Лабораторная работа №6

Решение моделей в непрерывном и дискретном времени

Ким Реачна[1]

13 декабря, 2023, Москва, Россия

[1]Российский Университет Дружбы Народов

Цели и задачи

Цель лабораторной работы

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

Задание

1. Используя Jupyter Lab, повторите примеры.
2. Выполните задания для самостоятельной работы.

Процесс выполнения лабораторной работы

Модель экспоненциального роста

```
[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. 1: Численное решение модель экспоненциального роста

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]:

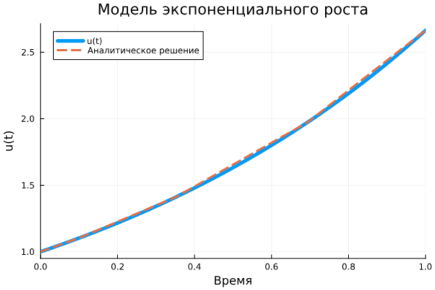

Рис. 2: График модель экспоненциального роста

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 знанчения 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. 3: Численное решение динамической системой Лоренца

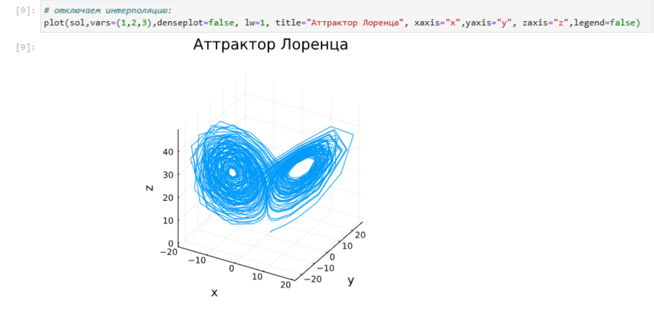

Рис. 4: Аттрактор Лоренца (интерполяция отключена)

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 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)

[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
        3.0380065851974147
        ⋮
        6.455762090996754
        6.780496138817711
        7.171040059920871
        7.584863345264154
        7.978068981329682
        8.48316543760351
        8.719248247740158
        8.949206788834692
        9.200185054623292
        9.438029017301554
        9.711808134779586
       10.0
```

Рис. 5: Численное решение модель Лотки–Вольтерры

Модель Лотки–Вольтерры

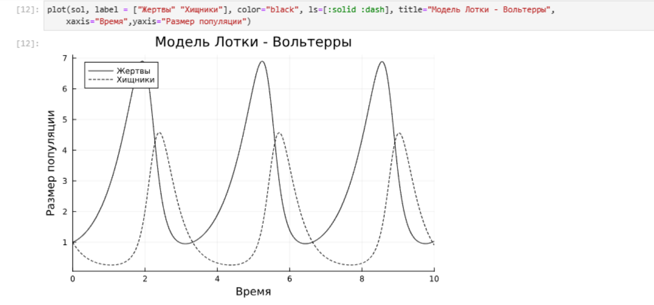

Рис. 6: Модель Лотки–Вольтерры: динамика изменения численности популяций

Модель Лотки–Вольтерры

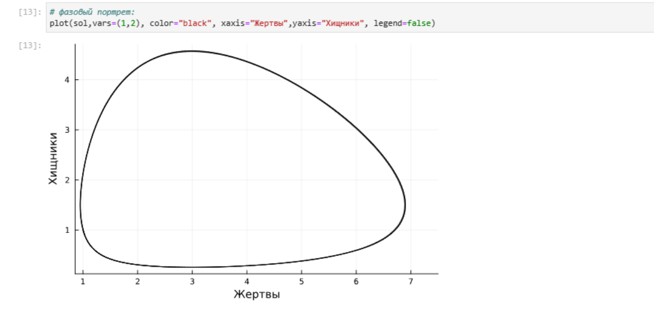

Рис. 7: Модель Лотки–Вольтерры: фазовый портрет

Модель Мальтуса

```
[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 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)
```

```
[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]
 [2.327358634990142]
 [3.0908767890047213]
 [4.391507855871063]
 [6.774976441549192]
 [11.251525438518586]
 [20.12503871207196]
 [38.513500897099114]
 [78.48706775956025]
 [169.25231460681178]
 [383.9709586782193]
 [806.8145670268354]
```

Рис. 8: Решение модели Мальтуса

Модель Мальтуса

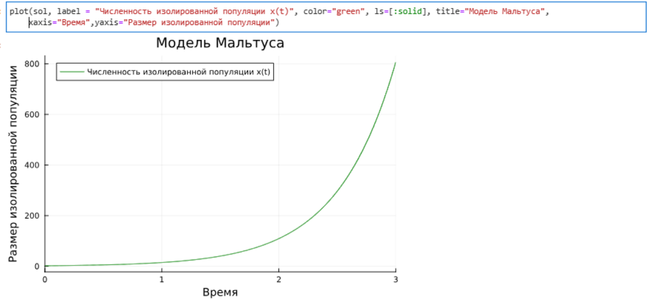

Рис. 9: График модели Мальтуса

Логистическая модель роста популяции

```
[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]
 [2.816088473652035]
 [4.379382291381814]
 [6.9638339217858904]
 [10.897151531483962]
 [15.262798385676023]
 [17.860400164058895]
 [19.28300459695191]
 [19.73872139608262]
 [19.928358494866384]
 [19.95293645513508]
```

Рис. 10: Решение логистическую модель роста популяции

Логистическая 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]:

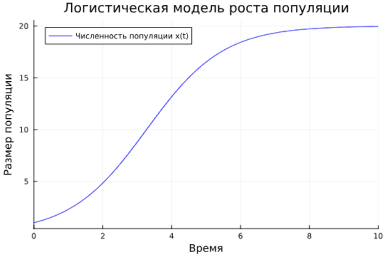

Рис. 11: График логистическую модель роста популяции

SIR-Модель

```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
  81.3772819745153б
  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]
```

Рис. 12: Решение SIR модель

SIR-Модель

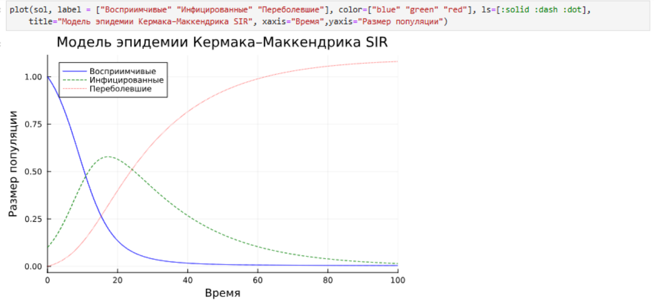

Рис. 13: График SIR модель

SEIR-Модель

```
[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 значенция 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
         0.672446110582935
         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]
```

Рис. 14: 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]:

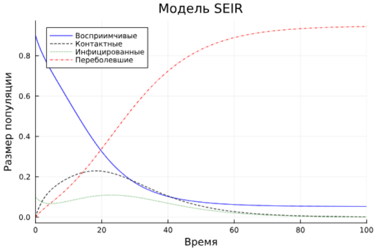

Рис. 15: График SEIR-модель

Дискретной модели Лотки–Вольтерры

```
[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 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
```

Рис. 16: Решение модели Лотки–Вольтерры

Дискретной 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")
```

[28]:

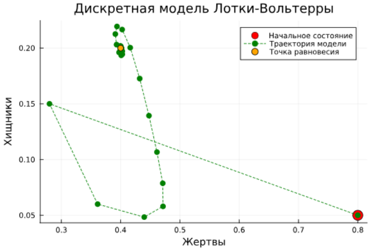

Рис. 17: График модели Лотки–Вольтерры

Модель роста популяции в условиях коконкуренции

```
[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 знанчения параметров:
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
  2.384274376329455
  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]
```

Рис. 18: Решение модели роста популяции в условиях коконкуренции

Модель роста популяции в условиях коконкуренции

Рис. 19: График модели роста популяции в условиях коконкуренции

Модель консервативного гармонического осциллятора

```
[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 знанчения 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
  0.35309669557584694
  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]
```

Рис. 20: Решение модель консервативного гармонического осциллятора

График модели консервативного гармонического осциллятора

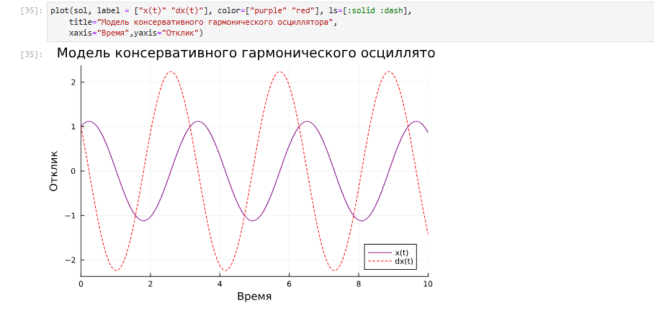

Рис. 21: График модели консервативного гармонического осциллятора

Фазовый портрет модели консервативного гармонического осциллятора

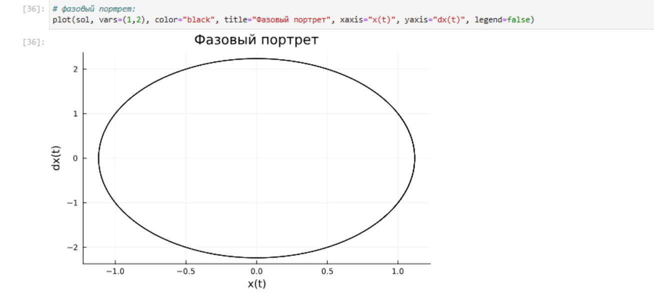

Рис. 22: Фазовый портрет

Модель свободных колебаний гармонического осциллятора

```
[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
        1.6345590088704463
        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]
```

Рис. 23: Решение модели свободных колебаний гармонического осциллятора

График 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")
```

[39]:

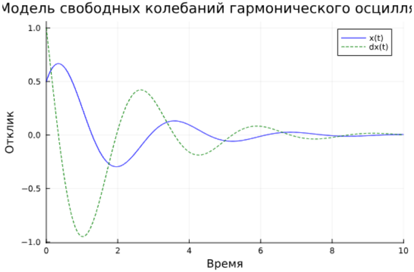

Рис. 24: График модели свободных колебаний гармонического осциллятора

Фазовый портрет модели свободных колебаний гармонического осциллятора

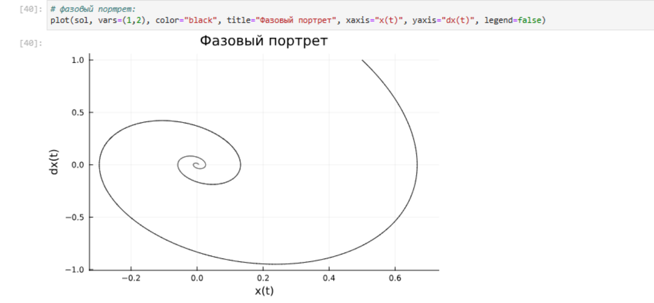

Рис. 25: Фазовый портрет

Выводы по проделанной работе

Вывод

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