Отчёт по лабораторной работе №6 Решение моделей в непрерывном и дискретном времени

Статический анализ данных

Выполнила: Коняева Марина Александровна,
НФИбд-01-21, 1032217044

Содержание

Список иллюстраций

Цели лабораторной работы

Освоить специализированных пакетов для решения задач в непрерывном и дискретном времени.

Теоретическое введение

Напомним, что обыкновенное дифференциальное уравнение (ОДУ) описывает изменение некоторой переменной u:

$$u'(t) = f(u(t), p, t),$$

где $f(u(t), p, t)$ — нелинейная модель (функция) изменения $u(t)$ с заданным начальным значением $u(t_0) = u_0$, p — параметры модели, t — время.

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

Задачи лабораторной работы

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

Выполнение лабораторной работы

Решение обыкновенных дифференциальных уравнений

Напомним, что обыкновенное дифференциальное уравнение (ОДУ) описывает изменение некоторой переменной u:

$$u'(t) = f(u(t), p, t),$$

где $f(u(t), p, t)$ — нелинейная модель (функция) изменения $u(t)$ с заданным начальным значением $u(t_0) = u_0$, p — параметры модели, t — время.

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

Модель экспоненциального роста

1. Перед использованием пакетов следует их установить и подключить в Julia.

подключаем необходимые пакеты: import Pkg Pkg.add("DifferentialEquations") Pkg.add("Plots")

2. Рассмотрим пример использования этого пакета для решение уравнения модели экспоненциального роста, описываемую уравнением □′(□) = □□(□), □(0) = □0, где □ — коэффициент роста. Предположим, что заданы следующие начальные данные □ = 0, 98, □(0) = 1, 0, □ □ [0; 1, 0]. Аналитическое решение модели (6.1) имеет вид: □(□) = □0 exp(□□)□(□).

```
using DifferentialEquations
# 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)
# подключаем необходимые пакеты:
using Plots
# строим графики:
plot(sol, linewidth=5,title="Модель экспоненциального роста", xaxis="Время",yaxis="
plot!(sol.t, t->1.0*exp(a*t),lw=3,ls=:dash,label="Аналитическое решение")
```

Рис. 1: Листинг

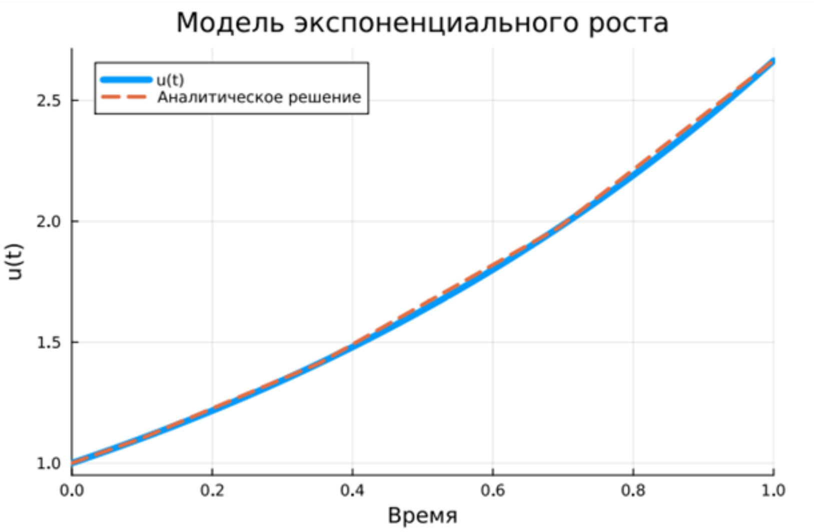

Рис. 2: Модель экспоненциального роста

3. Повторим аналогичный пример. 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.

```
# задаём точность решения:
sol = solve(prob,abstol=1e-8,reltol=1e-8)
println(sol)
# 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")
```

Рис. 3: Листинг

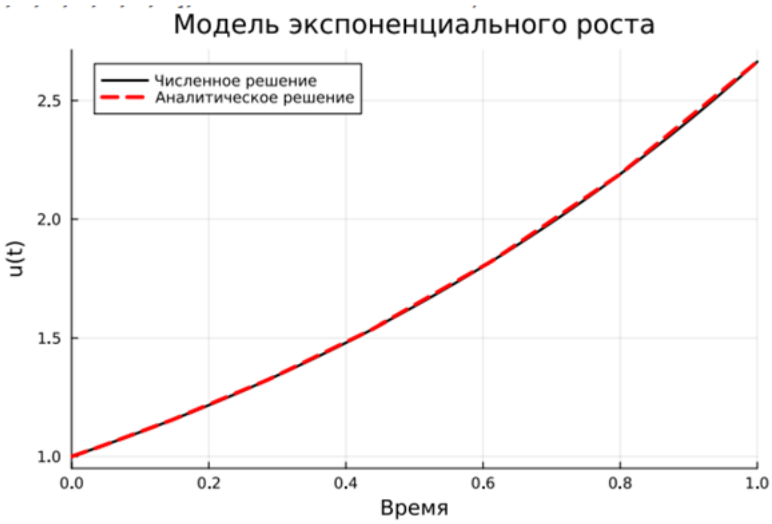

Рис. 4: Модель экспоненциального роста

Система Лоренца

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

$$\begin{cases} \dot{x} = \sigma(y - x), \\ \dot{y} = \rho x - y - xz, \\ \dot{z} = xy - \beta z, \end{cases} \tag{6.2}$$

где σ, ρ и β — параметры системы (некоторые положительные числа, обычно указывают $\sigma = 10$, $\rho = 28$ и $\beta = 8/3$).

4. Повторим пример. Система (6.2) 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.

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

Рис. 5: Листинг

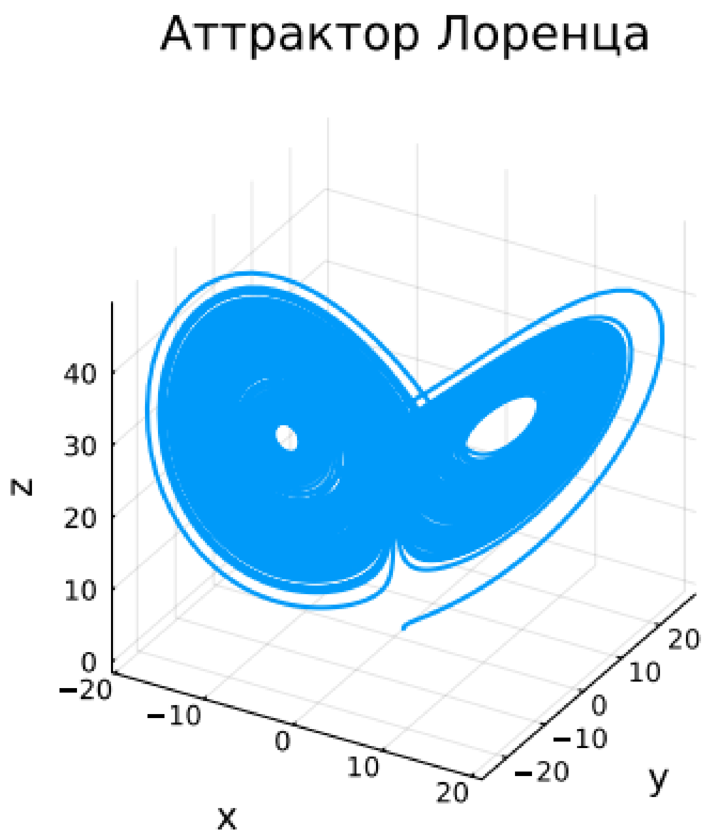

Рис. 6: Аттрактор Лоренца

5. Повторим пример системы Лоренца, но при этом отключим интерполяцию.

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

Рис. 7: Листинг

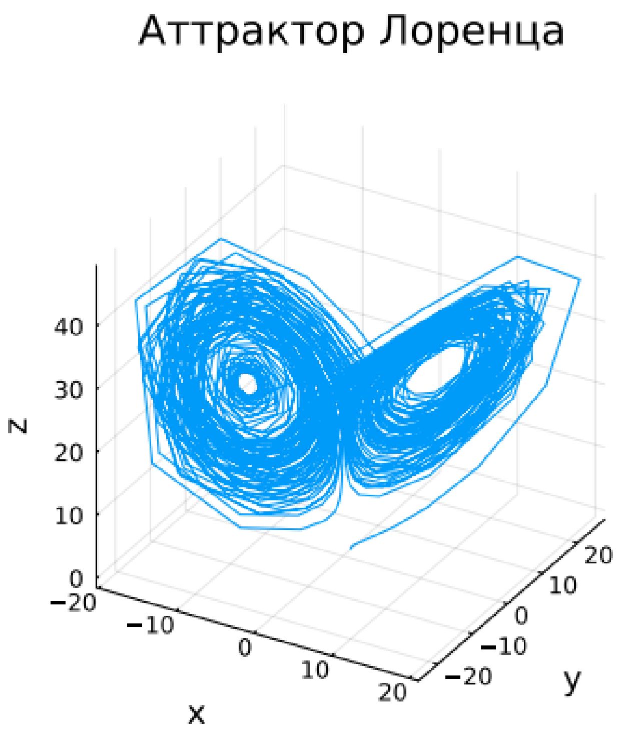

Рис. 8: Аттрактор Лоренца (интерполяция отключена)

Модель Лотки–Вольтерры

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

$$\begin{cases} \dot{x} = (\alpha - \beta y)x, \\ \dot{y} = (-\gamma + \delta x)y, \end{cases}$$

где x — количество жертв, y — количество хищников, t — время, $\alpha, \beta, \gamma, \delta$ — коэффициенты, отражающие взаимодействия между видами (в данном случае α — коэффициент рождаемости жертв, γ — коэффициент убыли хищников, β — коэффициент убыли жертв в результате взаимодействия с хищниками, δ — коэффициент роста численности хищников).

6. Повторим пример реализации модели Лотки-Вольтерры.

```
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаём описание модели:
lv! = @ode_def LotkaVolterra begin
dx = a*x - b*x*y
dy = -c*y + d*x*y
end a b c d
# задаём начальное условие:
u0 = [1.0,1.0]
# задаём значения параметров:
p = (1.5,1.0,3.0,1.0)
# задаём интервал времени:
tspan = (0.0,10.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
plot(sol, label = ["Жертвы" "Хищники"], color="black", ls=[:solid :dash], title="Модель Лотки - Вольтерры", xaxis="Время",yaxis="P
```

Рис. 9: Листинг

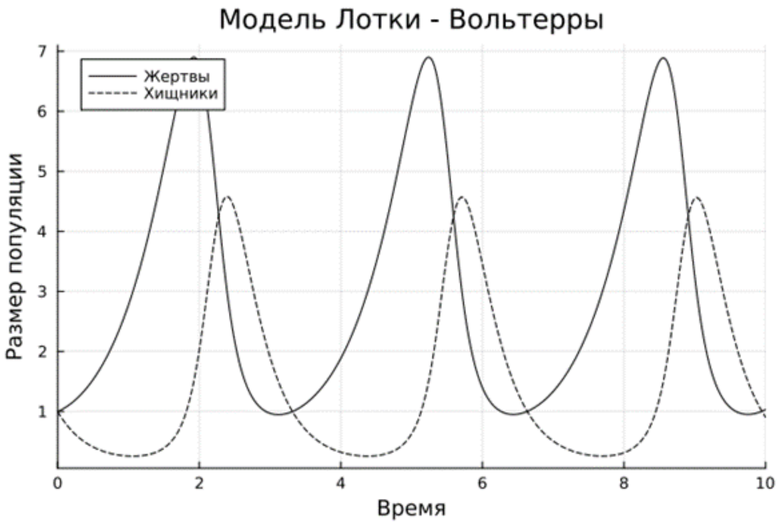

Рис. 10: Модель Лотки–Вольтерры

7. Повторим пример: построим фазовый портрет для модели из пункта 6.

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

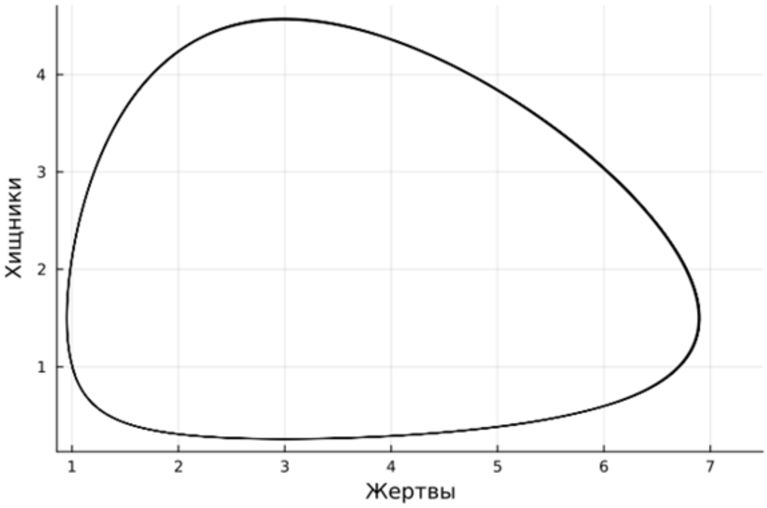

Рис. 11: Листинг и фазовый портрет

Задания для самостоятельного выполнения

1. Реализовать и проанализировать модель роста численности изолированной популяции (модель Мальтуса): ̇□ = □□, □ = □ − □. где □(□) — численность изолированной популяции в момент времени □, □ — коэффициент роста популяции, □ — коэффициент рождаемости, □ — коэффициент смертности. Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией).

Модель Мальтуса — модель роста численности изолированной популяции, где изменение роста популяции контролируется численностью уже существующей популяции, домноженной на коэффициент a, который является разницей между рождаемостью и смертностью ($b - c$). Коэффициенты b и c было предложено выбрать самостоятельно, и я выставлю для системы значения $b = 1.09$ и $c = 1.134$ (что является соответственно коэффициентами рождаемости и смертности за январь-август в 2022 году в Центральном федеральном округе РФ). Изначальная численность населения (39433556 человек) также взята из статистики Росстата за 2022 год (с учётом переписи населения).

Модель Мальтуса подразумевает, что коэффициенты рождаемости и смертности не изменяются, так что если b превышает c, численность популяции будет расти (и наоборот).

```
function Maltus!(du,u,p,t)
    du[1] = (p[1]-p[2])*u[1]
end
u0 = [39433556.0]
tspan = (0.0,100.0)
p = Float64[1.09, 1.134]
prob = ODEProblem(Maltus!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=1.0)
R1 = [tu[1] for tu in sol.u]
plot(sol.t, R1, title="Модель Мальтуса", xaxis="Время",yaxis="Численность",label=
```

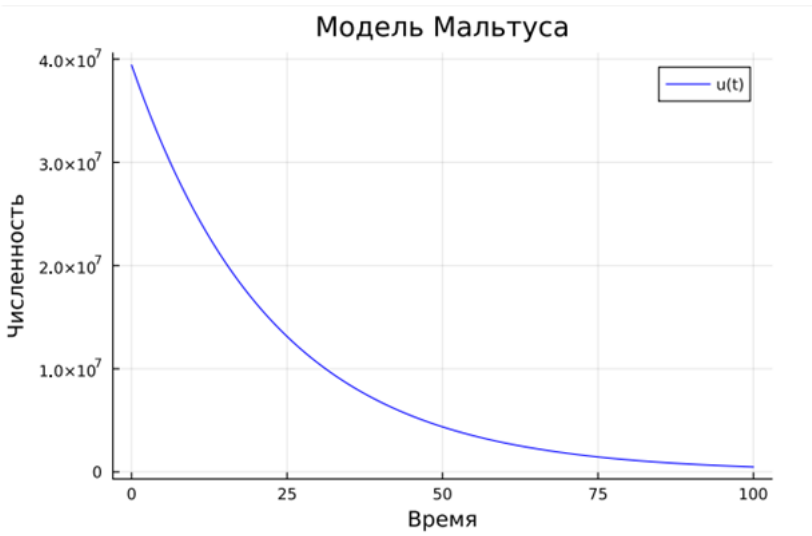

Рис. 12: Модель Мальтуса

```
anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], R1[1:i], title="Логистическая модель роста популяции", xaxis
end
```

```
gif(anim, "presentation//image//2.gif")
```

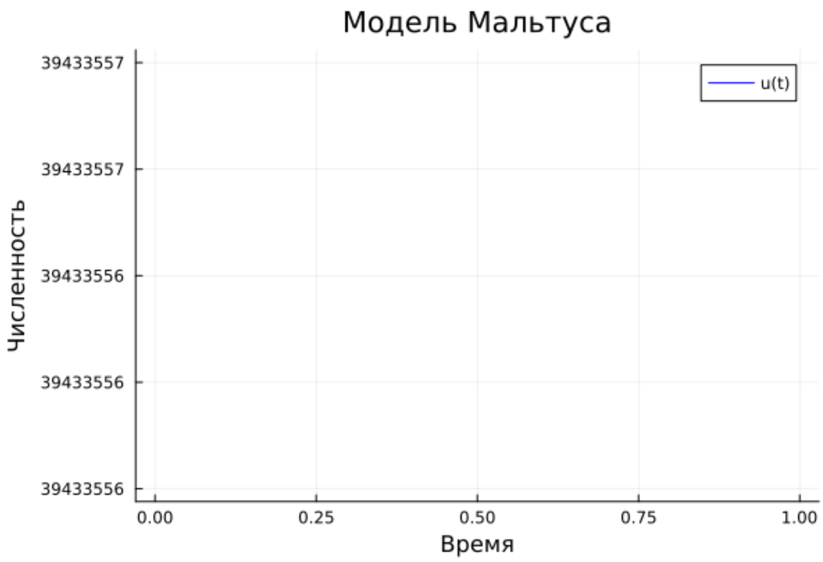

Рис. 13: Модель Мальтуса

2. Реализовать и проанализировать логистическую модель роста популяции, заданную уравнением: ˙□ = □□ (1 − □ □) , □ > 0, □ > 0, □ — коэффициент роста популяции, □ — потенциальная ёмкость экологической системы (предельное значение численности популяции). Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией).

```
function LogModPop!(du,u,p,t)
    du[1] = p[1]*u[1]*(1-u[1]/p[2])
end
u0 = [39433556.0]
tspan = (0.0,100.0)
p = Float64[1.09, 15e7]
prob = ODEProblem(LogModPop!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=1.0)
R1 = [tu[1] for tu in sol.u]
plot(sol.t, R1, title="Логистическая модель роста популяции", xaxis="Время",yaxis
```

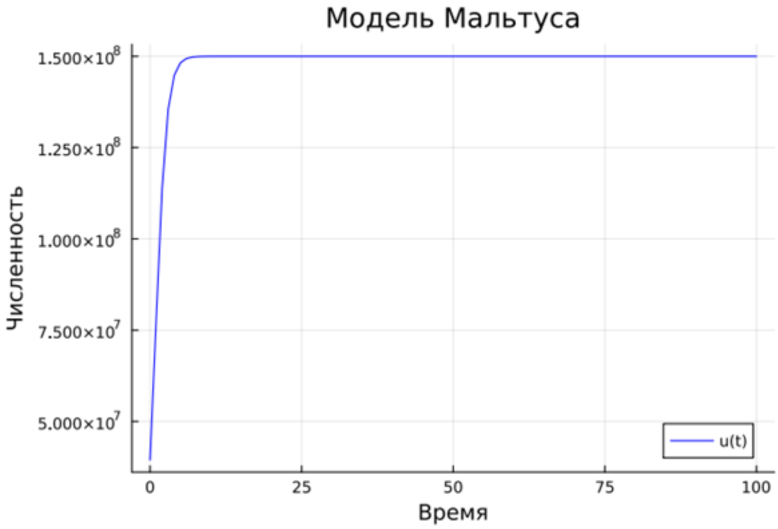

Рис. 14: Логистическая модель роста популяции

```
anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], R1[1:i], title="Логистическая модель роста популяции", xaxis
end
gif(anim, "presentation//image//2.gif")
```

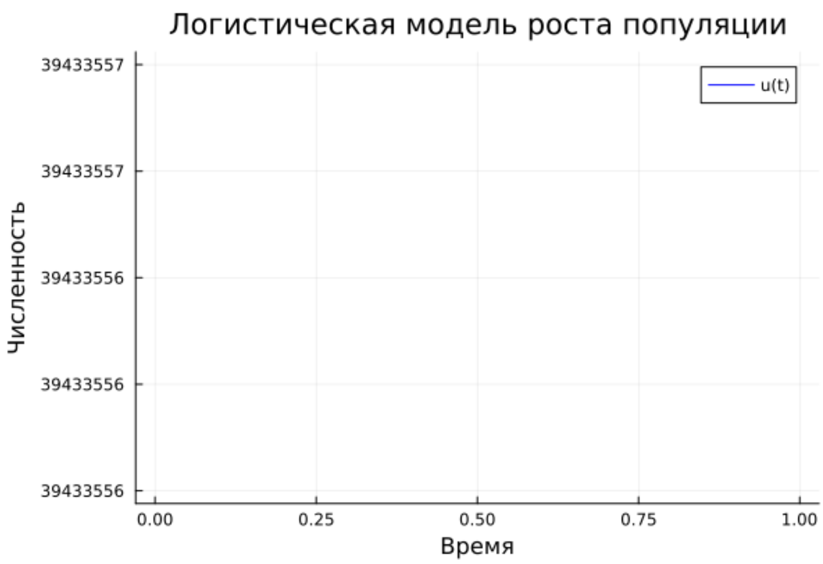

Рис. 15: Логистическая модель роста популяции

3. Модель эпидемии Кермака–Маккендрика (SIR-модель):

Реализовать и проанализировать SIR-модель:

$\dot{x}_s = -\beta is$

$\dot{x}_i = \beta is - \nu i$

$\dot{x}_r = \nu i$

где:

- s(t) — численность восприимчивых индивидов
- i(t) — численность инфицированных индивидов
- r(t) — численность переболевших индивидов
- β — коэффициент интенсивности контактов с инфицированием
- ν — коэффициент интенсивности выздоровления

$\dot{x}_s + \dot{x}_i + \dot{x}_r = 0$ (численность популяции постоянна).

Самостоятельно задать начальные данные и параметры, объяснив свой выбор. Построить графики численности каждой группы со временем (включая анимацию).

```
function SIR!(du,u,p,t)
    du[1] = -p[1]*u[1]*u[2] # S
    du[2] = p[1]*u[2]*u[1]-p[2]*u[2] # I
    du[3] = p[2]*u[2] # R
end
u0 = [39433553.0, 3.0, 0.0]
tspan = (0.0,20.0)
p = Float64[0.3,0.7]
prob = ODEProblem(SIR!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=0.01)
R1 = [tu[1] for tu in sol.u]
R2 = [tu[2] for tu in sol.u]
```

```
R3 = [tu[3] for tu in sol.u]
plot(sol.t, R1, title="SIR", xaxis="Время",yaxis="Численность",label="Susceptable
plot!(sol.t, R2, title="SIR", label="Infected", c=:red, leg=:topright)
plot!(sol.t, R3, label="Recovered", c=:blue, leg=:topright)
```

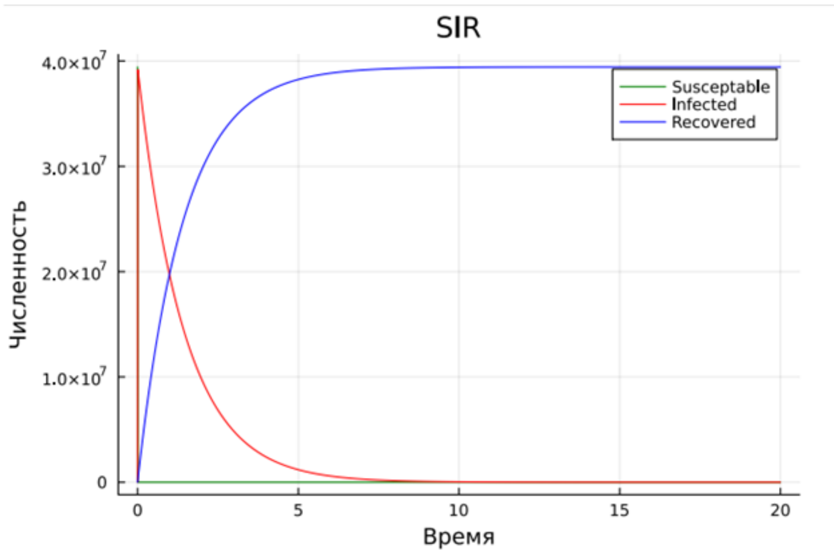

Рис. 16: SIR

```
anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], R1[1:i], title="SIR", xaxis="Время",yaxis="Численность",labe
    plot!(sol.t[1:i], R2[1:i], title="SIR", label="Infected", c=:red, leg=:toprig
    plot!(sol.t[1:i], R3[1:i], label="Recovered", c=:blue, leg=:topright)
end
gif(anim, "presentation//image//3.gif")
```

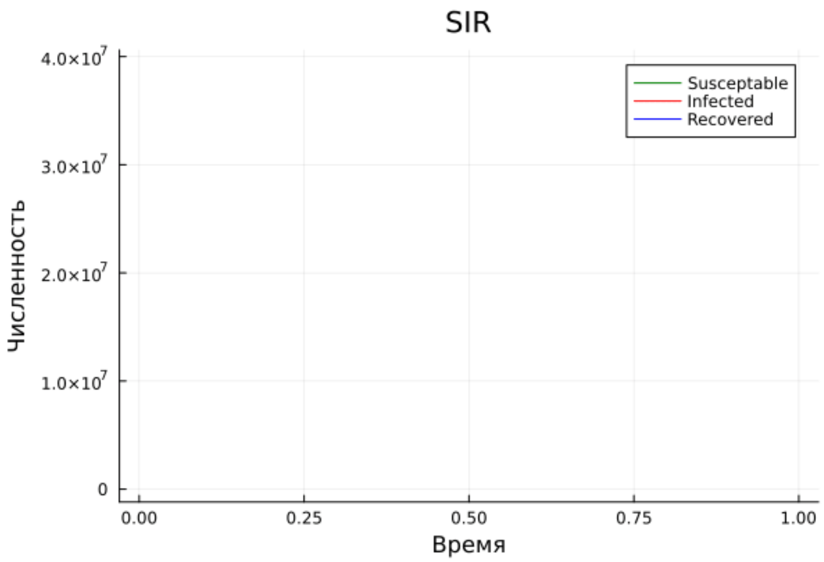

Рис. 17: SIR

4. Реализовать и проанализировать модель SEIR (Susceptible-Exposed-Infected-Removed):

```
ẋₛ(t) = -β/N * s(t)i(t)
ẋₑ(t) = β/N * s(t)i(t) - δe(t)
ẋᵢ(t) = δe(t) - γi(t)
ẋᵣ(t) = γi(t)
s(t) + e(t) + i(t) + r(t) = N
```

Сравнить результаты с SIR-моделью.

```
function SEIR!(du,u,p,t)
    betta, delta, gamma, N = p
    s, e, i, r = u
    du[1] = -betta / N * s * i
    du[2] = betta / N * s * i - delta * e
    du[3] = delta * e - gamma * i
    du[4] = gamma * i
```

```
end
u0 = [0.98, 0.02, 0.0, 0.0]
tspan = (0.0,200.0)
p = Float64[0.8,0.4,0.3,1.0]
prob = ODEProblem(SEIR!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=0.1)
R1 = [tu[1] for tu in sol.u]
R2 = [tu[2] for tu in sol.u]
R3 = [tu[3] for tu in sol.u]
R4 = [tu[4] for tu in sol.u]
plot(sol.t, R1, title="SEIR", xaxis="Время",yaxis="Численность",label="Susceptabl
plot!(sol.t, R2, label="Exposed", c=:orange, leg=:topright)
plot!(sol.t, R3, label="Infected", c=:red, leg=:topright)
plot!(sol.t, R4, label="Recovered", c=:blue, leg=:topright)
```

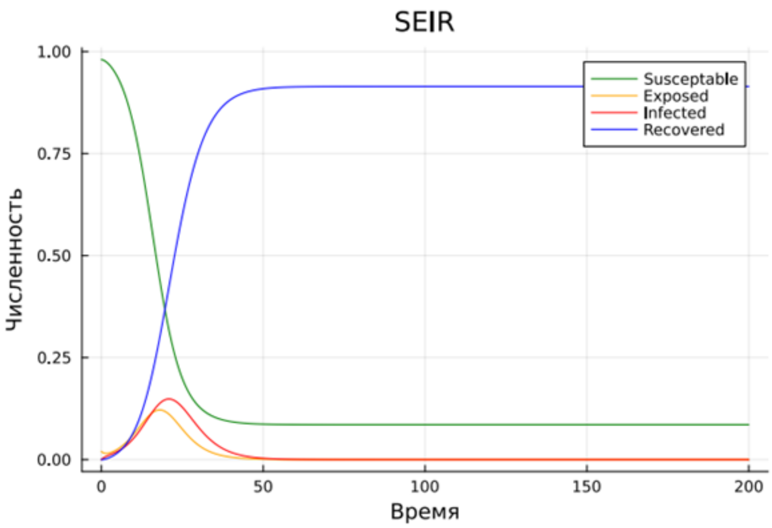

Рис. 18: SEIR

```
anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], R1[1:i], title="SEIR", xaxis="Время",yaxis="Численность",lab
```

```
    plot!(sol.t[1:i], R2[1:i], label="Exposed", c=:orange, leg=:topright)
    plot!(sol.t[1:i], R3[1:i], label="Infected", c=:red, leg=:topright)
    plot!(sol.t[1:i], R4[1:i], label="Recovered", c=:blue, leg=:topright)
end
gif(anim, "presentation//image//4.gif")
```

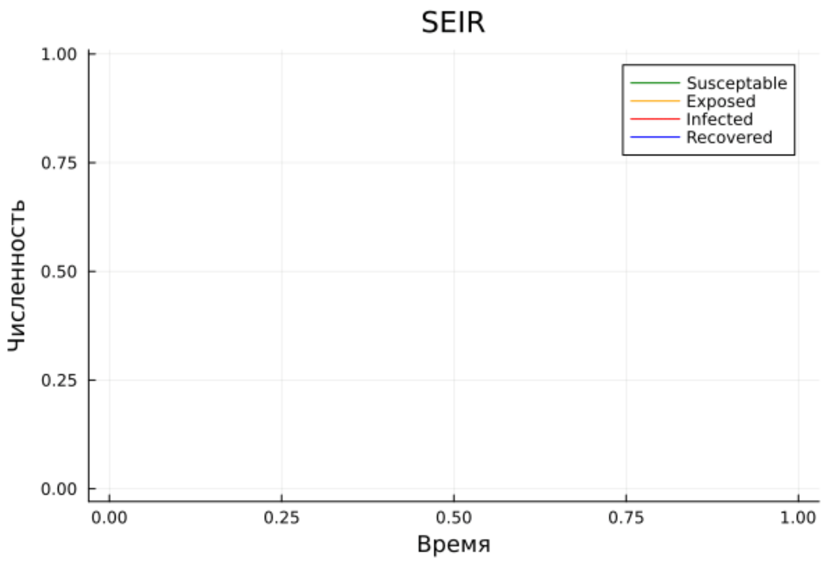

Рис. 19: SEIR

5. Дискретная модель Лотки–Вольтерры:

Для дискретной модели:

```
X₁(t + 1) = aX₁(t)(1 - X₁(t)) - X₁(t)X₂(t)
X₂(t + 1) = -cX₂(t) + dX₁(t)X₂(t)
```

с начальными данными a = 2, c = 1, d = 5 найти точку равновесия. Получить и сравнить аналитическое и численное решения. Изобразить численное решение на фазовом портрете.

```
using NLsolve
# Аналитическое решение
```

```
function find_equilibrium(a, c, d)
    function system!(du, u)
        du[1] = a*u[1]*(1-u[1]) - u[1]*u[2]
        du[2] = -c*u[2] + d*u[1]*u[2]
    end

    initial_guess = [0.5, 0.5]
    result = nlsolve(system!, initial_guess)

    equilibrium_point = result.zero
    return equilibrium_point
end
# Численное решение
function LotkiVolterry(a, c, d, x1_0, x2_0, dt, num_steps)
    x1 = x1_0
    x2 = x2_0
    results = [(x1, x2)]

    for _ in 1:num_steps
        x1_new = x1 + dt * (a * x1 * (1 - x1) - x1 * x2)
        x2_new = x2 + dt * (-c * x2 + d * x1 * x2)
        x1, x2 = x1_new, x2_new
        push!(results, (x1, x2))
    end

    return results
end

a = 2.0
```

```
c = 1.0
d = 5.0
x1_0 = 0.15
x2_0 = 0.25
dt = 0.01
num_steps = 10000

results = LotkiVolterry(a, c, d, x1_0, x2_0, dt, num_steps)
R1 = [x[1] for x in results]
R2 = [x[2] for x in results]

equilibrium = find_equilibrium(2,1,5)

plot(R1, R2, title="Лотки-Вольтерры (фазовый портрет)", leg=:topright)
scatter!([equilibrium[1]], [equilibrium[2]], color="red", label="Точка равновеси
```

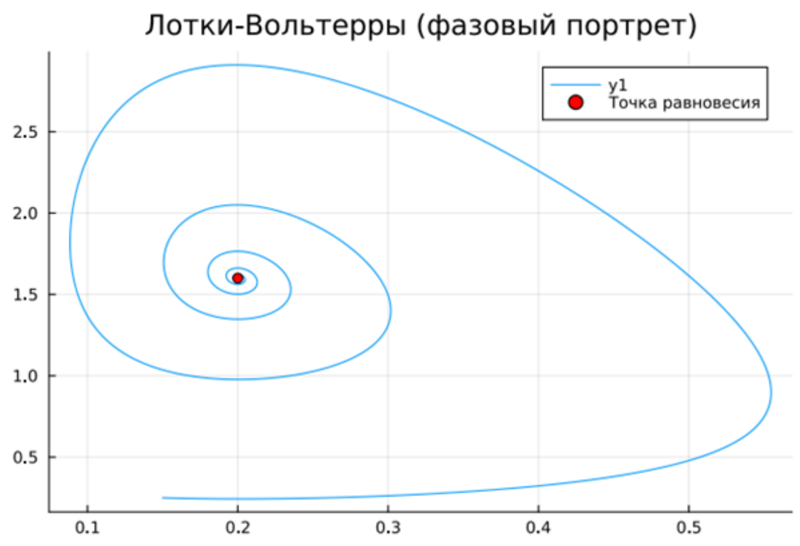

Рис. 20: модель Лотки–Вольтерры фазовый портрет

```
anim = @animate for i in 1:length(R1)
```

```
    plot(R1[1:i], R2[1:i], title="Лотки-Вольтерры (фазовый портрет)", leg=:toprig
    scatter!([equilibrium[1]], [equilibrium[2]], color="red", label="Точка равнов
end
gif(anim, "presentation//image//5.gif")
```

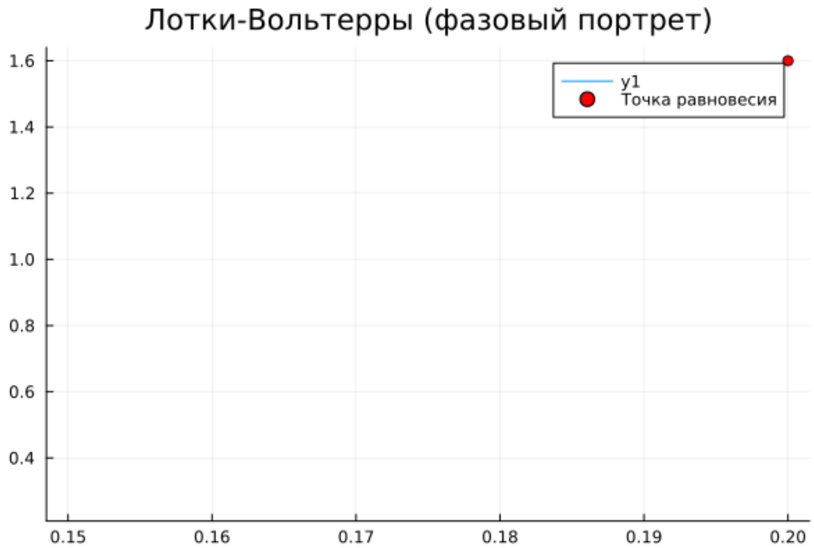

Рис. 21: модель Лотки–Вольтерры фазовый портер

6. Модель отбора на основе конкурентных отношений (Julia):

Реализовать модель:

```
ẋ = αx - βxy
ẏ = αy - βxy
```

Самостоятельно задать начальные данные и параметры, объяснив свой выбор. Построить графики и фазовый портрет (включая анимацию).

```
function KonkOtn!(du,u,p,t)
    du[1] = p[1] * u[1] - p[2] * u[1] * u[2]
    du[2] = -p[1] * u[2] + p[2] * u[1] * u[2]
end
```

```
u0 = [16.0,7.0]
tspan = (0.0,100.0)
p = Float64[0.02, 0.04]
prob = ODEProblem(KonkOtn!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=0.1)
R1 = [tu[1] for tu in sol.u]
R2 = [tu[2] for tu in sol.u]
plot(sol.t, R1, title="Конкурентные отношения", xaxis="Время",yaxis="Значение",la
plot!(sol.t, R2, label="y", c=:orange, leg=:topright)
```

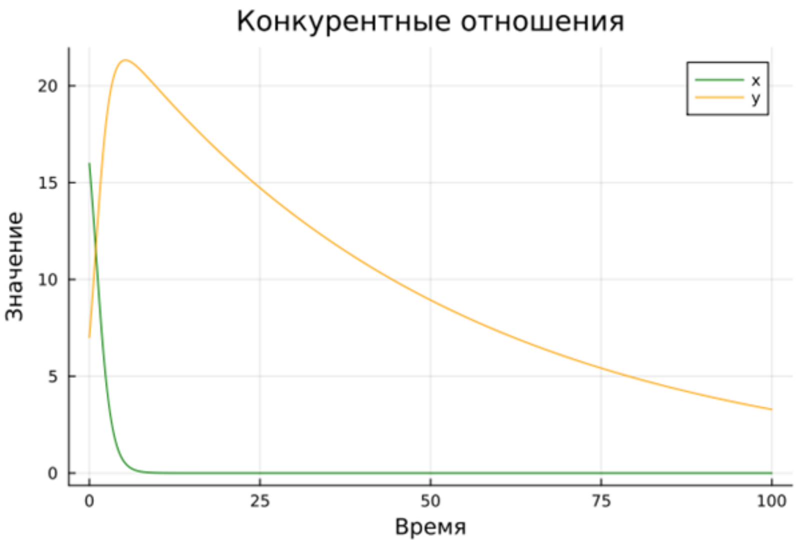

Рис. 22: Конкурентные отношения

```
anim = @animate for i in 1:length(R1)
    plot(sol.t[1:i], R1[1:i], title="Конкурентные отношения", xaxis="Время",yaxis
    plot!(sol.t[1:i], R2[1:i], label="y", c=:orange, leg=:topright)
end
gif(anim, "presentation//image//6.gif")
```

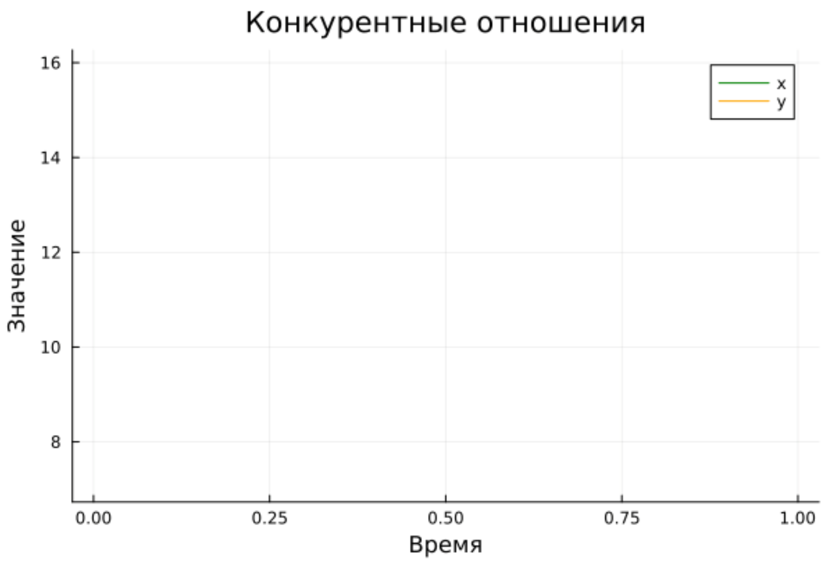

Рис. 23: Конкурентные отношения

```
plot(R1, R2, title="Конкурентные отношения (фазовый портрет)", leg=:topright)
```

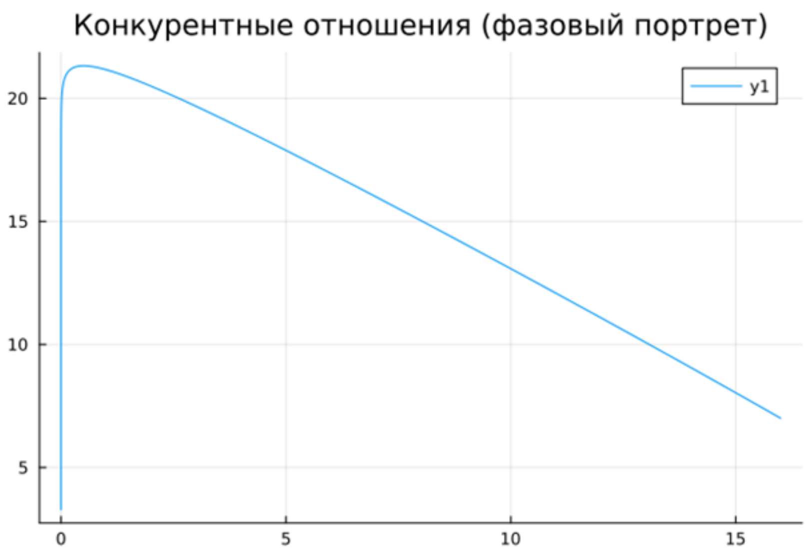

Рис. 24: Фазовый портрет

7. Консервативный гармонический осциллятор (Julia):

Реализовать модель:

$$\ddot{x} + \omega_0^2 x = 0,\quad x(t_0) = x_0,\quad \dot{x}(t_0) = y_0$$

Самостоятельно задать начальные параметры, объяснив свой выбор. Построить графики и фазовый портрет (включая анимацию).

```
function KGO!(du,u,p,t)
    du[1] = u[2]
    du[2] = -p[1]^2 * u[1]
end
u0 = [0.0, 1.0]
tspan = (0.0,10.0)
p = Float64[3.0]
prob = ODEProblem(KGO!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=0.1)
R1 = [tu[1] for tu in sol.u]
R2 = [tu[2] for tu in sol.u]
plot(sol.t, R1, title="Консервативный гармонический осциллятор", xaxis="Время",ya
plot!(sol.t, R2, label="v(t)", c=:orange, leg=:topright)
```

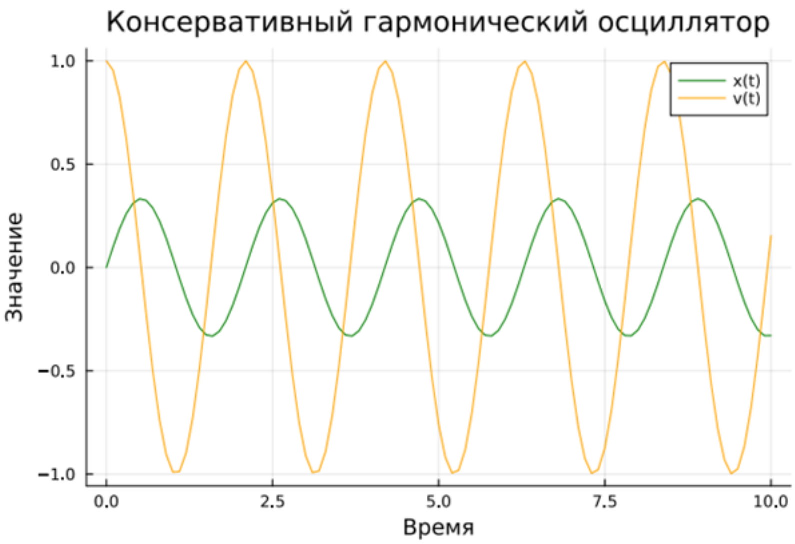

Рис. 25: Консервативный гармонический осциллятор

```
anim = @animate for i in 1:length(R1)
    plot(sol.t[1:i], R1[1:i], title="Консервативный гармонический осциллятор", xa
    plot!(sol.t[1:i], R2[1:i], label="v(t)", c=:orange, leg=:topright)
end
gif(anim, "presentation//image//7.gif")
```

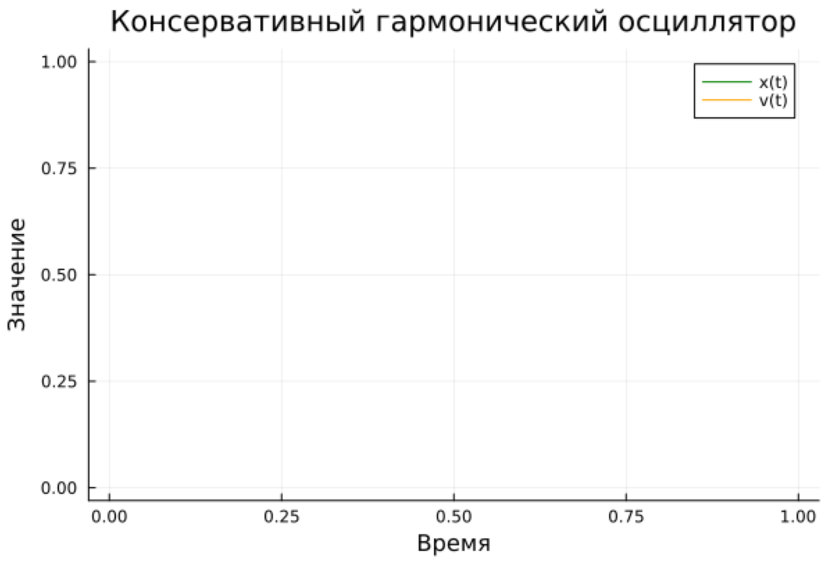

Рис. 26: Консервативный гармонический осциллятор

```
plot(R1, R2, title="Консервативный гармонический осциллятор (фазовый портрет)", l
```

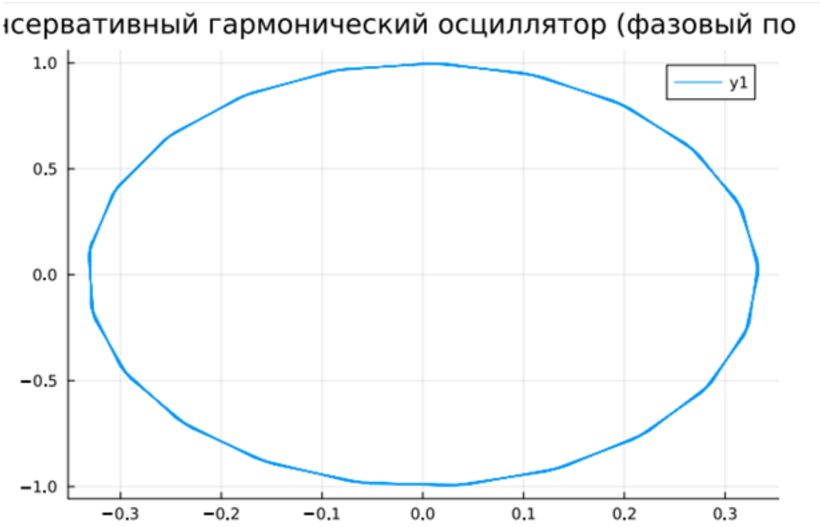

Рис. 27: Фазовый портрет

8. Свободные колебания гармонического осциллятора (Julia):

Реализовать модель затухающих колебаний:

$$\ddot{x} + 2\gamma\dot{x} + \omega_0^2 x = 0, \ x(t_0) = x_0, \ \dot{x}(t_0) = y_0$$

где γ — параметр затухания. Самостоятельно задать начальные параметры, объяснив свой выбор. Построить графики и фазовый портрет (включая анимацию).

```
function GO!(du,u,p,t)
    du[1] = u[2]
    du[2] = -2.0*p[2]*u[2] - p[1]^2*u[1]
end
u0 = [0.0, 1.0]
tspan = (0.0,10.0)
p = Float64[4.0, 0.2]
prob = ODEProblem(GO!,u0,tspan,p)
sol = solve(prob,abstol=1e-6,reltol=1e-6, saveat=0.1)
R1 = [tu[1] for tu in sol.u]
```

```
R2 = [tu[2] for tu in sol.u]
plot(sol.t, R1, title="Гармонический осциллятор", xaxis="Время",yaxis="Значение",
plot!(sol.t, R2, label="v(t)", c=:orange, leg=:topright)
```

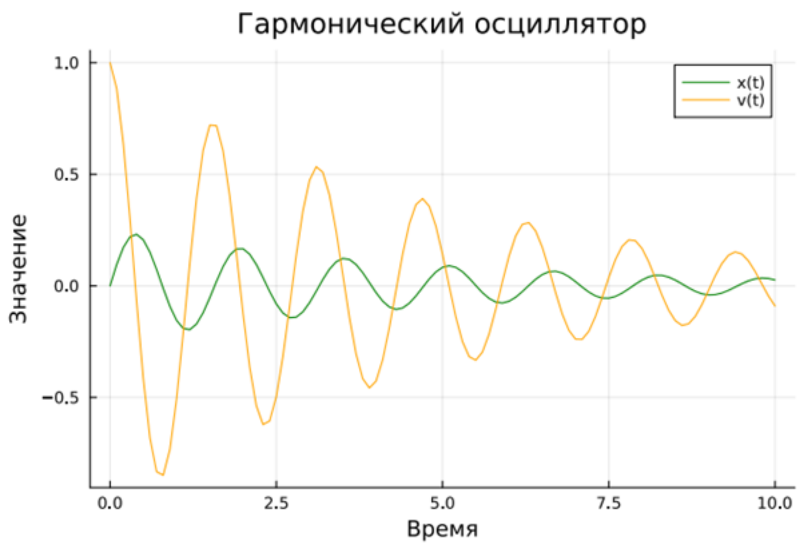

Рис. 28: Гармонический осциллятор

```
anim = @animate for i in 1:length(R1)
    plot(sol.t[1:i], R1[1:i], title="Гармонический осциллятор", xaxis="Время",yax
    plot!(sol.t[1:i], R2[1:i], label="v(t)", c=:orange, leg=:topright)
end
gif(anim, "presentation//image//8.gif")
```

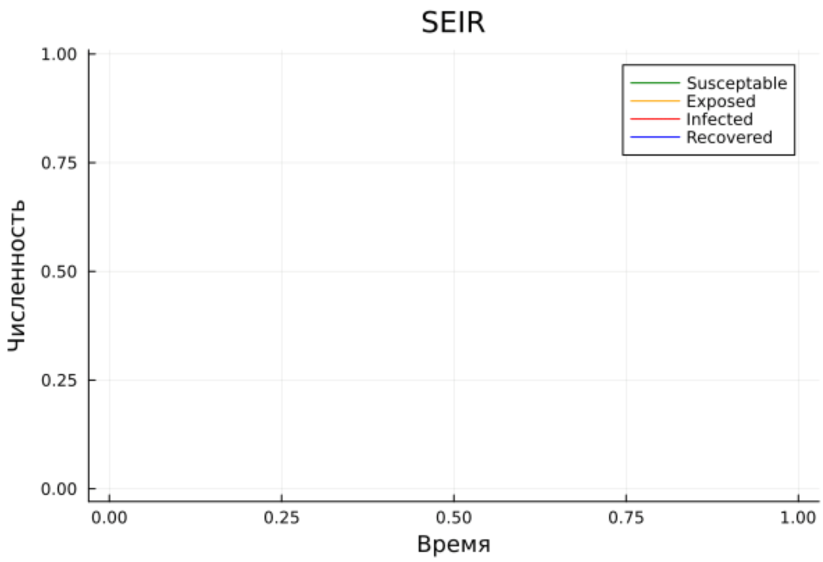

Рис. 29: Гармонический осциллятор

```
plot(R1, R2, title="Гармонический осциллятор (фазовый портрет)", leg=:topright)
```

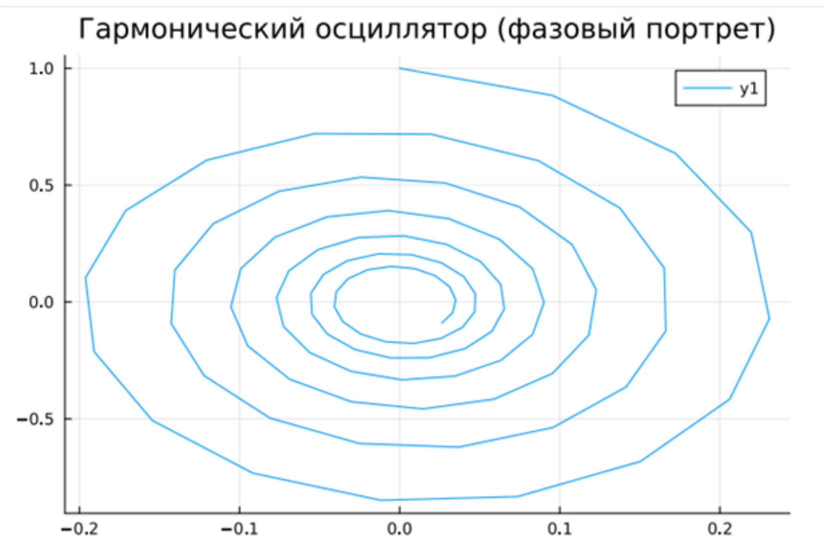

Рис. 30: Фазовый портрет

Выводы по проделанной работе

Вывод

В результате выполнения работы мы освоили специализированные пакеты для решения задач в непрерывном и дискретном времени. Были записаны скринкасты выполнения , создания отчета, презентации и защиты лабораторной работы.

Список литературы

- Julia: https://ru.wikipedia.org/wiki/Julia
- https://julialang.org/packages/
- https://juliahub.com/ui/Home
- https://juliaobserver.com/
- https://github.com/svaksha/Julia.jl