Отчёт по лабораторной работе №6

Решение моделей в непрерывном и дискретном времени

Ким Реачна

Содержание

Список иллюстраций

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

2.1.1 Модель экспоненциального роста

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

```
    Updating registry at `C:\Users\Reachna\.julia\registries\General.toml`
   Resolving package versions...
   Installed OffsetArrays ──────────────────── v1.12.10
   Installed EnumX ─────────────────────────── v1.0.4
   Installed Polyester ─────────────────────── v0.7.9
   Installed Sundials_jll ──────────────────── v5.2.1+0
   Installed Accessors ─────────────────────── v0.1.33
   Installed NonlinearSolve ────────────────── v2.8.2
   Installed DifferentialEquations ─────────── v7.11.0
   Installed RecursiveArrayTools ───────────── v2.38.10
   Installed CEnum ─────────────────────────── v0.5.0
   Installed FunctionWrappers ──────────────── v1.1.3
   Installed TriangularSolve ───────────────── v0.1.20
   Installed IntelOpenMP_jll ───────────────── v2024.0.0+0
   Installed Static ────────────────────────── v0.8.8
   Installed Distances ─────────────────────── v0.10.11
   Installed BoundaryValueDiffEq ───────────── v5.5.0
```

Рис. 2.1: Установка пакетов DifferentialEquations

```
[2]: 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)
```

```
[2]: retcode: Success
     Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
     t: 5-element Vector{Float64}:
      0.0
      0.10042494449239292
      0.35218603951893646
      0.6934436334555072
      1.0
     u: 5-element Vector{Float64}:
      1.0
      1.1034222047865465
      1.4121908848175448
      1.9730384867968267
      2.664456142481423
```

Рис. 2.2: Численное решение модель экспоненциального роста

```
[3]: # подключаем необходимые пакеты:
     using Plots
     # 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")
```

[3]:

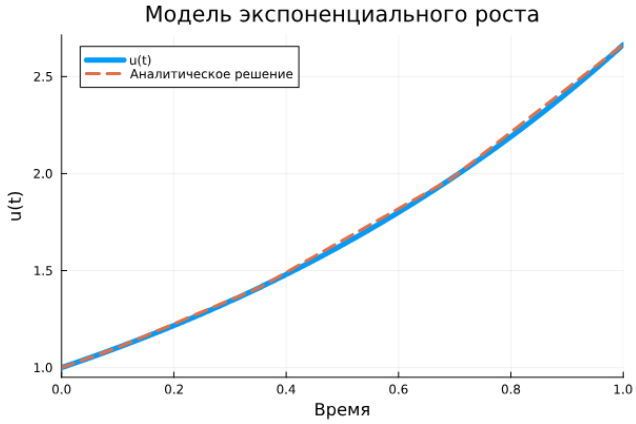

Рис. 2.3: График модель экспоненциального роста

[4]:
```
# задаём точность решения:
sol = solve(prob,abstol=1e-8,reltol=1e-8)
println(sol)
```

```
ODESolution{Float64, 1, Vector{Float64}, Nothing, Nothing, Vector{Float64}, Vector{Vector{Float64}}, ODEProblem{Float6
4, Tuple{Float64, Float64}, false, SciMLBase.NullParameters, ODEFunction{false, SciMLBase.AutoSpecialize, typeof(f), L
inearAlgebra.UniformScaling{Bool}, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, No
thing, Nothing, Nothing, Nothing, Nothing, typeof(SciMLBase.DEFAULT_OBSERVED), Nothing, Nothing}, Base.Pairs{Symbol, U
nion{}, Tuple{}, NamedTuple{(), Tuple{}}}, SciMLBase.StandardODEProblem}, CompositeAlgorithm{Tuple{Vern7{typeof(Ordina
ryDiffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limiter!), Static.False}, Rodas5P{0, false, Nothing, typeof
(OrdinaryDiffEq.DEFAULT_PRECS), Val{:forward}, true, nothing}}, OrdinaryDiffEq.AutoSwitchCache{Vern7{typeof(OrdinaryDi
ffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limiter!), Static.False}, Rodas5P{0, false, Nothing, typeof(Ordi
naryDiffEq.DEFAULT_PRECS), Val{:forward}, true, nothing}, Rational{Int64}, Int64}}, OrdinaryDiffEq.CompositeInterpolat
ionData{ODEFunction{false, SciMLBase.AutoSpecialize, typeof(f), LinearAlgebra.UniformScaling{Bool}, Nothing, Nothing,
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, typeof(Sci
MLBase.DEFAULT_OBSERVED), Nothing, Nothing}, Vector{Float64}, Vector{Float64}, Vector{Vector{Float64}}, OrdinaryDiffE
q.CompositeCache{Tuple{OrdinaryDiffEq.Vern7ConstantCache, OrdinaryDiffEq.Rosenbrock5ConstantCache{SciMLBase.TimeDeriva
tiveWrapper{false, ODEFunction{false, SciMLBase.AutoSpecialize, typeof(f), LinearAlgebra.UniformScaling{Bool}, Nothin
g, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothin
g, typeof(SciMLBase.DEFAULT_OBSERVED), Nothing, Nothing}, Float64, SciMLBase.NullParameters}, SciMLBase.UDerivativeWra
pper{false, ODEFunction{false, SciMLBase.AutoSpecialize, typeof(f), LinearAlgebra.UniformScaling{Bool}, Nothing, Nothi
ng, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, typeof
(SciMLBase.DEFAULT_OBSERVED), Nothing, Nothing}, Float64, SciMLBase.NullParameters}, OrdinaryDiffEq.Rodas5Tableau{Floa
t64, Float64}, Float64, OrdinaryDiffEq.StaticWOperator{true, Float64}, Nothing}}, OrdinaryDiffEq.AutoSwitchCache{Vern7
{typeof(OrdinaryDiffEq.trivial_limiter!), typeof(OrdinaryDiffEq.trivial_limiter!), Static.False}, Rodas5P{0, false, No
thing, typeof(OrdinaryDiffEq.DEFAULT_PRECS), Val{:forward}, true, nothing}, Rational{Int64}, Int64}}}, SciMLBase.DESta
ts, Vector{Int64}}([1.0, 1.0412786454705882, 1.1547261252949712, 1.3239095703537043, 1.5363819257509728, 1.82148946874
85153, 2.187139584776572 7, 2.6627635969237584, 2.6644562419335167], nothing, nothing, [0.0, 0.04127492324135852, 0.146
79917846877366, 0.28631546412766684, 0.4381941361169628, 0.6118924038903762, 0.7985658820706335, 0.9993515608907323,
1.0], [[1.0], [0.98, 0.9801982021814051, 0.9843259088450718, 0.9864960397068047, 0.9982235849699092, 1.004461525348705
3, 1.0156761793421414, 1.0173619945010424, 1.020453062048735, 1.0204531254788298], [1.0204530725611765, 1.020980717058
5115, 1.0320086945830 99, 1.0378358482529488, 1.0696715858355577, 1.0868444672206994, 1.118137332800357, 1.122887966170
9432, 1.1316311317327943, 1.1316339737658136], [1.1316316027890718, 1.132405219875053, 1.1486046772375742, 1.157187375
1018655, 1.204348922697842, 1.2299784341869338, 1.2770150778489493, 1.2841928268697484, 1.2974297784138256, 1.29743943
58656947], [1.2974313789466 3, 1.2983969345053052, 1.3186294324241983, 1.3293593653719393, 1.3884437156056277, 1.420639
3861163815, 1.4798796872175815, 1.4889366329148255, 1.505651708028238, 1.5056672716050201], [1.5056542872359533, 1.506
9357819895899, 1.5338210410621769, 1.548103795775711, 1.6270435296782662, 1.670264148528 6929, 1.7501540791016998, 1.76
240827248546 06, 1.7850545525357413, 1.785085493205899], [1.785059679373545, 1.7866924735391743, 1.820972474196545 6, 1.
8392024332742414, 1.9401801766356095, 1.9956239092433081, 2.098385357571685, 2.114178492697315, 2.143388680099189, 2.1
4343764694735 93], [2.1433967930810414, 2.1455055736434314, 2.18981333874682, 2.213402464090262, 2.344378500211417, 2.4
165155526067466, 2.5506111609309468, 2.5712634795852125, 2.6094952795355777, 2.6095740262028784], [2.609508324985283,
```

Рис. 2.4: Точность решения

[5]:
```
# 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")
```

[5]:

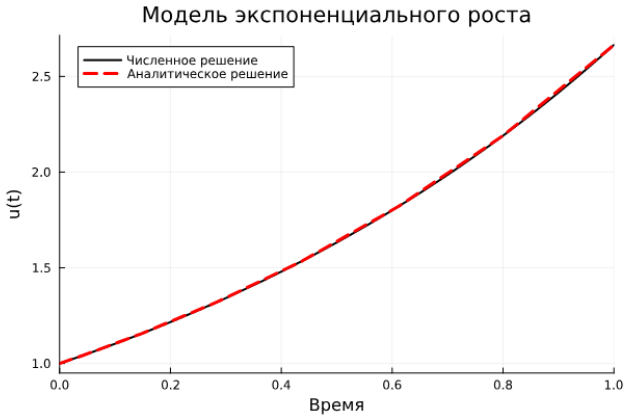

Рис. 2.5: Модель экспоненциального роста

2.1.2 Система Лоренца

[6]:
```
# подключаем необходимые пакеты:
import Pkg
Pkg.add("DifferentialEquations")
```

```
  Resolving package versions...
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
```

[7]:
```
using DifferentialEquations, Plots;
# 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 значнения параметров:
p = (10,28,8/3)
# задаём интервал времени:
tspan = (0.0,100.0)
# решение:
prob = ODEProblem(lorenz!,u0,tspan,p)
sol = solve(prob)
```

[7]:
```
retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 1263-element Vector{Float64}:
   0.0
   3.5678604836301404e-5
   0.0003924646531993154
   0.0032624077544510573
   0.009058075635317072
   0.01695646895607931
   0.02768995855685593
   0.04185635042021763
   0.06024041165841079
   0.08368541255159562
   0.11336499649094857
   0.1486218182609657
   0.18703978481550704
   ⋮
  99.05535949898116
  99.14118781914485
  99.22588252940076
  99.30760258626904
  99.39665422328268
  99.49536147459878
```

Рис. 2.6: Численное решение динамической системой Лоренца

```
[8]: # подключаем необходимые пакеты:
using Plots
# строим график:
plot(sol, vars=(1,2,3), lw=2, title="Аттрактор Лоренца", xaxis="x",yaxis="y", zaxis="z",legend=false)
```

```
┌ Warning: To maintain consistency with solution indexing, keyword argument vars will be removed in a future version.
Please use keyword argument idxs instead.
│   caller = ip:0x0
└ @ Core :-1
```

[8]:

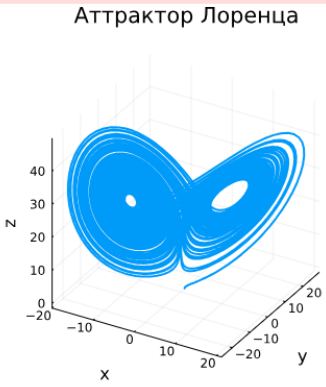

Рис. 2.7: Аттрактор Лоренца

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

[9]:

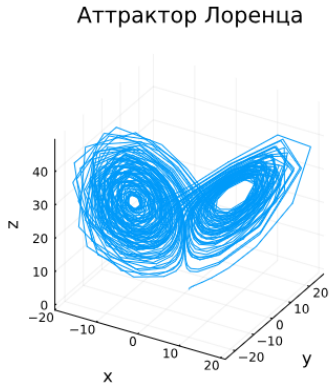

Рис. 2.8: Аттрактор Лоренца (интерполяция отключена)

2.2 Модель Лотки–Вольтерры

```
[10]: # подключаем необходимые пакеты:
      import Pkg
      Pkg.add("ParameterizedFunctions")
```

```
  Resolving package versions...
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
```

```
[11]: 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 значчения 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)
```

```
[11]: retcode: Success
      Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
      t: 34-element Vector{Float64}:
        0.0
        0.0776084743154256
        0.23264513699277584
        0.4291185174543143
        0.6790821987497083
        0.9444046158046306
        1.2674601546021105
        1.6192913303893046
        1.9869754428624007
        2.2640902393538296
        2.5125484290863063
        2.7468280298123062
        3.038006585197414 7
        ⋮
        6.455762090996754
        6.780496138817711
        7.171040059920871
        7.584863345264154
        7.978068981329682
        8.48316543760351
        8.719248247740158
        8.949206788834692
        9.200185054623292
        9.438029017301554
        9.711808134779586
       10.0
```

Рис. 2.9: Численное решение модель Лотки–Вольтерры

```
[12]: plot(sol, label = ["Жертвы" "Хищники"], color="black", ls=[:solid :dash], title="Модель Лотки - Вольтерры",
          xaxis="Время",yaxis="Размер популяции")
```

[12]:

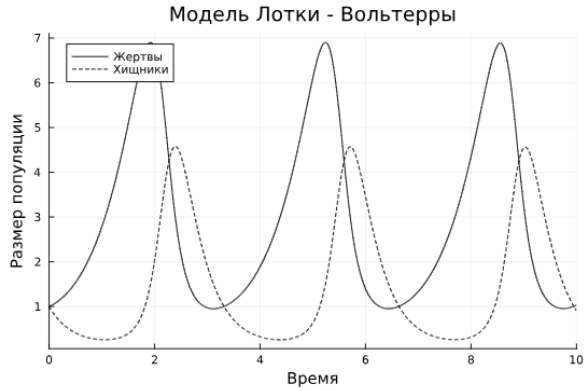

Рис. 2.10: Модель Лотки–Вольтерры: динамика изменения численности популяций

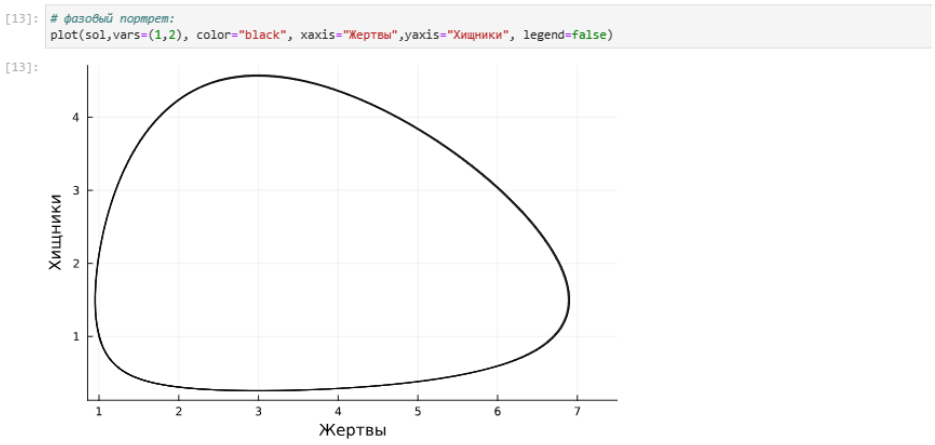

Рис. 2.11: Модель Лотки–Вольтерры: фазовый портрет

2.3 Задания для самостоятельного выполнения

1. Реализовать и проанализировать модель роста численности изолированной популяции (модель Мальтуса) : $\dot{x} = ax, a = b - c$

Использовала следующие коэффициенты: $a = b - c = 2.0, b = 3.0, c = 1.0$ и интервал от 0 до 3. Я решила взять довольно большой коэффициент рождаемости и роста популяции тем самым ожидая на графике очень быстрый рост населения, что в принципе я и получила. Так как за 3 единицы времени размер изолированной популяции с 2 увеличилось до 800 по экспоненте.

```
[14]: # Task1
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаем описание модели:
lv! = @ode_def Malthus begin
    dx = a*x
end a
# задаём начальное условие:
u0 = [2]
# задаём знанчения параметров:
b = 3.0
c = 1.0
p = (b-c)
# задаём интервал времени:
tspan = (0.0,3.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
```

```
[14]: retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 12-element Vector{Float64}:
 0.0
 0.07579340539309044
 0.2176538131796436
 0.39326275375009306
 0.6100444793398203
 0.8636787203353302
 1.1544101119687582
 1.4789340537388638
 1.8349001265017795
 2.219134461733416
 2.628731787861167
 3.0
u: 12-element Vector{Vector{Float64}}:
 [2.0]
 [2.3273586349901 42]
 [3.0908767890047213]
 [4.391507855871063]
 [6.7749764415491 92]
 [11.251525438518586]
 [20.12503871207196]
 [38.513500897099114]
 [78.48706775956025]
 [169.25231460681178]
 [383.9709586782193]
 [806.8145670268354]
```

Рис. 2.12: Решение модели Мальтуса

```
plot(sol, label = "Численность изолированной популяции x(t)", color="green", ls=[:solid], title="Модель Мальтуса",
    xaxis="Время",yaxis="Размер изолированной популяции")
```

Рис. 2.13: График модели Мальтуса

```
[16]: animate(sol, fps=7, "Malthus.gif", label = "Численность изолированной популяции x(t)",
    color="green", ls=[:solid], title="Модель Мальтуса", xaxis="Время", yaxis="Размер изолированной популяции")
```

Рис. 2.14: Построение анимации

Рис. 2.15: Анимация модели Мальтузы

2. Реализовать и проанализировать логистическую модель роста популяции, заданную уравнением: $\dot{x} = xr(1 - \frac{x}{k}), r > 0, k > 0$

Задала коэффициенты следующие $r = 0.9, k = 20$. Таким образом коэффициент роста равен 0.9, а предельное значение численности популяции равно 20, поэтому график на интервале от 0 до 10 должен достичь по оси у значение 20 и выше не под- ниматься.

[17]:
```
# Task2
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаем описание модели:
lv! = @ode_def Logistic_population begin
    dx = r*x*(1-x/k)
end r k
# задаём начальное условие:
u0 = [1.0]
# задаём значения параметров:
p = (0.9, 20)
# задаём интервал времени:
tspan = (0.0,10.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
```

[17]:
```
retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 14-element Vector{Float64}:
  0.0
  0.10320330193850687
  0.3855506045099877
  0.780748965506008
  1.262015691559725
  1.8586159628422565
  2.574933530608521
  3.471498551774082
  4.571529609523612
  5.629314234929612
  6.930091225213617
  8.078262639019435
  9.531767314565881
 10.0
u: 14-element Vector{Vector{Float64}}:
 [1.0]
 [1.092018818522065]
 [1.3860627615966585]
 [1.9212436077635002]
 [2.8160884736520035]
 [4.379382291381814]
 [6.9638339217858904]
 [10.897151531483962]
 [15.262798385676023]
 [17.860400164058895]
 [19.283004596951911]
 [19.73872139608262]
 [19.928358494866384]
 [19.95293645513508]
```

Рис. 2.16: Решение логистическую модель роста популяции

[18]:
```
plot(sol, label = "Численность популяции x(t)", color="blue", ls=[:solid], title="Логистическая модель роста популяции",
    xaxis="Время",yaxis="Размер популяции")
```

[18]:

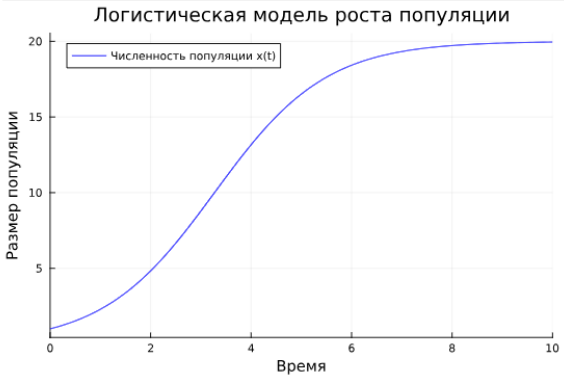

Рис. 2.17: График логистическую модель роста популяции

[19]:
```
animate(sol, fps=7, "Logistic_population.gif", label = "Численность популяции x(t)", color="blue",
    ls=[:solid], title="Логистическая модель роста популяции", xaxis="Время", yaxis="Размер популяции")
```

Рис. 2.18: Построение анимации

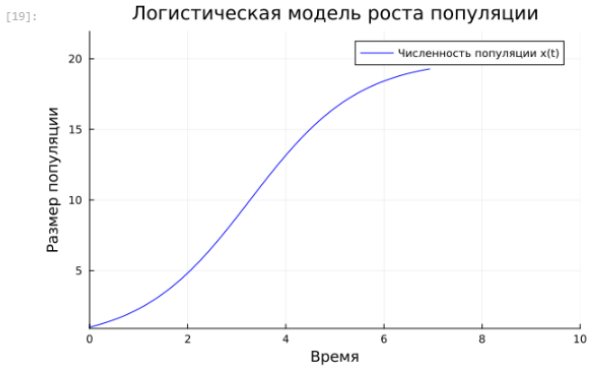

Рис. 2.19: Анимация логистическую модель роста популяции

3. Реализовать и проанализировать модель эпидемии Кермака–Маккендрика (SIR-модель):

$$\begin{cases} \dot{s} = -\beta i s, \\ \dot{i} = \beta i s - v i, \\ \dot{r} = v i \end{cases} \tag{2.1}$$

Задала $\beta = 0.25, v = 0.05$

```julia
# задаём описание модели:
lv! = @ode_def SIR begin
ds = - b*i*s
di = b*i*s - v*i
dr = v*i
end b v

# задаём начальное условие:
u0 = [1.0, 0.1, 0]
# задаём знанчения параметров:
p = (0.25, 0.05)
# задаём интервал времени:
tspan = (0.0, 100.0)

# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
```

```
retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 19-element Vector{Float64}:
   0.0
   0.08088145925786733
   0.674649456103469
   1.9774507638268786
   3.928608933045557
   6.371598738903415
   9.52414865378298
  13.099293783864182
  17.0272982736033
  22.927215420937856
  27.195723313986843
  33.36650873512655
  39.87152643660008
  49.090534040944405
  57.69126316873367
  69.09753551513025
  81.37728197451536
  95.06634205664659
 100.0
u: 19-element Vector{Vector{Float64}}:
 [1.0, 0.1, 0.0]
 [0.9979636107059043, 0.10162869618330198, 0.0004076931107937434]
 [0.9821139347502068, 0.11427647441254161, 0.0036095908372516l9]
 [0.9414409947662143, 0.1464902846000639, 0.012068720633721717]
 [0.8642596086672918, 0.2065639776528575, 0.029176413679850716]
 [0.74121657128916, 0.29889094291007307, 0.05989248580076701]
 [0.5567300467580422, 0.42613510184975667, 0.11713485139220119]
 [0.360654706841008, 0.5353792639828872, 0.2039660291761048l]
 [0.20739657233924386, 0.5779798290842139, 0.3146235985765423]
 [0.09060823642513902, 0.5292223125052752, 0.4801694510695858]
 [0.05338199009370889, 0.46063944558344505, 0.5859785643228461
 [0.028391022566481953, 0.35937261970727175, 0.7122363577262463]
```

Рис. 2.20: Решение SIR модель

```julia
plot(sol, label = ["Восприимчивые" "Инфицированные" "Переболевшие"], color=["blue" "green" "red"], ls=[:solid :dash :dot],
    title="Модель эпидемии Кермака-Маккендрика SIR", xaxis="Время",yaxis="Размер популяции")
```

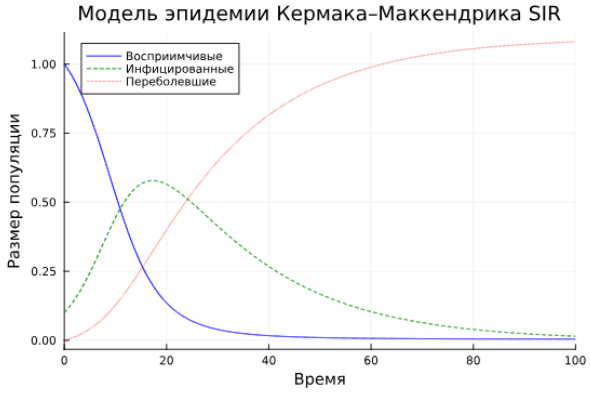

Рис. 2.21: График SIR модель

Видно, что число инфицированных растет намного медленнее, а также в целом

меньшее число людей было инфицировано по истечению времени, нежели в первом случае. Коэффициент интенсивности $R_0 = \frac{\beta}{v} = 3$.

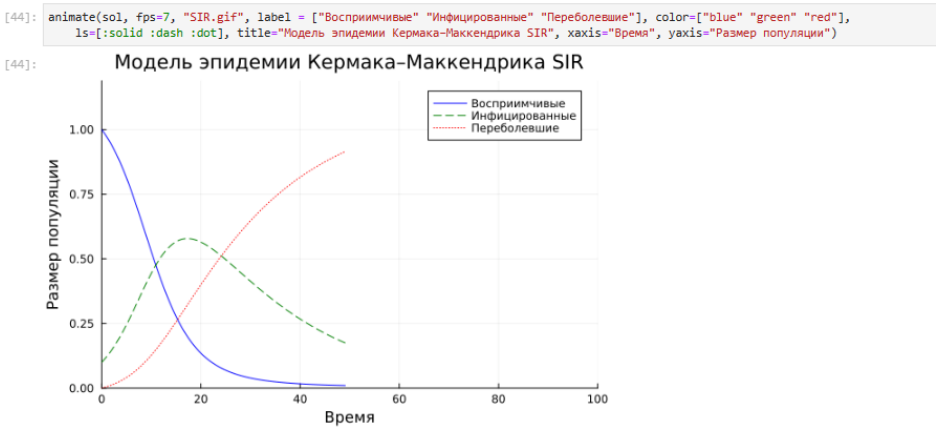

Рис. 2.22: Анимация SIR модель

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

$$\begin{cases} \dot{s}(t) = -\frac{\beta}{N} s(t) i(t), \\ \dot{e}(t) = \frac{\beta}{N} s(t) i(t) - \delta e(t), \\ \dot{i}(t) = \delta e(t) - \gamma i(t), \\ \dot{r}(t) = \gamma i(t) \end{cases} \tag{2.2}$$

Задала $\beta = 0.6, \gamma = 0.2, \delta = 0.1, N = 1.0$:

[23]:
```
# Task4
using ParameterizedFunctions, DifferentialEquations, Plots;
N = 1.0
# задаём описание модели:
lv! = @ode_def SEIR begin
    ds = -(β/N)*s*i
    de = (β/N)*s*i - δ*e
    di = δ*e - γ*i
    dr = γ*i
end β γ δ

initialInfect = 0.1
# задаём начальное условие:
u0 = [(N - initialInfect), 0.0, initialInfect, 0.0]
# задаём значнения параметров:
p = (0.6, 0.2, 0.1)
# задаём интервал времени:
tspan = (0.0, 100.0)

# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
```

[23]:
```
retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 25-element Vector{Float64}:
   0.0
   0.024423707511123237
   0.21983937298994613
   0.6724461105829 35
   1.3433111385495904
   2.2048531822239386
   3.3191976283637796
   4.685982405908369
   6.3524541259891905
   8.357303010682124
  10.779894718728759
  13.70832499771059
  17.247755058759296
  21.418982739513048
  26.11587299819979
  31.347120286227476
  37.434253818037966
  45.57027277276479
  51.92822542671895
  59.56906630189384
  67.00367237385267
  75.35077774263594
  84.11521635193901
  93.80030928820192
 100.0
u: 25-element Vector{Vector{Float64}}:
 [0.9, 0.0, 0.1, 0.0]
```

Рис. 2.23: Решение SEIR-модель

[24]:
```
plot(sol, label = ["Восприимчивые" "Контактные" "Инфицированные" "Переболевшие"], color=["blue" "black" "green" "red"],
    ls=[:solid :dash :dot :dashdot], title="Модель SEIR", xaxis="Время",yaxis="Размер популяции")
```

[24]:

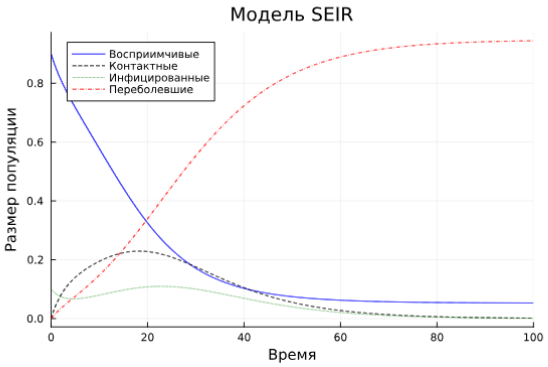

Рис. 2.24: График SEIR-модель

```
[25]: animate(sol, fps=7, "SIR.gif", label = ["Восприимчивые" "Контактные" "Инфицированные" "Переболевшие"],
          color=["blue" "black" "green" "red"], ls=[:solid :dash :dot :dashdot], title="Модель SEIR",
          xaxis="Время", yaxis="Размер популяции")
```

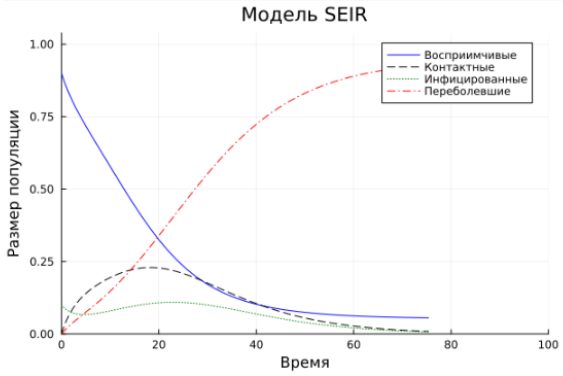

Рис. 2.25: Анимация SEIR-модель

5. Для дискретной модели Лотки–Вольтерры:

$$\begin{cases} X_1(t+1) = aX_1(t)(1 - X_1(t)) - X_1(t)X_2(t), \\ X_2(t+1) = -cX_2(t) + dX_1(t)X_2(t) \end{cases} \tag{2.3}$$

Задала $a = 2, c = 1, d = 5$:

```
[26]: import Pkg
      Pkg.add("LaTeXStrings")
```

```
  Resolving package versions...
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
```

```
[27]: # Task5
using DifferentialEquations, Plots, ParameterizedFunctions, LaTeXStrings

# задаём знанчения параметров:
a, c, d = 2, 1, 5

# задаем функцию для дискретной модели
next(x1, x2) = [(a*x1*(1 - x1) - x1*x2), (-c*x2 + d*x1*x2)]

# рассчитываем точку равновесия
balancePoint = [(1 + c)/d ,(d*(a - 1)-a*(1 + c))/d]

# задаём начальное условие:
u0 = [0.8, 0.05]
modelingTime = 100

simTrajectory = Array{Union{Nothing, Array}}(nothing, modelingTime)

for t in 1:modelingTime
    simTrajectory[t] = []
    if(t == 1)
        simTrajectory[t] = u0
    else
        simTrajectory[t] = next(simTrajectory[t-1]...)
    end
end
```

Рис. 2.26: Решение модели Лотки–Вольтерры

```
[28]: scatter([simTrajectory[1][1]], [simTrajectory[1][2]],
          c=:red, ms=9, label="Начальное состояние")

      plot!(first.(simTrajectory), last.(simTrajectory), color=:green, linestyle=:dash,
          marker = (:dot, 5, Plots.stroke(0)), label="Траектория модели", title = "Дискретная модель Лотки-Вольтерры")

      scatter!([balancePoint[1]], [balancePoint[2]], color=:orange, markersize=5,
          label="Точка равновесия",
          xlabel="Жертвы", ylabel="Хищники")
```

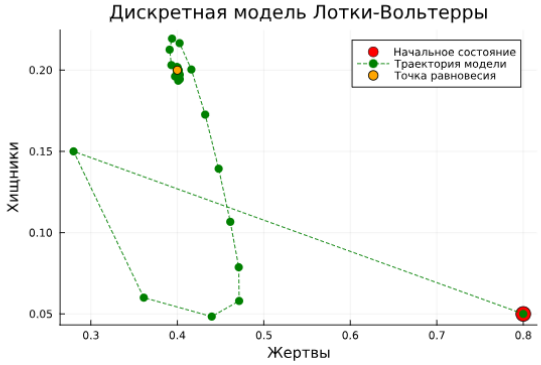

Рис. 2.27: График модели Лотки–Вольтерры

```
[29]: n = 100
      anim = @animate for i in 1:n
          modelingTime = (i)
          # задаём значения параметров:
          a, c, d = 2, 1, 5
          # задаём начальное условие:
          u0 = [0.8, 0.05]

          simTrajectory = Array{Union{Nothing, Array}}(nothing, modelingTime)

          for t in 1:modelingTime
              simTrajectory[t] = []
              if(t == 1)
                  simTrajectory[t] = u0
              else
                  simTrajectory[t] = next(simTrajectory[t-1]...)
              end
          end

          scatter([simTrajectory[1][1]], [simTrajectory[1][2]],
          c=:red, ms=9, label="Начальное состояние")

          plot!(first.(simTrajectory), last.(simTrajectory), color=:green, linestyle=:dash,
          marker = (:dot, 5, Plots.stroke(0)), label="Траектория модели",
          title = "Дискретная модель Лотки-Вольтерры", xlabel="Жертвы", ylabel="Хищники")

          scatter!([balancePoint[1]], [balancePoint[2]], color=:orange, markersize=5,
          label="Точка равновесия",
          xlabel="Жертвы", ylabel="Хищники")
      end
      gif(anim, "LotkaVolterra.gif", fps=7)

[ Info: Saved animation to C:\Users\Reachna\computer-analysis\LotkaVolterra.gif
```

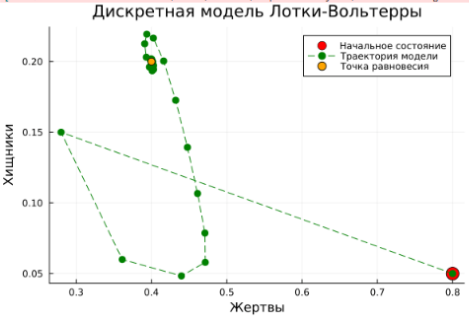

Рис. 2.28: Анимация модели Лотки–Вольтерры

6. Реализовать на языке Julia модель отбора на основе конкурентных отноше-

ний:

$$\begin{cases} \dot{x} = \alpha x - \beta xy, \\ \dot{y} = \alpha y - \beta xy \end{cases} \tag{2.4}$$

```
[30]: # Task6
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаем описание модели:
lv! = @ode_def CompetitiveSelectionModel begin
    dx = a*x - b*x*y
    dy = a*y - b*x*y
end a b
# задаём начальное условие:
u0 = [1.0,1.4]
# задаём значчения параметроб:
p = (0.5, 0.2)
# задаём интервал времени:
tspan = (0.0,10.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
```

```
[30]: retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 20-element Vector{Float64}:
  0.0
  0.13063515958673816
  0.6620095919016169
  1.4745519498111759
  2.3842743763294555
  3.4538271544345127
  4.5626739618521099
  5.678394757457 4588
  6.618742680948912
  7.174117808180357
  7.581652232104449
  7.907287404662807
  8.211743148258813
  8.476131118651619
  8.723933719403542
  8.961493008998673
  9.203605421462836
  9.457562739292689
  9.735425791326456
 10.0
u: 20-element Vector{Vector{Float64}}:
 [1.0, 1.4]
 [1.028414415994334, 1.4554136105342876]
 [1.1349867943706582, 1.6919333806954628]
 [1.2538768556942972, 2.0899703101538307]
 [1.2907022905384151, 2.608347267277165]
 [1.1633365332504924, 3.412641939340822]
 [0.8307911711859107, 4.746685014866651]
 [0.3930025927897952, 7.233789081223332]
 [0.11653304820955845, 11.063651811087418]
 [0.03764043695860551, 14.488643868861088]
 [0.012491831797030182, 17.7296058455366 96]
 [0.004202666799768924, 20.85409224693543]
 [0.0012459925909265 48, 24.27939005309544]
 [0.0003618573640917 8275, 27.709728708980833]
 [9.560318903313416e-5, 31.364442854341014]
 [2.24450975432822 3e-5, 35.32009587775201]
 [4.313586083190942e-6, 39.86535971687741]
 [6.934576244296251e-7, 45.26283767286127]
```

Рис. 2.29: Решение модели роста популяции в условиях кокнкуренции

```
[31]: plot(sol, label = ["1-ый биологический вид" "2-ой биологический вид"], color=["blue" "red"], ls=[:solid :dash],
          title="Модель роста популяции в условиях конкуренции", xaxis="Время",yaxis="Размер популяции")
```

[31]:

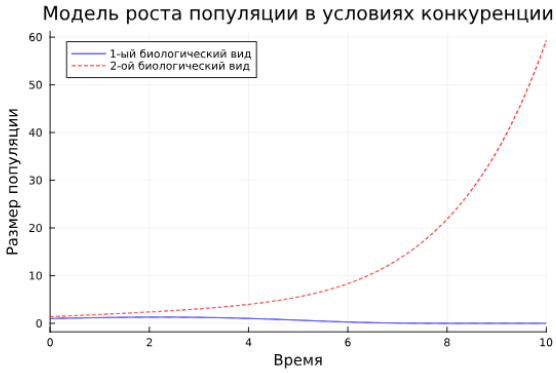

Рис. 2.30: График модели роста популяции в условиях коконкуренции

```
[32]: # фазовый портрет:
plot(sol,vars=(1,2), color="black", title="Фазовый портрет", xaxis="1-ый биологический вид",
     yaxis="2-ой биологический вид", legend=false)
```

[32]:

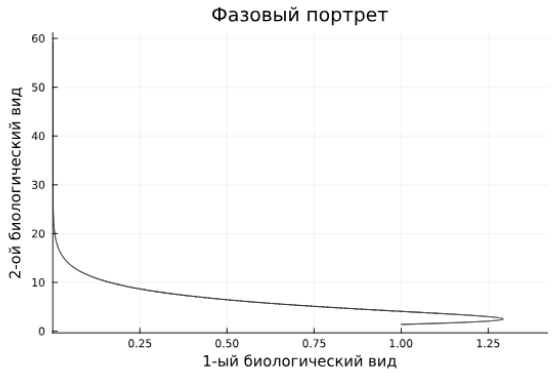

Рис. 2.31: Фазовый портрет

```
[33]: animate(sol, fps=7, "CompetitiveSelection.gif", label = ["1-ый биологический вид" "2-ой биологический вид"],
          color=["blue" "red"], ls=[:solid :dash], title="Модель роста популяции в условиях конкуренции",
          xaxis="Время", yaxis="Размер популяции")
```

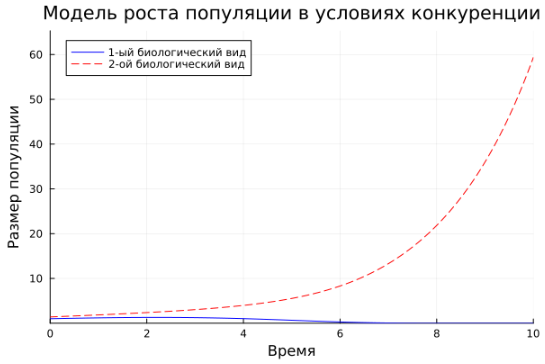

Рис. 2.32: Анимация модели роста популяции в условиях коконкуренции

7. Реализовать на языке Julia модель консервативного гармонического осциллятора: $\ddot{x} = \omega_0^2 x = 0, x(t_0) = x_0, \dot{x}(t_0) = y_0$

```
[34]: # задаём описание модели:
lv! = @ode_def classicOscillator begin
dx = y
dy = -(w0^2)*x
end w0

# задаём начальное условие:
u0 = [1.0, 1.0]
# задаём значения параметров:
p = (2.0)
# задаём интервал времени:
tspan = (0.0, 10.0)

# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
```

```
[34]: retcode: Success
Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
t: 31-element Vector{Float64}:
  0.0
  0.07580097943195412
  0.2069885812216689
  0.353096695575846 94
  0.5285194634228536
  0.7514914358697009
  1.0072081570278821
  1.2779920001493048
  1.5687719973957874
  1.9026764818913302
  2.22297371986793 34
  2.5850540372165933
  2.9526997573066778
  ⋮
  5.742944245724529
  6.171924002188556
  6.584586020436235
  7.010469902680138
  7.433067666627458
  7.849769901055738
  8.282275752485367
  8.684259757356292
  9.126171387660838
  9.52416872011131
  9.970147020711996
 10.0
u: 31-element Vector{Vector{Float64}}:
 [1.0, 1.0]
 [1.0640413705392677, 0.6864865930281919]
 [1.1166550813709244, 0.1110207837006 2098]
 [1.0853087396797187, -0.5370470078797774]
 [0.9269048870800588, -1.2503554234259173]
```

Рис. 2.33: Решение модель консервативного гармонического осциллятора

```
[35]: plot(sol, label = ["x(t)" "dx(t)"], color=["purple" "red"], ls=[:solid :dash],
          title="Модель консервативного гармонического осциллятора",
          xaxis="Время",yaxis="Отклик")
```

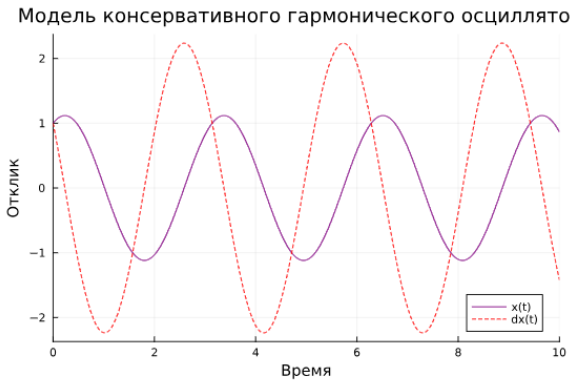

Рис. 2.34: График модели консервативного гармонического осциллятора

```
[36]: # фазовый портрет:
      plot(sol, vars=(1,2), color="black", title="Фазовый портрет", xaxis="x(t)", yaxis="dx(t)", legend=false)
```

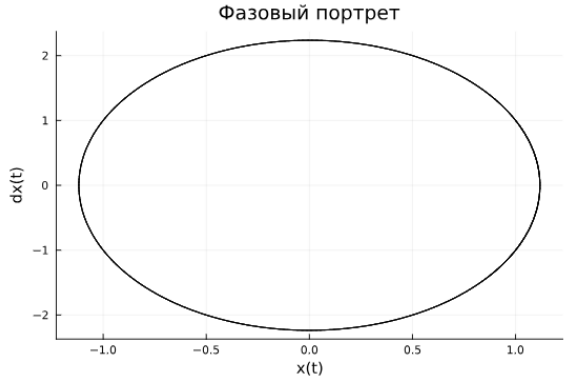

Рис. 2.35: Фазовый портрет

```
[37]: animate(sol, fps=7, "harmonic_oscillator_model.gif", label = ["x(t)" "dx(t)"], color=["red" "green"], ls=[:solid :dash],
          title="Модель консервативного гармонического осциллятора", xaxis="Время",yaxis="Отклик")
```

[37]:

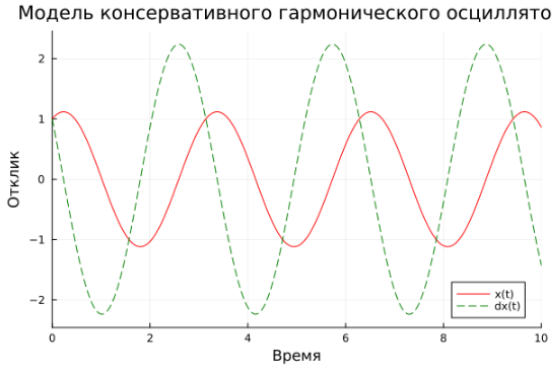

Рис. 2.36: Анимация модели консервативного гармонического осциллятора

8. Реализовать на языке Julia модель свободных колебаний гармонического осциллятора: $\ddot{x} + 2\gamma\dot{x} + \omega_0^2 x = 0, x(t_0) = x_0, \dot{x}(t_0) = y_0$

```
[38]: # Task8
      # задаём описание модели:
      lv! = @ode_def Oscillator begin
      dx = y
      dy = -2*v*y - (w0^2)*x
      end v w0

      # задаём начальное условие:
      u0 = [0.5, 1.0]
      # задаём значения параметров:
      p = (0.5, 2.0)
      # задаём интервал времени:
      tspan = (0.0, 10.0)

      # решение:
      prob = ODEProblem(lv!,u0,tspan,p)
      sol = solve(prob)
```

```
[38]: retcode: Success
      Interpolation: specialized 4th order "free" interpolation, specialized 2nd order "free" stiffness-aware interpolation
      t: 31-element Vector{Float64}:
        0.0
        0.07472295275624102
        0.2111474073618621
        0.370998115535819
        0.5569932690427111
        0.7935651279847608
        1.0592494385311997
        1.3434677044587358
        1.63455900887044463
        1.96957712789461115
        2.298725814870145
        2.66376914252198O7
        3.0363220157596453
        ⋮
        5.737115484628017
        6.172863902331125
        6.563858473632751
        6.9841201699413915
        7.37336534430487
        7.815553300011508
        8.212080450073843
        8.639453151000238
        9.029651151704007
        9.479283944498775
        9.87771453843803
       10.0
      u: 31-element Vector{Vector{Float64}}:
       [0.5, 1.0]
       [0.566294838471426, 0.7739308462880993]
```

Рис. 2.37: Решение модели свободных колебаний гармонического осциллятора

```
[39]: plot(sol, label = ["x(t)" "dx(t)"], color=["blue" "green"], ls=[:solid :dash],
          title="Модель свободных колебаний гармонического осциллятора",
          xaxis="Время",yaxis="Отклик")
```

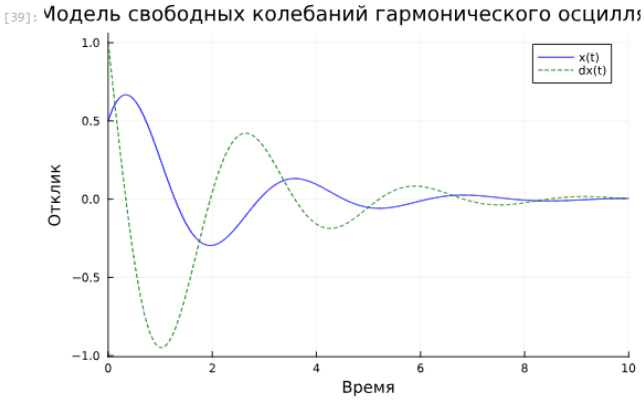

Рис. 2.38: График модели свободных колебаний гармонического осциллятора

```
[40]: # фазовый портрет:
      plot(sol, vars=(1,2), color="black", title="Фазовый портрет", xaxis="x(t)", yaxis="dx(t)", legend=false)
```

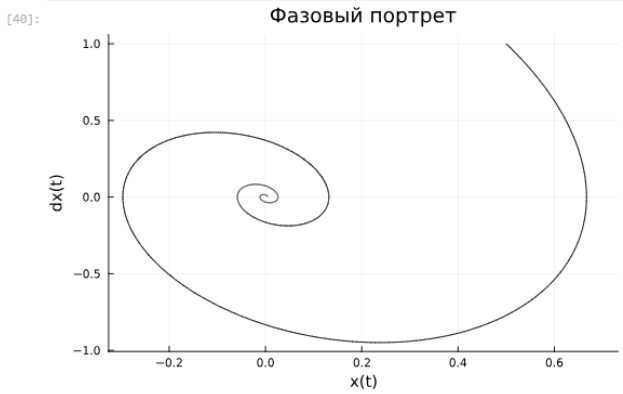

Рис. 2.39: Фазовый портрет

```
[41]: animate(sol, fps=7, "oscillator_model.gif", label = ["x(t)" "dx(t)"], color=["blue" "green"], ls=[:solid :dash],
          title="Модель свободных колебаний гармонического осциллятора",
          xaxis="Время",yaxis="Отклик")
```

[41]:

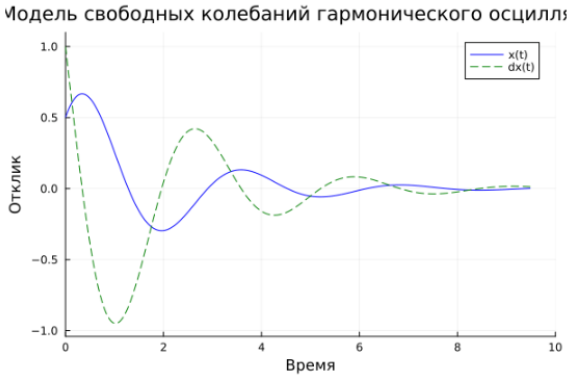

Рис. 2.40: Анимация модели свободных колебаний гармонического осциллятора

3 Листинги программы

```
# подключаем необходимые пакеты:
import Pkg
Pkg.add("DifferentialEquations")
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)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 5-element Vector{Float64}:
 0.0
 0.10042494449239292
 0.35218603951893646
```

```
 0.6934436334555072
 1.0
u: 5-element Vector{Float64}:
 1.0
 1.1034222047865465
 1.4121908848175448
 1.9730384867968267
 2.664456142481423

# подключаем необходимые пакеты:
using Plots
# 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")
# 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")
# подключаем необходимые пакеты:
import Pkg
Pkg.add("DifferentialEquations")
$\beta$
using DifferentialEquations, Plots;
```

```
# задаём описание модели:
function lorenz!(du,u,p,t)
    $\sigma$,$\rho$,$\beta$ = p
    du[1] = $\sigma$*(u[2]-u[1])
    du[2] = u[1]*($\rho$-u[3]) - u[2]
    du[3] = u[1]*u[2] - $\beta$*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)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 1263-element Vector{Float64}:
   0.0
   3.5678604836301404e-5
   0.0003924646531993154
   0.0032624077544510573
   0.009058075635317072
   0.01695646895607931
   0.02768995855685593
   0.04185635042021763
   0.06024041165841079
```

```
  0.08368541255159562
  0.11336499649094857
  0.1486218182609657
  0.18703978481550704
  .
 99.05535949898116
 99.14118781914485
 99.22588252940076
 99.30760258626904
 99.39665422328268
 99.49536147459878
 99.58822928767293
 99.68983993598462
 99.77864535713971
 99.85744078539504
 99.93773320913628
100.0
u: 1263-element Vector{Vector{Float64}}:
 [1.0, 0.0, 0.0]
 [0.9996434557625105, 0.0009988049817849058, 1.781434788799208e-8]
 [0.9961045497425811, 0.010965399721242457, 2.146955365838907e-6]
 [0.9693591634199452, 0.08977060667778931, 0.00014380183422669 37]
 [0.9242043615038835, 0.24228912482984957, 0.0010461623302512404]
 [0.8800455868998046, 0.43873645009348244, 0.0034242593451028745]
 [0.8483309847495312, 0.6915629321083602, 0.008487624590227805]
 [0.8495036669651213, 1.0145426355349096, 0.01821208962127994]
 [0.9139069574560097, 1.4425599806525806, 0.03669382197085303]
 [1.088863826836895, 2.052326595543049, 0.0740257368585531]
 [1.4608627354936607, 3.0206721193016133, 0.16003937020467585]
```

```
 [2.162723488309695, 4.633363843843712, 0.37711740539408584]
 [3.3684644104189387, 7.26769410983553, 0.936355641713984]
 .
 [12.265454131109882, 12.598146409807255, 31.546057337607913]
 [10.48677626670755, 6.494631680470132, 33.669742813875764]
 [6.893277189568002, 3.1027383340030155, 29.77818388970318]
 [4.669609096878053, 3.061564434452441, 25.14247350179 59]
 [4.188801916573263, 4.617474401440693, 21.09864175382292]
 [5.559603854699961, 7.905631612648314, 18.79323210016923]
 [8.556629716266505, 12.533041060088328, 20.6623639692711]
 [12.280585075547771, 14.505154761545633, 29.332088452699942]
 [11.736883151600804, 8.279294641640229, 34.68007510231878]
 [8.10973327066804, 3.2495066495235854, 31.97052076740117]
 [4.958629886040755, 2.194919965065022, 26.948439650907677]
 [3.8020065515435855, 2.787021797920187, 23.420567509786622]

# подключаем необходимые пакеты:
using Plots
# строим график:
plot(sol, vars=(1,2,3), lw=2, title="Аттрактор Лоренца",
xaxis="x",yaxis="y", zaxis="z",legend=false)
# отключаем интерполяцию:
plot(sol,vars=(1,2,3),denseplot=false, lw=1, title="Аттрактор Лоренца",
xaxis="x",yaxis="y", zaxis="z",legend=false)
# подключаем необходимые пакеты:
import Pkg
Pkg.add("ParameterizedFunctions")
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)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 34-element Vector{Float64}:
  0.0
  0.0776084743154256
  0.23264513699277584
  0.4291185174543143
  0.6790821987497083
  0.9444046158046306
  1.2674601546021105
  1.6192913303893046
  1.9869754428624007
  2.2640902393538296
  2.5125484290863063
  2.7468280298123062
```

```
  3.0380065851974147
  .
  6.455762090996754
  6.780496138817711
  7.171040059920871
  7.584863345264154
  7.978068981329682
  8.48316543760351
  8.719248247740158
  8.949206788834692
  9.200185054623292
  9.438029017301554
  9.711808134779586
 10.0
u: 34-element Vector{Vector{Float64}}:
 [1.0, 1.0]
 [1.0454942346944578, 0.8576684823217127]
 [1.1758715885138267, 0.639459570317544]
 [1.4196809607170826, 0.4569962601282084]
 [1.876719395008001, 0.32473342927911314]
 [2.5882500645533466, 0.26336255535952163]
 [3.8607089092207665, 0.2794458098285253]
 [5.750812667710396, 0.5220072537934558]
 [6.814978999130169, 1.9177826328390666]
 [4.3929992925714245, 4.194670792850584]
 [2.1008562663496626, 4.31694049248469]
 [1.2422757654297396, 3.1073646247560807]
 [0.9582720921023357, 1.7661433892230374]
  .
```

```
 [0.952206525526163, 1.4383448433913901]
 [1.1004623776276266, 0.7526620730760382]
 [1.5991134291557523, 0.3903181675223147]
 [2.614253967788294, 0.26416945387525886]
 [4.241076127191749, 0.3051236762921916]
 [6.791123785297795, 1.1345287797146113]
 [6.265370675764892, 2.74169350754023]
 [3.7807651118880545, 4.431165685863461]
 [1.816420140681761, 4.064056625315978]
 [1.1465021407690728, 2.7911706616216976]
 [0.9557986135403302, 1.6235622951850799]
 [1.0337581256020607, 0.9063703842886133]

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)
# Task1
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаем описание модели:
lv! = @ode_def Malthus begin
    dx = a*x
end a
# задаём начальное условие:
u0 = [2]
# задаём значения параметров:
b = 3.0
c = 1.0
```

```
p = (b-c)
# задаём интервал времени:
tspan = (0.0,3.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 12-element Vector{Float64}:
 0.0
 0.07579340539309044
 0.2176538131796436
 0.39326275375009306
 0.6100444793398203
 0.8636787203353302
 1.1544101119687582
 1.4789340537388638
 1.8349001265017795
 2.219134461733416
 2.628731787861167
 3.0
u: 12-element Vector{Vector{Float64}}:
 [2.0]
 [2.327358634990142]
 [3.0908767890047213]
 [4.391507855871063]
 [6.774976441549192]
 [11.251525438518586]
```

```
 [20.12503871207196]
 [38.513500897099114]
 [78.48706775956025]
 [169.25231460681178]
 [383.9709586782193]
 [806.8145670268354]
plot(sol, label = "Численность изолированной популяции x(t)",
    color="green", ls=[:solid], title="Модель Мальтуса",
    xaxis="Время",yaxis="Размер изолированной популяции")
animate(sol, fps=7, "Malthus.gif",
    label = "Численность изолированной популяции x(t)",
    color="green", ls=[:solid], title="Модель Мальтуса",
    xaxis="Время", yaxis="Размер изолированной популяции")
# Task2
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаем описание модели:
lv! = @ode_def Logistic_population begin
    dx = r*x*(1-x/k)
end r k
# задаём начальное условие:
u0 = [1.0]
# задаём знанчения параметров:
p = (0.9, 20)
# задаём интервал времени:
tspan = (0.0,10.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
retcode: Success
```

```
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 14-element Vector{Float64}:
  0.0
  0.10320330193850687
  0.3855506045099877
  0.780748965506008
  1.262015691559725
  1.8586159628422565
  2.574933530608521
  3.471498551774082
  4.571529609523612
  5.629314234929612
  6.930091225213617
  8.078262639019435
  9.531767314565881
 10.0
u: 14-element Vector{Vector{Float64}}:
 [1.0]
 [1.092018818522065]
 [1.3860627615966585]
 [1.9212436077635002]
 [2.816088473652035]
 [4.379382291381814]
 [6.9638339217858904]
 [10.897151531483962]
 [15.262798385676023]
 [17.860400164058895]
 [19.28300459695191]
```

```
 [19.73872139608262]
 [19.928358494866384]
 [19.95293645513508]
plot(sol, label = "Численность популяции x(t)", color="blue", ls=[:solid],
    title="Логистическая модель роста популяции",
    xaxis="Время",yaxis="Размер популяции")
animate(sol, fps=7, "Logistic_population.gif",
    label = "Численность популяции x(t)", color="blue", ls=[:solid],
    title="Логистическая модель роста популяции",
    xaxis="Время", yaxis="Размер популяции")
#Task3
# задаём описание модели:
lv! = @ode_def SIR begin
ds = - b*i*s
di = b*i*s - v*i
dr = v*i
end b v

# задаём начальное условие:
u0 = [1.0, 0.1, 0]
# задаём значения параметров:
p = (0.25, 0.05)
# задаём интервал времени:
tspan = (0.0, 100.0)

# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
retcode: Success
```

```
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 19-element Vector{Float64}:
   0.0
   0.08088145925786733
   0.674649456103469
   1.9774507638268786
   3.928608933045557
   6.371598738903415
   9.52414865378298
  13.099293783864182
  17.0272982736033
  22.927215420937856
  27.195723313986843
  33.36650873512655
  39.87152643660008
  49.090534040944405
  57.69126316873367
  69.09753551513025
  81.37728197451536
  95.06634205664659
 100.0
u: 19-element Vector{Vector{Float64}}:
 [1.0, 0.1, 0.0]
 [0.9979636107059043, 0.10162869618330198, 0.0004076931107937434]
 [0.9821139347502068, 0.11427647441254161, 0.003609590837251619]
 [0.9414409947662143, 0.1464902846000639, 0.012068720633721717]
 [0.8642596086672918, 0.2065639776528575, 0.029176413679850716]
 [0.74121657128916, 0.29889094291007307, 0.05989248580076701]
```

```
 [0.5567300467580422, 0.42613510184975667, 0.11713485139220119]
 [0.360654706841008, 0.5353792639828872, 0.20396602917610487]
 [0.20739657233924386, 0.5779798290842139, 0.3146235985765423]
 [0.09060823642513902, 0.5292223125052752, 0.4801694510695858]
 [0.05338199009370889, 0.46063944558344505, 0.585978564322846]
 [0.028391022566481953, 0.35937261970727175, 0.7122363577262463]
 [0.0170789878367201, 0.26904206253122753, 0.8138789496320523]
 [0.010313325603209397, 0.17491074430234183, 0.9147759300944487]
 [0.007575677466988872, 0.11594548156094338, 0.9764788409720677]
 [0.005875265433141473, 0.0667974153589635, 1.027327319207895]
 [0.00503471355663355, 0.03675496060375371, 1.0582103258396127]
 [0.004593344331796414, 0.018844607044236347, 1.0765620486239673]
 [0.004499446877146492, 0.014807724434560448, 1.080692828688293]
plot(sol, label = ["Восприимчивые" "Инфицированные" "Переболевшие"],
    color=["blue" "green" "red"], ls=[:solid :dash :dot],
    title="Модель эпидемии Кермака-Маккендрика SIR",
    xaxis="Время",yaxis="Размер популяции")
animate(sol, fps=7, "SIR.gif", label = ["Восприимчивые" "Инфицированные"
    "Переболевшие"], color=["blue" "green" "red"],
    ls=[:solid :dash :dot], title="Модель эпидемии Кермака-Маккендрика SIR",
    xaxis="Время", yaxis="Размер популяции")
# Task4
using ParameterizedFunctions, DifferentialEquations, Plots;
N = 1.0
# задаём описание модели:
lv! = @ode_def SEIR begin
    ds = -($\beta$/N)*s*i
    de = ($\beta$/N)*s*i - $\delta$*e
    di = $\delta$*e - $\gamma$*i
```

```
    dr = $\gamma$*i
end $\beta$ $\gamma$ $\delta$

initialInfect = 0.1
# задаём начальное условие:
u0 = [(N - initialInfect), 0.0, initialInfect, 0.0]
# задаём значения параметров:
p = (0.6, 0.2, 0.1)
# задаём интервал времени:
tspan = (0.0, 100.0)

# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 25-element Vector{Float64}:
   0.0
   0.024423707511123237
   0.21983937298994613
   0.672446110582935
   1.3433111385495904
   2.2048531822239386
   3.3191976283637796
   4.685982405908369
   6.3524541259891905
   8.357303010682124
  10.779894718728759
```

```
  13.70832499771059
  17.247755058759296
  21.418982739513048
  26.11587299819979
  31.347120286227476
  37.434253818037966
  45.57027277276479
  51.92822542671895
  59.56906630189384
  67.00367237385267
  75.35077774263594
  84.11521635193901
  93.80030928820192
 100.0
u: 25-element Vector{Vector{Float64}}:
 [0.9, 0.0, 0.1, 0.0]
 [0.8986852898870974, 0.0013131042225504326, 0.09951432022141568, 0.000487285668
 [0.8884555417234649, 0.011417324323729522, 0.09582374914118306, 0.00430338481162
 [0.866527396902727, 0.03234294107695571, 0.08849598302303603, 0.012633678997281
 [0.8376686538121818, 0.05813585741007436, 0.08027144058351698, 0.023924048194226
 [0.8051950526908224, 0.08442177816749856, 0.07327976147138879, 0.037103407670290
 [0.7680943013707942, 0.11059043221384872, 0.06848784016256021, 0.052827426252796
 [0.7267952053167084, 0.13502937458285547, 0.06692539109637212, 0.071250029004064
 [0.6792664312390849, 0.15805852226928974, 0.0688812349182705, 0.093793811573354
 [0.623382992351777, 0.17997364305440322, 0.07423208615365647, 0.122411278440163
 [0.5563247884595703, 0.2007972721617986, 0.08253037899272447, 0.160347560385906
 [0.4768637125359172, 0.2186438129000962, 0.09277106500685588, 0.211721409557130
 [0.3871661137834433, 0.22877728944666506, 0.10287624546720801, 0.281180351302683
 [0.2965346996802043 5, 0.22432013816924648, 0.10906830780960446, 0.37007685434094
```

```
 [0.21812154248717963, 0.20203117991295588, 0.10739980642601175, 0.4724474711738
 [0.15806973255508666, 0.16563080905944505, 0.09651368465549173, 0.5797857737299
 [0.11491898104500536, 0.1215634160999261, 0.077464413236204, 0.6860531896188644
 [0.08400800945909402, 0.07403353279384636, 0.05147687832977306, 0.7904815794172
 [0.07130232240330717, 0.04833829861708447, 0.035216955841511605, 0.845142423138
 [0.06276940082693701, 0.02824847582139928, 0.021352251676490398, 0.887629871675
 [0.05825882147195975, 0.0165002046928897, 0.012753410930975914, 0.9124875629041
 [0.055521002402506674, 0.008936742931616089, 0.007009643484943623, 0.9285326111
 [0.054029149007287586, 0.004666816658196447, 0.003692034829599681, 0.9376119995
 [0.05320737945653188, 0.002268337061083503, 0.0018033195237675385, 0.9427209639
 [0.05292214534843295, 0.001427809361623715, 0.0011373400856070426, 0.9445127052
```

```
plot(sol, label = ["Восприимчивые" "Контактные" "Инфицированные" "Переболевшие"],
    color=["blue" "black" "green" "red"],
    ls=[:solid :dash :dot :dashdot], title="Модель SEIR",
    xaxis="Время",yaxis="Размер популяции")
animate(sol, fps=7, "SEIR.gif", label = ["Восприимчивые" "Контактные"
    "Инфицированные" "Переболевшие"],
    color=["blue" "black" "green" "red"], ls=[:solid :dash :dot :dashdot],
    title="Модель SEIR", xaxis="Время", yaxis="Размер популяции")
import Pkg
Pkg.add("LaTeXStrings")
# Task5
using DifferentialEquations, Plots, ParameterizedFunctions, LaTeXStrings
# задаём значения параметров:
a, c, d = 2, 1, 5

# задаем функцию для дискретной модели
next(x1, x2) = [(a*x1*(1 - x1) - x1*x2), (-c*x2 + d*x1*x2)]
```

```
# рассчитываем точку равновесия
balancePoint = [(1 + c)/d ,(d*(a - 1)-a*(1 + c))/d]

# задаём начальное условие:
u0 = [0.8, 0.05]
modelingTime = 100

simTrajectory = Array{Union{Nothing, Array}}(nothing, modelingTime)

for t in 1:modelingTime
    simTrajectory[t] = []
    if(t == 1)
        simTrajectory[t] = u0
    else
        simTrajectory[t] = next(simTrajectory[t-1]...)
    end
end
scatter([simTrajectory[1][1]], [simTrajectory[1][2]],
    c=:red, ms=9, label="Начальное состояние")

plot!(first.(simTrajectory), last.(simTrajectory), color=:green,
    linestyle=:dash, marker = (:dot, 5, Plots.stroke(0)),
    label="Траектория модели",
    title = "Дискретная модель Лотки-Вольтерры")
scatter!([balancePoint[1]], [balancePoint[2]], color=:orange, markersize=5,
    label="Точка равновесия",
    xlabel="Жертвы", ylabel="Хищники")
n = 100
```

```
anim = @animate for i in 1:n
    modelingTime = (i)
    # задаём значения параметров:
    a, c, d = 2, 1, 5
    # задаём начальное условие:
    u0 = [0.8, 0.05]

    simTrajectory = Array{Union{Nothing, Array}}(nothing, modelingTime)

    for t in 1:modelingTime
        simTrajectory[t] = []
        if(t == 1)
            simTrajectory[t] = u0
        else
            simTrajectory[t] = next(simTrajectory[t-1]...)
        end
    end

    scatter([simTrajectory[1][1]], [simTrajectory[1][2]],
    c=:red, ms=9, label="Начальное состояние")

    plot!(first.(simTrajectory), last.(simTrajectory), color=:green,
    linestyle=:dash, marker = (:dot, 5, Plots.stroke(0)),
    label="Траектория модели", title = "Дискретная модель Лотки-Вольтерры",
    xlabel="Жертвы", ylabel="Хищники")
    scatter!([balancePoint[1]], [balancePoint[2]], color=:orange, markersize=5,
    label="Точка равновесия", xlabel="Жертвы", ylabel="Хищники")
end
gif(anim, "LotkaVolterra.gif", fps=7)
```

```
[ Info: Saved animation to C:\Users\Reachna\computer-analysis\LotkaVolterra.gif
# Task6
using ParameterizedFunctions, DifferentialEquations, Plots;
# задаем описание модели:
lv! = @ode_def CompetitiveSelectionModel begin
    dx = a*x - b*x*y
    dy = a*y - b*x*y
end a b
# задаём начальное условие:
u0 = [1.0,1.4]
# задаём значения параметров:
p = (0.5, 0.2)
# задаём интервал времени:
tspan = (0.0,10.0)
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 20-element Vector{Float64}:
  0.0
  0.13063515958673816
  0.6620095919016169
  1.4745519498111759
  2.384274376329455
  3.4538271544345127
  4.562673961852109
  5.67839475574588
```

```
  6.618742680948912
  7.174117808180357
  7.581652232104449
  7.907287404662807
  8.211743148258813
  8.476131118651619
  8.723933719403542
  8.961493008998673
  9.203605421462836
  9.457562739292689
  9.735425791326456
 10.0
u: 20-element Vector{Vector{Float64}}:
 [1.0, 1.4]
 [1.028414415994334, 1.4554136105342876]
 [1.1349867943706582, 1.6919333806954628]
 [1.2538768556942972, 2.0899703101538307]
 [1.2907022905384151, 2.608347267277165]
 [1.1633365332504924, 3.412641939340822]
 [0.8307911711859107, 4.746685014866651]
 [0.3930025927897952, 7.233789081223332]
 [0.11653304820955845, 11.063651811087418]
 [0.03764043695860551, 14.488643868861088]
 [0.012491831797030182, 17.729605845536696]
 [0.004202666799768924, 20.85409224693543]
 [0.001245992590926548, 24.27939005309544]
 [0.00036185736409178275, 27.709728708980833]
 [9.560318903313416e-5, 31.364442854341014]
 [2.244509754328223e-5, 35.32009587775201]
```

```
 [4.313586083190942e-6, 39.86535971687741]
 [6.934576244296251e-7, 45.26283767286127]
 [1.4912772355035453e-7, 52.009055572552825]
 [4.679085610076088e-8, 59.365007376922144]
plot(sol, label = ["1-ый биологический вид" "2-ой биологический вид"],
    color=["blue" "red"], ls=[:solid :dash],
    title="Модель роста популяции в условиях конкуренции",
    xaxis="Время",yaxis="Размер популяции")
# фазовый портрет:
plot(sol,vars=(1,2), color="black", title="Фазовый портрет",
    xaxis="1-ый биологический вид",
    yaxis="2-ой биологический вид", legend=false)
animate(sol, fps=7, "CompetitiveSelection.gif", label = ["1-ый биологический вид"
    "2-ой биологический вид"], color=["blue" "red"], ls=[:solid :dash],
    title="Модель роста популяции в условиях конкуренции",
    xaxis="Время", yaxis="Размер популяции")
# Task7
# задаём описание модели:
lv! = @ode_def classicOscillator begin
dx = y
dy = -(w0^2)*x
end w0

# задаём начальное условие:
u0 = [1.0, 1.0]
# задаём значнения параметров:
p = (2.0)
# задаём интервал времени:
tspan = (0.0, 10.0)
```

```
# решение:
prob = ODEProblem(lv!,u0,tspan,p)
sol = solve(prob)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 31-element Vector{Float64}:
  0.0
  0.07580097943195412
  0.2069885812216689
  0.35309669557584694
  0.5285194634228536
  0.7514914358697009
  1.0072081570278821
  1.2779920001493048
  1.5687719973957874
  1.9026764818913302
  2.2229737198679334
  2.5850540372165933
  2.9526997573066778
  .
  5.742944245724529
  6.171924002188556
  6.584586020436235
  7.010469902680138
  7.433067666627458
  7.849769901055738
  8.282275752485367
```

```
  8.684259757356292
  9.126171387660838
  9.52416872011131
  9.970147020711996
 10.0
u: 31-element Vector{Vector{Float64}}:
 [1.0, 1.0]
 [1.0640413705392677, 0.6864865930281919]
 [1.1166550813709244, 0.11102078370062098]
 [1.0853087396797187, -0.5370470078797774]
 [0.9269048870800588, -1.2503554234259173]
 [0.5666112375440778, -1.9276425383253635]
 [0.022386691007709538, -2.2356196821467753]
 [-0.5570284464613865, -1.9387831861313267]
 [-0.9979739926547287, -1.0080686210010363]
 [-1.0957225558152268, 0.4445374258736321]
 [-0.7456354461054244, 1.6661869368567293]
 [-0.006569139131189493, 2.2360614231772953]
 [0.7451324586587859, 1.6671355348183015]
 .
 [0.03021521168044187, 2.235606930351544]
 [0.8654637606214528, 1.41627272049878]
 [1.1073959714327866, -0.3112605647462868]
 [0.6122516331842249, -1.871658434967589]
 [-0.29388388213013045, -2.1580890344574137]
 [-0.9963842964392918, -1.015855708266747]
 [-1.0328816676509016, 0.8579542164981876]
 [-0.4077221808745367, 2.0829571936841744]
 [0.5467601666298106, 1.9514927878253023]
```

```
 [1.0797592054767065, 0.583755858914729]
 [0.9051310875033446, -1.3144311246694527]
 [0.8643018846988861, -1.4201082114276489]
plot(sol, label = ["x(t)" "dx(t)"], color=["purple" "red"], ls=[:solid :dash],
    title="Модель консервативного гармонического осциллятора",
    xaxis="Время",yaxis="Отклик")
# фазовый портрет:
plot(sol, vars=(1,2), color="black", title="Фазовый портрет",
xaxis="x(t)", yaxis="dx(t)", legend=false)
animate(sol, fps=7, "harmonic_oscillator_model.gif",
    label = ["x(t)" "dx(t)"], color=["red" "green"], ls=[:solid :dash],
    title="Модель консервативного гармонического осциллятора",
    xaxis="Время",yaxis="Отклик")
# Task8
# задаём описание модели:
lv! = @ode_def Oscillator begin
dx = y
dy = -2*v*y - (w0^2)*x
end v w0

# задаём начальное условие:
u0 = [0.5, 1.0]
# задаём значения параметров:
p = (0.5, 2.0)
# задаём интервал времени:
tspan = (0.0, 10.0)

# решение:
prob = ODEProblem(lv!,u0,tspan,p)
```

```
sol = solve(prob)
retcode: Success
Interpolation: specialized 4th order "free" interpolation,
specialized 2nd order "free" stiffness-aware interpolation
t: 31-element Vector{Float64}:
  0.0
  0.07472295275624102
  0.2111474073618621
  0.370998115535819
  0.5569932690427111
  0.7935651279847608
  1.0592494385311997
  1.3434677044587358
  1.6345590088704463
  1.9695771278946115
  2.298725814870145
  2.6637691425219807
  3.0363220157596453
  .
  5.737115484628017
  6.172863902331125
  6.563858473632751
  6.9841201699413915
  7.37336534430487
  7.815553300011508
  8.212080450073843
  8.639453151000238
  9.029651151704007
  9.479283944498775
```

```
 9.87771453843803
 10.0
u: 31-element Vector{Vector{Float64}}:
 [0.5, 1.0]
 [0.566294838471426, 0.7739308462880993]
 [0.6437437851223823, 0.3637766487994777]
 [0.6656226258660081, -0.08049773973260232]
 [0.6094379096181167, -0.503508004573969]
 [0.4452831172025973, -0.8447598314155575]
 [0.20099102764791993, -0.9462443065396515]
 [-0.04954803582308005, -0.7771790317510109]
 [-0.22751020774505532, -0.42797996515316294]
 [-0.2962791192338872, 0.00818480202141821]
 [-0.23845282512709462, 0.3134414248513738]
 [-0.09679021863435247, 0.42129625389844744]
 [0.04524414886471238, 0.3125279503774128]
 .
 [-0.03315737624036357, 0.07927158408600336]
 [0.0017326274212230485, 0.07121962845594192]
 [0.022062892859074344, 0.030158917447841632]
 [0.024809347789294035, -0.014651796201659706]
 [0.014235235204598871, -0.0355305572028829]
 [-0.0014083593645169308, -0.0309103127427868]
 [-0.01013099472742578, -0.01203260641848518]
 [-0.010785598009219697, 0.0077159074914185674]
 [-0.005781690145433664, 0.016065370061608866]
 [0.0011806103102176132, 0.013025832366786661]
 [0.004710304453757682, 0.004335633036576887]
 [0.005069760406208022, 0.0015704622491427875]
```

```
plot(sol, label = ["x(t)" "dx(t)"], color=["blue" "green"], ls=[:solid :dash],
    title="Модель свободных колебаний гармонического осциллятора",
    xaxis="Время",yaxis="Отклик")
# фазовый портрет:
plot(sol, vars=(1,2), color="black", title="Фазовый портрет",
    xaxis="x(t)", yaxis="dx(t)", legend=false)
animate(sol, fps=7, "oscillator_model.gif", label = ["x(t)" "dx(t)"],
    color=["blue" "green"], ls=[:solid :dash],
    title="Модель свободных колебаний гармонического осциллятора",
    xaxis="Время",yaxis="Отклик")
```

4 Вывод

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