Отчёт по лабораторной работе №6

Решение моделей в непрерывном и дискретном времени

Козлов Всеволод Павлович НФИбд-02-22

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

1. Используя Jupyter Lab, повторите примеры из раздела 6.2.
2. Выполните задания для самостоятельной работы (раздел 6.4).

3 Выполнение лабораторной работы

Библиотеки (рис. 3.1)

```
# Подключение необходимых пакетов
import Pkg
Pkg.add("DifferentialEquations")
Pkg.add("Plots")
Pkg.add("ParameterizedFunctions")

using DifferentialEquations, Plots, ParameterizedFunctions
```

```
    Updating registry at `C:\Users\vsvld\.julia\registries\General.toml`
   Resolving package versions...
   Installed SciMLPublic ──────────────────────── v1.0.0
   Installed Accessors ────────────────────────── v0.1.42
   Installed OrdinaryDiffEqRosenbrock ─────────── v1.18.1
   Installed LoggingExtras ────────────────────── v1.2.0
   Installed OrdinaryDiffEqRKN ────────────────── v1.5.0
```

Рис. 3.1: Библиотеки

Экспоненциальный рост (рис. 3.2)

```
# Пример 1: Модель экспоненциального роста
a = 0.98
f(u,p,t) = a*u
u0 = 1.0
tspan = (0.0,1.0)

prob = ODEProblem(f,u0,tspan)
sol = solve(prob)

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]:

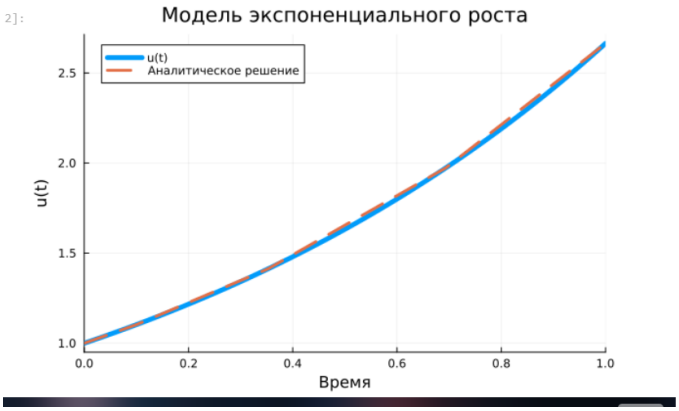

Рис. 3.2: Экспоненциальный рост

Повышенная точность (рис. 3.3)

[3]:

```
# Решение с повышенной точностью
sol = solve(prob,abstol=1e-8,reltol=1e-8)
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")
```

[3]:

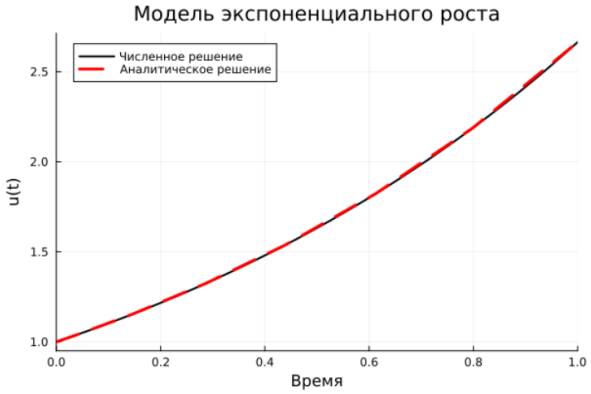

Рис. 3.3: Повышенная точность

Система Лоренца (рис. 3.4)

```julia
# Пример 2: Система Лоренца
function lorenz1(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

u0 = [1.0,0.0,0.0]
p = (10,28,8/3)
tspan = (0.0,100.0)

prob = ODEProblem(lorenz1,u0,tspan,p)
sol = solve(prob)

plot(sol, vars=(1,2,3), lw=2, title="Аттрактор Лоренца",
     xaxis="x",yaxis="y", zaxis="z",legend=false)

plot(sol,vars=(1,2,3),denseplot=false, lw=1, title="Аттрактор Лоренца",
     xaxis="x",yaxis="y", zaxis="z",legend=false)
```

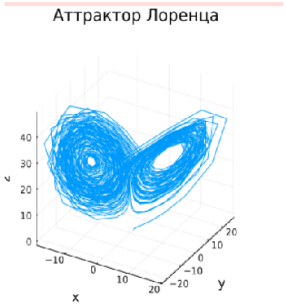

Рис. 3.4: Система Лоренца

Модель Лотки-Вольтерры (рис. 3.5)

```julia
# Пример 3: Модель Лотки-Вольтерры
lvl = @ode_def LotkaVolterra begin
    dx = a*x - b*x*y
    dy = -c*y + d*x*y
end a b c d

u0 = [1.0,1.0]
p = (1.5,1.0,3.0,1.0)
tspan = (0.0,10.0)

prob = ODEProblem(lvl,u0,tspan,p)
sol = solve(prob)
plot(sol, label = ["Жертвы" "Хищники"], color="black", ls=[:solid :dash],
     title="Модель Лотки - Вольтерры", xaxis="Время", yaxis="Размер популяции")

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

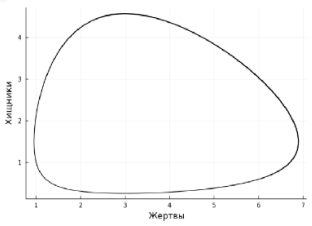

Рис. 3.5: Модель Лотки-Вольтерры

Задание №2 (рис. 3.6)

```
[11]: # ===== ЗАДАНИЕ 2: Модель Мальтуса (экспоненциальный рост) =====
# Реализовать модель роста численности изолированной популяции
function malthus_model(du, u, p, t)
    a = p
    du[1] = a * u[1]
end

u0 = [100.0]  # начальная численность популяции
a = 0.1       # коэффициент роста (рождаемость - смертность)
tspan = (0.0, 50.0)
prob = ODEProblem(malthus_model, u0, tspan, a)
sol = solve(prob)

plot(sol, label="Модель Мальтуса", xaxis="Время", yaxis="Численность популяции",
     title="Экспоненциальный рост популяции", size=(500, 300))
```

[11]:

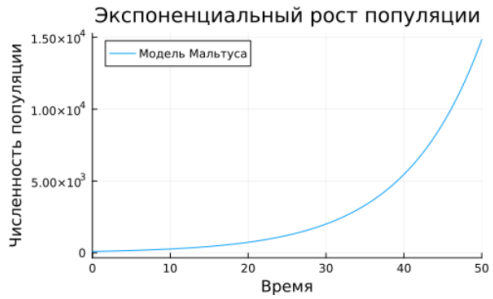

Рис. 3.6: Задание №2

Задание №3 (рис. 3.7)

```
# ===== ЗАДАНИЕ 3: Логистическая модель роста популяции =====
# Реализовать логистическую модель роста популяции
function logistic_model(du, u, p, t)
    r, k = p
    du[1] = r * u[1] * (1 - u[1]/k)
end

u0 = [10.0]    # начальная численность популяции
r = 0.2        # коэффициент роста
k = 1000.0     # емкость среды
tspan = (0.0, 100.0)
prob = ODEProblem(logistic_model, u0, tspan, (r, k))
sol = solve(prob)

plot(sol, label="Логистический рост", xaxis="Время", yaxis="Численность популяции",
     title="Логистическая модель роста популяции", size=(500, 300))
```

[ ]:

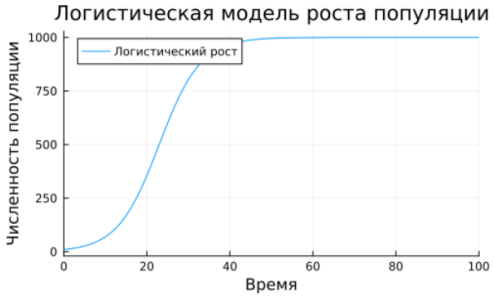

Рис. 3.7: Задание №3

Задание №4 (рис. 3.8)

```
# ===== ЗАДАНИЕ 4: Модель эпидемии Кермака-Маккендрика (SIR) =====
# Реализовать SIR-модель эпидемии
function sir_model(du, u, p, t)
    s, i, r = u
    β, ν = p
    du[1] = -β * i * s      # ds/dt
    du[2] = β * i * s - ν * i  # di/dt
    du[3] = ν * i           # dr/dt
end
u0 = [0.99, 0.01, 0.0]  # s, i, r
β = 0.3                 # коэффициент заражения
ν = 0.1                 # коэффициент выздоровления
tspan = (0.0, 100.0)
prob = ODEProblem(sir_model, u0, tspan, (β, ν))
sol = solve(prob)

plot(sol, label=["Восприимчивые" "Инфицированные" "Выздоровевшие"],
     title="SIR-модель эпидемии", xaxis="Время", yaxis="Доля популяции", size=(500, 300))
```

4]:

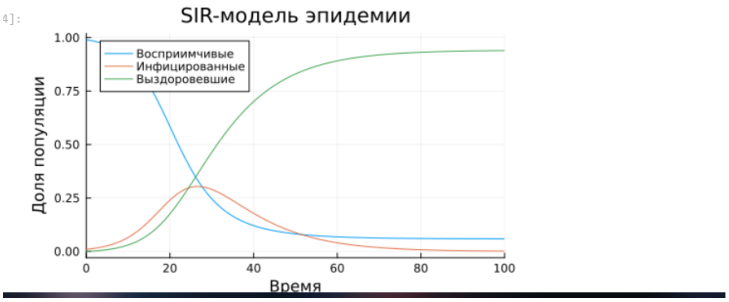

Рис. 3.8: Задание №4

Задание №5 (рис. 3.9)

```
# ===== ЗАДАНИЕ 5: Модель SEIR =====
# Реализовать и исследовать SEIR-модель эпидемии
function seir_model(du, u, p, t)
    s, e, i, r = u
    β, δ, γ, N = p

    du[1] = -β/N * s * i        # ds/dt
    du[2] = β/N * s * i - δ * e  # de/dt
    du[3] = δ * e - γ * i       # di/dt
    du[4] = γ * i              # dr/dt
end

N = 1.0  # общая численность популяции (нормированная)
u0 = [0.95, 0.02, 0.02, 0.01]  # s, e, i, r
β = 0.5   # коэффициент заражения
δ = 0.2   # коэффициент перехода из экспонированных в инфицированные
γ = 0.1   # коэффициент выздоровления
tspan = (0.0, 150.0)
prob = ODEProblem(seir_model, u0, tspan, (β, δ, γ, N))
sol = solve(prob)

plot(sol, label=["Восприимчивые" "Экспонированные" "Инфицированные" "Выздоровевшие"],
     title="SEIR-модель эпидемии", xaxis="Время", yaxis="Доля популяции")
```

]:

Рис. 3.9: Задание №5

Задание №5 (рис. 3.10)

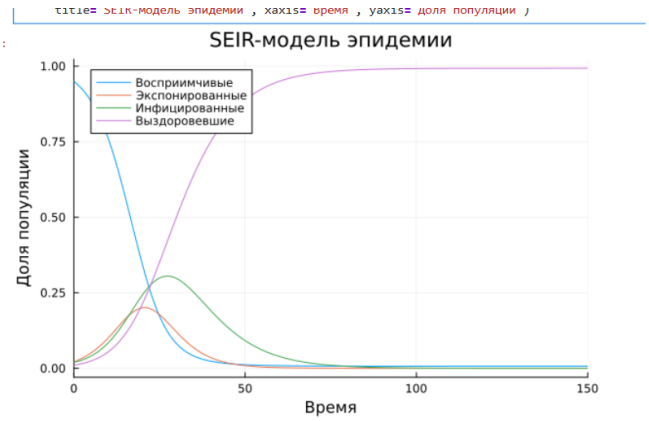

Рис. 3.10: Задание №5

Задание №6 (рис. 3.11)

```julia
# ===== ЗАДАНИЕ 6: Дискретная модель Лотки-Вольтерры =====
# Фазовый портрет дискретной модели Лотки-Вольтерры

using LinearAlgebra

# Параметры модели:
a = 2.0    # коэффициент роста жертв
c = 1.0    # коэффициент смертности хищников
d = 5.0    # коэффициент превращения жертв в хищников

# Несколько начальных условий для построения фазового портрета
initial_conditions = [
    [0.1, 0.1],  # мало жертв и хищников
    [0.3, 0.2],  # умеренное количество
    [0.6, 0.1],  # много жертв, мало хищников
    [0.2, 0.5],  # мало жертв, много хищников
    [0.5, 0.4]   # сбалансированно
]

# Функция дискретной модели Лотки-Вольтерры
function discrete_lotka_volterra(X, p, t)
    a, c, d = p
    x1, x2 = X

    X1_next = a * x1 * (1 - x1) - x1 * x2  # изменение популяции жертв
    X2_next = -c * x2 + d * x1 * x2        # изменение популяции хищников

    return [X1_next, X2_next]
end

# Создаем фазовый портрет
plt = plot(size=(800, 600), title="Фазовый портрет дискретной модели Лотки-Вольтерры",
           xlabel="Жертвы (x₁)", ylabel="Хищники (x₂)",
```

Рис. 3.11: Задание №6

График (рис. 3.12)

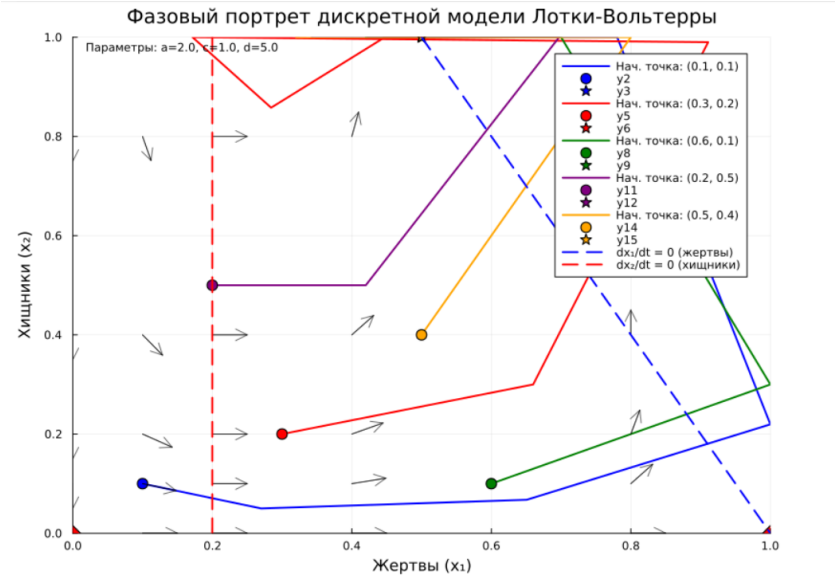

Рис. 3.12: График

Задание №7 (рис. 3.13)

```
# ===== ЗАДАНИЕ 7: Модель отбора на основе конкурентных отношений =====
# Реализовать модель конкурентных отношений с разными параметрами для видов

using DifferentialEquations, Plots, LinearAlgebra

# Модель конкурентных отношений с разными коэффициентами
function competition_model(du, u, p, t)
    x, y = u
    α1, α2, β = p  # α1 - рост вида 1, α2 - рост вида 2, β - конкуренция

    du[1] = α1 * x - β * x * y  # изменение вида 1
    du[2] = α2 * y - β * x * y  # изменение вида 2
end

# Параметры модели:
# α1 = 0.15 - более высокий коэффициент роста вида 1
# α2 = 0.08 - более низкий коэффициент роста вида 2
# β = 0.005 - умеренная конкуренция
α1 = 0.15
α2 = 0.08
β = 0.005

u0 = [50.0, 30.0]  # начальные численности: вид 1 = 50, вид 2 = 30
tspan = (0.0, 50.0)
prob = ODEProblem(competition_model, u0, tspan, (α1, α2, β))
sol = solve(prob)

println("Параметры модели:")
println("α1 = $α1 (коэффициент роста вида 1)")
println("α2 = $α2 (коэффициент роста вида 2)")
println("β = $β (коэффициент конкуренции)")
println("Начальные условия: вид 1 = $(u0[1]), вид 2 = $(u0[2])")

# График динамики популяций во времени
```

Рис. 3.13: Задание №7

График (рис. 3.14)

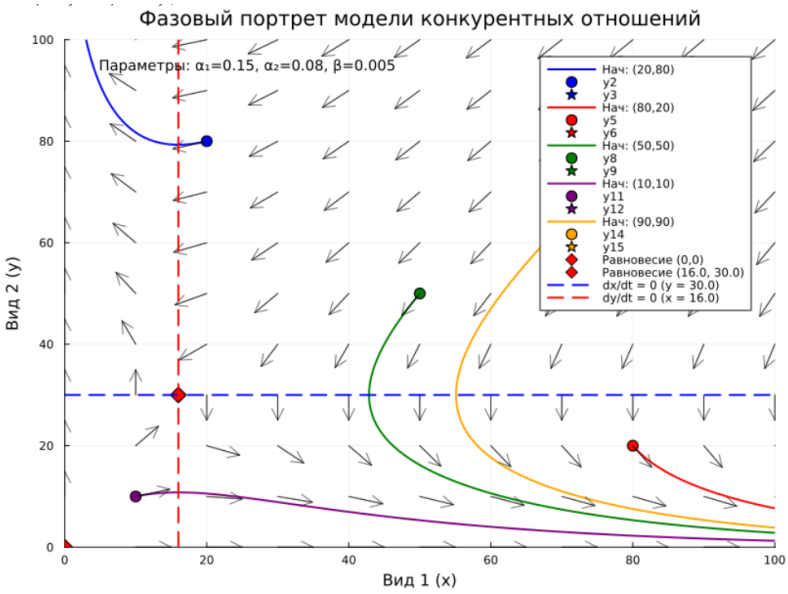

Рис. 3.14: График

Задание №8 (рис. 3.15)

```
# ===== ЗАДАНИЕ 8: Модель консервативного гармонического осциллятора =====
# Реализовать модель гармонического осциллятора без затухания
function harmonic_oscillator(du, u, p, t)
    x, v = u
    ω₀ = p

    du[1] = v           # dx/dt = v
    du[2] = -ω₀^2 * x  # dv/dt = -ω₀²x
end

u0 = [1.0, 0.0]    # начальное положение и скорость
ω₀ = 1.0           # циклическая частота
tspan = (0.0, 10π)
prob = ODEProblem(harmonic_oscillator, u0, tspan, ω₀)
sol = solve(prob)

plot(sol, label=["Положение" "Скорость"], title="Гармонический осциллятор",
     xaxis="Время", yaxis="Значение")

# Фазовый портрет
plot(sol, vars=(1,2), label="Фазовый портрет", xaxis="Положение", yaxis="Скорость",
     title="Фазовый портрет гармонического осциллятора", size=(500, 300))
```

Рис. 3.15: Задание №8

График (рис. 3.16)

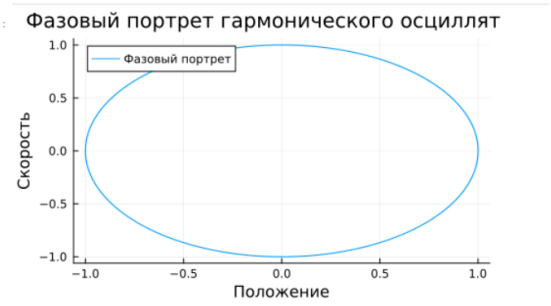

Рис. 3.16: График

Задание №9 (рис. 3.17)

```julia
# ===== ЗАДАНИЕ 9: Модель свободных колебаний гармонического осциллятора =====
# Реализовать модель гармонического осциллятора с затуханием
function damped_oscillator(du, u, p, t)
    x, v = u
    ω₀, γ = p

    du[1] = v                    # dx/dt = v
    du[2] = -2γ*v - ω₀^2 * x     # dv/dt = -2γv - ω₀²x
end

u0 = [1.0, 0.0]    # начальное положение и скорость
ω₀ = 1.0           # циклическая частота
γ = 0.1            # коэффициент затухания
tspan = (0.0, 20.0)
prob = ODEProblem(damped_oscillator, u0, tspan, (ω₀, γ))
sol = solve(prob)

plot(sol, label=["Положение" "Скорость"], title="Затухающий гармонический осциллятор",
     xaxis="Время", yaxis="Значение")

# Фазовый портрет
plot(sol, vars=(1,2), label="Фазовый портрет", xaxis="Положение", yaxis="Скорость",
     title="Фазовый портрет затухающего осциллятора")
```

Рис. 3.17: Задание №9

График (рис. 3.18)

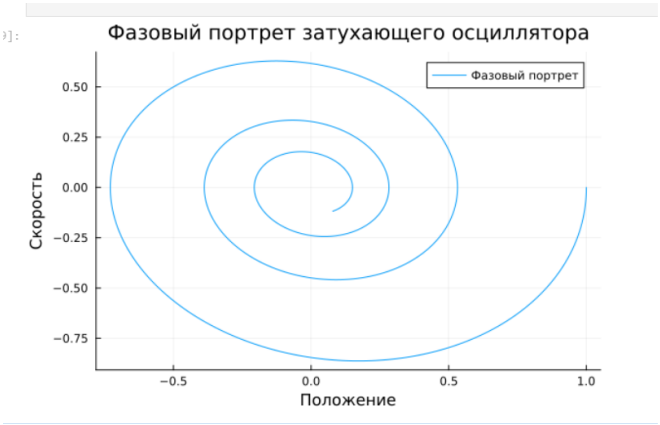

Рис. 3.18: График

4 Выводы

Освоил специализированные пакеты для решения задач в непрерывном и дискретном времени.

5 Список литературы

1. Julia 1.5 Documentation. — 2020. — URL: https://docs.julialang.org/en/v1/.
2. Klok H.,Nazarathy Y. Statistics with Julia: Fundamentals for Data Science,Machine Learning and Artificial Intelligence. — 2020. — URL: https://statisticswithjulia.org/.
3. Ökten G. First Semester in Numerical Analysis with Julia. — Florida State University, 2019. — DOI: 10.33009/jul.
4. Антонюк В. А. Язык Julia как инструмент исследователя. — М. : Физический факультет МГУ им. М. В. Ломоносова, 2019.
5. Шиндин А. В. Язык программирования математических вычислений Julia. Базовое руководство. — Нижний Новгород : Нижегородский госуниверситет, 2016.