Отчёт по лабораторной работе №6
Компьютерный практикум по статистическому анализу данных

Решение моделей в непрерывном и дискретном времени

Выполнил: Зиязетдинов Алмаз Радикович,
НПИбд-01-22, 1132222010

Содержание

1 Цель работы

Основной целью работы является освоение специализированных пакетов для решения задач в непрерывном и дискретном времени.

2 Выполнение лабораторной работы

2.1 Решение обыкновенных дифференциальных уравнений

Вспомним, что обыкновенное дифференциальное уравнение (ОДУ) описывает изменение некоторой переменной u.

Для решения обыкновенных дифференциальных уравнений (ОДУ) в Julia можно использовать пакет diffrentialEquations.jl.

2.2 Модель экспоненциального роста

Рассмотрим пример использования этого пакета для решение уравнения модели экспоненциального роста, описываемую уравнением, где a — коэффициент роста.

Численное решение в Julia будет иметь следующий вид, а также график, соответствующий полученному решению (рис. 1):

1. Решение обыкновенных дифференциальных уравнений

1.1. Модель экспоненциального роста

[2]:
```
# задаём описание модели с начальными условиями:
a = 0.98
f(u,p,t) = a*u
u0 = 1.0
# задаём интервал времени:
tspan = (0.0,1.0)
# решение:
prob = ODEProblem(f,u0,tspan)
sol = solve(prob)
# строим графики:
plot(sol, linewidth=5,title="Модель экспоненциального роста", xaxis="Время",yaxis="u(t)",label="u(t)")
plot!(sol.t, t->1.0*exp(a*t),lw=3,ls=:dash,label="Аналитическое решение")
```

[2]:

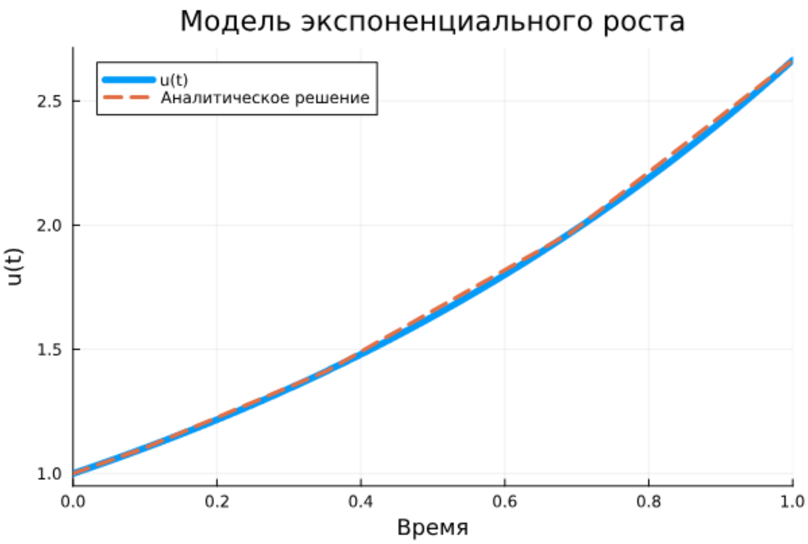


Рис. 1: График модели экспоненциального роста

При построении одного из графиков использовался вызов sol.t, чтобы захватить массив моментов времени. Массив решений можно получить, воспользовавшись sol.u.

Если требуется задать точность решения, то можно воспользоваться параметрами abstol (задаёт близость к нулю) и reltol (задаёт относительную точность). По умолчанию эти параметры имеют значение abstol = 1e-6 и reltol = 1e-3.

Для модели экспоненциального роста (рис. 2):

[4]:
```
# задаём точность решения:
sol = solve(prob,abstol=1e-8,reltol=1e-8)
# строим график:
plot(sol, lw=2, color="black", title="Модель экспоненциального роста", xaxis="Время",yaxis="u(t)",label="Численное решение")
plot!(sol.t, t->1.0*exp(a*t),lw=3,ls=:dash,color="red",label="Аналитическое решение")
```

[4]:

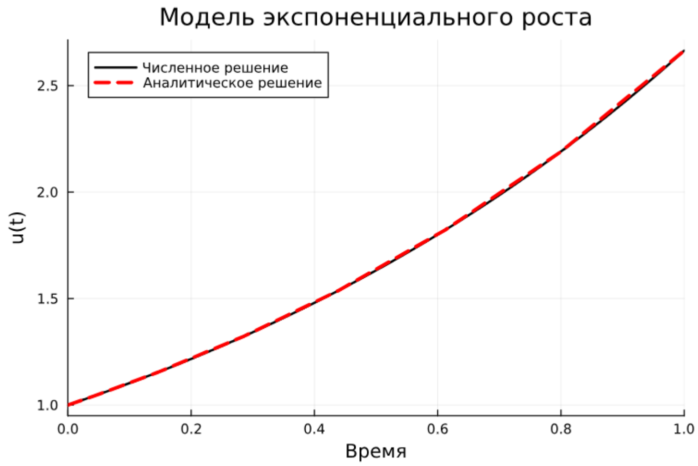


Рис. 2: График модели экспоненциального роста (задана точность решения)

2.3 Система Лоренца

Динамической системой Лоренца является нелинейная автономная система обыкновенных дифференциальных уравнений третьего порядка.

Система получена из системы уравнений Навье–Стокса и описывает движение воздушных потоков в плоском слое жидкости постоянной толщины при разложении скорости течения и температуры в двойные ряды Фурье с последующем усечением до первых-вторых гармоник.

Решение системы неустойчиво на аттракторе, что не позволяет применять классические численные методы на больших отрезках времени, требуется использовать высокоточные вычисления.

Численное решение в Julia будет иметь следующий вид (рис. 3):

1.2. Система Лоренца

[12]:
```
# Задаём описание модели
function lorenz!(du, u, p, t)
    σ, ρ, β = p
    du[1] = σ * (u[2] - u[1])
    du[2] = u[1] * (ρ - u[3]) - u[2]
    du[3] = u[1] * u[2] - β * u[3]
end
# Задаём начальное условие
u0 = [1.0, 0.0, 0.0]
# Задаём значения параметров
p = (10, 28, 8 / 3)
# Задаём интервал времени
tspan = (0.0, 100.0)
# Решение
prob = ODEProblem(lorenz!, u0, tspan, p)
sol = solve(prob, Tsit5())
# Строим график
plot(sol, idxs=(1, 2, 3), lw=1, title="Аттрактор Лоренца", xaxis="x", yaxis="y", zaxis="z", legend=false)
```

[12]:

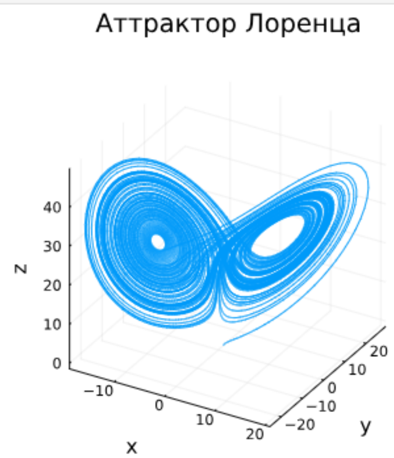


Рис. 3: Аттрактор Лоренца

Можно отключить интерполяцию (рис. 4):

[14]:
```
# отключаем интерполяцию:
plot(sol,vars=(1,2,3),denseplot=false, lw=1, title="Аттрактор Лоренца", xaxis="x",yaxis="y", zaxis="z", legend=false)
```

[14]:

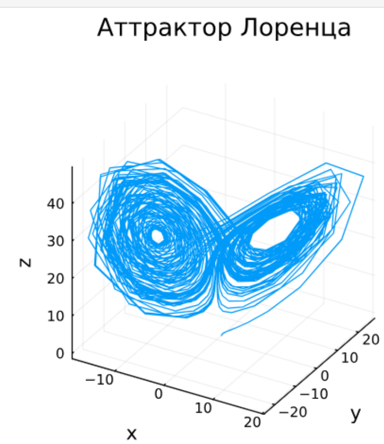


Рис. 4: Аттрактор Лоренца (интерполяция отключена)

2.4 Модель Лотки–Вольтерры

Модель Лотки–Вольтерры описывает взаимодействие двух видов типа «хищник – жертва».

Численное решение в Julia будет иметь следующий вид (рис. 5):

2. Модель Лотки–Вольтерры

[33]:

```julia
# Определяем модель Лотки-Вольтерры
function lotka_volterra!(du, u, p, t)
    x, y = u
    a, b, c, d = p
    du[1] = a * x - b * x * y  # Уравнение для жертв
    du[2] = -c * y + d * x * y  # Уравнение для хищников
end
# Начальные условия
u0 = [1.0, 1.0]  # начальные популяции жертв и хищников
# Параметры модели
p = [1.5, 1.0, 3.0, 1.0]  # a, b, c, d
# Интервал времени
tspan = (0.0, 10.0)
# Создаём задачу
prob = ODEProblem(lotka_volterra!, u0, tspan, p)
# Решаем задачу
sol = solve(prob, Tsit5())
# Построение графика
plot(sol, label=["Жертвы" "Хищники"], color="black",
     linestyle = [:solid :dash], title="Модель Лотки-Вольтерры",
     xlabel="Время", ylabel="Размер популяции")
```

[33]:

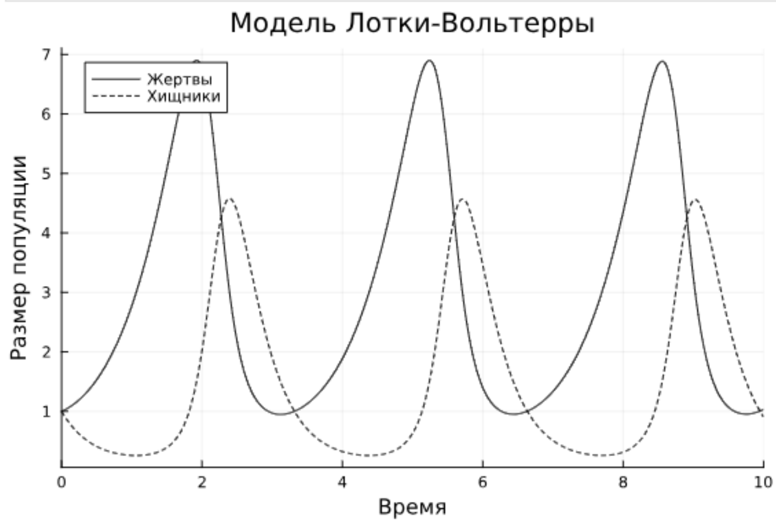


Рис. 5: Модель Лотки–Вольтерры: динамика изменения численности популяций

Фазовый портрет (рис. 6):

[34]:

```
# фазовый портрет:
plot(sol,vars=(1,2), color="black", xaxis="Жертвы",yaxis="Хищники", legend=false)
```

[34]:

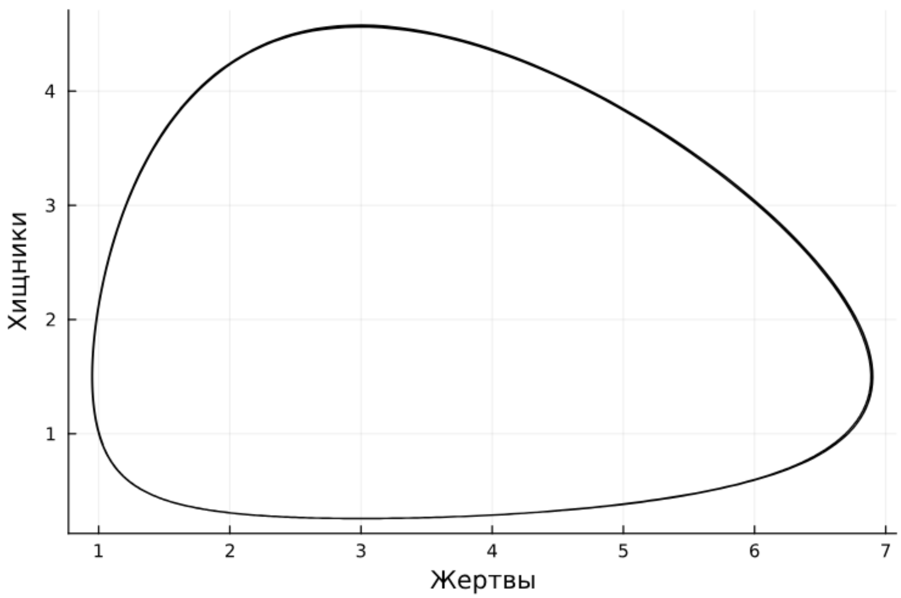


Рис. 6: Модель Лотки–Вольтерры: фазовый портрет

2.5 Самостоятельное выполнение

Выполнение задания №1 (рис. 7):

```julia
using DifferentialEquations, Plots

function malthus!(du, u, p, t)
    b, c = p
    du[1] = (b - c) * u[1]
end

u0 = [1.0]
tspan = (0.0, 10.0)
p = (1.0, 0.2)

prob = ODEProblem(malthus!, u0, tspan, p)
sol = solve(prob)

# Статический график (работает сразу)
plot(sol, linewidth=3, title="Модель Мальтуса", label="Численность",
     xlabel="Время", ylabel="Популяция")


anim = @animate for i in 1:length(sol.t)

    current_pop = [sol.u[j][1] for j in 1:i]

    plot(sol.t[1:i], current_pop,
         linewidth=3, xlim=(0,10), ylim=(0,8),
         title="Модель Мальтуса", label="",
         xlabel="Время", ylabel="Численность популяции")
end

gif(anim, "malthus.gif", fps=15)
```

```
[ Info: Saved animation to C:\Users\Олеся\malthus.gif
```

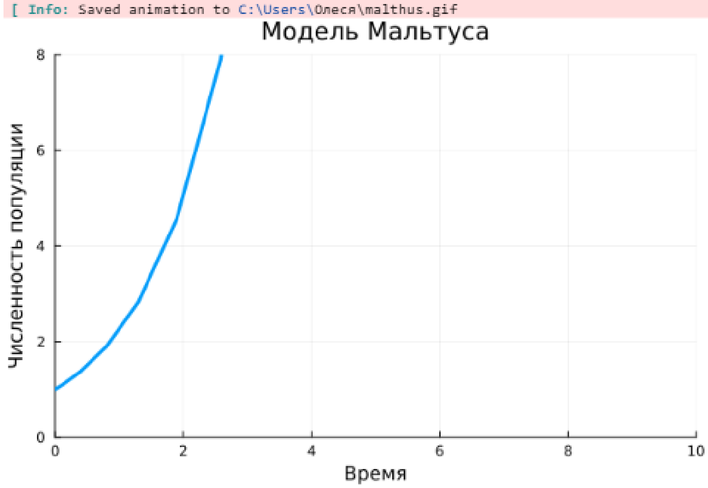


Рис. 7: Решение задания №1

Выполнение задания №2 (рис. 8):

```
•[4]: using DifferentialEquations, Plots

function malthus!(du, u, p, t)
    b, c = p
    du[1] = (b - c) * u[1]
end

u0 = [1.0]
tspan = (0.0, 10.0)
p = (1.0, 0.2)

prob = ODEProblem(malthus!, u0, tspan, p)
sol = solve(prob)

# Статический график (работает сразу)
plot(sol, linewidth=3, title="Модель Мальтуса", label="Численность",
     xlabel="Время", ylabel="Популяция")

# АНИМАЦИЯ — ИСПРАВЛЕННЫЙ ВАРИАНТ
anim = @animate for i in 1:length(sol.t)

    current_pop = [sol.u[j][1] for j in 1:i]   # <-- правильная распаковка

    plot(sol.t[1:i], current_pop,
         linewidth=3, xlim=(0,10), ylim=(0,8),
         title="Модель Мальтуса", label="",
         xlabel="Время", ylabel="Численность популяции")
end

gif(anim, "malthus.gif", fps=15)
```

```
[ Info: Saved animation to C:\Users\Олеся\malthus.gif
```

[4]:

Рис. 8: Решение задания №2

Выполнение задания №3 (рис. 9):

```
5]: using DifferentialEquations, Plots

# Логистическая модель роста
function logistic!(du, u, p, t)
    r, K = p
    du[1] = r * u[1] * (1 - u[1]/K)
end

# Параметры
u0 = [1.0]          # начальная численность
tspan = (0.0, 20.0)
r = 0.5             # коэффициент роста
K = 10.0            # ёмкость экосистемы
p = (r, K)

# Задача и решение
prob = ODEProblem(logistic!, u0, tspan, p)
sol = solve(prob, Tsit5())

# Анимация
anim = @animate for i in 1:length(sol.t)
    plot(sol, idxs=(0,1), tspan=(0, sol.t[i]), linewidth=3, color=:blue,
         title="Логистическая модель роста", xlabel="Время", ylabel="Популяция",
         xlim=(0,20), ylim=(0,11), label="")
    scatter!([sol.t[i]], [sol.u[i][1]], color=:red, markersize=5,
             label="Добавляем текущую точку")
end

# Сохранение
gif(anim, "logistic_growth.gif", fps=18)
```

```
[ Info: Saved animation to C:\Users\Олеся\logistic_growth.gif
```

5]:

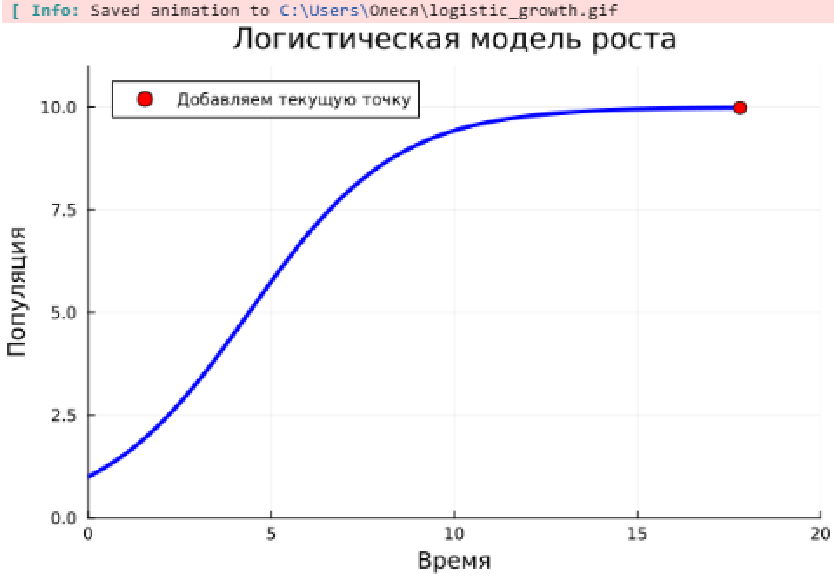


Рис. 9: Решение задания №3

Выполнение задания №4 (рис. 10):

[7]:
```
using DifferentialEquations, Plots

function sir!(du, u, p, t)
    S, I, R = u
    β, γ = p
    du[1] = -β*S*I
    du[2] = β*S*I - γ*I
    du[3] = γ*I
end

u0 = [0.99, 0.01, 0.0]
p = (0.5, 0.2)
tspan = (0.0, 20.0)

prob = ODEProblem(sir!, u0, tspan, p)
sol = solve(prob)

anim = @animate for i in eachindex(sol.t)
    plot(sol, idxs=(0,1), tspan=(0,sol.t[i]), color=:blue,   lw=2, label="Восприимчивые", ylim=(0,1))
    plot!(sol, idxs=(0,2), tspan=(0,sol.t[i]), color=:red,   lw=2, label="Инфицированные")
    plot!(sol, idxs=(0,3), tspan=(0,sol.t[i]), color=:green, lw=2, label="Выздоровевшие")
end

gif(anim, "sir_model.gif", fps=20)
```

```
[ Info: Saved animation to C:\Users\Олеся\sir_model.gif
```

[7]:

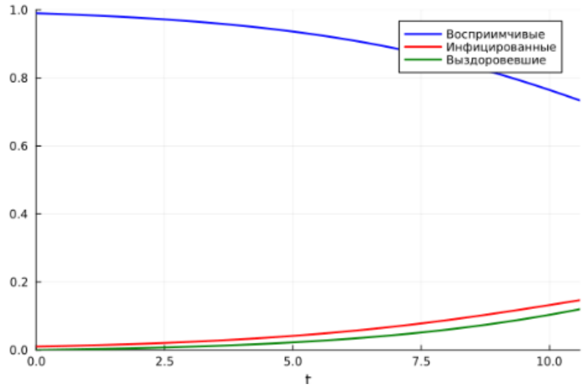


Рис. 10: Решение задания №4

Выполнение задания №5 (рис. 11):

[8]:
```
using Plots

a, b, c, d = 1.0, 0.1, 0.2, 0.05
x, y = 10.0, 5.0
times = 0:0.1:100
X = zeros(length(times), 2)
X[1,:] .= [x, y]

for i in 1:length(times)-1
    x += (a*x - b*x*y) * 0.1
    y += (-c*y + d*x*y) * 0.1
    X[i+1,:] .= [x, y]
end

plot(X[:,1], X[:,2], label="Фазовый портрет", xlabel="x1", ylabel="x2")
```

[8]:

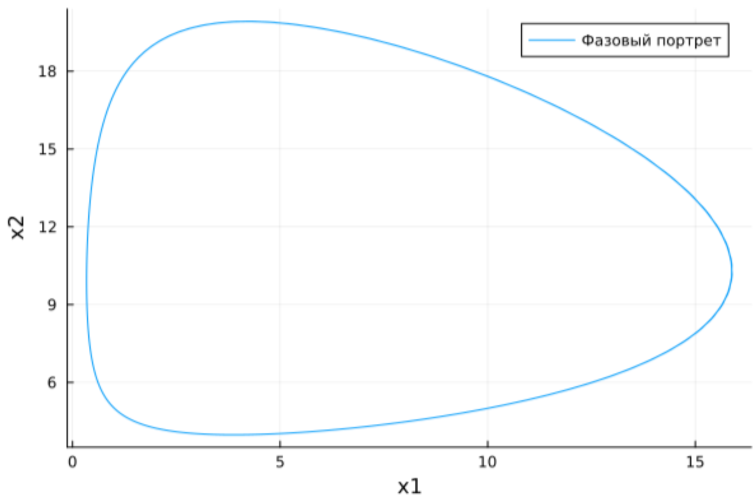


Рис. 11: Решение задания №5

Выполнение задания №6 (рис. 12):

```
[9]: using DifferentialEquations, Plots

function competition!(du, u, p, t)
    x, y = u
    a, b = p
    du[1] = x*(1 - x) - a*x*y
    du[2] = y*(0.8 - y) - b*x*y
end

u0 = [0.1, 0.1]
p = (0.3, 0.5)
tspan = (0.0, 100.0)

prob = ODEProblem(competition!, u0, tspan, p)
sol = solve(prob, Tsit5())

plot(sol, label=["Популяция x" "Популяция y"], title="Модель конкуренции",
     xlabel="Время", linewidth=2)
```

[9]:

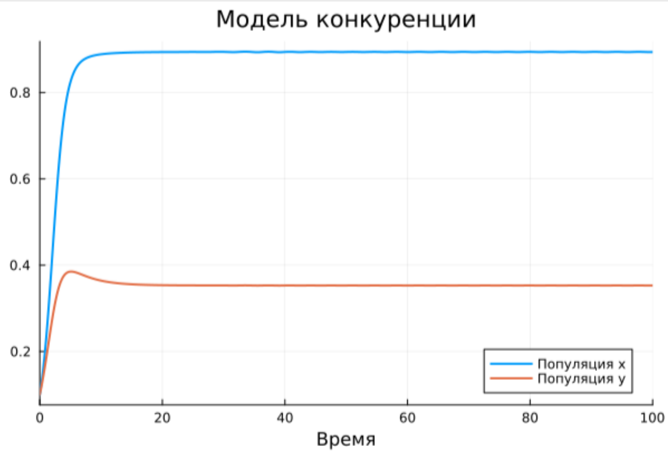


Рис. 12: Решение задания №6

Выполнение задания №7 (рис. 13):

[12]:
```
using DifferentialEquations, Plots

function harmonic!(du, u, p, t)
    x, v = u
    du[1] = v
    du[2] = -u[1]
end

u0 = [0.0, 1.0]
tspan = (0.0, 50.0)
prob = ODEProblem(harmonic!, u0, tspan)
sol = solve(prob, Tsit5())

plot(sol, idxs=(1,2), label="Фазовый портрет", xlabel="Положение x", ylabel="Скорость v")
plot!(sol[1,:], sol[2,:], label="Траектория x(t), v(t)")
```

[12]:

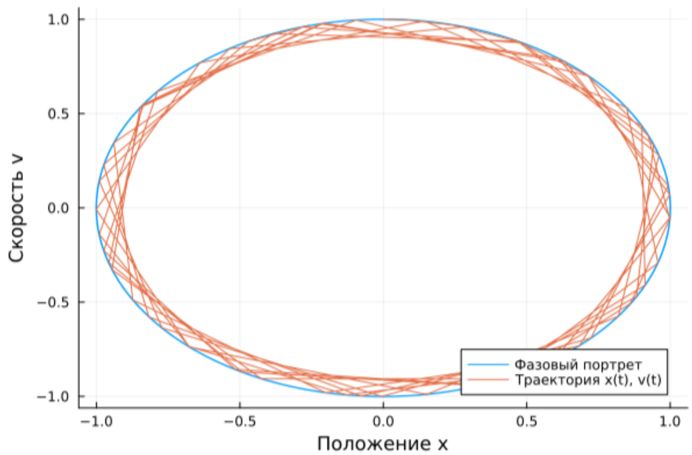


Рис. 13: Решение задания №7

Выполнение задания №8 (рис. 14):

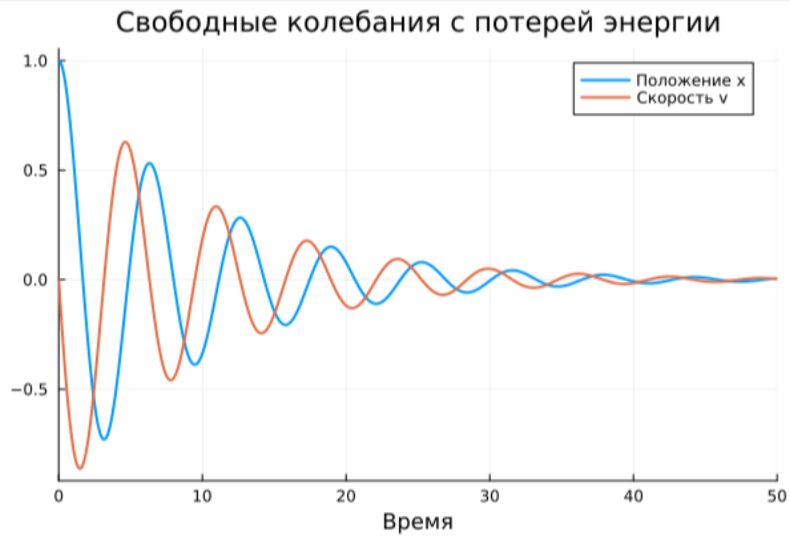
```
[11]: using DifferentialEquations, Plots

function damped!(du, u, p, t)
    x, v = u
    γ = 0.1
    du[1] = v
    du[2] = -2*γ*v - u[1]
end

u0 = [1.0, 0.0]
tspan = (0.0, 50.0)
prob = ODEProblem(damped!, u0, tspan)
sol = solve(prob, Tsit5())

plot(sol, label=["Положение x" "Скорость v"], title="Свободные колебания с потерей энергии",
     xlabel="Время", linewidth=2)
```

[11]:




```
[ ]:
```

Рис. 14: Решение задания №8

3 Вывод

В ходе выполнения лабораторной работы были освоены специализированные пакеты для решения задач в непрерывном и дискретном времени.

4 Список литературы. Библиография

[1] Julia Documentation: https://docs.julialang.org/en/v1/