

# **Отчёт по лабораторной работе №2**

**Задача о погоне**

Латыпова Диана. НФИбд-02-21

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Ход лабораторной работы</b>	<b>11</b>
<b>6</b>	<b>Выводы</b>	<b>16</b>
	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

5.1	Установка Julia . . . . .	11
5.2	Запуск Julia . . . . .	11
5.3	Пакеты Julia . . . . .	12

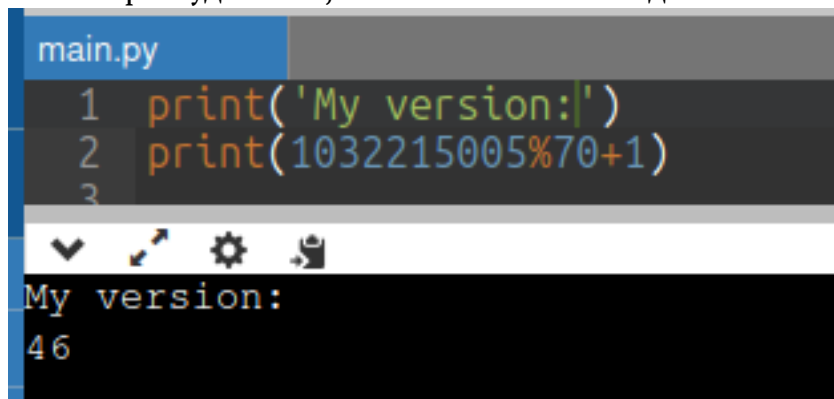
## **Список таблиц**

# 1 Цель работы

Реализовать задачу о погоне. Изучить язык программирования Julia.

## 2 Задание

По формуле из ТУИСа я выявила свой вариант:  $(S_n \bmod N) + 1$ , где  $S_n$  — номер студбилета,  $N$  — количество заданий. Мой вариант - 46. (рис. ??)



```
main.py
1 print('My version:|')
2 print(1032215005%70+1)
3
My version:
46
```

### *Условие задачи о погоне*

На море в тумане катер береговой охраны преследует лодку браконьеров. Через определенный промежуток времени туман рассеивается, и лодка обнаруживается на расстоянии 16,5 км от катера. Затем лодка снова скрывается в тумане и уходит прямолинейно в неизвестном направлении. Известно, что скорость катера в 4,3 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

### 3 Теоретическое введение

*Задача о погоне* - это классическая задача оптимизации, в которой группа охотников пытается поймать добычу (например, зайца), который быстрее их. Они могут двигаться с разной скоростью, и их цель - сделать так, чтобы добыча никогда не смогла достичь безопасного убежища. *Цель задачи* о погоне состоит в том, чтобы найти оптимальные стратегии для охотников, чтобы они смогли захватить добычу, прежде чем та достигнет безопасного места. Одной из возможных метрик успеха может быть время, затраченное на погоню, или расстояние, пройденное добычей до ее поимки. *Решение задачи* о погоне часто включает в себя использование алгоритмов оптимизации или алгоритмов поиска, чтобы найти оптимальные стратегии для охотников. Это может включать в себя моделирование движения добычи и охотников, определение оптимальных путей и принятие решений на основе текущих условий.

В контексте программирования задачу о погоне можно решить, используя различные методы оптимизации, такие как генетические алгоритмы, методы градиентного спуска или алгоритмы поиска в пространстве состояний. Эти методы могут быть реализованы на языках программирования, таких как Python, Julia или MATLAB, и использоваться для симуляции и анализа различных стратегий охотников и добычи[1].

*Julia* - это высокопроизводительный динамический язык программирования, разработанный для численных и научных вычислений. Он объединяет простоту и выразительность Python с производительностью компилируемых языков, таких как C и Fortran. Julia позволяет писать чистый и читаемый код, который

работает быстро[2].

Особенности:

- Динамическая типизация: Julia является языком с динамической типизацией, что означает, что типы переменных определяются во время выполнения программы, а не во время компиляции.
- Высокая производительность: Благодаря специализированному компилятору, Julia предоставляет высокую производительность, сравнимую с языками низкого уровня, такими как C и Fortran.
- Многопоточность: Julia поддерживает параллельные вычисления и многопоточность, что позволяет эффективно использовать многопроцессорные системы.
- Широкие возможности: Julia имеет обширную стандартную библиотеку и активное сообщество разработчиков, что обеспечивает широкие возможности для научных вычислений, статистики, машинного обучения и других областей.

Команды для установки последней версии Julia на Linux:

```
wget https://julialang-s3.julialang.org/bin/linux/x64/1.10/julia-1.10.1-linux-x86_64.tar.gz
tar zxvf julia-1.10.1-linux-x86_64.tar.gz
```

Запуск Julia: `/bin/julia`

В нашем случае: `julia-1.10.1/bin/julia`

Для нашей задачи о полигоне понадобятся пакеты `Plots` и `DifferentialEquations`.

Запустив Julia, для установки пакетов нужно ввести:

```
using Pkg
Pkg.add("Plots")
Pkg.add("DifferentialEquations")
```



## 4 Выполнение лабораторной работы

### *Построение математической модели*

1. Начнем с момента обнаружения лодки браконьеров, когда туман рассеялся. Вводим полярные координаты, где точка обнаружения лодки - полюс, а ось проходит через береговую охрану. Таким образом, координаты катера(16,5; 0)
2. Находим расстояние, при котором катер начнет двигаться вокруг полюса. Это происходит, когда катер и лодка находятся на одинаковом расстоянии от полюса.
3. Составляем систему уравнений, учитывая скорость движения лодки браконьеров и время, которое катер и лодка проводят на одинаковом расстоянии от полюса. За время  $t$  лодка пройдет  $x$ , а катер береговой охраны  $16.5 - x$ . Примем скорость лодки браконьеров за  $v$ . Следовательно время будет равно  $\frac{x}{v}$  для лодки и  $\frac{16.5-x}{4.3v}$  или  $\frac{16.5+x}{4.3v}$  для катера. Учитывая, что время должно быть равно, получается:

$$\begin{cases} \frac{x}{v} = \frac{16.5 - x}{4.3v} \\ \frac{x}{v} = \frac{16.5 + x}{4.3v} \end{cases}$$

Решив систему, мы получили два значения  $x$ :  $x_1 = \frac{165}{53}$ , а  $x_2 = 5$

4. Когда катер достигает той же дистанции от полюса, что и лодка, он начинает движение вокруг полюса, удаляясь от лодки со скоростью  $v$ . Скорость  $v$  раскладывается на 2 значения:  $v_r = \frac{dr}{dt}$  - радиальная скорость и  $v_\tau = r * \frac{d\theta}{dt}$  - тангенциальная скорость.

5. Формулируем систему дифференциальных уравнений для определения радиальной и тангенциальной скоростей катера относительно полюса. Первое уравнение у нас уже есть:  $v_r = \frac{dr}{dt}$ . Второе уравнение мы найдем из разложения скорости на две составляющие с помощью теоремы Пифагора:

$$v_\tau = \sqrt{(4.3v)^2 - v_r^2} = \sqrt{18.49v^2 - v^2} = \frac{\sqrt{1749}v}{10}$$

Следовательно второе уравнение выглядит так:  $r * \frac{d\theta}{dt} = \frac{\sqrt{1749}v}{10}$

Тогда система уравнений получается:

$$\begin{cases} \frac{dr}{dt} = v \\ r * \frac{d\theta}{dt} = \frac{\sqrt{1749}v}{10} \end{cases}$$

С начальными условиями:

(для первого случая)

$$\begin{cases} \theta = 0 \\ r_0 = \frac{165}{53} \end{cases}$$

(для второго случая)

$$\begin{cases} \theta = -\pi \\ r_0 = 5 \end{cases}$$

Путем математических манипуляций приводим систему к такому виду:

$$\frac{dr}{d\theta} = \frac{10r}{\sqrt{1749}}$$

Математическая модель готова.

## 5 Ход лабораторной работы

Для начала я установила Julia на Linux(рис. 5.1).

```
dlatypova@linux:~$ wget https://julialang-s3.julialang.org/bin/linux/x64/1.10/julia-1.10.1-linux-x86_64.tar.gz
--2024-02-17 18:06:32-- https://julialang-s3.julialang.org/bin/linux/x64/1.10/julia-1.10.1-linux-x86_64.tar.gz
Распознаётся julialang-s3.julialang.org (julialang-s3.julialang.org)... 151.101.246.49, 2a04:4e42:3a::561
Подключение к julialang-s3.julialang.org (julialang-s3.julialang.org)|151.101.246.49|:443...
. соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: 167854807 (160M) [application/x-tar]
Сохранение в: 'julia-1.10.1-linux-x86_64.tar.gz'

julia-1.10.1-linux-x86 100%[=====] 160,08M  10,8MB/s   за 15s

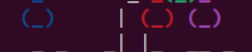
2024-02-17 18:06:47 (10,9 MB/s) - 'julia-1.10.1-linux-x86_64.tar.gz' сохранён [167854807/167854807]

dlatypova@linux:~$ tar xzvf julia-1.10.1-linux-x86_64.tar.gz
julia-1.10.1/
julia-1.10.1/etc/
julia-1.10.1/etc/julia/
julia-1.10.1/etc/julia/startup.jl
julia-1.10.1/libexec/
```

Рис. 5.1: Установка Julia

Запустила Julia(рис. 5.2).

```
dlatypova@linux:~$ julia-1.10.1/bin/julia
```



```
| Documentation: https://docs.julialang.org
|
| Type "?" for help, "]?" for Pkg help.
|
| Version 1.10.1 (2024-02-13)
| Official https://julialang.org/ release
|_/_/
```

Рис. 5.2: Запуск Julia

### Подключение пакетов Plots и DifferentialEquations (рис. 5.3).

```
julia> using Plots
julia> using DifferentialEquations
```

Рис. 5.3: Пакеты Julia

Написала код на языке Julia для реализации нашей задачи о погоне:

```
using Plots
using DifferentialEquations

# Расстояние от лодки до катера
const distance_to_kater = 16.5
const ratio = 4.3

# Расстояние начала спирали для движения катера впереди лодки и позади лодки
const r0_forward = distance_to_kater / (ratio + 1)
const r0_backward = distance_to_kater / (ratio - 1)

# Интервал для времени
const T = (0, 2*pi)          # Для движения катера впереди лодки
const T_backward = (-pi, pi) # Для движения катера позади лодки

# Определение функции для уравнений движения
function F(u, p, t)
    return u / sqrt(ratio * ratio - 1)
end

# 1 случай: Катер впереди лодки
problem = ODEProblem(F, r0_forward, T)
result = solve(problem, abstol=1e-8, reltol=1e-8)
```

```

random_index = rand(1:size(result.t)[1]) # Выбираем случайный индекс для отображения
rAngles = [result.t[random_index] for i in 1:size(result.t)[1]] # Углы

# Создание графика для случая 1
plt = plot(proj=:polar, aspect_ratio=:equal, dpi=1000, legend=true, bg=:white)

# Настройка параметров графика
plot!(plt, xlabel="theta", ylabel="r(t)", title="Задача о погоне. 1 случай", legend=:none)
plot!(plt, [rAngles[1], rAngles[2]], [0.0, result.u[size(result.u)[1]]], label="Путь катера",
scatter!(plt, rAngles, result.u, label="", mc=:black, ms=0.0005)
plot!(plt, result.t, result.u, xlabel="theta", ylabel="r(t)", label="Путь катера",
scatter!(plt, result.t, result.u, label="", mc=:purple, ms=0.0005)

savefig(plt, "lab2_1.png")

# 2 случай: Катер позади лодки
problem = ODEProblem(F, r0_backward, T_backward)
result = solve(problem, abstol=1e-8, reltol=1e-8)
random_index = rand(1:size(result.t)[1]) # Выбираем случайный индекс для отображения
rAngles = [result.t[random_index] for i in 1:size(result.t)[1]] # Углы

# Создание графика для случая 2
plt1 = plot(proj=:polar, aspect_ratio=:equal, dpi=1000, legend=true, bg=:white)

# Настройка параметров графика
plot!(plt1, xlabel="theta", ylabel="r(t)", title="Задача о погоне. 2 случай", legend=:none)
plot!(plt1, [rAngles[1], rAngles[2]], [0.0, result.u[size(result.u)[1]]], label="Путь катера",
scatter!(plt1, rAngles, result.u, label="", mc=:black, ms=0.0005)
plot!(plt1, result.t, result.u, xlabel="theta", ylabel="r(t)", label="Путь катера",

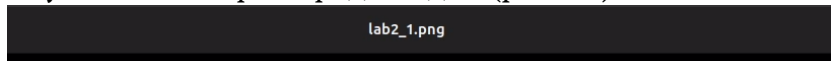
```

```
scatter!(plt1, result.t, result.u, label="", mc=:purple, ms=0.0005)
```

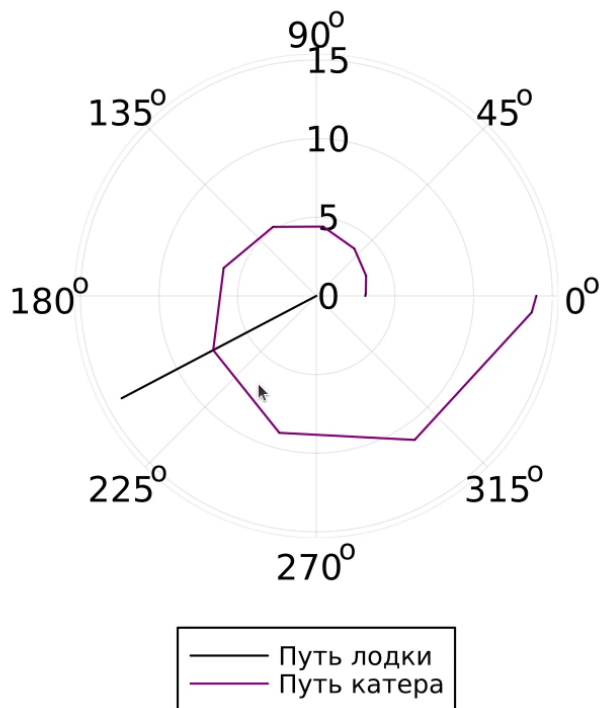
```
savefig(plt1, "lab2_2.png")
```

После запуска кода, сгенерировалось 2 картинки(2 случая)

Случай 1 - Катер впереди лодки (рис. ??):



### Задача о погоне. 1 случай

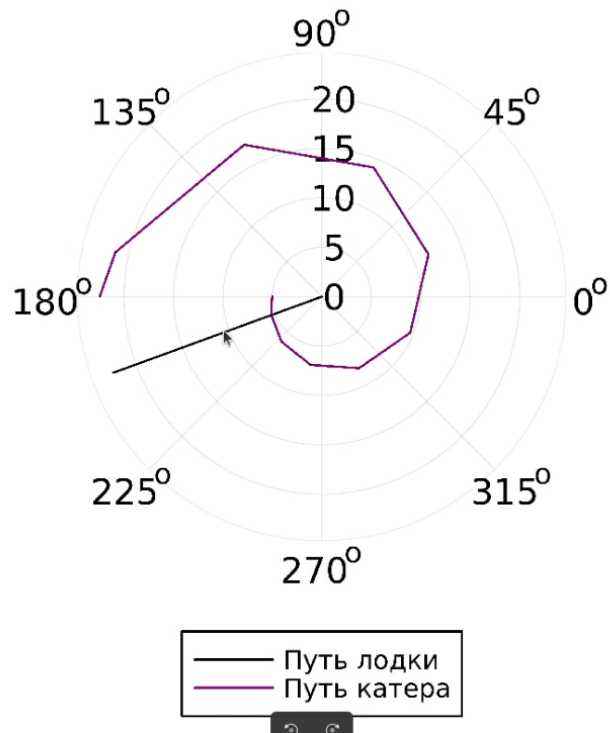


Исходя из случа1 на

картинке мы видим, что полярный радиус равен 7, а полярный угол равен 210 градусов.

Случай 2 - Катер позади лодки (рис. ??):

## Задача о погоне. 2 случай



Исходя из случа2 на картинке мы видим, что полярный радиус равен 5.1, а полярный угол равен 202.5 градусов.

## 6 Выводы

Я реализовала (решила) задачу о погоне варианта 46. Кроме того, теперь знаю основы языка программирования Julia.



## Список литературы

1. Кривая погони [Электронный ресурс]. Wikimedia Foundation, Inc., 2018. URL: [https://ru.wikipedia.org/wiki/Кривая\\_погони](https://ru.wikipedia.org/wiki/Кривая_погони).
2. Julia Documentation [Электронный ресурс]. Documenter.jl; the Julia Programming Language, 2024. URL: <https://docs.julialang.org/en/v1/>.