2.088 км

1: Точка пересечения $\approx (1.68, 2.93)$ км

Случай: 1
($n=2.4$, $k=7.1$ км)

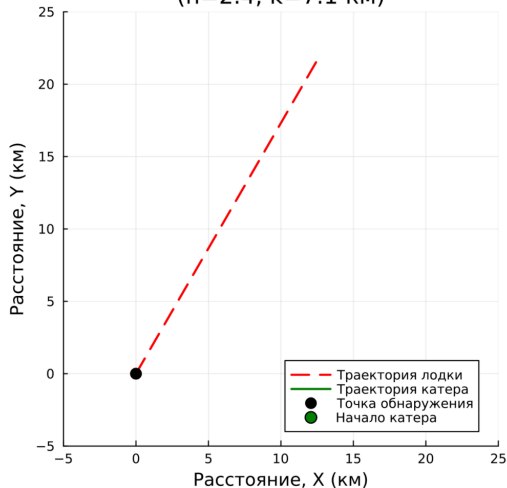


2.5 А во втором случае столкновения не произошло.

Случай 2: Катер начинает с расстояния 7.1 км

2: Точка пересечения не найдена (минимальное расстояние = 7.1 км)

Случай: 2
($n=2.4$, $k=7.1$ км)



2.6 Прописываем код на языке Modelica в OMEdit для первого случая.

```
model PursuitCase1 "Погоня - Случай 1: катер ближе"
import Modelica.Constants.pi;
import Modelica.Math.Vectors.length;

parameter Real v = 1 "Скорость лодки, км/мин";
parameter Real n = 2.4 "Отношение скоростей";
parameter Real k = 7.1 "Начальное расстояние, км";
parameter Real phi = pi/3 "Курс лодки, 60 градусов";

// Случай 1:  $\theta_0 = 0$ ,  $r_0 = k/(n+1) = 2.088$  км
Real theta(start = 0, fixed = true);
Real r(start = k/(n+1), fixed = true);

// Координаты для построения траектории
Real x_cathet, y_cathet; // Координаты катера
Real x_boat, y_boat; // Координаты лодки
Real t "Время"; // Переименовали time -> t

// Расстояние между катером и лодкой
Real distance;

equation
// Дифференциальные уравнения движения катера
der(r) = v;
r * der(theta) = v * sqrt(n^2 - 1);

// Координаты в декартовой системе
x_cathet = r * cos(theta);
y_cathet = r * sin(theta);
x_boat = v * t * cos(phi);
y_boat = v * t * sin(phi);

// Расстояние между катером и лодкой
distance = sqrt((x_boat - x_cathet)^2 + (y_boat - y_cathet)^2);

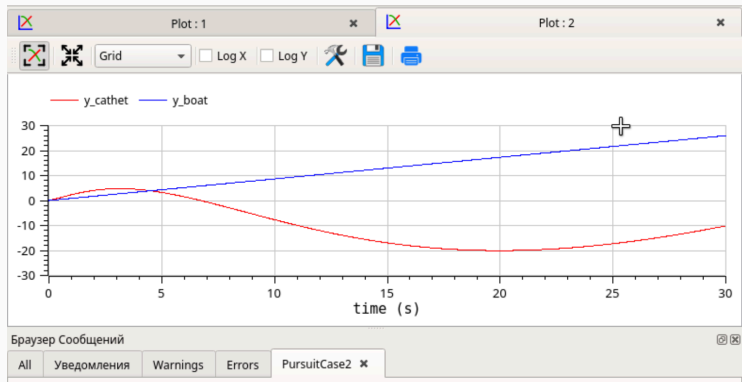
der(t) = 1;
```

2.7 Продолжение кода

```
// Остановка при перехвате
when distance < 0.01 then
  terminate("Катер нагнал лодку! Время: " + String(t) +
    ", Координаты: (" + String(x_cathet) + ", " + String(y_cathet) + ")");
end when;

annotation(experiment(StartTime=0, StopTime=15, Tolerance=1e-6, Interval=0.01));
end PursuitCase1;
```

2.8 Видим пересечение на графике примерно там же, что и при моделировании на Julia



2.9 Прописываем код для второго случая

```
model PursuitCase2 "Погоня - Случай 2: катер дальше"
import Modelica.Constants.pi;
import Modelica.Math.Vectors.length;

parameter Real v = 1 "Скорость лодки, км/мин";
parameter Real n = 2.4 "Отношение скоростей";
parameter Real k = 7.1 "Начальное расстояние, км";
parameter Real phi = pi/3 "Курс лодки, 60 градусов";

// Случай 2:  $\theta_0 = -\pi$ ,  $r_0 = k = 7.1$  км
Real theta(start = -pi, fixed = true);
Real r(start = k, fixed = true);

// Координаты для построения траектории
Real x_cathet, y_cathet; // Координаты катера
Real x_boat, y_boat; // Координаты лодки
Real t "Время"; // Переименовали time -> t

// Расстояние между катером и лодкой
Real distance;

equation
// Дифференциальные уравнения движения катера
der(r) = v;
r * der(theta) = v * sqrt(n^2 - 1);

// Координаты в декартовой системе
x_cathet = r * cos(theta);
y_cathet = r * sin(theta);
x_boat = v * t * cos(phi);
y_boat = v * t * sin(phi);

// Расстояние между катером и лодкой
distance = sqrt((x_boat - x_cathet)^2 + (y_boat - y_cathet)^2);

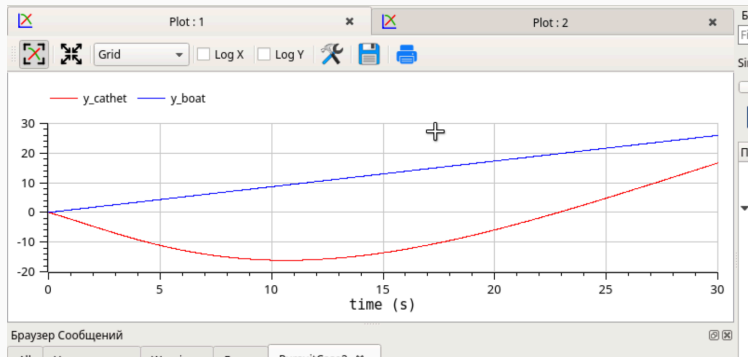
der(t) = 1;
```

2.10 Продолжение кода

```
// Остановка при перехвате
when distance < 0.01 then
    terminate("Катер нагнал лодку! Время: " + String(t) +
        ", Координаты: (" + String(x_cathet) + ", " + String(y_cathet) + ")");
end when;

annotation(experiment(StartTime=0, StopTime=25, Tolerance=1e-6, Interval=0.01));
end PursuitCase2;
```

2.11 В итоге получаем график, который опять же иллюстрирует отсутствие пересечений



3. Результаты

3. Результаты

Мы решили задачу о погоне. Получили на выходе два файла из OpenModelica: `first.occ`, `secondocc.mo`. Получили две картинки с графиками: `pursuit_case1.png`, `pursuit_case2.png`, а так же код в оболочке Julia, прописанный в JupiterNotebook.