

Отчет по лабораторной работе № 2

Задача о погоне

Лебедева Ольга Андреевна

Содержание

Цель работы	4
Теоретическое введение	5
Задание	6
Выполнение лабораторной работы	7
Код лабораторной работы	10
Примечание	14
Заключение	16
Библиографическая справка	17

Список иллюстраций

1	Вариант задания	6
1	Установка Julia	7
2	Окно Julia	7
3	Команды по установке библиотеки Plots	8
4	Команды по установке библиотеки Plots	8
5	Установка библиотеки Plots	8
6	Команды по установке библиотеки DifferentialEquations	8
7	Установка библиотеки DifferentialEquations	9
1	Запуск программы	12
2	График 1	12
3	График 2	13

Цель работы

Решить задачу о погоне, используя Julia и OpenModelica. Построить траекторию движения катера и лодки для двух случаев.

Теоретическое введение

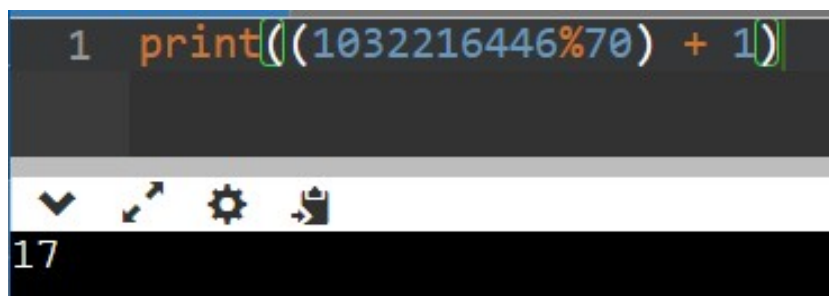
Julia - это высокопроизводительный динамический язык программирования общего назначения, который изначально разрабатывался для решения задач научных вычислений и анализа данных. Julia отличается от других языков программирования своей скоростью выполнения и простотой синтаксиса, что делает его привлекательным выбором для решения широкого спектра задач, включая математическое моделирование, анализ данных, машинное обучение и многое другое.

OpenModelica - это свободная и открытая система для моделирования и симуляции динамических систем. Она предоставляет интегрированную среду разработки, где пользователи могут создавать, редактировать и анализировать модели на основе языка Modelica. OpenModelica поддерживает множество различных областей применения, включая инженерные системы, энергетику, теплофизику и другие.

Задание

Формула определения номера задания: $(S_n \bmod N) + 1$, где S_n — номер студбилета, N — количество заданий.

Для вычисления нашего варианта воспользуемся питоном:(рис. [-@fig:001])

A screenshot of a Python terminal window. The first line shows the command `print((1032216446%70) + 1)` being executed. Below the command line, there is a toolbar with icons for a dropdown menu, a cursor, a gear (settings), and a copy icon. The output of the command, the number 17, is displayed on the line below the toolbar.

```
1 print((1032216446%70) + 1)
```

17

Рис. 1: Вариант задания

Вариант 17

На море в тумане катер береговой охраны преследует лодку браконьеров. Через определенный промежуток времени туман рассеивается, и лодка обнаруживается на расстоянии 7,6 км от катера. Затем лодка снова скрывается в тумане и уходит прямолинейно в неизвестном направлении. Известно, что скорость катера в 2,6 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

Выполнение лабораторной работы

Для выполнения лабораторной работы нам потребуется установка приложения Julia. Для этого скачаем нужную нам версию (для ОС Windows) на официальном сайте. (рис. [-@fig:002])

Platform	64-bit	32-bit
Windows [help]	installer, portable	installer, portable

Рис. 1: Установка Julia

После загрузки и установки наше приложение откроется следующим образом: (рис. [-@fig:003])

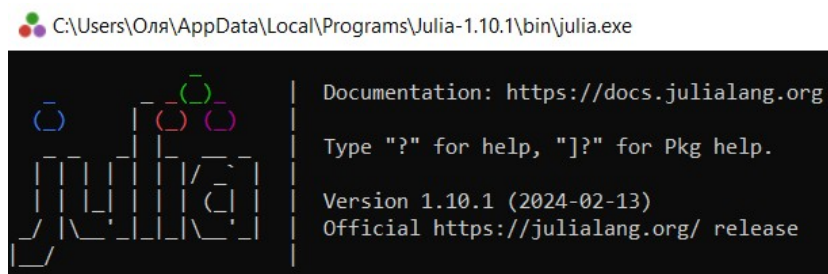


Рис. 2: Окно Julia

Далее, для решения поставленной задачи, необходимо установить несколько библиотек для дальнейшей работы. Библиотека Plots предоставляет простой и гибкий интерфейс для создания графиков, а библиотека (рис. [-@fig:004]) (рис. [-@fig:005]) (рис. [-@fig:006]) (рис. [-@fig:007]) (рис. [-@fig:008])

Install

First, add the package:

```
import Pkg
Pkg.add("Plots")

# if you want the latest features:
Pkg.pkg"add Plots#master"
```

Рис. 3: Команды по установке библиотеки Plots

Initialize

```
using Plots # or StatsPlots
# using GraphRecipes # if you wish to use GraphRecipes package too
```

Рис. 4: Команды по установке библиотеки Plots

```
julia> import Pkg

julia> Pkg.add("Plots")
  Resolving package versions...
  No Changes to `C:\Users\Оля\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\Оля\.julia\environments\v1.10\Manifest.toml`
  Precompiling project...
  106 dependencies successfully precompiled in 511 seconds. 209 already precompiled.

julia> using Plots
```

Рис. 5: Установка библиотеки Plots

Installing from Julia

To install the package, use the following command inside the Julia REPL:

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

To load the package, use the command:

```
using DifferentialEquations
```

Рис. 6: Команды по установке библиотеки DifferentialEquations


```
julia> Pkg.add("DifferentialEquations")
    Resolving package versions...
    No Changes to `C:\Users\Оля\.julia\environments\v1.10\Project.toml`
    No Changes to `C:\Users\Оля\.julia\environments\v1.10\Manifest.toml`
julia> using DifferentialEquations
```

Рис. 7: Установка библиотеки DifferentialEquations

Код лабораторной работы

```
using Plots
using DifferentialEquations
```

Подключаем библиотеки для дальнейшей работы.

```
const a, n = 7.6, 2.6
const r0, r0_2 = a / (n + 1), a / (n - 1)
const T, T_2 = (0, 2 * pi), (-pi, pi)
```

Определение констант и начальных условий. Задаём константы a и n из условия задачи.

r_0 и r_{0_2} — начальные значения радиуса для двух разных задач ОДУ, рассчитанные на основе a и n .

T и T_2 — временные интервалы для решения каждой из задач.

```
F(u, p, t) = u / sqrt(n^2 - 1)
```

Определение Функции Для ОДУ

F — функция, определяющая ОДУ, где u — зависимая переменная (радиус), t — независимая переменная (время), и p — параметры уравнения, которые в данном случае не используются. Уравнение описывает изменение радиуса со временем.

```
problem1 = ODEProblem(F, r0, T)
result1 = solve(problem1, abstol=1e-8, reltol=1e-8)
```

```
problem2 = ODEProblem(F, r0_2, T_2)
result2 = solve(problem2, abstol=1e-8, reltol=1e-8)
```

Создаются две задачи ОДУ (problem1 и problem2) с использованием функции F, начальных значений и временных интервалов. solve решает эти задачи с заданными абсолютной (abstol) и относительной (reltol) точностями.

```
plt1 = plot(proj=:polar, aspect_ratio=:equal, dpi=1000, legend=true, bg=:white)
plot!(plt1, result1.t, result1.u, label="Путь катера", color=:red, lw=1, linestyle=
```

```
max_radius1 = maximum(result1.u)
random_angle1 = result1.t[rand(1:length(result1.t))]
plot!(plt1, [0, random_angle1], [0, max_radius1], label="Путь лодки", color=:magent
```

Создаем полярный график с равным соотношением сторон, высоким разрешением, легендой и белым фоном. Добавляем на график plt1 кривую, представляющую решение result1 (путь катера), с красной пунктирной линией.

Вычисляем максимальный радиус из решения и выбирает случайный угол для иллюстрации предполагаемого пути лодки. Добавляем линию для пути лодки с штрихпунктирным стилем.

```
plt2 = plot(proj=:polar, aspect_ratio=:equal, dpi=1000, legend=true, bg=:white)
plot!(plt2, result2.t, result2.u, label="Путь катера", color=:red, lw=1, linestyle=
```

```
max_radius2 = maximum(result2.u)
random_angle2 = result2.t[rand(1:length(result2.t))]
plot!(plt2, [0, random_angle2], [0, max_radius2], label="Путь лодки", color=:magent
```

Аналогичные шаги выполняются для второй задачи (plt2).

```
savefig(plt1, "lab02_01.png")
savefig(plt2, "lab02_02.png")
```

Сохраняем созданные графики в файлы PNG для дальнейшего использования или анализа.

Теперь мы переходим в консоль Windows и запускаем наш файл. (рис. [-@fig:009])



Рис. 1: Запуск программы

В результате выполнения в той же папке, где находилась наша программа, появится два графика. (рис. [-@fig:010]) (рис. [-@fig:011])

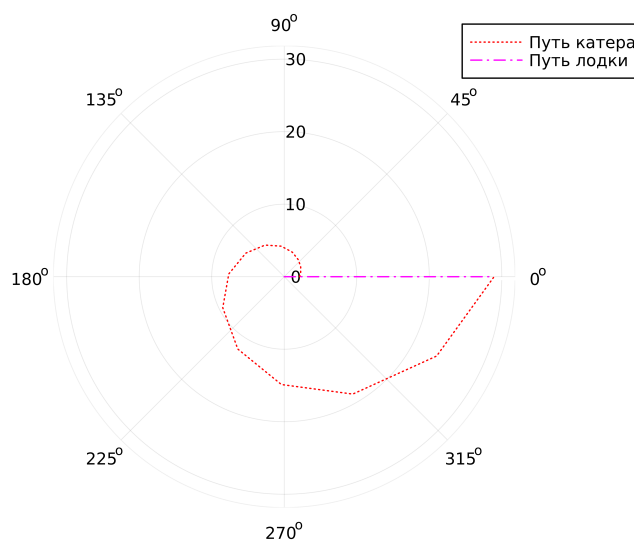


Рис. 2: График 1

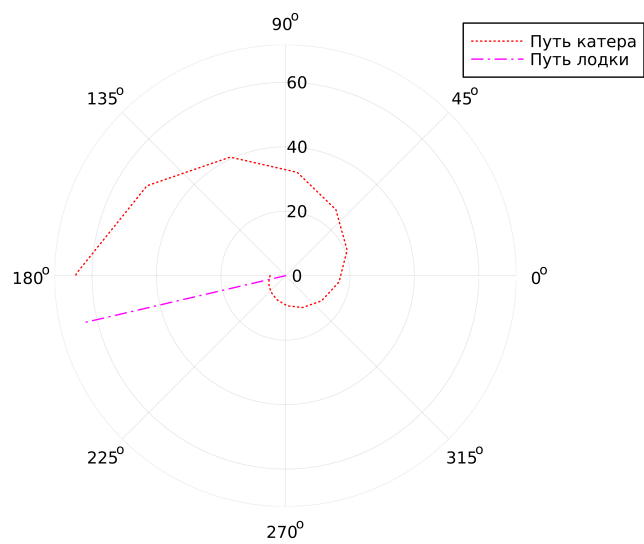


Рис. 3: График 2

Примечание

К сожалению, OpenModelica не предоставляет графики для полярных координат. Но мы можем написать реализацию программы.

```
model Problem1
  constant Real a = 7.6;
  constant Real n = 2.6;
  constant Real r0 = a / (n + 1);
  constant Real T_start = 0;
  constant Real T_end = 2 * Modelica.Constants.pi;
  Real u;

  equation
    der(u) = u / sqrt(n^2 - 1);

  initial equation
    u = r0;

end Problem1;
```

```
model Problem2
  constant Real a = 7.6;
  constant Real n = 2.6;
  constant Real r0_2 = a / (n - 1);
```

```
constant Real T_start = -Modelica.Constants.pi;  
constant Real T_end = Modelica.Constants.pi;  
Real u;  
  
equation  
der(u) = u / sqrt(n^2 - 1);  
  
initial equation  
u = r0_2;  
  
end Problem2;
```

Заключение

Познакомились с приложениями Julia и OpenModelica. Реализовали задачу о погоне, вывели траектории при помощи графиков.

Библиографическая справка

[1] Документация по Julia: <https://docs.julialang.org/en/v1/>

[2] Документация по OpenModelica: <https://openmodelica.org/>