

Лабораторная работа 6

Решение моделей в непрерывном и дискретном времени

Ланцова Яна Игоревна

Содержание

1 Цель работы	6
2 Задание	7
3 Выполнение лабораторной работы	8
3.1 Задание 1	11
3.2 Задание 2	14
3.3 Задание 3	15
3.4 Задание 4	17
3.5 Задание 5	19
3.6 Задание 6	20
3.7 Задание 7	22
3.8 Задание 8	23
4 Выводы	26

Список иллюстраций

3.1 Подключим основные пакеты	8
3.2 Модель экспоненциального роста	8
3.3 График модели экспоненциального роста	9
3.4 График модели экспоненциального роста	9
3.5 Система Лоренца	9
3.6 График системы Лоренца	10
3.7 Подключим основные пакеты	10
3.8 Модель Лотки–Вольтерры	10
3.9 График модели Лотки–Вольтерры	11
3.10 Фазовый портрет Лотки–Вольтерры	11
3.11 Задание №1	12
3.12 Задание №1	12
3.13 График №1	13
3.14 Анимация №1	13
3.15 Анимация №1	13
3.16 Задание №2	14
3.17 График №2	15
3.18 Анимация №2	15
3.19 Задание №3	16
3.20 График №3	16
3.21 Анимация №3	17
3.22 Анимация №3	17
3.23 Задание №4	18
3.24 График №4	19
3.25 Задание №5	19
3.26 График №5	20
3.27 Задание №6	21
3.28 График №6	21
3.29 Анимация №6	21
3.30 Задание №7	22
3.31 График №7	22
3.32 График №7	23
3.33 График №7	23
3.34 Анимация №7	23
3.35 Задание №8	24
3.36 График №8	24
3.37 График №8	25

Список таблиц

1 Цель работы

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

2 Задание

1. Используя JupyterLab, повторить примеры. При этом дополнить графики обозначениями осей координат, легендой с названиями траекторий, названиями графиков и т.п.
2. Выполнить задания для самостоятельной работы.

3 Выполнение лабораторной работы

Выполним примеры из лабораторной работы для знакомства с работой с различными моделями и способами их задания решения (рис. 3.1 - 3.10).

```
[*]: # подключаем необходимые пакеты:
import Pkg
Pkg.add("DifferentialEquations")
using DifferentialEquations

Updating registry at `~/julia/registries/General.toml'
Resolving package versions...
Installed DiffEqBase.jl ━━━━━━ v1.5.2
Installed OrdinaryDiffEqRPN ━━━━━━ v1.5.0
Installed OrdinaryDiffEqRosenbrock ━━━━ v1.18.1
Installed SciMLPublic ━━━━━━ v1.0.0
Installed HypergeometricFunctions ━━ v0.3.28
Installed DifferentialEquations ━━ v7.17.0
Installed OrdinaryDiffEqStabilizedRK ━━ v1.4.0
Installed OrdinaryDiffEqSolve ━━ v1.12.0
Installed SciMLLogging ━━ v1.5.0
Installed Accessors ━━ v0.1.42
Installed AlmostBlockDiagonals ━━ v0.1.10
Installed BoundaryValueDiffEqFIRK ━━ v1.9.0
```

Рис. 3.1: Подключим основные пакеты

```
[3]: # 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)

[3]: retcode: Success
Interpolation: 3rd order Hermite
t: 5-element Vector{Float64}:
 0.0
 0.19842494449239292
 0.3521866297865888
 0.5034436122197829
 1.0
u: 5-element Vector{Float64}:
 1.0
 1.1034222047865465
 1.42121908713484919
 1.9738384457359198
 2.6644561424814266
```

Рис. 3.2: Модель экспоненциального роста

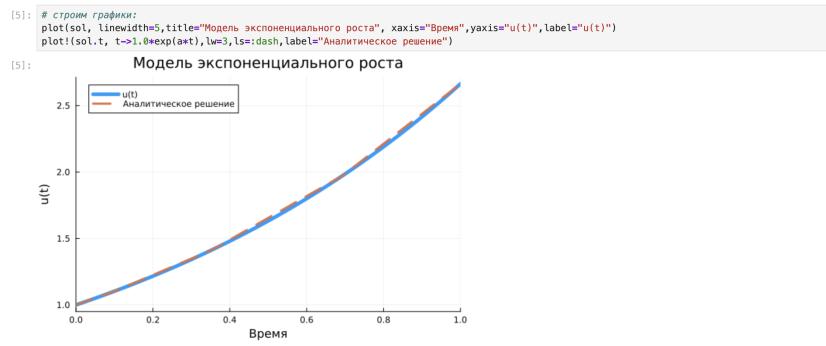


Рис. 3.3: График модели экспоненциального роста

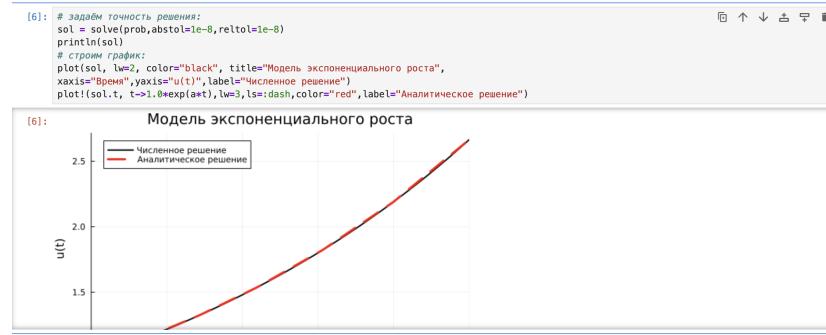


Рис. 3.4: График модели экспоненциального роста



Рис. 3.5: Система Лоренца

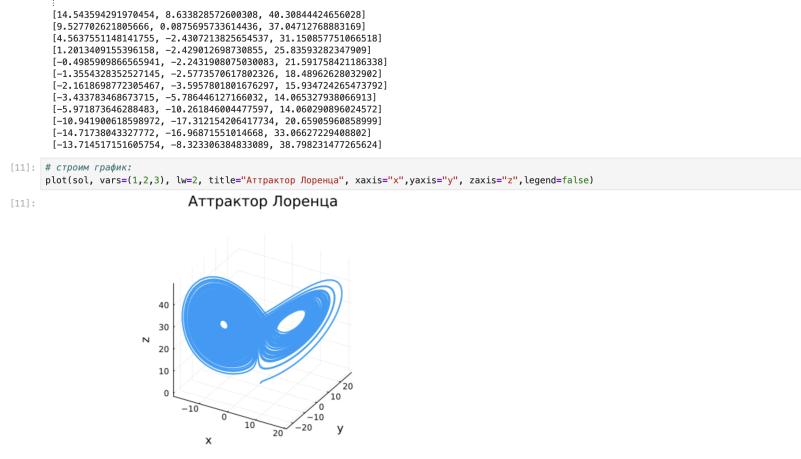


Рис. 3.6: График системы Лоренца

Модель Лотки–Вольтерры

```
[21]: # подключаем необходимые пакеты:
import Pkg
Pkg.add("ParameterizedFunctions")
using ParameterizedFunctions, DifferentialEquations, Plots;

Resolving package versions...
Installed Unitful ━━━━━━ v1.25.1
Installed OffGridCurves ━━━━━━ v1.17.0
Installed GridTools ━━━━━━ v0.1.13
Installed FindFirstFunctions ━━━━━━ v1.4.2
Installed Bijections ━━━━━━ v0.2.2
Installed DomainSets ━━━━━━ v0.7.16
Installed MutableArithmetics ━━━━━━ v1.6.7
Installed JuliaFormatter ━━━━━━ v2.2.0
Installed MultivariatePolynomials ━━━━ v0.1.13
Installed CompatiblityTypes ━━━━ v1.1.4
Installed Glob ━━━━━━ v1.3.1
Installed JuliaSyntax ━━━━━━ v0.4.10
Installed DispatchDoctor ━━━━━━ v0.4.26
Installed MLStyleDoctor ━━━━━━ v0.4.17
Installed Symbolics ━━━━━━ v6.57.0
Installed Combinatorics ━━━━━━ v0.9.1
Installed DynamicPolynomials ━━━━ v0.1.4
Installed Combinatorics ━━━━ v1.0.2
Installed TaskLocalValues ━━━━ v0.1.3
Installed AbstractTrees ━━━━ v0.4.5
```

Рис. 3.7: Подключим основные пакеты

```
[12]: # 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)

r Warning: Independent variable t should be defined with @independent_variables t.
└ @ ModelingToolkit ~/julia/packages/ModelingToolkit/BTOXs/src/utils.jl:121
```

Рис. 3.8: Модель Лотки–Вольтерры

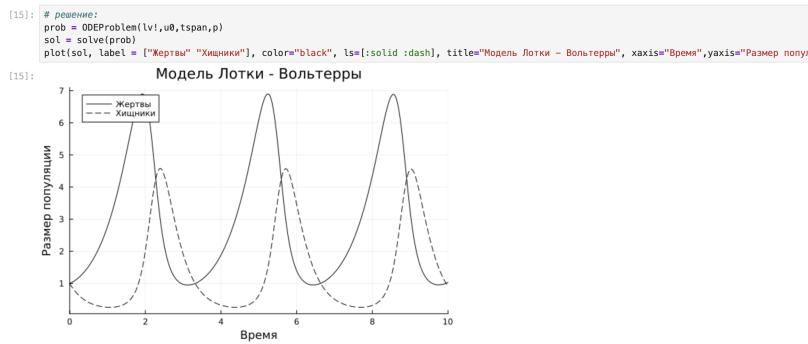


Рис. 3.9: График модели Лотки–Вольтерры

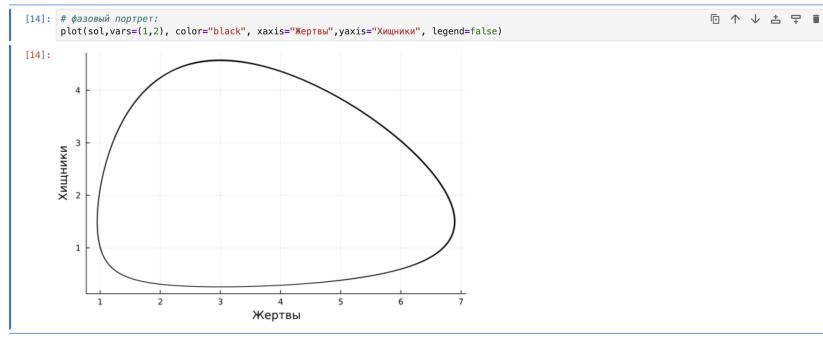


Рис. 3.10: Фазовый портрет Лотки–Вольтерры

Теперь перейдем к заданиям для самостоятельного выполнения

3.1 Задание 1

Реализуем и проанализируем модель роста численности изолированной популяции (модель Мальтуса):

$$\dot{x} = ax, \quad a = b - c$$

где:

- $x(t)$ – численность изолированной популяции в момент времени t
- a – коэффициент роста популяции
- b – коэффициент рождаемости
- c – коэффициент смертности

Начальные данные и параметры зададим самостоятельно (рис. 3.11 - 3.15).

При коэффициенте роста 0.1 рост умеренный, малая начальная популяция была выбрана для наглядности экспоненциального роста.

Задания для самостоятельного выполнения

Задание 1

Реализовать и проанализировать модель роста численности изолированной популяции (модель Мальтуса): $\dot{x} = ax$, $a = b - c$. Где $x(t)$ — численность изолированной популяции в моменте времени t .

```
[16]: # Настройки графиков
gr(size=800, 600), legend=:topleft)

[16]: Plots.GRBackend()

[17]: function malthus_model!(du, u, p, t)
    x = u[1]
    a = p[1]

    du[1] = a * x
end

[17]: malthus_model! (generic function with 1 method)

[18]: # Параметры для модели Мальтуса
a = 0.1 # коэффициент роста (рождаемость 0.15 – смертность 0.85)
x0_malthus = [10.0] # начальная численность популяции
tspan_malthus = (0.0, 50.0)

[18]: (0.0, 50.0)
```

Рис. 3.11: Задание №1

```
[19]: # Решение ОДУ
prob_malthus = ODEProblem(malthus_model!, x0_malthus, tspan_malthus, [a])
sol_malthus = solve(prob_malthus, Tsit5())

[19]: retcode: Success
Interpolation: specialized 4th order "free" interpolation
t: 12-element Vector{Float64}:
 0.0
 0.15849248898571244
 1.5625439226754578
 4.329557234194189
 7.310159848499677
 12.9915574913694
 17.123537618951076
 22.893925314886125
 29.34606507464884
 36.42928307562584
 44.08200395722155
 50.0
u: 12-element Vector{Vector{Float64}}:
 [10.0]
 [10.159755144258059]
 [11.691235813782926]
 [15.417552927260108]
 [21.836805177072534]
 [33.124804124053]
 [55.418959708084995]
 [98.69916064589857]
 [188.13983116174498]
 [382.0304751088252]
 [821.1952074572912]
 [1484.0920812399486]
```

Рис. 3.12: Задание №1

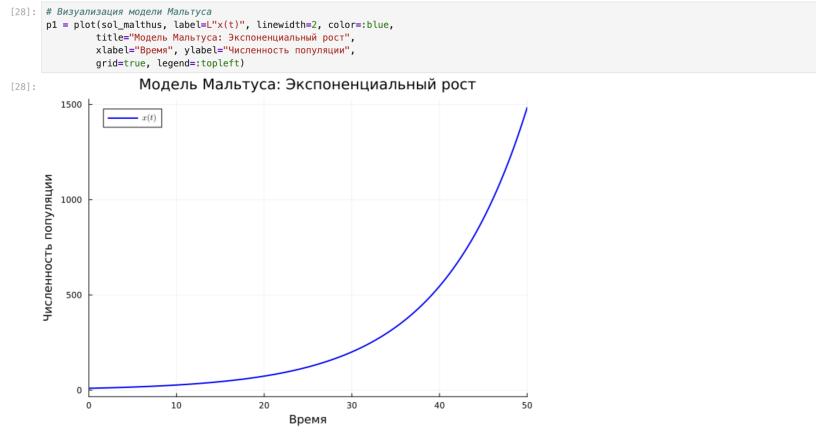


Рис. 3.13: График №1

```
[29]: # Анимация для модели Мальтуса
anim_malthus = @animate for t in range(0, 50, length=200)
    plot!(sol_malthus, label="x(t)", linewidth=2, color:blue,
          title="Модель Мальтуса: Экспоненциальный рост",
          xlabel="Время", ylabel="Численность популяции",
          xlims=(0, 50), ylims=(0, 1500))

    current_t = min(t, 50)
    current_x = x0_malthus[1] * exp(a * current_t)

    scatter!([current_t], [current_x], color:red, markersize=6,
            label="Текущее состояние")

    annotate!(40, 1200, text("t = $(round(current_t, digits=1))", 10))
    annotate!(40, 1000, text("x = $(round(Int, current_x))", 10))
end

gif(anim_malthus, "malthus_model.gif", fps=15)
[I: Info: Saved animation to /Users/yana/malthus_model.gif]
```

Рис. 3.14: Анимация №1

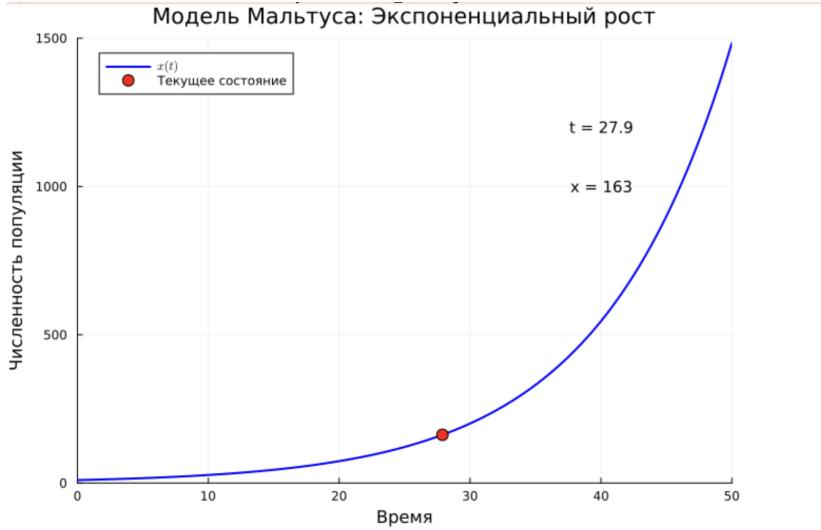


Рис. 3.15: Анимация №1

3.2 Задание 2

Реализуем и проанализируем логистическую модель роста популяции, заданную уравнением:

$$\dot{x} = rx \left(1 - \frac{x}{k}\right), \quad r > 0, \quad k > 0$$

где:

- r — коэффициент роста популяции
- k — потенциальная ёмкость экологической системы (предельное значение численности популяции)

Начальные данные и параметры зададим самостоятельно (рис. 3.16 - 3.18). В данной модели $r = 0.2$ - реалистичный коэффициент роста, в то время как $K = 1000$ - ограниченная ёмкость среды,

```
Задание 2
Реализовать и проанализировать логистическую модель роста популяции, заданную уравнением, где r — коэффициент роста популяции, k — потенциальная ёмкость экологической системы (предельное значение численности популяции). Начальные данные и параметры задать самостоятельно и пояснить их выбор

[30]: function logistic_model!(du, u, p, t)
    x = u[1]
    r, k = p
    du[1] = r * x * (1 - x/k)
end

[30]: logistic_model! (generic function with 1 method)

[31]: # Параметры для логистической модели
      r = 0.2 # коэффициент роста
      k = 1000.0 # ёмкость среды
      x0_logistic = [10.0] # начальная численность
      tspan_logistic = [0.0, 100.0]

prob_logistic = ODEProblem(logistic_model!, x0_logistic, tspan_logistic, [r, k])
sol_logistic = solve(prob_logistic, Tsit5())

[31]: retcode: Success
Interpolation: specialized 4th order "free" interpolation
t: 21-element Vector{Float64}:
  0.1382533448945986
  0.9774069399840323
  2.4583059495178625
  4.260928341498329
  6.5038514637708425
  9.128411459671941
  12.169582447083334
  15.624078225564404
```

Рис. 3.16: Задание №2

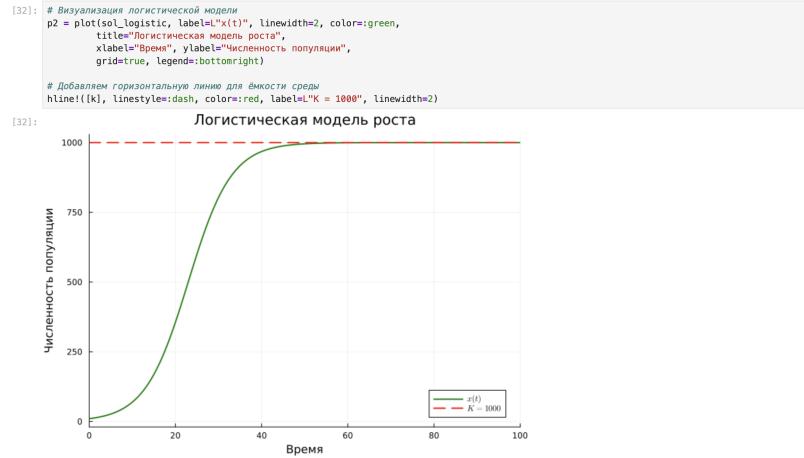


Рис. 3.17: График №2

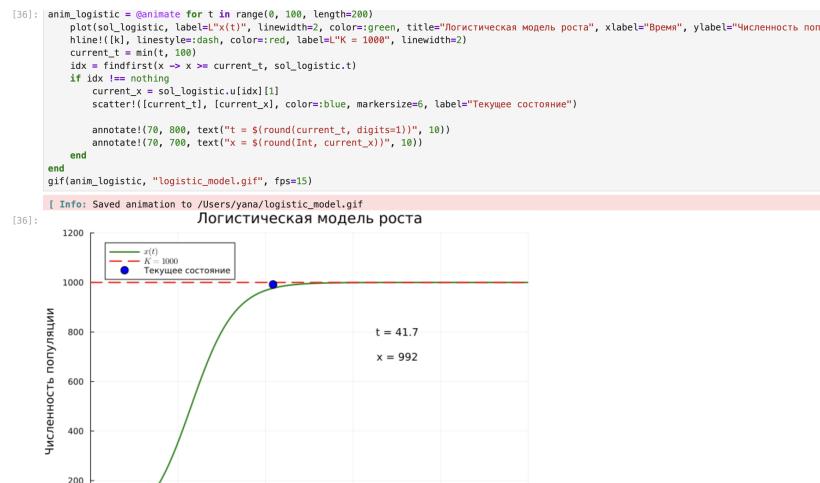


Рис. 3.18: Анимация №2

3.3 Задание 3

Реализуем и проанализировав модель эпидемии Кермака–Маккендрика (рис.

3.19 - 3.22):

$$\begin{cases} \dot{s} = -\beta i s, \\ \dot{i} = \beta i s - \nu i, \\ \dot{r} = \nu i, \end{cases}$$

где:

- $s(t)$ — численность восприимчивых к болезни индивидов в момент времени t
- $i(t)$ — численность инфицированных индивидов в момент времени t
- $r(t)$ — численность переболевших индивидов в момент времени t
- β — коэффициент интенсивности контактов индивидов с последующим инфицированием
- ν — коэффициент интенсивности выздоровления инфицированных индивидов

Задание 3

Реализовать и проанализировать модель эпидемии Кермака–Маккендрика (SIRмодель), где $s(t)$ — численность восприимчивых к болезни индивидов в момент времени t , $i(t)$ — численность инфицированных индивидов в момент времени t , $r(t)$ — численность переболевших индивидов в момент времени t , β — коэффициент интенсивности контактов индивидов с последующим инфицированием, ν — коэффициент интенсивности выздоровления инфицированных индивидов. Численность популяции считается постоянной. Начальные данные и параметры задать самостоятельно и пояснить их выбор.

```
[37]: 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

[37]: sir_model! (generic function with 1 method)

[38]: # Параметры для SIR модели
      β = 0.3 # коэффициент заражения
      ν = 0.1 # коэффициент выздоровления
      N = 1000 # общая численность популяции

# Начальные условия: 1 зараженный, остальные восприимчивые
s0 = (N - 1) / N
i0 = 1 / N
r0 = 0.0

u0_sir = [s0, i0, r0]
tspan_sir = (0.0, 200.0)

prob_sir = ODEProblem(sir_model!, u0_sir, tspan_sir, [β, ν])
sol_sir = solve(prob_sir, Tsit5())

[38]: retcode: Success
Interpolation: specialized 4th order "free" interpolation
```

Рис. 3.19: Задание №3

```
[44]: # Визуализация SIR модели
p3 = plot(sol_sir, label=[L"s(t)" L"i(t)" L"r(t)"], linewidth=2,
          palette=[:purple, :red, :green],
          title="Модель эпидемии SIR",
          xlabel="Время", ylabel="Доля популяции",
          grid=true, legend=:right)
```

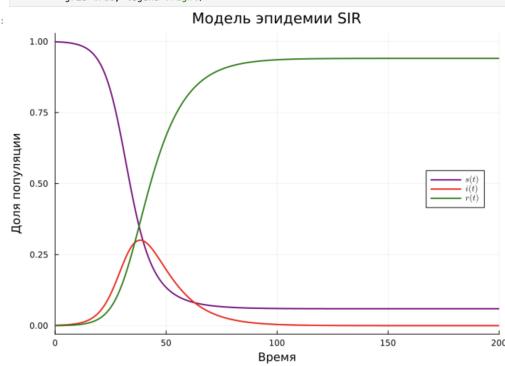


Рис. 3.20: График №3

```
[46]: # Анимация для SIR модели
anim_sir = @animate for t in range(0, 200, length=300)
    plot(sol_sir, label=[L"s(t)" L"i(t)" L"r(t)"], linewidth=2, palette=:blue, :red, :green), title="Модель эпидемии SIR",
        xlabel="Время", ylabel="Доля популяции",
        xlims=(0, 200), ylims=(0, 1),
        grid=true, legend=:right)
    current_t = min(t, 200)
    idx = findfirst(x -> x >= current_t, sol_sir.t)
    if idx == nothing
        current_s = sol_sir.u[idx][1]
        current_i = sol_sir.u[idx][2]
        current_r = sol_sir.u[idx][3]
    else
        scatter!([current_t, current_t], [current_s, current_i, current_r], color=:blue, :red, :green], markersize=6, label="")
        annotate!(150, 0.8, text("t = $(round(current_t, digits=3))", 10))
        annotate!(150, 0.7, text("S = $(round(current_s, digits=3))", 10, :blue))
        annotate!(150, 0.6, text("I = $(round(current_i, digits=3))", 10, :red))
        annotate!(150, 0.5, text("R = $(round(current_r, digits=3))", 10, :green))
    end
end
```

Рис. 3.21: Анимация №3

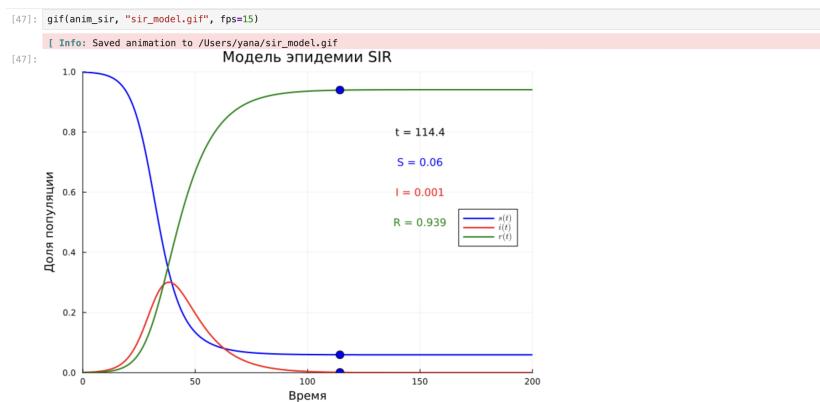


Рис. 3.22: Анимация №3

3.4 Задание 4

Как расширение модели SIR (Susceptible-Infected-Removed) по результатам эпидемии испанки была предложена модель SEIR (Susceptible-Exposed-Infected-Removed):

$$\begin{cases} \dot{s}(t) = -\frac{\beta}{N}s(t)i(t), \\ \dot{e}(t) = \frac{\beta}{N}s(t)i(t) - \delta e(t), \\ \dot{i}(t) = \delta e(t) - \gamma i(t), \\ \dot{r}(t) = \gamma i(t). \end{cases}$$

где:

- $s(t)$ — численность восприимчивых особей

- $e(t)$ — численность экспонированных (латентно инфицированных) особей
- $i(t)$ — численность инфицированных и заразных особей
- $r(t)$ — численность выздоровевших или умерших особей
- β — коэффициент заражения
- δ — коэффициент перехода из экспонированного в инфицированное состояние
- γ — коэффициент выздоровления
- N — общая численность популяции

Размер популяции сохраняется:

$$s(t) + e(t) + i(t) + r(t) = N$$

Исследуем модель SEIR и сравним с классической SIR-моделью (рис. 3.23 - 3.24):

Задание 4

Как расширение модели SIR (Susceptible-Infected-Removed) по результатам эпидемии испанки была предложена модель SEIR (Susceptible-Exposed-Infected-Removed). Исследуйте, сравните с SIR.

```
[48]: 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
seir_model! (generic function with 1 method)

[49]: # Параметры для SEIR модели
β_seir = 0.4 # коэффициент заражения
δ_seir = 0.2 # скорость перехода из экспонированных в инфицированные (1/инкубационный период)
γ_seir = 0.1 # скорость выздоровления
N_seir = 1000 # общая численность популяции

# Начальные условия: 1 экспонированный, остальные восприимчивые
s0_seir = (N_seir - 1) / N_seir
e0_seir = 1 / N_seir
i0_seir = 0.0
r0_seir = 0.0
u0_seir = [s0_seir, e0_seir, i0_seir, r0_seir]
tspan_seir = (0.0, 200.0)

prob_seir = ODEProblem(seir_model!, u0_seir, tspan_seir, [β_seir, δ_seir, γ_seir, N_seir])
sol_seir = solve(prob_seir, Tsit5());

[51]: retcod: Success
Interpolation: specialized 4th order "free" interpolation
t: 23-element Vector{Float64}:
```

Рис. 3.23: Задание №4

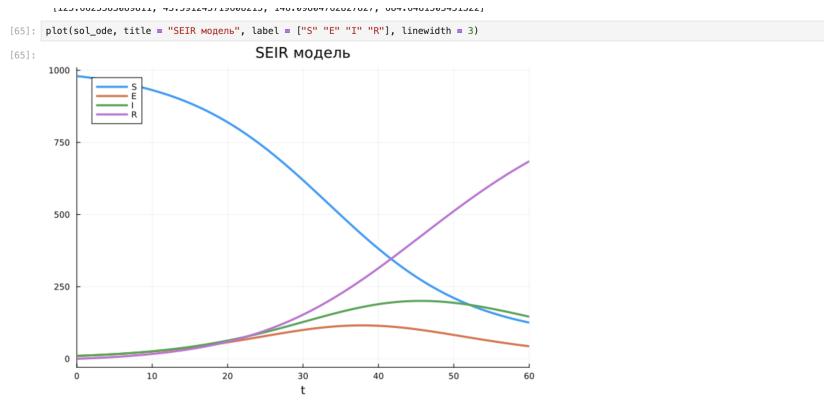


Рис. 3.24: График №4

3.5 Задание 5

Для дискретной модели Лотки–Вольтерры:

$$\begin{cases} X_1(t+1) = aX_1(t)(1 - X_1(t)) - X_1(t)X_2(t), \\ X_2(t+1) = -cX_2(t) + dX_1(t)X_2(t). \end{cases}$$

с начальными данными $a = 2, c = 1, d = 5$:

Найдите точку равновесия, получите и сравните аналитическое и численное решения, изобразите последнее на фазовом портрете (рис. 3.25 - 3.26):

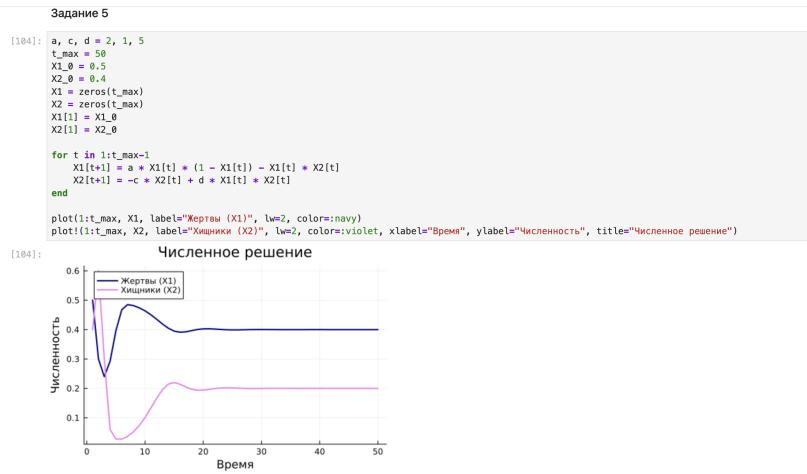


Рис. 3.25: Задание №5

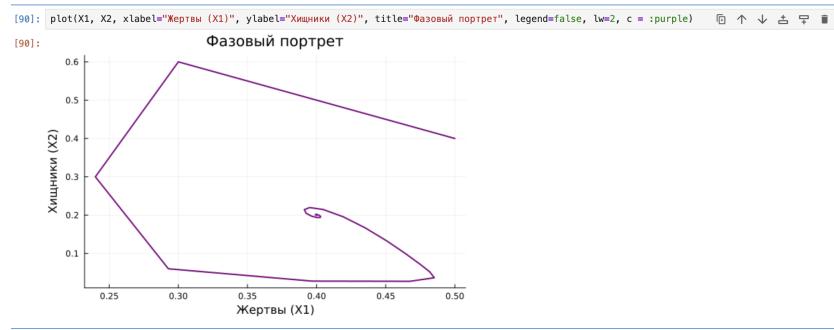


Рис. 3.26: График №5

3.6 Задание 6

Реализовать на языке Julia модель отбора на основе конкурентных отношений:

$$\begin{cases} \dot{x} = \alpha x - \beta xy, \\ \dot{y} = \alpha y - \beta xy. \end{cases}$$

где:

- x, y — численности конкурирующих популяций
- α — коэффициент естественного прироста
- β — коэффициент конкуренции

Начальные данные и параметры зададим самостоятельно (рис. 3.27 - 3.29).

Параметр альфа задан таким образом, чтобы рост обоих видов был умеренным.

Интенсивность конкуренции усредненная.

```

Задание 6

[120]: function competition(u,p,t)
    % (x, y) - u
    % a = p
    dx = a*x - b*x*y
    dy = a*y - b*x*y
    return [dx, dy]
end

[120]: competition (generic function with 1 method)

[126]: tspan = [0, 50.0]
        p = [20, 0.01]
        prob = ODEProblem(competition, u0, tspan, p)
        sol = solve(prob, Rodas5())
[126]: retcode: Success
Interpolation: specialized 4th (Rodas5P = 5th) order "free" stiffness-aware interpolation
t: 52-element Vector{Float64}:
0.0
0.182695914446915
0.342196311611425
0.6394180980338537
1.0851046629768019
1.4545533187746
1.82485390898905
2.3929405708749084
2.862028907581632
3.33115610642326
3.7531492323408796
4.02990876517505
4.249530583622125
45.73323599666626
46.13215492307385
46.53898384648107

```

Рис. 3.27: Задание №6

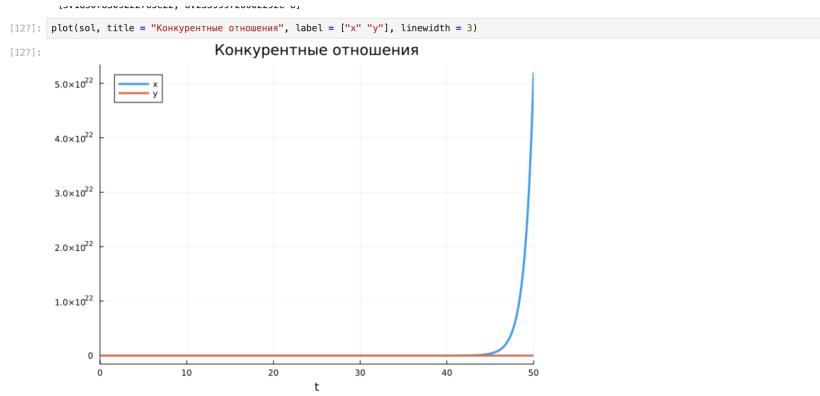


Рис. 3.28: График №6

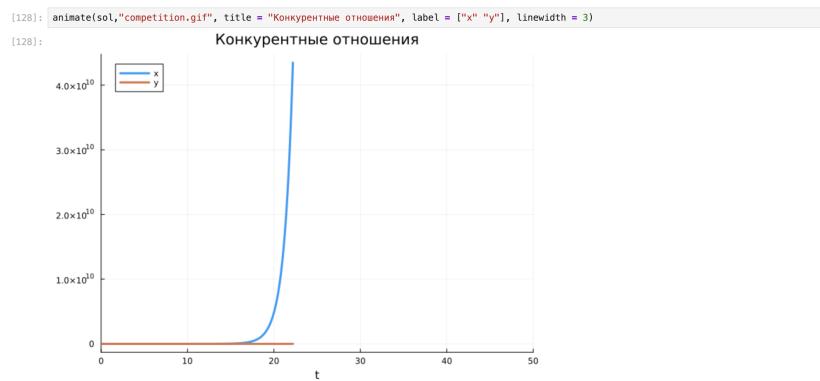


Рис. 3.29: Анимация №6

3.7 Задание 7

Реализовать на языке Julia модель консервативного гармонического осциллятора:

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

где:

- ω_0 – циклическая частота
- x_0 – начальное положение
- y_0 – начальная скорость

Начальные данные и параметры зададим самостоятельно (рис. 3.30 - 3.34).

```
Задание 7
29]: function harmonic_oscillator!(du, u, p, t)
    x, y = u # y = dx/dt
    ux = p[1]
    du[1] = y # dx/dt = y
    du[2] = -ux^2 * x # dy/dt = -ux^2 x
end
29]: harmonic_oscillator! (generic function with 1 method)

31]: # Параметры консервативного осциллятора
      ω_сons = 2.0 # циклическая частота

      # Начальные условия: различные амплитуды и фазы
      initial_conditions_osc = [
          [1.0, 0.0], # Начальное отклонение, нулевая скорость
          [2.0, 0.0], # Большее отклонение
          [0.0, 3.0], # Нулевое отклонение, начальная скорость
          [1.5, 2.0] # Комбинированные условия
      ]
      tspan_osc = (0.0, 10.0);
```

Рис. 3.30: Задание №7

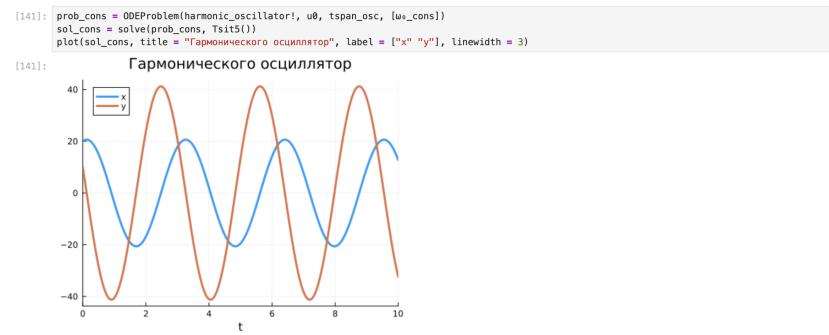


Рис. 3.31: График №7

```
[142]: p_cons_time = plot(layout=(2,2), size=(1000, 800))
p_cons_phase = plot(size=(800, 600), legend=:bottomleft)

for (idx, u0) in enumerate(initial_conditions_osc)
    prob_cons = ODEProblem(harmonic_oscillator!, u0, tspan_osc, [ω_cons])
    sol_cons = solve(prob_cons, Tsit5())

    t_points = 0:0.01:10 # Временные ряды (положение и скорость)
    x_sol = [sol_cons(t)[1] for t in t_points]
    y_sol = [sol_cons(t)[2] for t in t_points]

    plot!(p_cons_time[idx], t_points, x_sol, label="x(t)", linewidth=2, color=:blue)
    plot!(p_cons_time[idx], t_points, y_sol, label="y(t) = dot(x)(t)", linewidth=2, color=:red)
    plot!(p_cons_time[idx], title="Нач. усл. ($u_0[1], $u_0[2])", xlabel="Время", ylabel="Значение", grid=true, legend=:right)

    # фазовый портрет
    plot!(p_cons_phase, x_sol, y_sol, label="Траектория $idx", linewidth=2)
    scatter!(p_cons_phase, [u0[1]], [u0[2]], markersize=4, marker=:square, label="")
end

plot!(p_cons_phase, title="Фазовый портрет консервативного осциллятора",
      xlabel="x", ylabel="dot(x)", grid=true)
```

Рис. 3.32: График №7

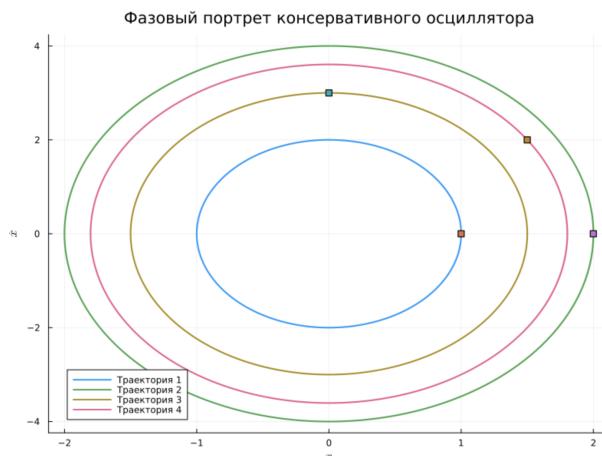


Рис. 3.33: График №7

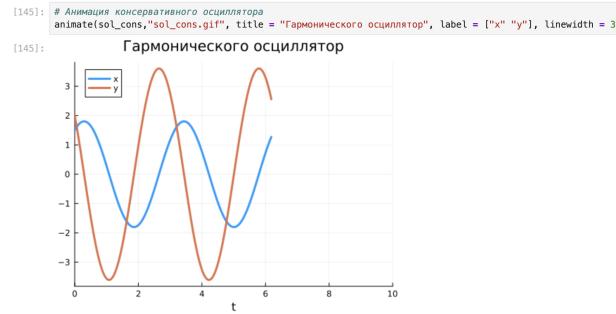


Рис. 3.34: Анимация №7

3.8 Задание 8

Реализовать на языке Julia модель свободных колебаний гармонического осциллятора:

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

где:

- ω_0 – циклическая частота
- γ – параметр, характеризующий потери энергии
- x_0 – начальное положение
- y_0 – начальная скорость

Начальные данные и параметры зададим самостоятельно (рис. 3.35 - 3.38). Как параметры для графика были выбраны слабое затухание, среднее, критическое и сильное.

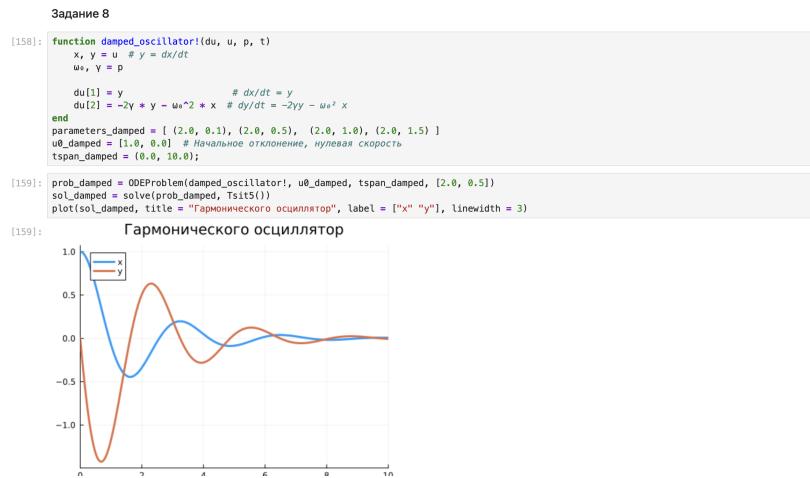


Рис. 3.35: Задание №8

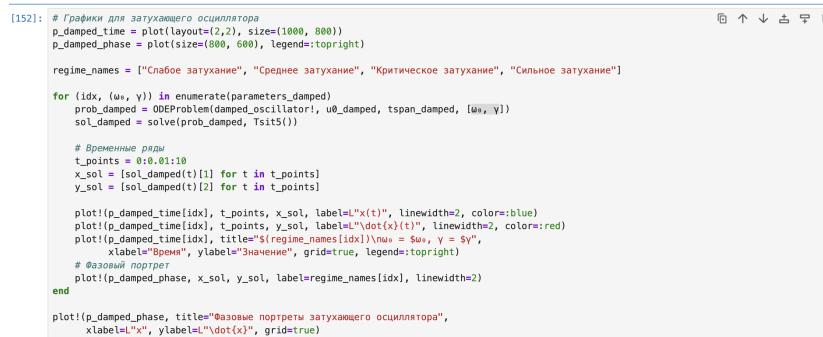


Рис. 3.36: График №8

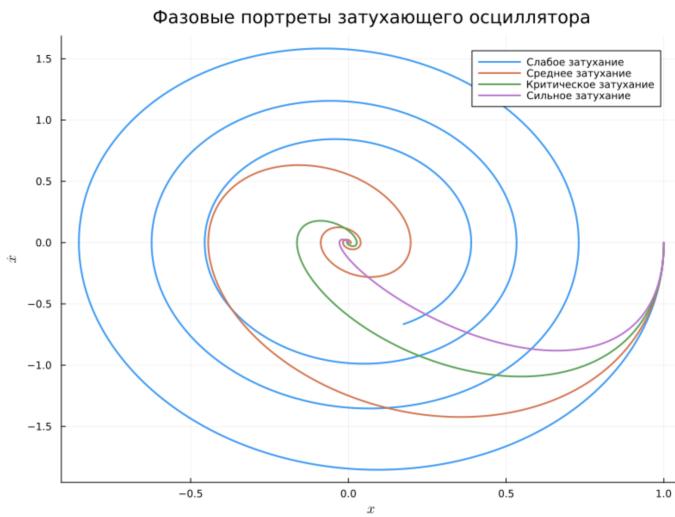


Рис. 3.37: График №8

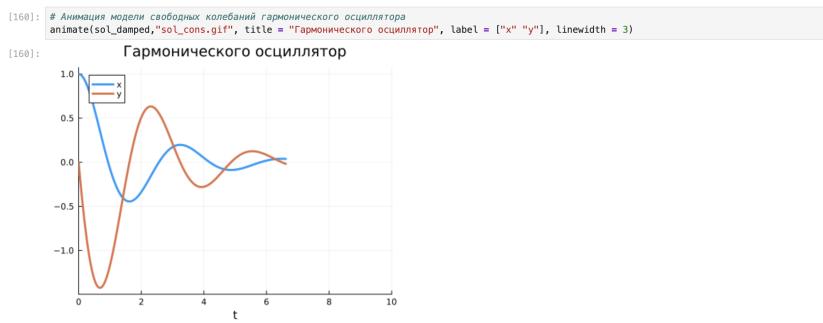


Рис. 3.38: Анимация №8

4 Выводы

В результате выполнения данной лабораторной работы я освоила специализированные пакеты для решения задач в непрерывном и дискретном времени.