Лабораторная работа №6

Леснухин Даниил Дмитриевич Российский университет дружбы народов Москва

Цель работы

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

Задание

1. Используя JupyterLab, повторите примеры, дополнив графики обозначениями осей, легендой и названиями графиков
2. Выполните задания для самостоятельной работы

Теоретическое введение

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений

Эффективен также для написания программ общего назначения

Использовалась официальная документация Julia

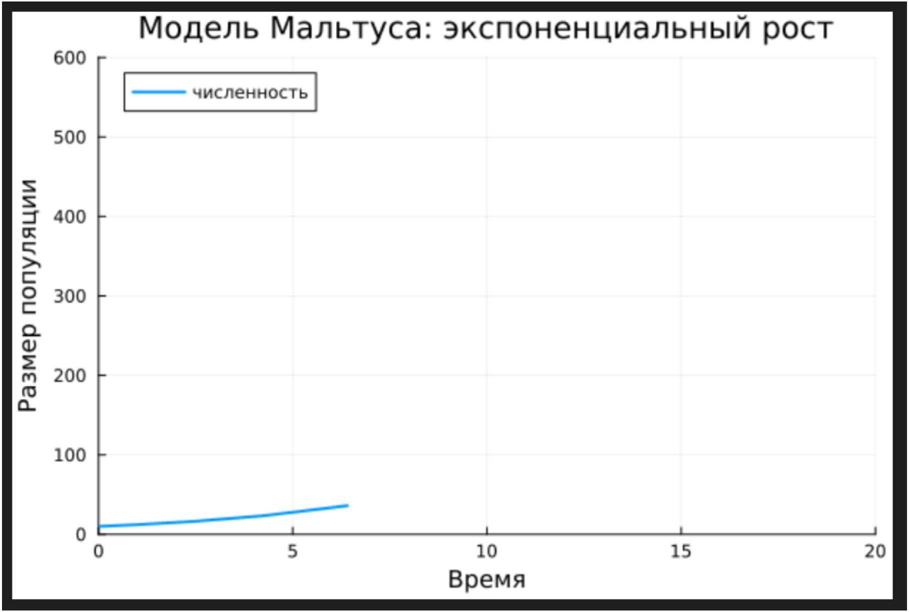
Задание №1 — Модель Мальтуса

Модель роста численности изолированной популяции:

dot x = a x, a = b - c

где x(t) — численность популяции, b — коэффициент рождаемости, c — коэффициент смертности

```julia
using DifferentialEquations, Plots

function maltus_model(du, u, p, t)
    a = p
    du[1] = a * u[1]
end

a = 0.2
u0 = [10.0]
tspan = (0.0, 20.0)

prob = ODEProblem(maltus_model, u0, tspan, a)
sol = solve(prob)


plot(sol, lw=2, title="Модель Мальтуса: экспоненциальный рост", xaxis="Время", yaxis="Размер популяции", label="численность", legend=:topleft)

anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], sol[1, 1:i], lw=2, title="Модель Мальтуса: экспоненциальный рост",
     xaxis="Время", yaxis="Размер популяции", label="численность", legend=:topleft,
     xlim=(0, 20), ylim=(0, maximum(sol[1, :]) * 1.1))
end

gif(anim, "malthus_model.gif", fps=10)
```

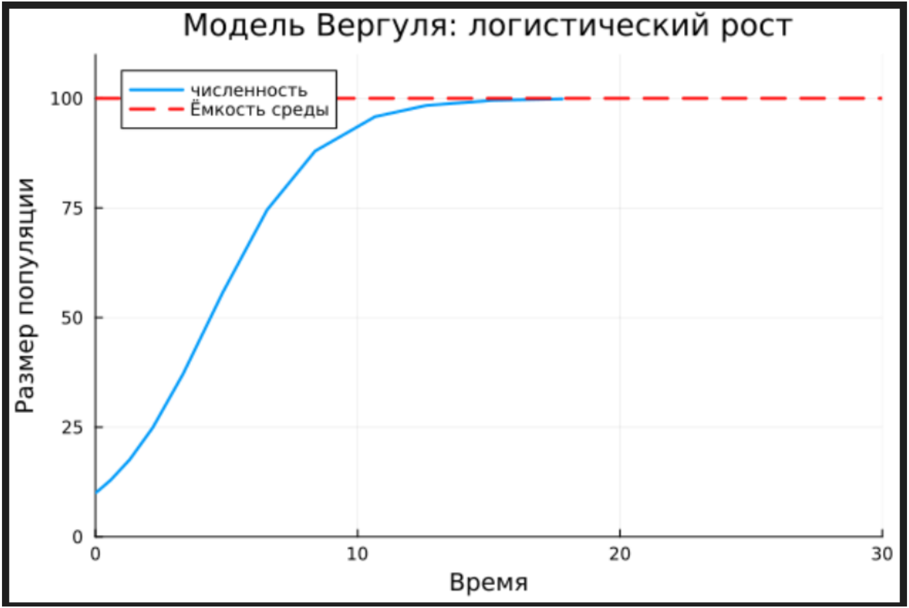

Модель Мальтуса: экспоненциальный рост
численность
Размер популяции
Время
0
100
200
300
400
500
600
0
5
10
15
20

Задание №2 — Логистическая модель роста

dot x = r x (1 - x/k), r > 0, k > 0

где r — коэффициент роста, k — предельная численность

```julia
function logistic_model(du, u, p, t)
    r, k = p
    du[1] = r * u[1] * (1 - u[1] / k)
end

r = 0.5
k = 100.0
u0 = [10.0]

tspan = (0.0, 30.0)

p = (r, k)

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

plot(sol, lw=2, title="Модель Вергуля: логистический рост", xaxis="Время", yaxis="Размер популяции", label="численность", legend=:topleft)

hline!([k], lw=2, ls=:dash, color=:red, label="Ёмкость среды")

anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], sol[1, 1:i], lw=2, title="Модель Вергуля: логистический рост",
     xaxis="Время", yaxis="Размер популяции", label="численность", legend=:topleft,
     xlim=(0, 30), ylim=(0, k * 1.1))
    hline!([k], lw=2, ls=:dash, color=:red, label="Ёмкость среды")
end

gif(anim, "logistic_model.gif", fps=10)
```


Модель Вергуля: логистический рост
численность
Ёмкость среды
Размер популяции
0
25
50
75
100
0
10
20
30
Время

Задание №3 — SIR-модель

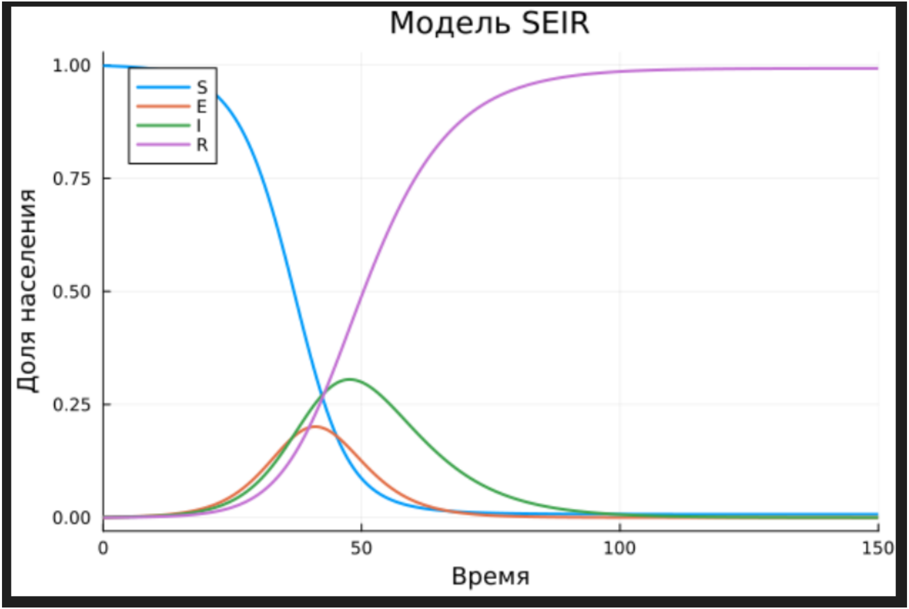
begin cases
dot S = - beta I S
dot I = beta I S - gamma I
dot R = gamma I
end cases

где S — восприимчивые, I — инфицированные, R — выздоровевшие, beta — коэффициент заражения, gamma — коэффициент выздоровления

```julia
function seir_model!(du, u, p, t)
    S, E, I, R = u
    β, σ, γ, N = p
    du[1] = -β * S * I / N
    du[2] = β * S * I / N - σ * E
    du[3] = σ * E - γ * I
    du[4] = γ * I
end

β = 0.5
σ = 0.2
γ = 0.1
N = 1000.0
u0 = [999.0/N, 0.0, 1.0/N , 0.0]  # S, E, I, R
tspan = (0.0, 150.0)

p = (β, σ, γ, N)
prob = ODEProblem(seir_model!, u0, tspan, p)
sol = solve(prob)

plot(sol, lw=2, title="Модель SEIR распространения инфекции", xaxis="Время", yaxis="Доля населения", label=["Восприимчивые (S)" "Экспонированные (E)" "Инфі
```

3.5s

Julia


Модель SEIR
S
E
I
R
Доля населения
1.00
0.75
0.50
0.25
0.00
0
50
100
150
Время

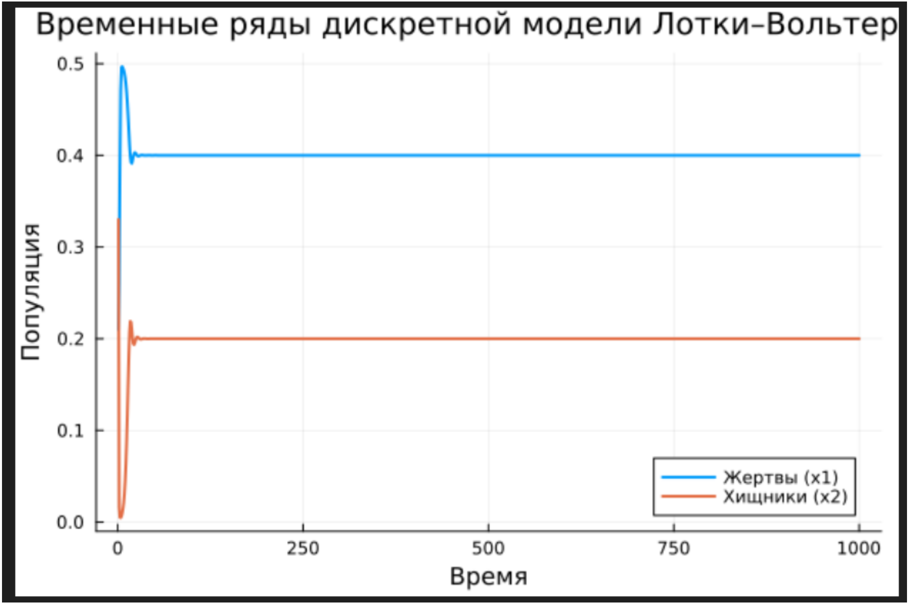
Задание №4 — Модель Лотки-Вольтерры дискретная

begin cases
X1(t+1) = a X1(t)(1-X1(t)) - X1(t) X2(t)
X2(t+1) = - c X2(t) - d X1(t) X2(t)
end cases

Начальные данные: a = 2, c = 1, d = 5

```julia
using Plots

function discrete_lotka_volterra(x1, x2, a, c, d)
    x1_next = a * x1 * (1 - x1) - x1 * x2
    x2_next = -c * x2 + d * x1 * x2
    return x1_next, x2_next
end

a = 2
c = 1
d = 5

# равновесная точка
x1_eq = c / d
x2_eq = a * x1_eq * (1 - x1_eq)

println("Равновесная точка: (x1, x2) = ($x1_eq, $x2_eq)")

T = 1000

# создаём массивы
x1_vals = zeros(T)
x2_vals = zeros(T)

# начальные условия (небольшое отклонение)
x1_vals[1] = x1_eq + 0.01
x2_vals[1] = x2_eq + 0.01
```

```julia
# итерации
for t in 1:T-1
    x1_vals[t+1], x2_vals[t+1] =
        discrete_lotka_volterra(x1_vals[t], x2_vals[t], a, c, d)
end

# фазовый портрет
scatter(x1_vals, x2_vals,
        markersize=1,
        alpha=0.5,
        title="Фазовый портрет дискретной модели Лотки-Вольтерры",
        xaxis="Жертвы (x1)",
        yaxis="Хищники (x2)",
        legend=false)

scatter!([x1_eq], [x2_eq], markersize=8, color=:red,
         label="Равновесная точка")
# временные ряды
plot(1:T, x1_vals,
     lw=2,
     title="Временные ряды дискретной модели Лотки-Вольтерры",
     xaxis="Время",
     yaxis="Популяция",
     label="Жертвы (x1)")
plot!(1:T, x2_vals, lw=2, label="Хищники (x2)")
```

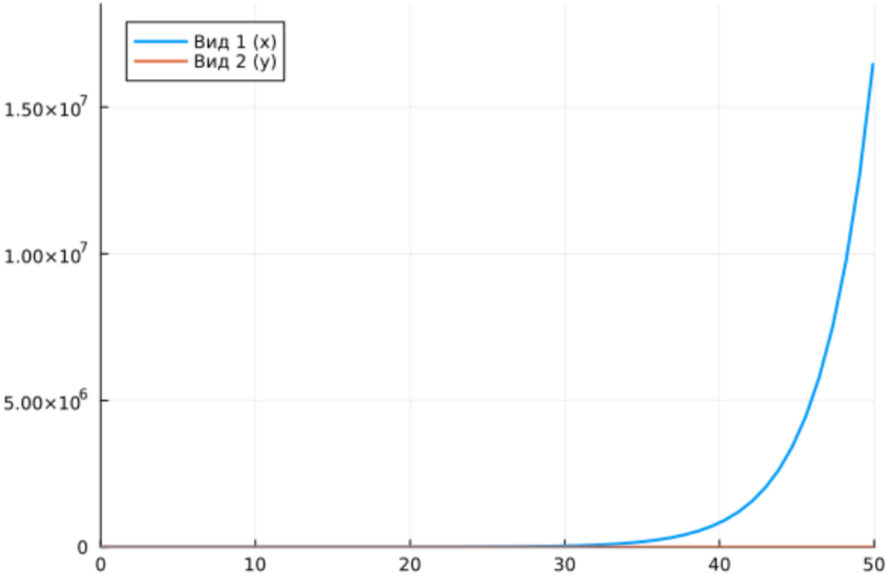

Временные ряды дискретной модели Лотки–Вольтер
Популяция
0.5
0.4
0.3
0.2
0.1
0.0
0
250
500
750
1000
Время
Жертвы (x1)
Хищники (x2)

Задание №5 — Конкурентные отношения

begin cases
dot x = alpha x - beta x y
dot y = alpha y - beta x y
end cases

```julia
using DifferentialEquations, Plots

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

a = 0.3
b = 0.1
p = (a, b)

u0 = [10.0, 5.0]
tspan = (0.0, 50.0)

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

# График во времени
plot(sol, lw=2,
     title="Модель конкуренции",
     xaxis="Время",
     yaxis="Размер популяции",
     label=["Вид 1 (x)" "Вид 2 (y)"],
     legend=:topleft)

# Фазовый портрет
```

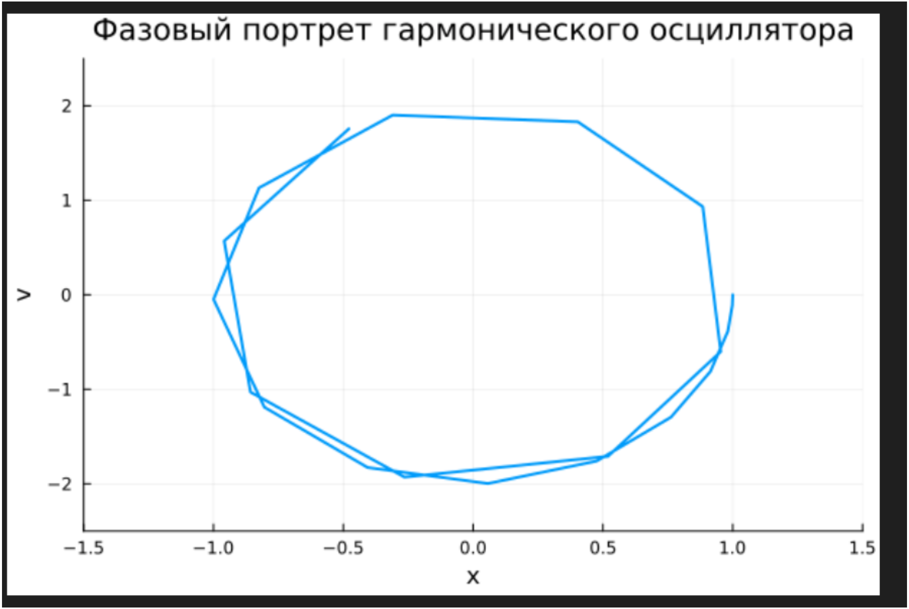
✓ 3.4s


Вид 1 (x)
Вид 2 (y)
1.50×10^7
1.00×10^7
5.00×10^6
0
0
10
20
30
40
50

Задание №6 — Гармонический осциллятор

ddot x + omega0^2 x = 0, x(t0) = x0, dot x(t0) = y0

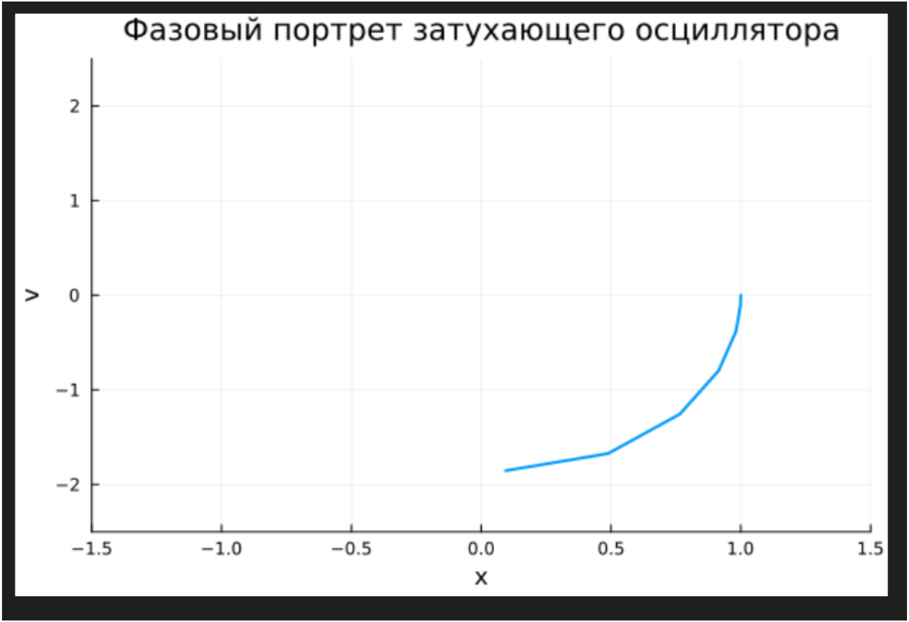
```
function harmonic_oscillator(du, u, p, t)
    x, v = u
    w0 = p

    du[1] = v
    du[2] = -w0^2 * x
end

w0 = 2.0

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


prob = ODEProblem(harmonic_oscillator, u0, tspan, w0)
sol = solve(prob)

plot(sol, lw=2, title="Гармонический осциллятор", xaxis="Время", yaxis="x(t)", label="Численное решени

plot(sol, vars=(1,2), lw=2, title="Фазовый портрет гармонического осциллятора", xaxis="x", yaxis="v", l

anim = @animate for i in 1:length(sol.t)
    plot(sol[1,1:i], sol[2,1:i], lw=2,
         title="Фазовый портрет гармонического осциллятора",
         xaxis="x",
         yaxis="v",
         legend=false,
```


Фазовый портрет гармонического осциллятора
v
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x

Задание №7 — Гармонический осциллятор с затуханием

ddot x + 2 gamma dot x + omega0^2 x = 0, x(t0) = x0, dot x(t0) = y0

```julia
function damped_oscillator(du, u, p, t)
    x, v = u
    w0, γ = p

    du[1] = v
    du[2] = -2*γ*v - w0^2 * x
end

w0 = 2.0
γ = 0.1

u0 = [1.0, 0.0]
tspan = (0.0, 20.0)
p = (w0, γ)
prob = ODEProblem(damped_oscillator, u0, tspan, p)

sol = solve(prob)

plot(sol, lw=2, title="Затухающие колебания", xaxis="Время", yaxis="x(t)", label="Численное решение")
plot(sol, vars=(1,2), lw=2, title="Фазовый портрет затухающего осциллятора", xaxis="x", yaxis="v", legend=false)

anim = @animate for i in 1:length(sol.t)
    plot(sol[1,1:i], sol[2,1:i], lw=2,
        title="Фазовый портрет затухающего осциллятора",
        xaxis="x",
        yaxis="v",
        legend=false,
        xlim=(-1.5, 1.5),
        ylim=(-2.5, 2.5))
```
✓ 2.1s


Фазовый портрет затухающего осциллятора
v
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x

Выводы

В результате выполнения лабораторной работы:

- Освоены специализированные пакеты Julia для моделирования систем в непрерывном и дискретном времени
- Построены графики и фазовые портреты моделей популяций и осцилляторов
- Получены навыки анализа и визуализации динамических систем