# Отчёт по лабораторной работе

**Лабораторная работа №4 (вариант 10)**

Сергеев Тимофей Сергеевич

# Содержание

**1 Цель работы** **6**

**2 Задание** **7**

**3 Теоретическое введение** **8**

**4 Выполнение лабораторной работы** **9**

**5 Выводы** **25**

**Список литературы** **26**

# Список иллюстраций


4.1 Реализация первого случая на языке Modelica . . . . . 9
4.2 Результат первого случая на языке Modelica . . . . . 10
4.3 Фазовый портрет первого случая на языке Modelica . . . . . 10
4.4 Реализация второго случая на языке Modelica . . . . . 11
4.5 Результат второго случая на языке Modelica . . . . . 11
4.6 Фазовый портрет второго случая на языке Modelica . . . . . 12
4.7 Реализация третьего случая на языке Modelica . . . . . 12
4.8 Результат третьего случая на языке Modelica . . . . . 13
4.9 Фазовый портрет третьего случая на языке Modelica . . . . . 13
4.10 Подключаем библиотеки, задаём функцию, решающую дифференциальные уравнения. Затем зададим начальные условия. . . . . 14
4.11 Выполним функцию с данными значениями. Создадим два пустых массива, в которые мы передадим полученные значения. Затем с помощью функционала библиотеки Plots создадим поле для вывода результата. . . . . 14
4.12 Выведем на экран полученные графы и сохраним результат в формате png . . . . . 15
4.13 Результат работы программы для первого случая . . . . . 15
4.14 Меняем только последнюю часть кода . . . . . 16
4.15 Фазовый портрет для первого случая . . . . . 16
4.16 Подключаем библиотеки, задаём функцию, решающую дифференциальные уравнения. Затем зададим начальные условия. . . . . 17
4.17 Выполним функцию с данными значениями. Создадим два пустых массива, в которые мы передадим полученные значения. Затем с помощью функционала библиотеки Plots создадим поле для вывода результата. . . . . 17
4.18 Выведем на экран полученные графы и сохраним результат в формате png . . . . . 18
4.19 Результат работы программы для второго случая . . . . . 19
4.20 Меняем только последнюю часть кода . . . . . 19
4.21 Фазовый портрет для второго случая . . . . . 20
4.22 Подключаем библиотеки, задаём функцию, решающую дифференциальные уравнения. Затем зададим начальные условия. . . . . 20
4.23 Выполним функцию с данными значениями. Создадим два пустых массива, в которые мы передадим полученные значения. Затем с помощью функционала библиотеки Plots создадим поле для вывода результата. . . . . 21

4.24 Выведем на экран полученные графы и сохраним результат в формате png . . . . . 22
4.25 Результат работы программы для третьего случая . . . . . 23
4.26 Меняем только последнюю часть кода . . . . . 23
4.27 Фазовый портрет для третьего случая . . . . . 24

# Список таблиц

# 1 Цель работы

Построить фазовый портрет гармонического осциллятора и решение уравнения гармонического осциллятора.

# 2 Задание

- Написать код на языке Julia для случаев:
  - без затуханий и без действий внешней силы;
  - с затуханием и без действий внешней силы;
  - с затуханием и под действием внешней силы;
- Написать код на языке Modelica для случаев:
  - без затуханий и без действий внешней силы;
  - с затуханием и без действий внешней силы;
  - с затуханием и под действием внешней силы;
- Составить отчёт на языке Markdown и сконвертировать его в docx и pdf.
- Подготовить презентацию на языке Markdown и защитить её.

# 3 Теоретическое введение

**Julia** – высокоуровневый высокопроизводительный свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков (например, MATLAB и Octave), однако имеет некоторые существенные отличия. Julia написан на Си, C++ и Scheme. Имеет встроенную поддержку многопоточности и распределённых вычислений, реализованные в том числе в стандартных конструкциях. [1]

**OpenModelica** – свободное открытое программное обеспечение для моделирования, симуляции, оптимизации и анализа сложных динамических систем. Основано на языке Modelica. Активно развивается Open Source Modelica Consortium, некоммерческой неправительственной организацией. Open Source Modelica Consortium является совместным проектом RISE SICS East AB и Линчёпингского университета. [2]

**Гармоническим осциллятором** называется система, способная совершать гармонические колебания. В физике модель гармонического осциллятора играет важную роль, особенно при исследовании малых колебаний систем около положения устойчивого равновесия. Примером таких колебаний в квантовой механике являются колебания атомов в твердых телах, молекулах и т.д.[3]

# 4 Выполнение лабораторной работы

1. Рассмотрим код на языке Modelica для первого случая. Объявим переменные типа Real (потому что это тип с плавающим знаком, наиболее подходящий для решения дифференциальных уравнений). Затем инициализурем x и y, подставив данные из условия. После этого пропишем решение наших дифференциальных уравнений (рис. 4.1).

```
1  model lab4_1
2  Real x, y;
3  Real t=time;
4  initial equation
5  x=2;
6  y=0;
7  equation
8  der(x)= y;
9  der(y)= -7*x;
10 end lab4_1;
```

Рис. 4.1: Реализация первого случая на языке Modelica

2. Затем установим настройки симуляции (начальное (0) и конечное (30) время и шаг (0.05)) и запустим симуляцию. Получим следующий результат (рис. 4.2).

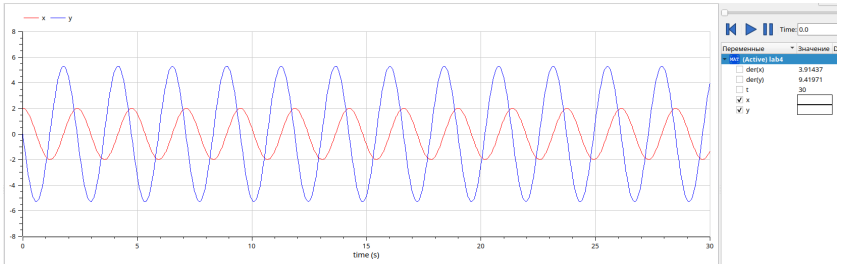



Рис. 4.2: Результат первого случая на языке Modelica

3. Затем необходимо вывести фазовый портрет. В Open Modelica для этого необходимо октрыть новое окно параметрического вывода график, нажать Shift и, удерживая его, поставить галочку у x, после чего поставить галочку у y. В результате на экране появится фазовый портрет (рис. 4.3).

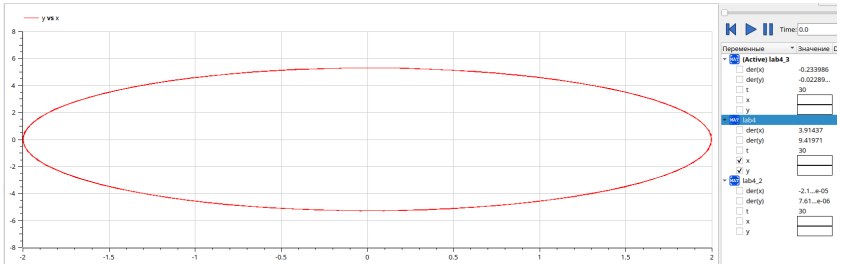



Рис. 4.3: Фазовый портрет первого случая на языке Modelica

4. Аналогичным образом представим код для второго случая (рис. 4.4).

```
1   model lab4_2
2   Real x, y;
3   Real t=time;
4   initial equation
5   x=2;
6   y=0;
7   equation
8   der(x)= y;
9   der(y)= -9*y-3*x;
10  end lab4_2;
```

Рис. 4.4: Реализация второго случая на языке Modelica

5. Затем установим настройки симуляции (начальное (0) и конечное (30) время и шаг (0.05)) и запустим симуляцию. Получим следующий результат (рис. 4.5).

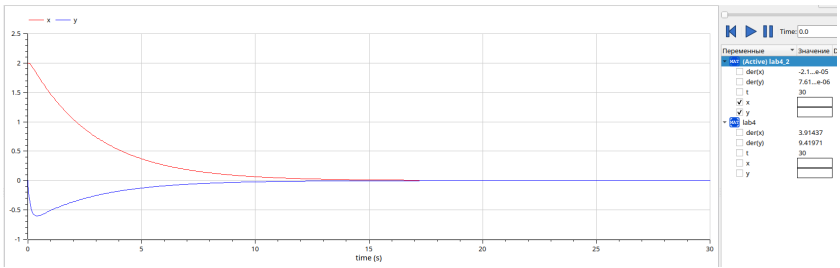



Рис. 4.5: Результат второго случая на языке Modelica

6. Затем выведем фазовый портрет (рис. 4.6).

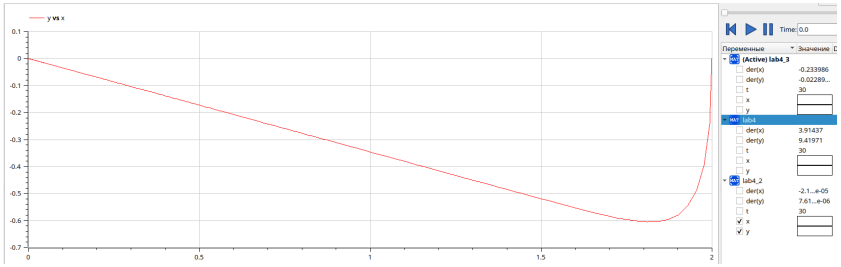


Рис. 4.6: Фазовый портрет второго случая на языке Modelica

7. Аналогичным образом представим код для третьего случая (в отличие от первых двух случаев, добавляется тригонометрическая функция косинуса) (рис. 4.7).

```
1   model lab4_3
2   Real x, y;
3   Real t=time;
4   initial equation
5   x=2;
6   y=0;
7   equation
8   der(x)= y;
9   der(y)= -4*y-x+cos(2*t);
10  end lab4_3;
```

Рис. 4.7: Реализация третьего случая на языке Modelica

8. Затем установим настройки симуляции (начальное (0) и конечное (30) время и шаг (0.05)) и запустим симуляцию. Получим следующий результат (рис. 4.8).

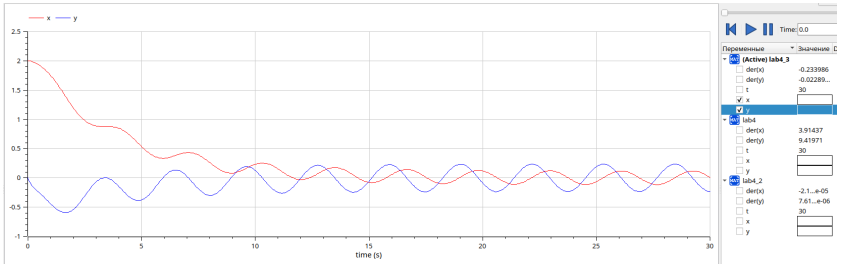



Рис. 4.8: Результат третьего случая на языке Modelica

9. Затем выведем фазовый портрет (рис. 4.9).

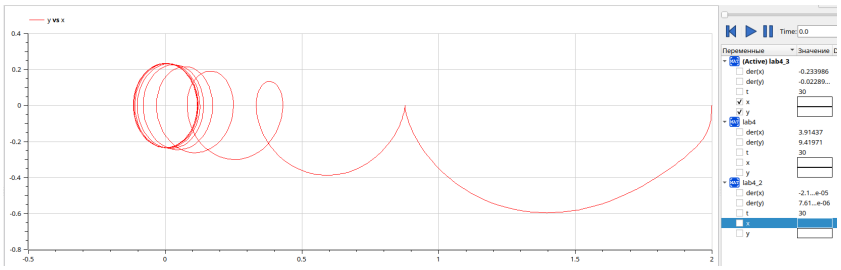
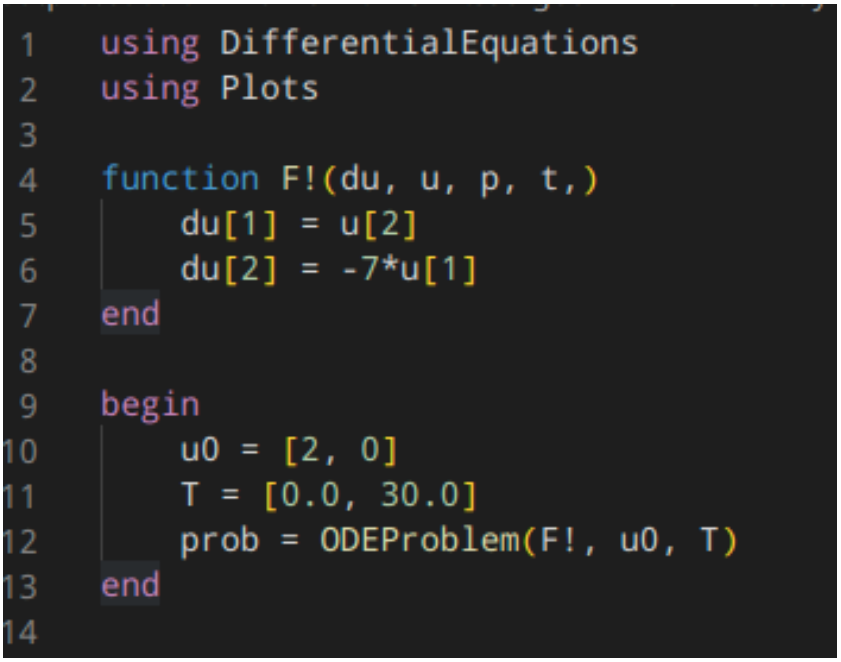



Рис. 4.9: Фазовый портрет третьего случая на языке Modelica

10. Теперь опишем эти случаи на языке Julia. Сперва опишем первый случай (рис. 4.10, 4.11, 4.12).



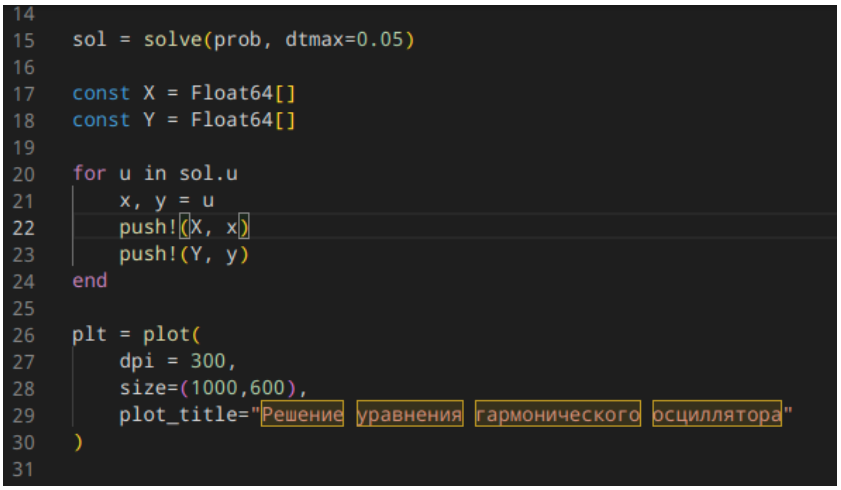
```
using DifferentialEquations
using Plots

function F!(du, u, p, t,)
    du[1] = u[2]
    du[2] = -7*u[1]
end

begin
    u0 = [2, 0]
    T = [0.0, 30.0]
    prob = ODEProblem(F!, u0, T)
end

```

Рис. 4.10: Подключаем библиотеки, задаём функцию, решающую дифференциальные уравнения. Затем зададим начальные условия.



```

sol = solve(prob, dtmax=0.05)

const X = Float64[]
const Y = Float64[]

for u in sol.u
    x, y = u
    push!(X, x)
    push!(Y, y)
end

plt = plot(
    dpi = 300,
    size=(1000,600),
    plot_title="Решение уравнения гармонического осциллятора"
)

```

Рис. 4.11: Выполним функцию с данными значениями. Создадим два пустых массива, в которые мы передадим полученные значения. Затем с помощью функционала библиотеки Plots создадим поле для вывода результата.

```
32  plot!(
33      plt,
34      sol.t,
35      X,
36      color=:blue,
37      label="x(t)"
38  )
39
40  plot!(
41      plt,
42      sol.t,
43      Y,
44      color=:red,
45      label="y(t)"
46  )
47
48  savefig(plt, "lab41.png")
```

Рис. 4.12: Выведем на экран полученные графы и сохраним результат в формате png

11. Получим следующий результат (рис. 4.13).

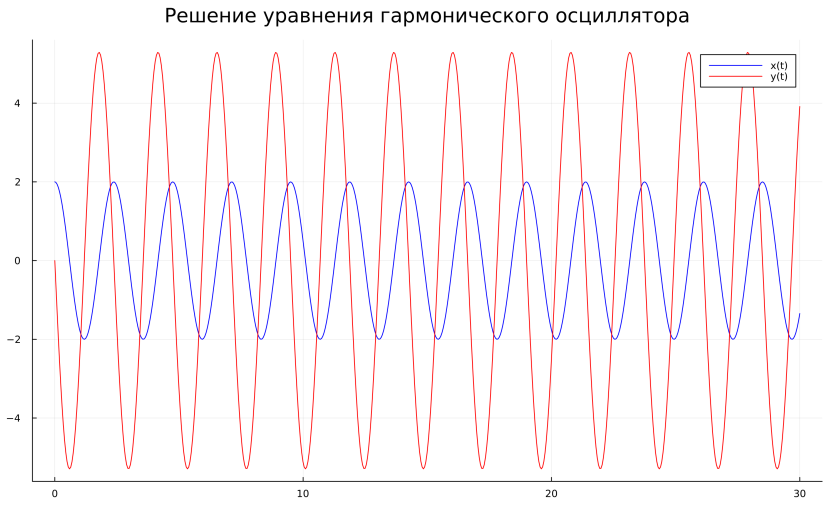



Рис. 4.13: Результат работы программы для первого случая

12. Код для вывода на экран фазового портрета отличается лишь в том, что мы хотим вывести на экран (рис. 4.14).

```
26  plt = plot(
27      dpi = 300,
28      size=(1000,600),
29      plot_title="Решение уравнения гармонического осциллятора"
30  )
31
32  plot!(
33      plt,
34      X,
35      Y,
36      color=:green,
37      label="Фазовый портрет"
38  )
39
40  savefig(plt, "lab41_portret.png")
```

Рис. 4.14: Меняем только последнюю часть кода

13. Получим следующий результат (рис. 4.15).

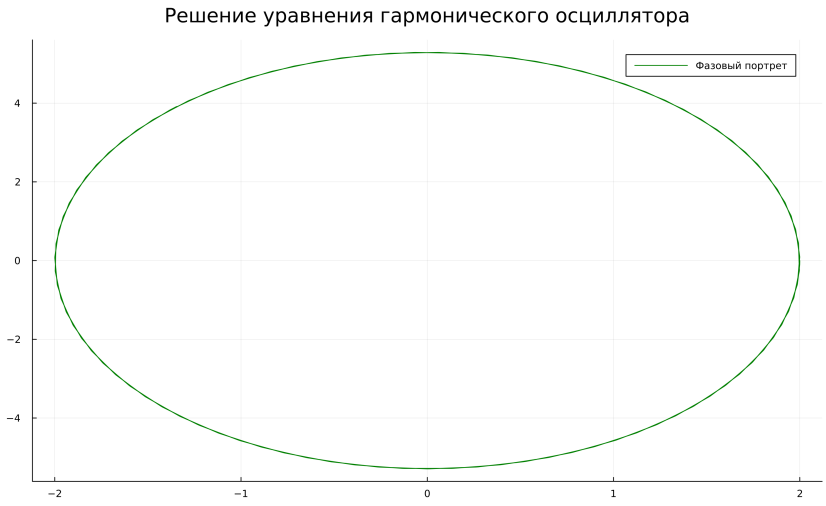
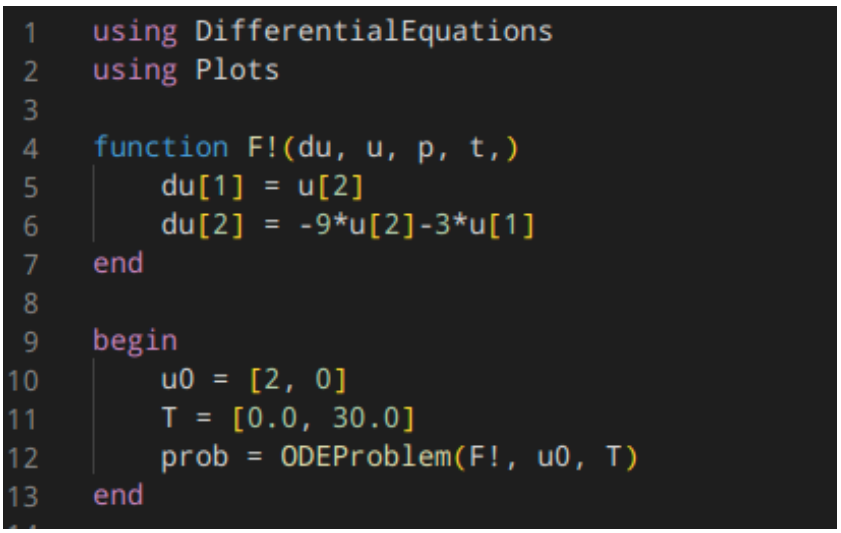



Рис. 4.15: Фазовый портрет для первого случая

14. Теперь опишем второй случай (рис. 4.16, 4.17, 4.18).

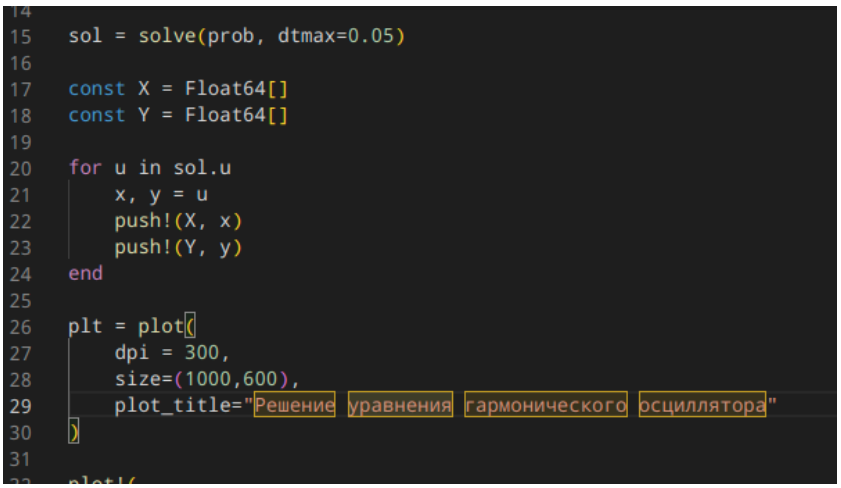


```
using DifferentialEquations
using Plots

function F!(du, u, p, t,)
    du[1] = u[2]
    du[2] = -9*u[2]-3*u[1]
end

begin
    u0 = [2, 0]
    T = [0.0, 30.0]
    prob = ODEProblem(F!, u0, T)
end
```

Рис. 4.16: Подключаем библиотеки, задаём функцию, решающую дифференциальные уравнения. Затем зададим начальные условия.



```
sol = solve(prob, dtmax=0.05)

const X = Float64[]
const Y = Float64[]

for u in sol.u
    x, y = u
    push!(X, x)
    push!(Y, y)
end

plt = plot(
    dpi = 300,
    size=(1000,600),
    plot_title="Решение уравнения гармонического осциллятора"
)
```

Рис. 4.17: Выполним функцию с данными значениями. Создадим два пустых массива, в которые мы передадим полученные значения. Затем с помощью функционала библиотеки Plots создадим поле для вывода результата.

```
32  plot!(
33      plt,
34      sol.t,
35      X,
36      color=:blue,
37      label="x(t)"
38  )
39
40  plot!(
41      plt,
42      sol.t,
43      Y,
44      color=:red,
45      label="y(t)"
46  )
47
48  savefig(plt, "lab42.png")
```

Рис. 4.18: Выведем на экран полученные графы и сохраним результат в формате png

15. Получим следующий результат (рис. 4.19).

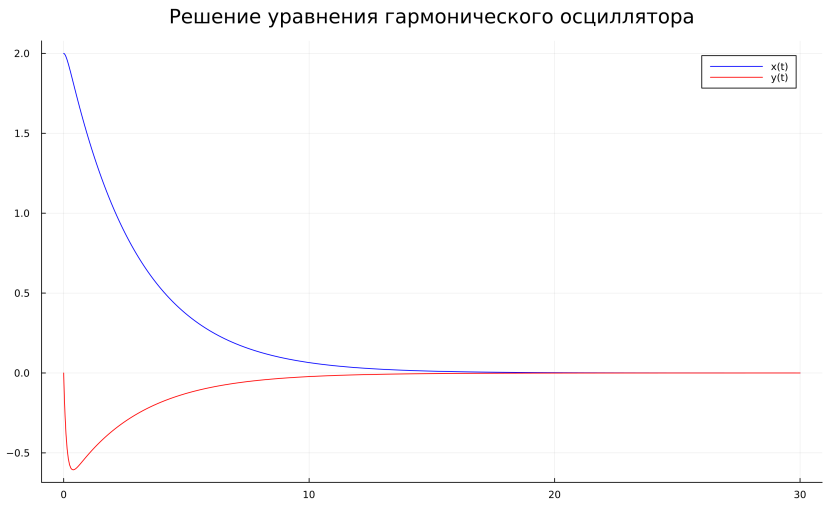



Рис. 4.19: Результат работы программы для второго случая

16. Код для вывода на экран фазового портрета отличается лишь в том, что мы хотим вывести на экран (рис. 4.20).

```
plt = plot(
    dpi = 300,
    size=(1000,600),
    plot_title="Решение уравнения гармонического осциллятора"
)

plot!(
    plt,
    X,
    Y,
    color=:green,
    label="Фазовый портрет"
)

savefig(plt, "lab42_portret.png")
```

Рис. 4.20: Меняем только последнюю часть кода

17. Получим следующий результат (рис. 4.21).

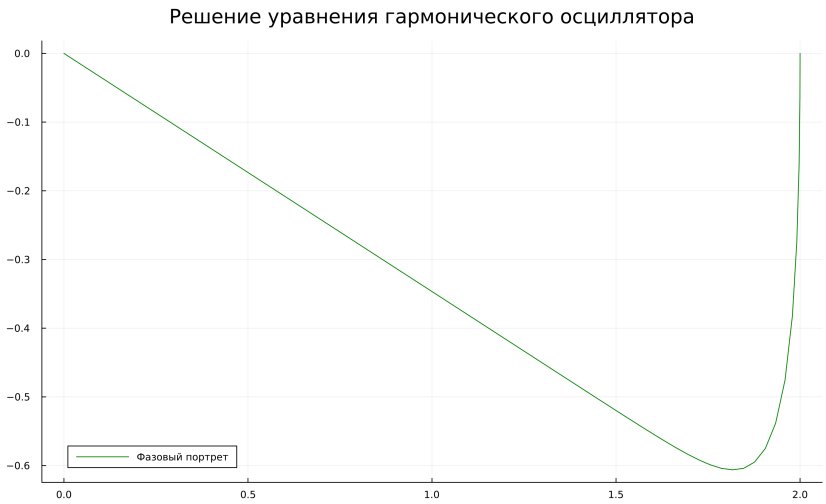
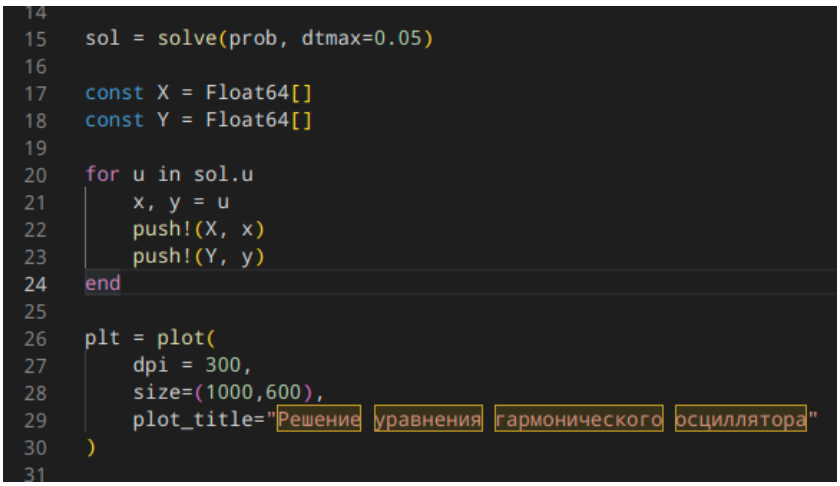



Рис. 4.21: Фазовый портрет для второго случая

14. Теперь опишем третий случай (рис. 4.22, 4.23, 4.24).

```
1   using DifferentialEquations
2   using Plots
3
4   function F!(du, u, p, t,)
5       du[1] = u[2]
6       du[2] = -4*u[2]-u[1]+cos(2*t)
7   end
8
9   begin
10      u0 = [2, 0]
11      T = [0.0, 30.0]
12      prob = ODEProblem(F!, u0, T)
13  end
14
```

Рис. 4.22: Подключаем библиотеки, задаём функцию, решающую дифференциальные уравнения. Затем зададим начальные условия.



```
14
15  sol = solve(prob, dtmax=0.05)
16
17  const X = Float64[]
18  const Y = Float64[]
19
20  for u in sol.u
21      x, y = u
22      push!(X, x)
23      push!(Y, y)
24  end
25
26  plt = plot(
27      dpi = 300,
28      size=(1000,600),
29      plot_title="Решение уравнения гармонического осциллятора"
30  )
31
```

Рис. 4.23: Выполним функцию с данными значениями. Создадим два пустых массива, в которые мы передадим полученные значения. Затем с помощью функционала библиотеки Plots создадим поле для вывода результата.

```
32  plot!(
33      plt,
34      sol.t,
35      X,
36      color=:blue,
37      label="x(t)"
38  )
39
40  plot!(
41      plt,
42      sol.t,
43      Y,
44      color=:red,
45      label="y(t)"
46  )
47
48  savefig(plt, "lab43.png")
```

Рис. 4.24: Выведем на экран полученные графы и сохраним результат в формате png

15. Получим следующий результат (рис. 4.25).

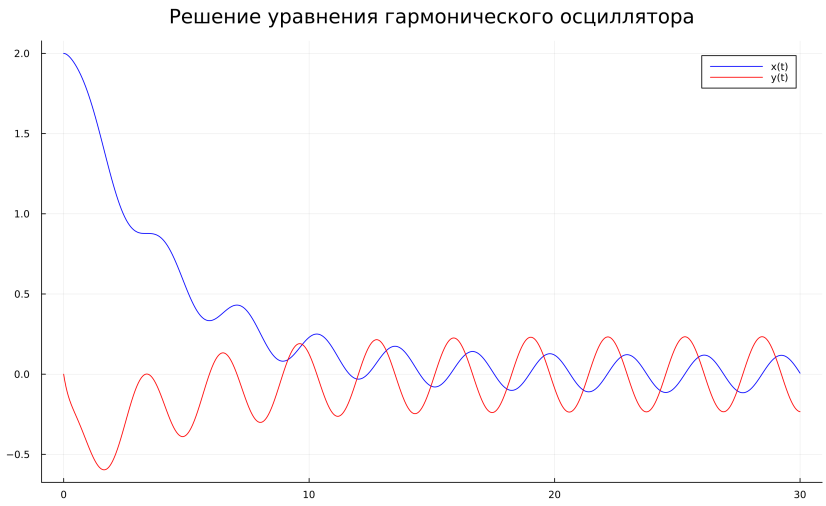



Рис. 4.25: Результат работы программы для третьего случая

16. Код для вывода на экран фазового портрета отличается лишь в том, что мы хотим вывести на экран (рис. 4.26).

```
26  plt = plot(
27      dpi = 300,
28      size=(1000,600),
29      plot_title="Решение уравнения гармонического осциллятора"
30  )
31
32  plot!(
33      plt,
34      X,
35      Y,
36      color=:green,
37      label="Фазовый портрет"
38  )
39
40  savefig(plt, "lab43_portret.png")
```

Рис. 4.26: Меняем только последнюю часть кода

17. Получим следующий результат (рис. 4.27).

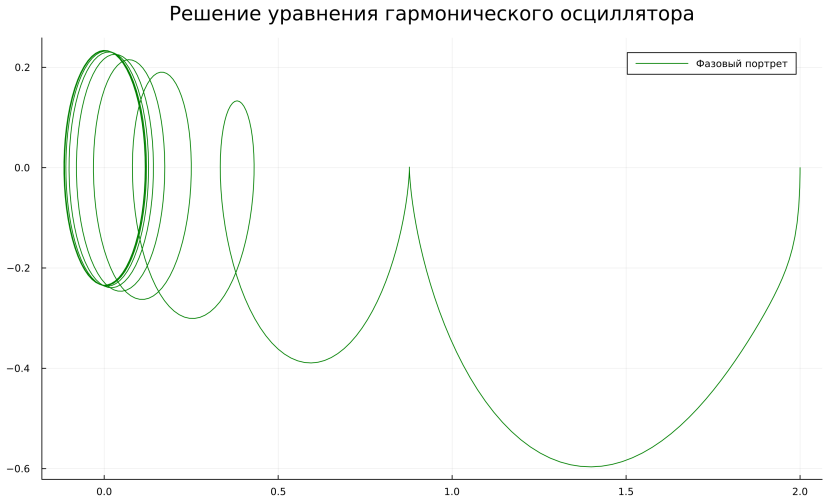



Рис. 4.27: Фазовый портрет для третьего случая

# 5 Выводы

Выполнив данную лабораторную работу, мы продолжили знакомство с языками программирования Julia и Modelica. Сравнивая реализацию одной программы на этих двух языках, можно заметить, что реализация на языке Modelica заметно проще и более точно показывает результат, поскольку можно отследить значения переменных с максимальной точностью на любом отрезке графика.

# Список литературы


1. Julia [Электронный ресурс]. Free Software Foundation, 2023. URL: https://ru.wikipedia.org/wiki/Julia_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).
2. OpenModelica [Электронный ресурс]. Free Software Foundation, 2023. URL: https://ru.wikipedia.org/wiki/OpenModelica.
3. Гармонический осциллятор [Электронный ресурс]. Free Software Foundation, 2023. URL: http://fn.bmstu.ru/data-physics/library/physbook/tom5/ch4/texthtml/ch4_5.htm.