Отчёт по лабораторной работе №6

Решение моделей в непрерывном и дискретном времени

Гань Чжаолун

Цель работы

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

Выполнение лабораторной работы

Решение обыкновенных дифференциальных уравнений

```
u′(t) = f(u(t), p, t),
```

где $f(u(t), p, t)$ — нелинейная модель (функция) изменения $u(t)$ с заданным начальным значением $u(t0$) = $u0$, p — параметры модели,t — время. Для решения обыкновенных дифференциальных уравнений в Julia можно использовать пакет diffrentialEquations.jl.

Модель экспоненциального роста

Численное решение в Julia будет иметь следующий вид:

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

```
   Updating registry at `~/.julia/registries/General`
######################################################################## 100.0%
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
```

```
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)
```

```
┌ Info: Precompiling DifferentialEquations [0c46a032-eb83-5123-abaf-570d42b7fbaa]
└ @ Base loading.jl:1278
┌ Warning: The call to compilecache failed to create a usable precompiled cache file for DifferentialEquations [0c46a032-eb83-5123-abaf-570d42b7fbaa]
│   exception = ErrorException("Required dependency ModelingToolkit [961ee093-0014-501f-94e3-6117800e7a78] failed to load from a cache file.")
└ @ Base loading.jl:1042
retcode: Success
Interpolation: automatic order switching interpolation
t: 5-element Array{Float64,1}:
 0.0
 0.10042494449239292
 0.35218603951893646
 0.6934436028208104
 1.0
u: 5-element Array{Float64,1}:
 1.0
 1.1034222047865465
 1.4121908848175448
 1.9730384275623003
 2.664456142481452
```

Рисунок 0.1 Модель экспоненциального роста 1

Построение графика, соответствующего полученному решению:

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

```
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
```

Рисунок 0.2 Модель экспоненциального роста 2

Результат:

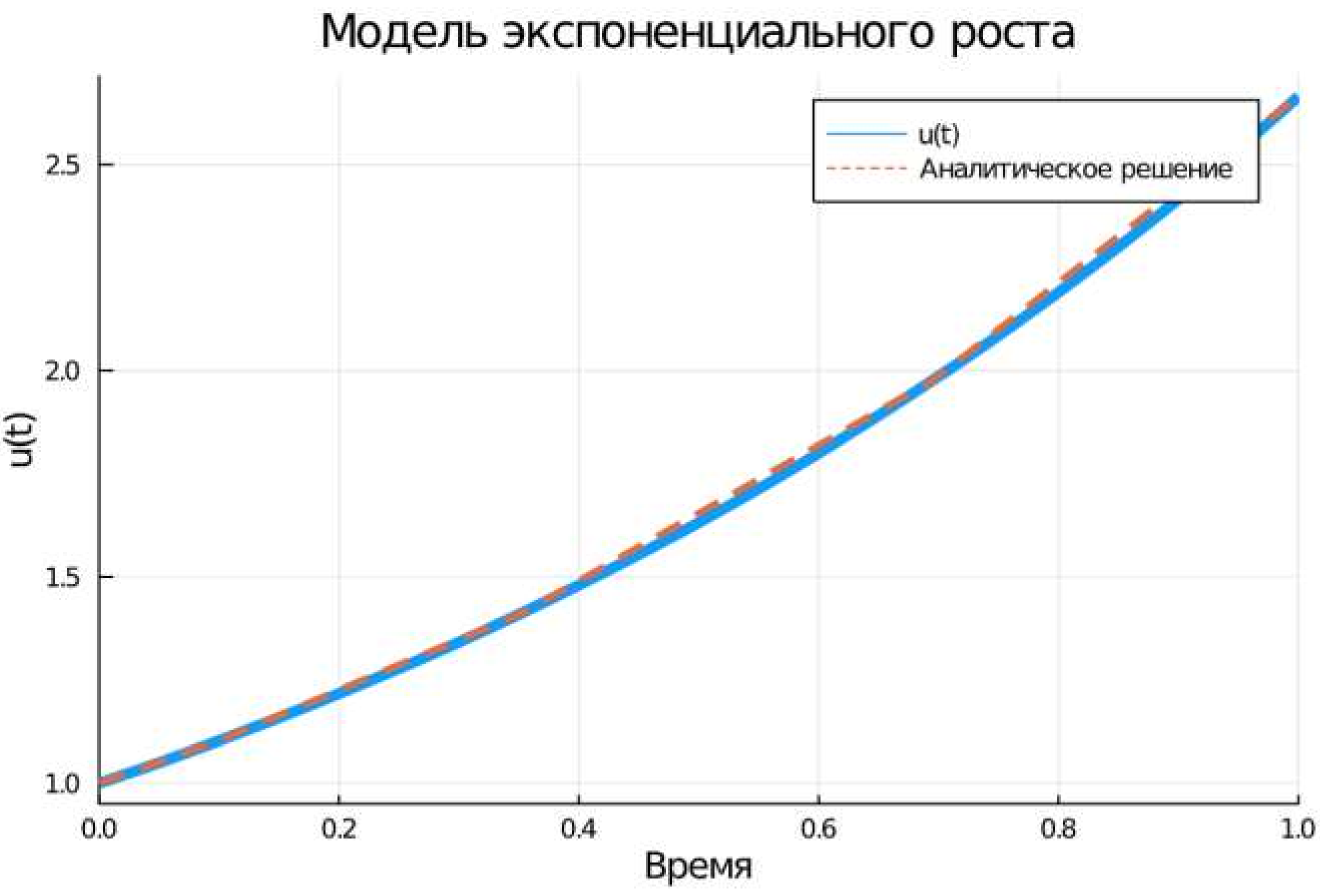

Рисунок 0.3 Модель экспоненциального роста 3

Если требуется задать точность решения, то можно воспользоваться параметрами abstol (задаёт близость к нулю) и reltol (задаёт относительную точность). По умолчанию эти параметры имеют значение abstol = 1e-6 и reltol = 1e-3. Для модели экспоненциального роста:

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

```
retcode: Success
Interpolation: automatic order switching interpolation
t: [0.0, 0.04127492324135852, 0.14679917846877366, 0.28631546412766684, 0.4381941361169628, 0.6118924302028597, 0.7985659100883337, 0.9993516479536952, 1.0]
u: [1.0, 1.0412786454705882, 1.1547261252949712, 1.3239095703537043, 1.5363819257509728, 1.8214895157178692, 2.1871396448296223, 2.662763824115295, 2.66445624193
3517]
```

```
# 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")
```

Рисунок 0.4 Модель экспоненциального роста 4

Результат:

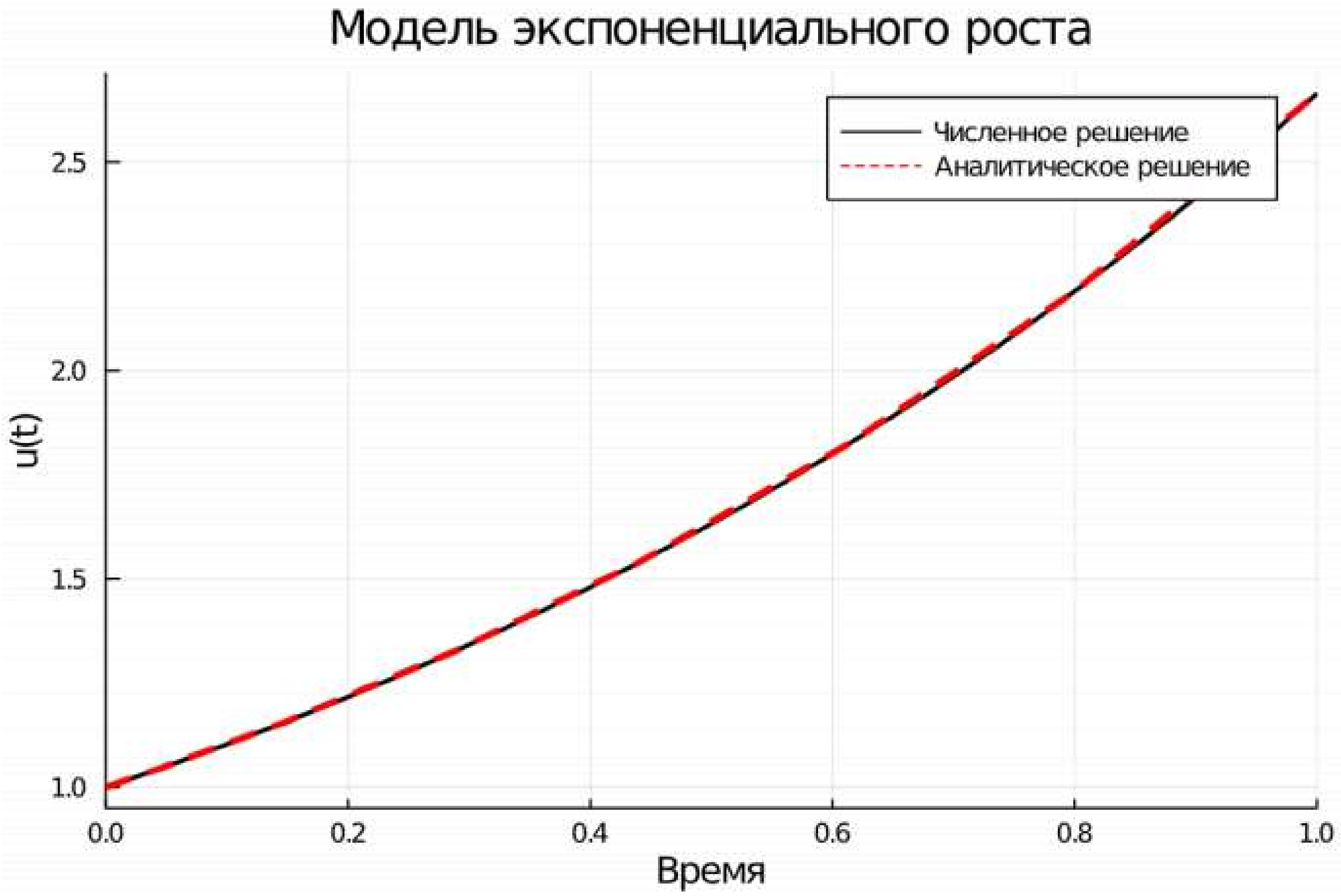

Рисунок 0.5 Модель экспоненциального роста 5

Система Лоренца

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

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

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

Численное решение в Julia будет иметь следующий вид:

```
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 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)
```

Рисунок 0.6 Система Лоренца 1

Фазовый портрет:

```
# 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)
```

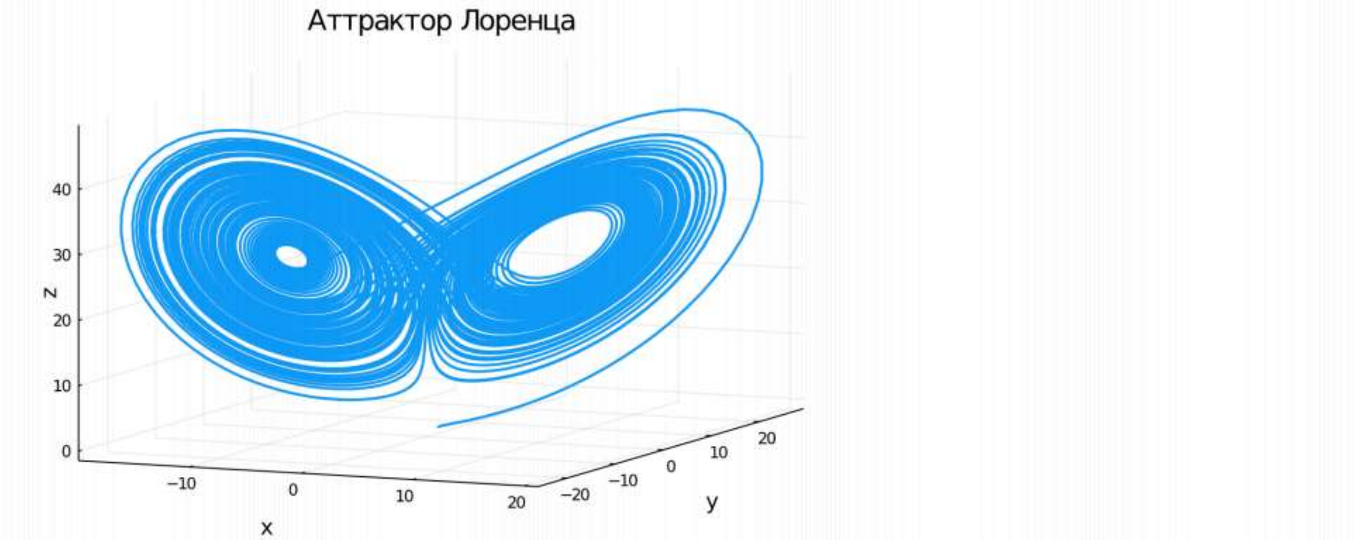

Рисунок 0.7 Система Лоренца 2

Можно отключить интерполяцию:

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

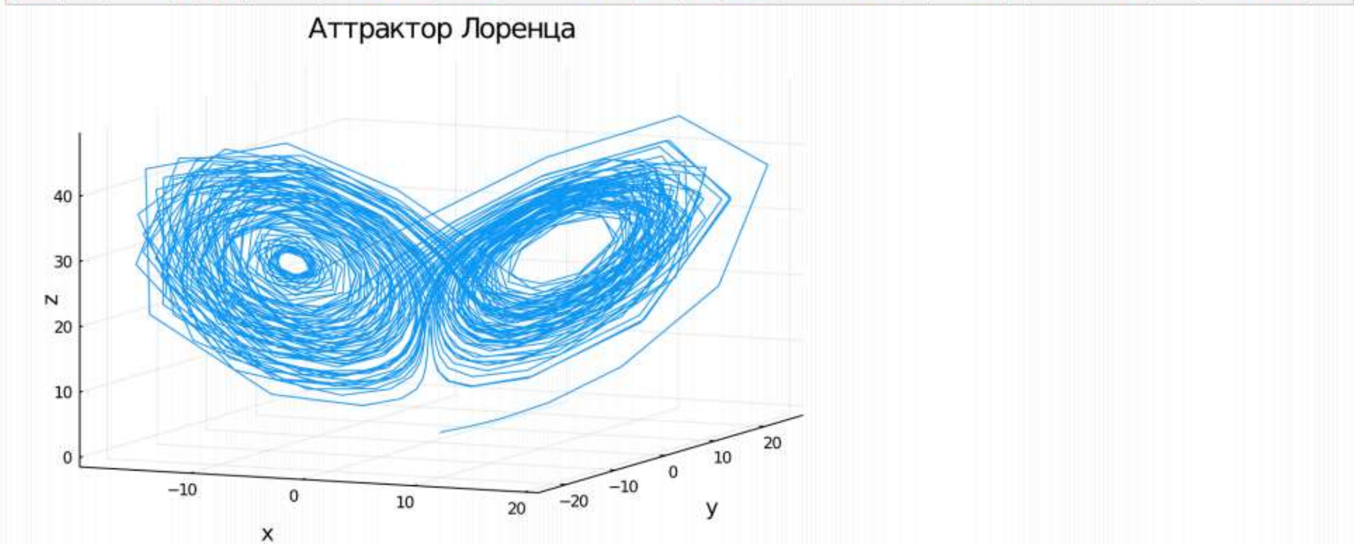

Рисунок 0.8 Система Лоренца 3

Модель Лотки–Вольтерры

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

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

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

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

```
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
```

```
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)
```

Рисунок 0.9 Модель Лотки–Вольтерры 1

Результат:

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

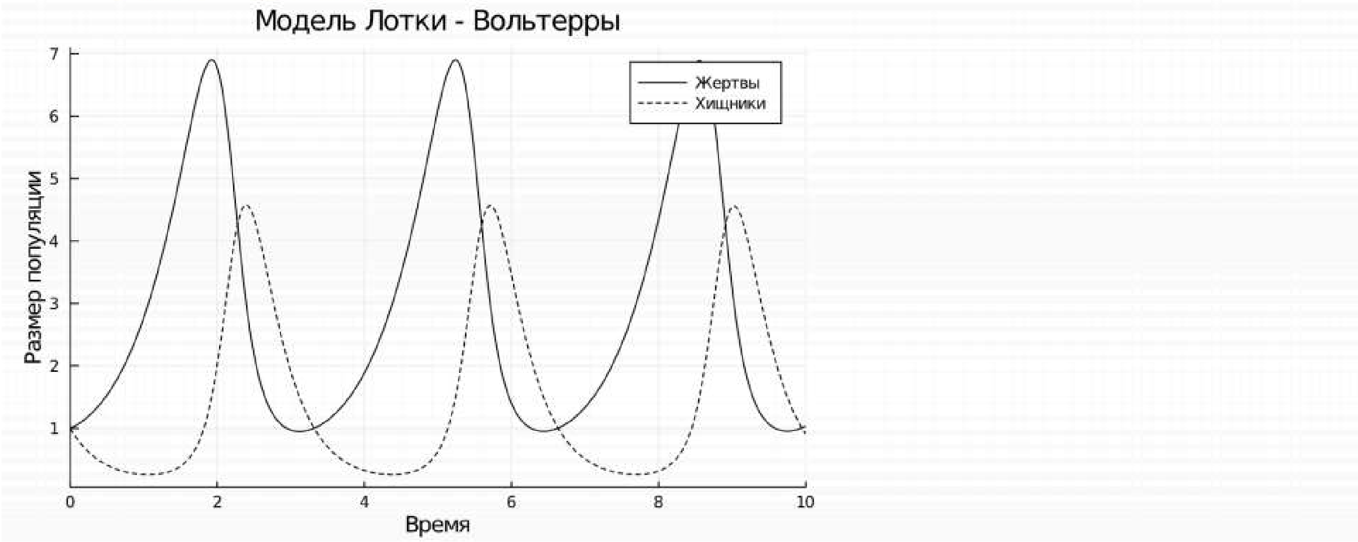

Рисунок 0.10 Модель Лотки–Вольтерры 2

Фазовый портрет:

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

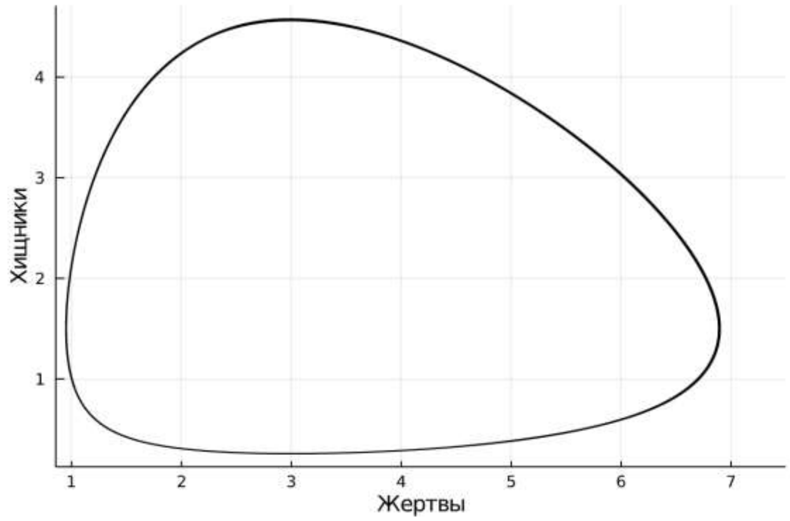

Рисунок 0.11 Модель Лотки–Вольтерры 3

Задания для самостоятельного выполнения

1. Реализовать и проанализировать модель роста численности изолированной популяции (модель Мальтуса): ¶

```
using ParameterizedFunctions, DifferentialEquations, Plots;

# задаём 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)
```

```
retcode: Success
Interpolation: automatic order switching interpolation
t: 12-element Array{Float64,1}:
 0.0
 0.07579340539309044
 0.2176538131796436
 0.3932627681470158
 0.6100444937861662
 0.8636787348395012
 1.154410126539212
 1.478934092373798
 1.8349002173915139
 2.2191346090276505
 2.6287319952831165
 3.0
u: 12-element Array{Array{Float64,1},1}:
 [2.0]
 [2.327358634990142]
 [3.0908767890047213]
```

Рисунок 1.1 Код и результат Задания 1-1

Я использовала следующие коэффициенты: a = b - c = 2.0, b = 3.0, c = 1.0 и смоделируем на интервале 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.

Результат:

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

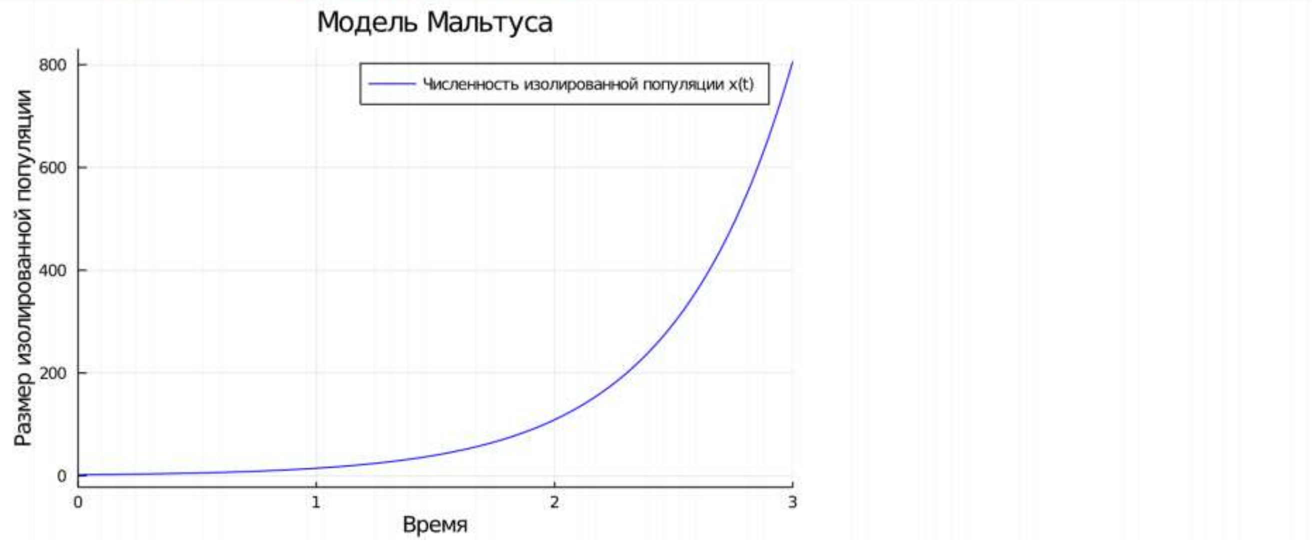

Рисунок 1.2 Код и результат Задания 1-2

Попробую подобрать другие параметры, чтобы сравнить скорость роста популяции. Допустим в этот раз я возьму довольно маленький коэффициент рождаемости и большой коэффициент смертности. Пусть b = 1, c = 3.

Результат:

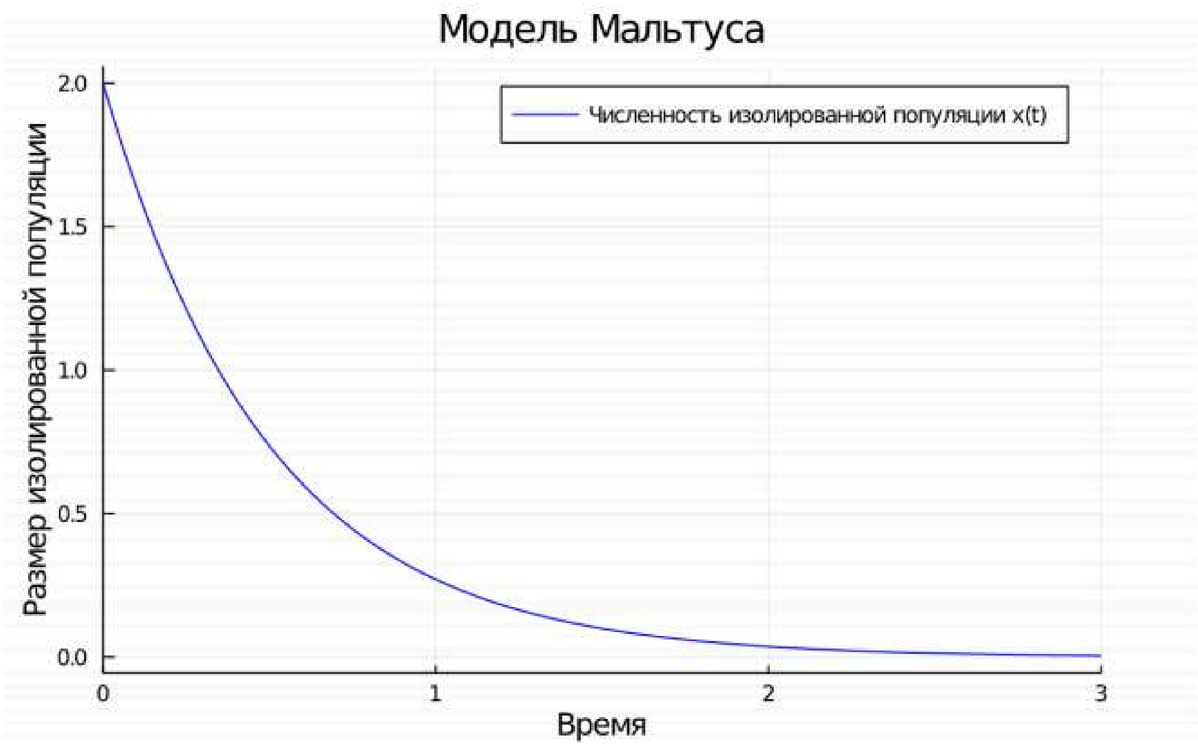

Рисунок 1.3 Код и результат Задания 1-3

Размер популяции за 3 ед. времени резко снизился до 0. Также реализована анимация графика и представлена вместе с программой в архиве.

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

```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/6 lab/Malthus.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

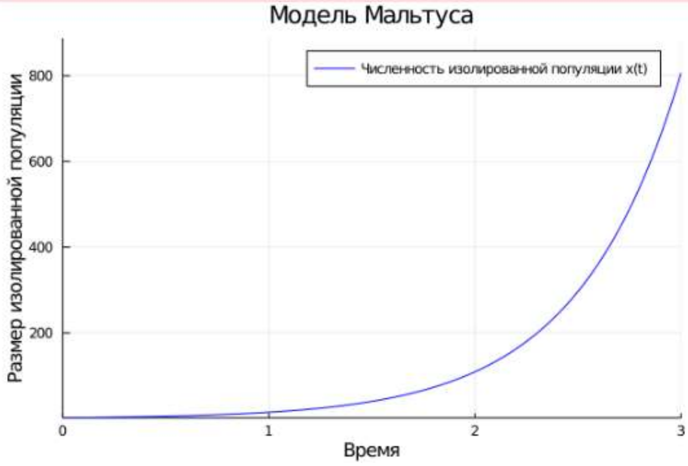

Рисунок 1.4 Код и результат Задания 1-4

2. Реализовать и проанализировать логистическую модель роста популяции, заданную уравнением:

```
# задаём 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)
```

```
retcode: Success
Interpolation: automatic order switching interpolation
t: 14-element Array{Float64,1}:
  0.0
  0.10320330193850687
  0.3855506045099877
  0.780748965506008
  1.262015691559725
  1.8586158648823017
  2.5749333150313944
  3.4714981889836993
  4.5715292448819005
  5.629313666416045
  6.930090935678242
  8.078262058777629
  9.531766731892224
 10.0
u: 14-element Array{Array{Float64,1},1}:
 [1.0]
```

Рисунок 2.1 Код и результат Задания 2-1

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 y значение 20 и выше не подниматься.

Результат:

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

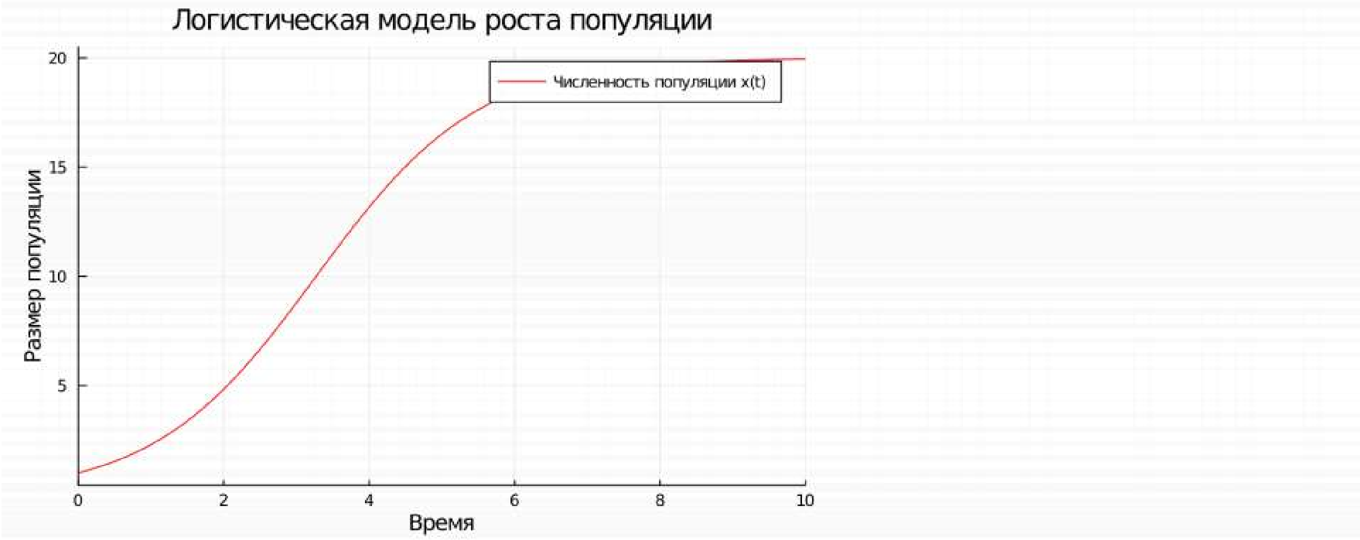

Рисунок 2.2 Код и результат Задания 2-2

Теперь разберем ситуация в которой коэффициент роста популяции намного меньше, допустим 0.1, а предельное значение допустим равно 25 на таком же временном интервале от 0 до 10.

Результат:

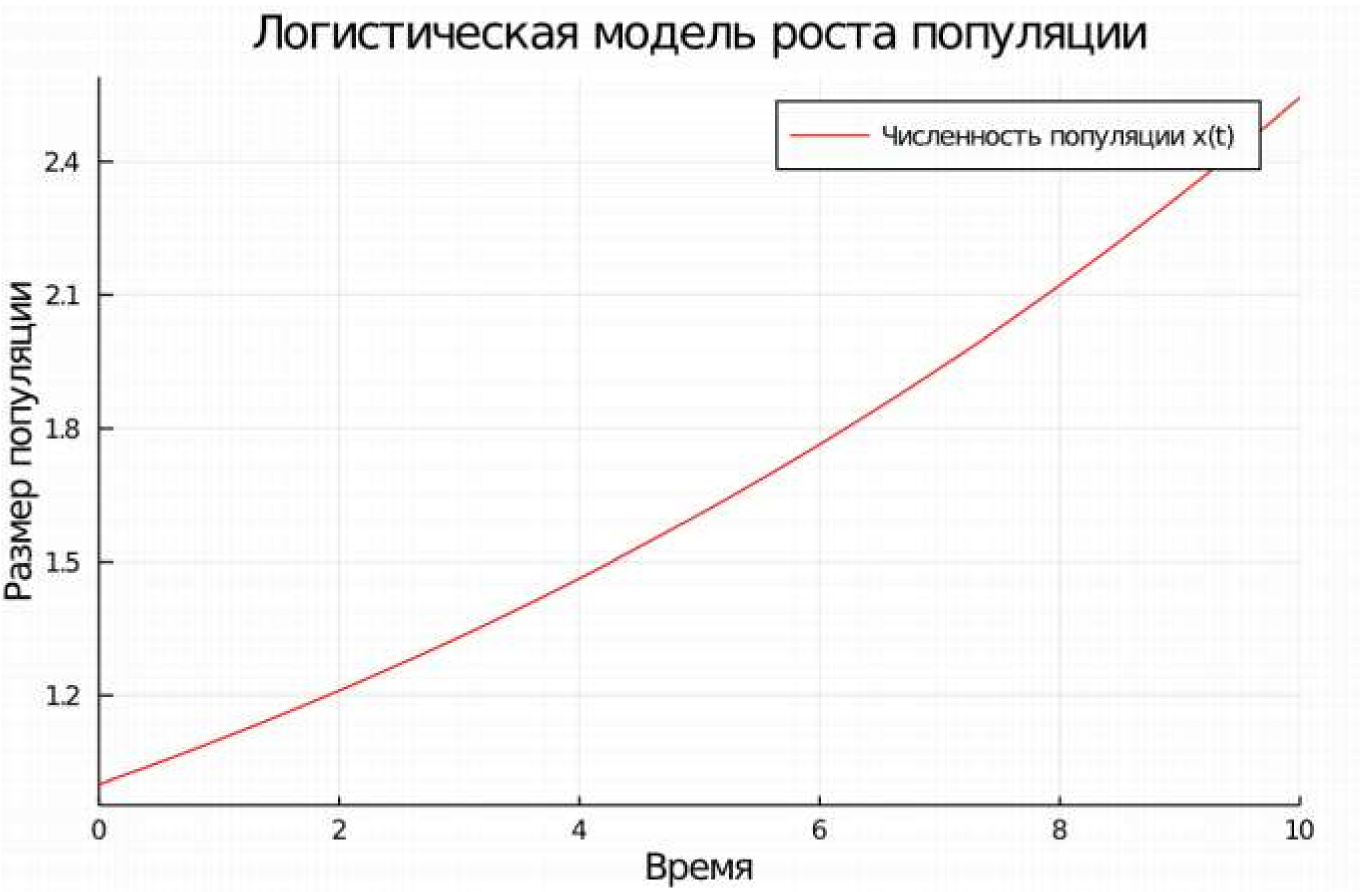

Рисунок 2.3 Код и результат Задания 2-3

Как мы видим, за 10 ед. времени с таким маленьким коэффициентом роста популяции она не успела дойти до предельного значения, увеличим временной промежуток до 80 и посмотрим результат.

Результат:

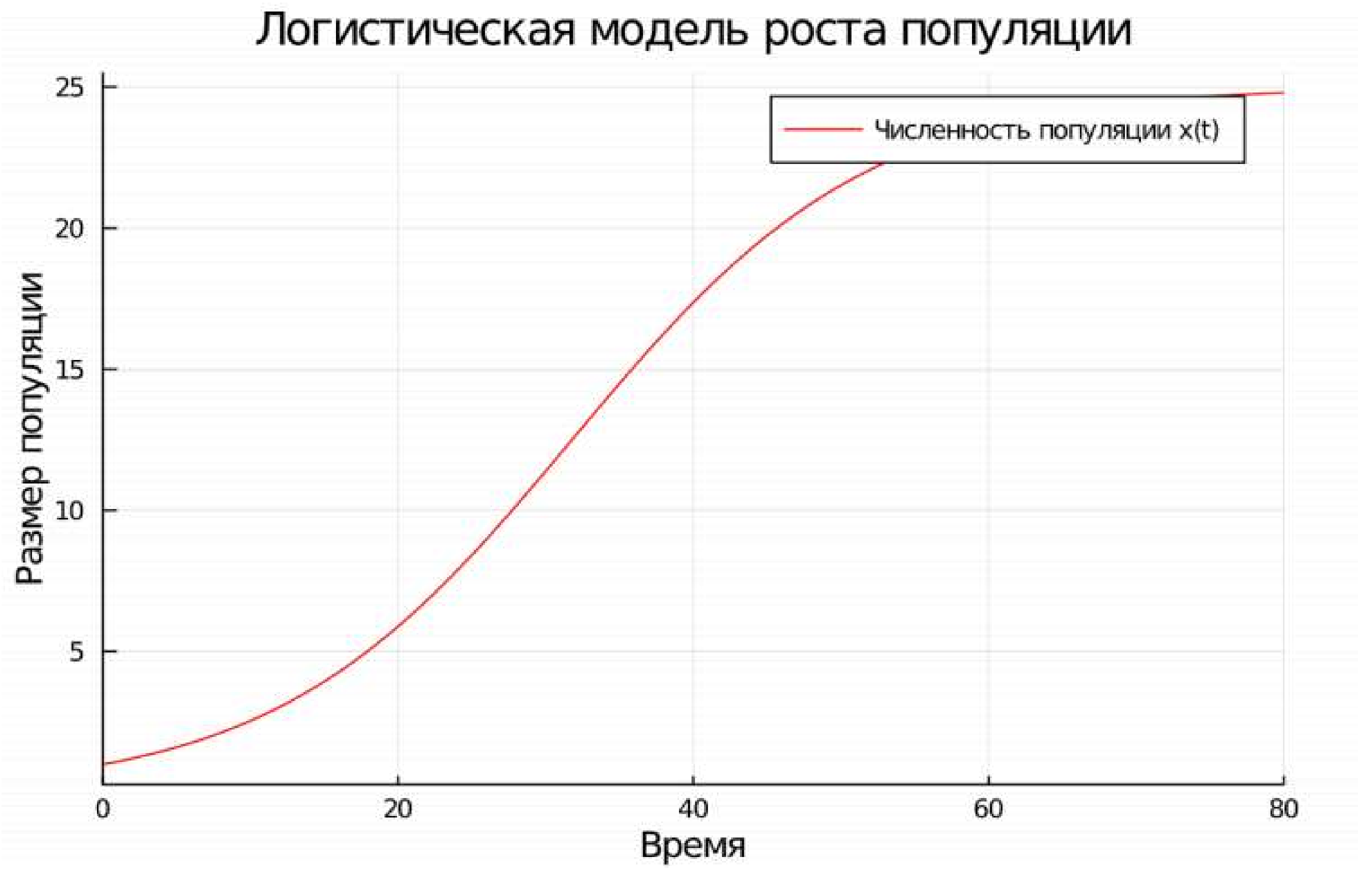

Рисунок 2.4 Код и результат Задания 2-4

Тем самым можно сделать вывод, что чем меньше мы возьмем коэффициент роста популяции, тем медленнее она будет расти.

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

```
# 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: automatic order switching interpolation
t: 19-element Array{Float64,1}:
   0.0
   0.08088145925786733
   0.674649456103469
   1.9774508584826755
   3.928609169463901
   6.371599152818624
   9.524149296748561
  13.099294418072489
  17.027299755640826
  22.92721876331779
  27.195725548013602
  33.36651350247801
  39.87153114962302
  49.09053867697217
  57.691269327139125
  69.09754269069009
```

Рисунок 3.1 Код и результат Задания 3-1

Изначально задам β = 0.25, ν = 0.05. Таким образом мы имеем достаточно маленький коэффициент интенсивности выздоровления и достаточно большой коэффициент интенсивности контактов индивидов с последующим инфицированием.

Результат:

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

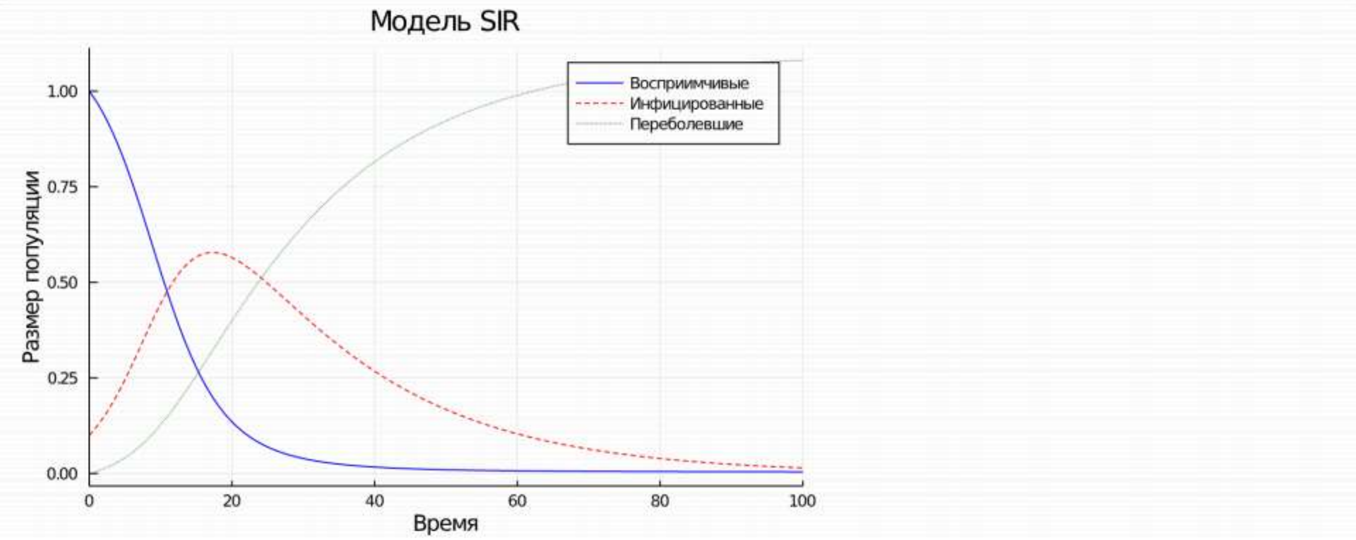

Рисунок 3.2 Код и результат Задания 3-2

Красный график интенсивности эпидемии, показывающей количество одномоментно болеющих индивидов, определяется параметром: R0 = β/ν = 5. Попробуем уменьшить данный коэффициент интенсивности и посмотрим на различия. Сделаем это за счет уменьшения коэффициента β, например, 0.15.

Результат:

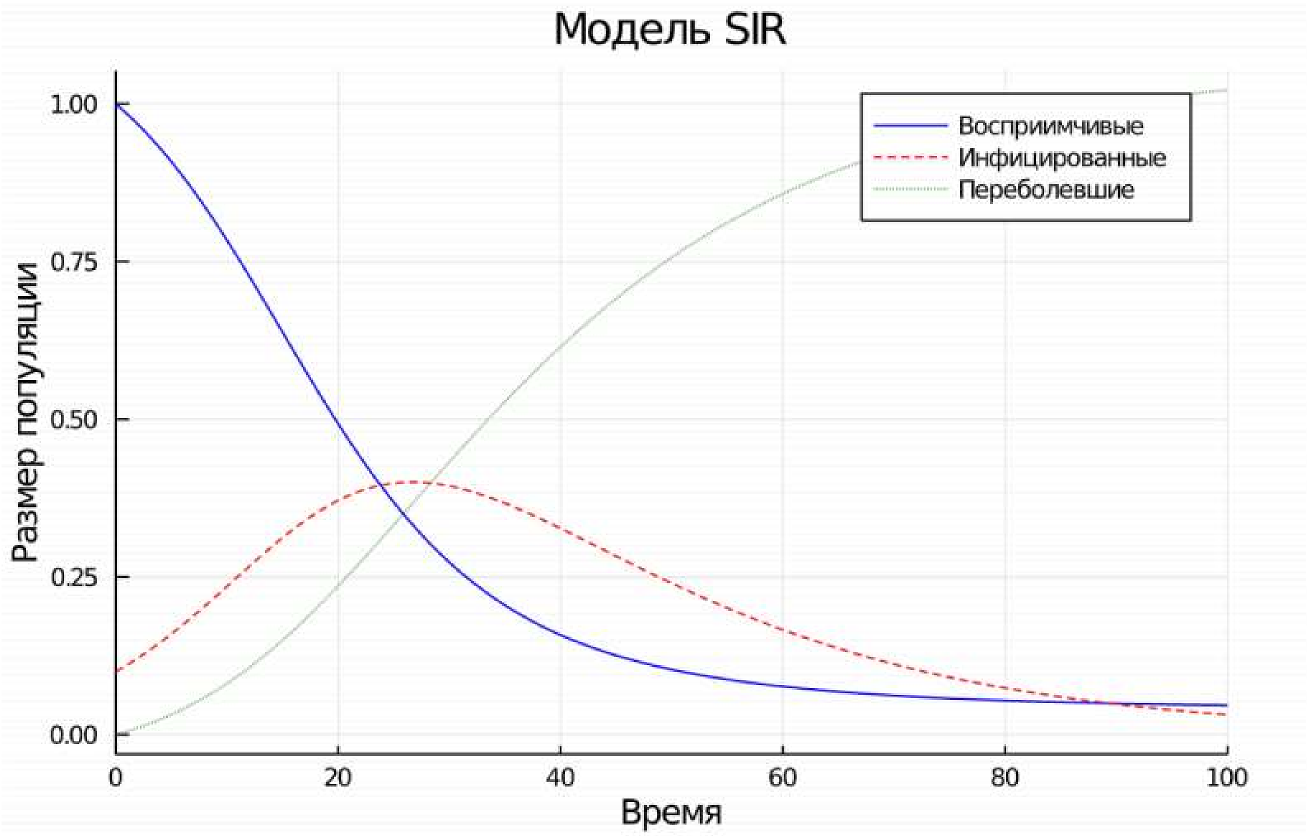

Рисунок 3.3 Код и результат Задания 3-3

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 R0 = β/ν = 3.

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

```
M = 1.0

# задаём описание модели:
lv! = @ode_def SEIR begin
ds = -(β/M)*s*i
de = (β/M)*s*i - δ*e
di = δ*e - γ*i
dr = γ*i
end β γ δ

initialInfect = 0.1
# задаём начальное условие:
u0 = [(M - 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: automatic order switching interpolation
t: 25-element Array{Float64,1}:
   0.0
   0.024423707511123237
   0.2198399425095939
   0.6724471660709395
   1.343312585078274
   2.2048549843871585
   3.3192000814261293
   4.6859854352737536
   6.3524579925856735
   8.357307405301253
  10.77990033301068
  13.70833185188398
  17.24776341152023
  21.41899285836336
  26.11588475544062
  31.347134254122864
```

Рисунок 4.1 Код и результат Задания 4-1

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

```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/6 lab/SEIR.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

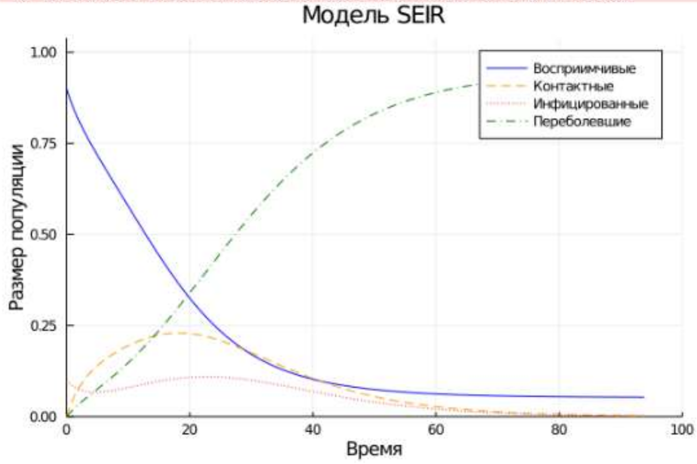

Рисунок 4.2 Код и результат Задания 4-2

1.Определение системы уравнений SEIR.

Я задал систему дифференциальных уравнений, в которой: - (s(t) — доля (или количество) восприимчивых,

- (e(t) — доля «контактных», то есть находящихся в инкубационном периоде,
- (i(t) — доля инфицированных,
- (r(t) — доля переболевших (удалённых из эпидемического процесса).

Дополнительно присутствуют параметры:

- beta — скорость контакта (заражения),
- delta — скорость перехода из «контактных» в инфицированные,
- gamma — скорость выздоровления (или удаления).

2.Выбор начальных условий.

Я задал начальные условия так, чтобы в начале почти вся популяция находилась во «восприимчивом» состоянии (s = M - initialInfect), небольшая часть сразу была инфицирована (i = initialInfect), а «контактных» и переболевших изначально нет (e = 0 и r = 0).

3.Задание интервала интегрирования и параметров.

Я выбрал временной промежуток t in [0, 100]. Параметрам beta, gamma, delta присвоил конкретные числовые значения, соответствующие гипотетическому сценарию развития эпидемии.

4.Решение системы ODE. Я сформировал объект `ODEProblem` из библиотеки `DifferentialEquations.jl`, передав ему описание системы, начальные условия, интервал времени и кортеж параметров. Затем с помощью функции `solve()` получил численное решение.

5.Визуализация и анимация.

- Итоговая GIF-анимация иллюстрирует, как SEIR-модель описывает развитие эпидемии с учётом «скрытого» периода состояние (e(t).

5. Для дискретной модели Лотки–Вольтерры:

```
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
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/6 lab/LotkaVolterra.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

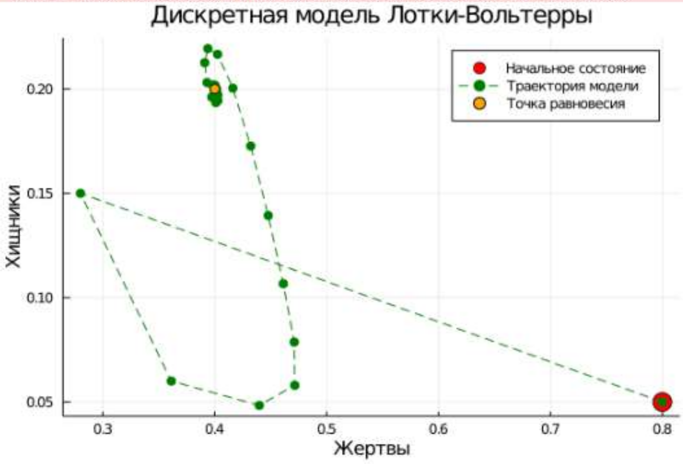

Рисунок 5.1 Код и результат Задания 5-1

1.Вычисление точки равновесия. Точка равновесия (X*1^), (X2^*) получается из условия, что Delta X*1 = 0 и Delta X*2 = 0. Я аналитически нашёл это решение и сохранил в переменную `balancePoint`.

2.Численное решение и построение траектории. - Я выбрал начальное состояние X*1(0)=0.8, X*2(0)=0.05 и задал количество итераций (`modelingTime`), после чего в цикле по шагам вычислял следующие значения жертв и хищников с помощью функции `next`. - Для визуализации создал фазовый портрет (ось абсцисс — жертвы, ось ординат — хищники) и отобразил: - начальную точку, - траекторию, по которой система перемещается, - точку равновесия.

3.Анимация. - Чтобы показать, как траектория «растёт» во времени, я использовал макрос `@animate`. На каждом шаге анимации я пересчитывал систему вплоть до текущего момента и рисовал результат. - Наконец, собрал все кадры в GIF-файл с помощью `gif(anim, "LotkaVolterra.gif", fps=7)`.

6. Реализовать на языке Julia модель отбора на основе конкурентных отношений:

```
# задаём 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)
```

```
 [1.0, 1.4]
 [1.028414415994334, 1.4554136105342876]
 [1.1349867878063837, 1.6919333641656147]
 [1.2538768548488464, 2.0899703058711037]
 [1.2907022936785661, 2.608347196451814]
 [1.1633366007475396, 3.4126416493413063]
 [0.8307913694023051, 4.746684173355672]
 [0.3930028424288755, 7.233787035234246]
 [0.11653311934280954, 11.06364992846164]
 [0.037640453121700033, 14.488642594432175]
 [0.012491835533500078, 17.729604986829926]
 [0.004202667929468129, 20.85409149286473]
 [0.0012459928627116352, 24.279389456771103]
 [0.00036185741893655868, 27.7097283083918]
 [9.560320356809188e-5, 31.364442456981713]
 [2.2445098860639743e-5, 35.32009575720295]
 [4.3135862944311176e-6, 39.86535963116519]
 [6.9345765585767666e-7, 45.262837635509484]
 [1.4912773550991116e-7, 52.009055605022986]
 [4.6790858856619414e-8, 59.365007377129125]
```

Рисунок 6.1 Код и результат Задания 6-1

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

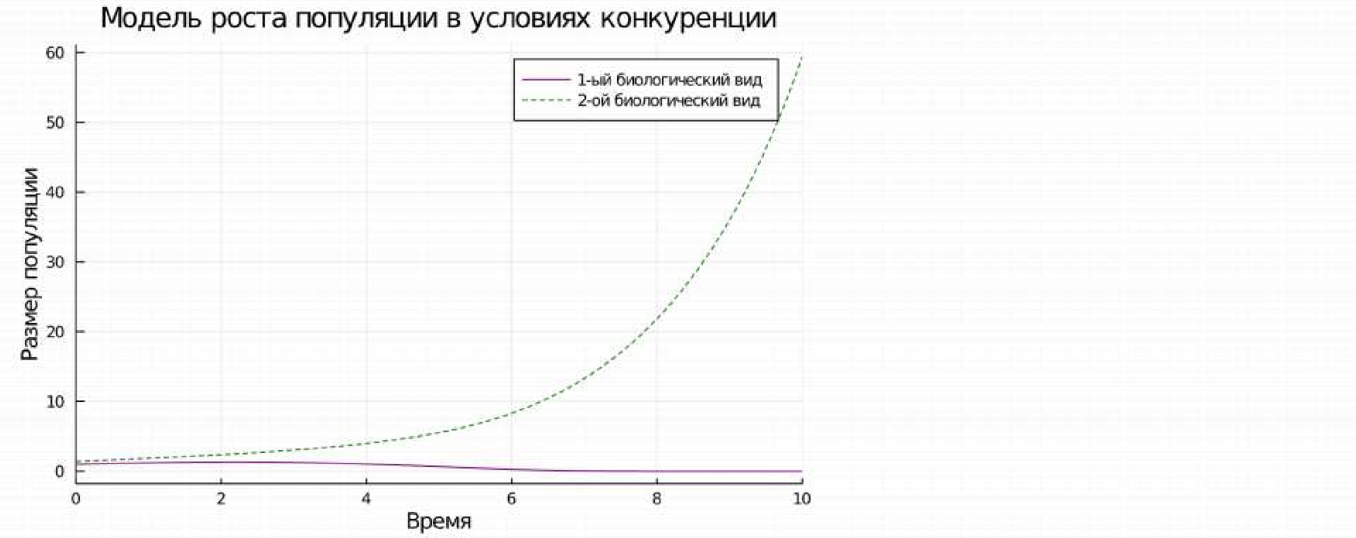

Рисунок 6.2 Код и результат Задания 6-2

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

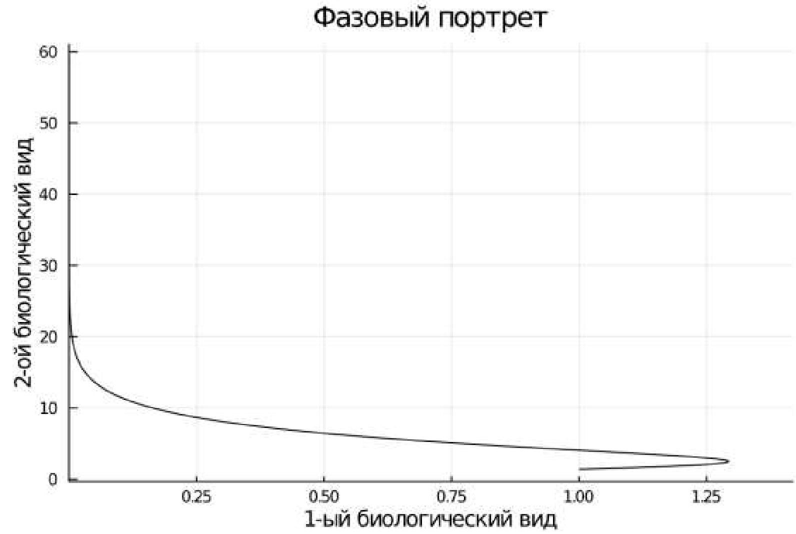

Рисунок 6.3 Код и результат Задания 6-3

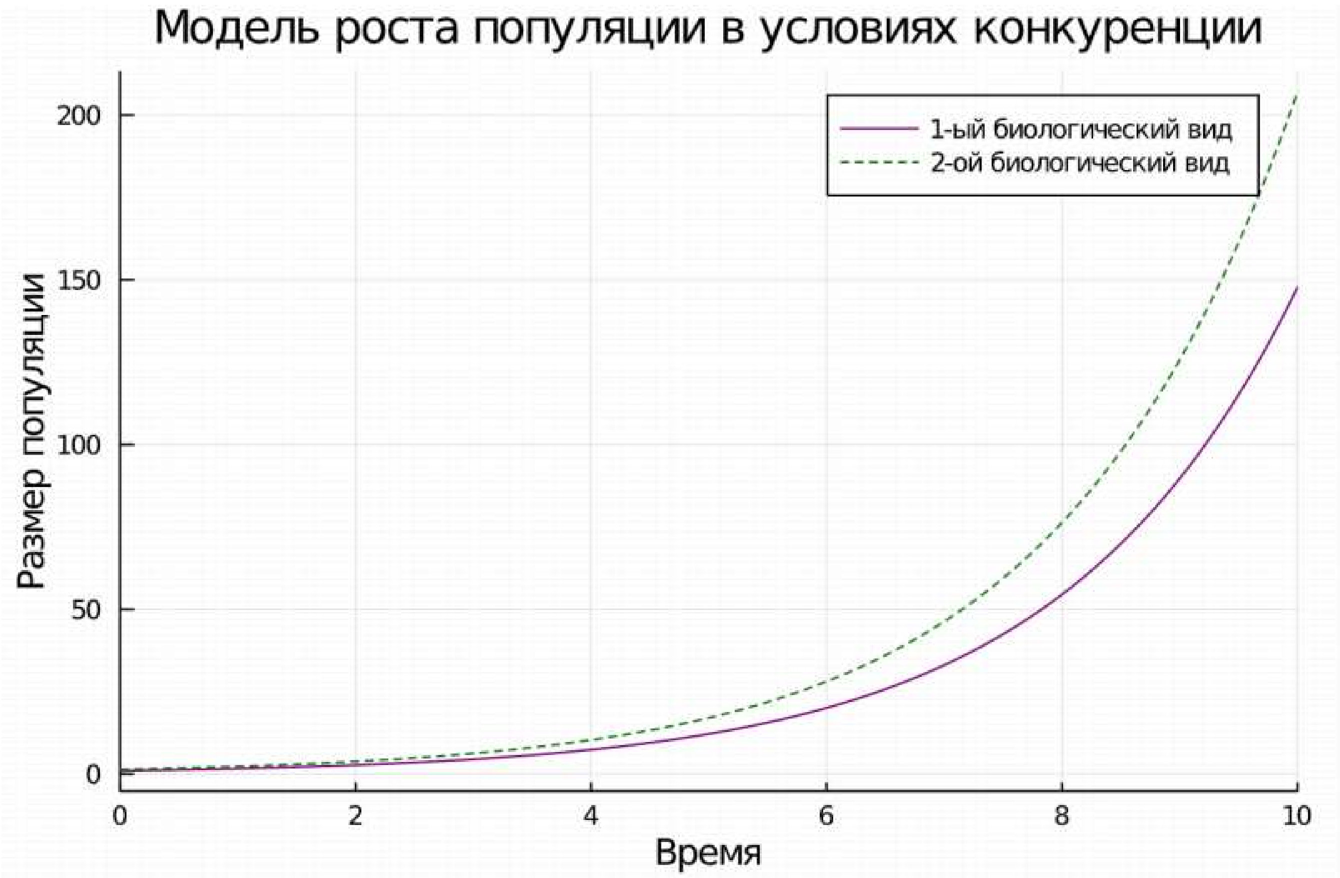

Рисунок 6.4 Код и результат Задания 6-4

В данной модели мы задаем 2 параметра, насколько я понимаю первый отвечает за рост популяции обеих групп, а второй это коэффициент конкурентности. То есть насколько одна группа конкурирует с другой. Также в данной модели я заметила, что сильно влияют начальные значения. То есть количество особей в двух моделируемых группах на начало времени. В моем случае 1-я группа имеет начальное значение 1, а вторая группа - 1.4. Следовательно по моим предположениям, второй биологический вид должен выиграть в данной конкурентной среде и начать быстро расти. вышло так, что второй вид полностью подавил первый биологический вид.

7. Реализовать на языке Julia модель консервативного гармонического осциллятора

```
# 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)
```

```
  2.952699559134839
  ⋮
  5.742943903730911
  6.171923680145046
  6.584585638714117
  7.010469575802823
  7.433067310856991
  7.849769652855734
  8.282275467285212
  8.684259561485183
  9.126171129123518
  9.524168428874143
  9.970146725929531
 10.0
u: 31-element Array{Array{Float64,1},1}:
 [1.0, 1.0]
 [1.0640413705392677, 0.6864865930281919]
 [1.1166550813709244, 0.11102078370062098]
 [1.0853087396797187, -0.5370470078797774]
```

Рисунок 7.1 Код и результат Задания 7-1

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

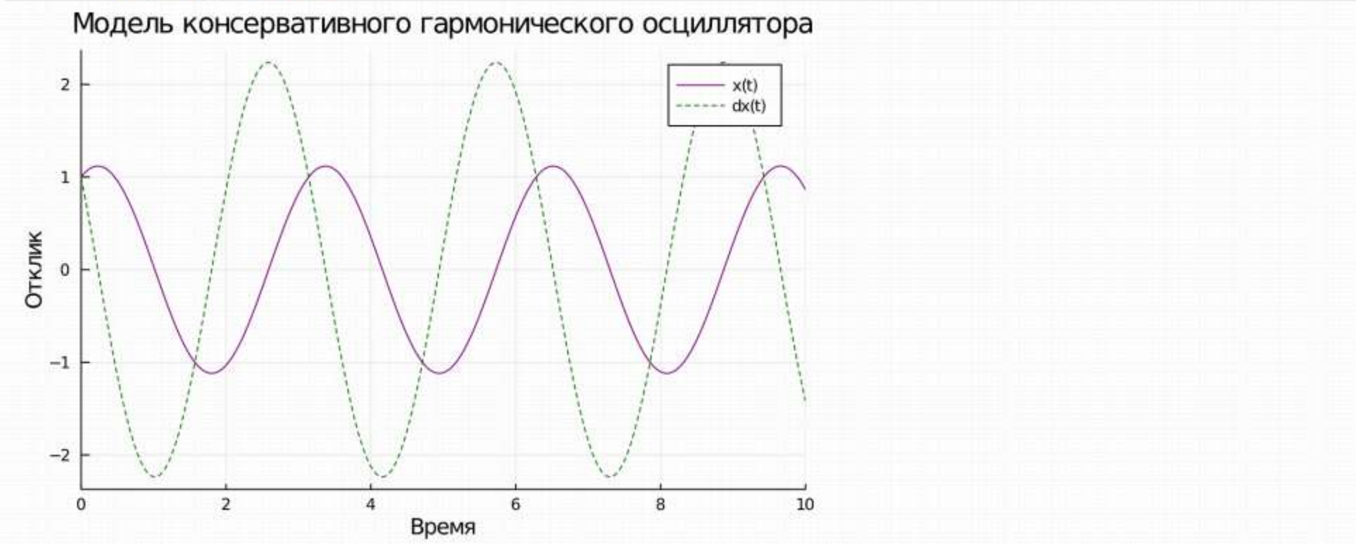

Рисунок 7.2 Код и результат Задания 7-2

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

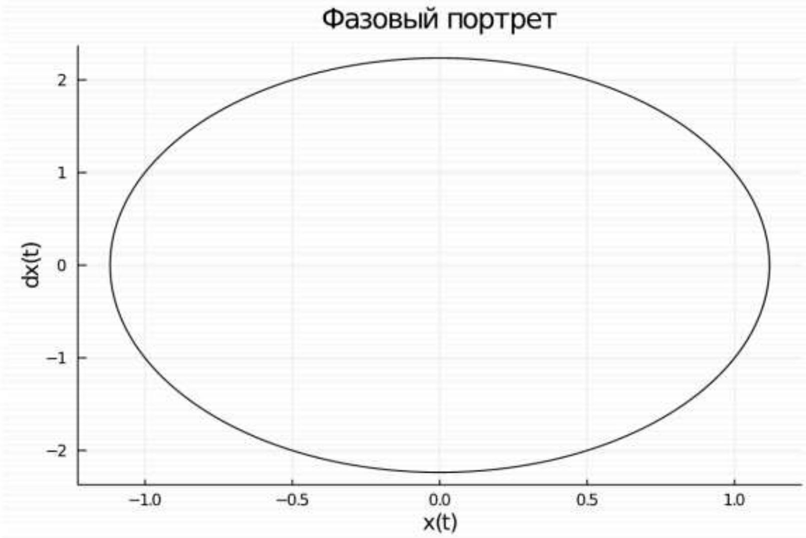

Рисунок 7.3 Код и результат Задания 7-3

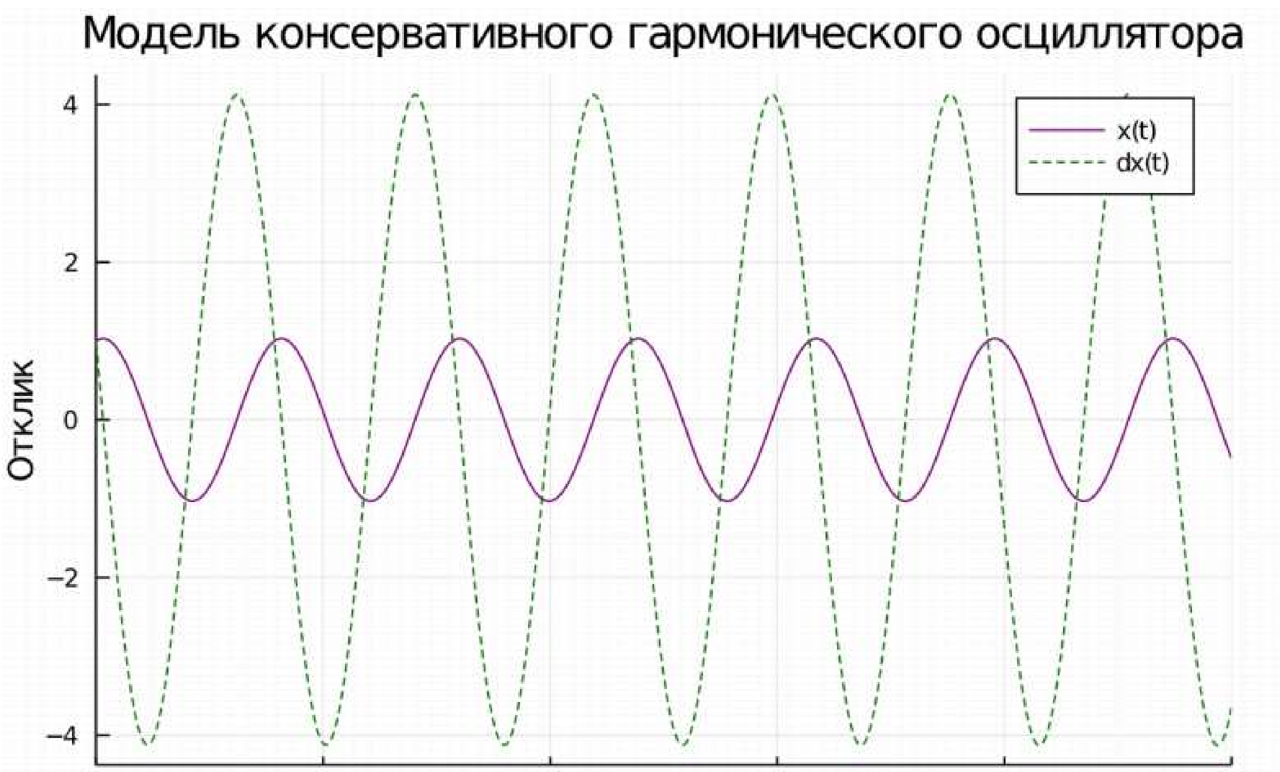

Рисунок 7.4 Код и результат Задания 7-4

1.Начальные условия.
Я задал начальное смещение x(0)=1.0 и начальную скорость dot{x}(0)=1.0. Это позволяет наблюдать «свободные» колебания без дополнительного затухания или внешнего воздействия.

2.Численное решение. Я оформил задачу как `ODEProblem(lv!, u0, tspan, p)` и решил её с помощью `solve(prob)`. Получил решение в виде временных рядов для x(t) и dot{x}(t).

3.Построение графиков и анимации. - Построил график x(t) и dot{x}(t) от времени, чтобы увидеть колебания и убедиться, что они действительно гармонические.

- Построил фазовый портрет (ось абсцисс — x, ось ординат — dot{x}, где траектория представляет собой замкнутую эллиптическую кривую, характерную для консервативного осциллятора.
- Создал анимацию, чтобы наглядно показать, как решения меняются во времени, и сохранил её в виде GIF-файла.

В итоге удалось продемонстрировать, что в отсутствии диссипации (затухания) система совершает периодические колебания.

8. Реализовать на языке Julia модель свободных колебаний гармонического осциллятора ¶

```
# 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)
```

```
  3.036321896353611
  ⋮
  5.7371152023935705
  6.172863569625527
  6.563858142109736
  6.984119749940885
  7.373364961737517
  7.815552860164412
  8.212080053058667
  8.639452678126846
  9.029650751134675
  9.479283489991738
  9.877714129664202
 10.0
u: 31-element Array{Array{Float64,1},1}:
 [0.5, 1.0]
 [0.566294838471426, 0.7739308462880993]
 [0.6437437851223823, 0.36377664879947777]
 [0.665622626864566, -0.08049770770465431]
```

Рисунок 8.1 Код и результат Задания 8-1

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

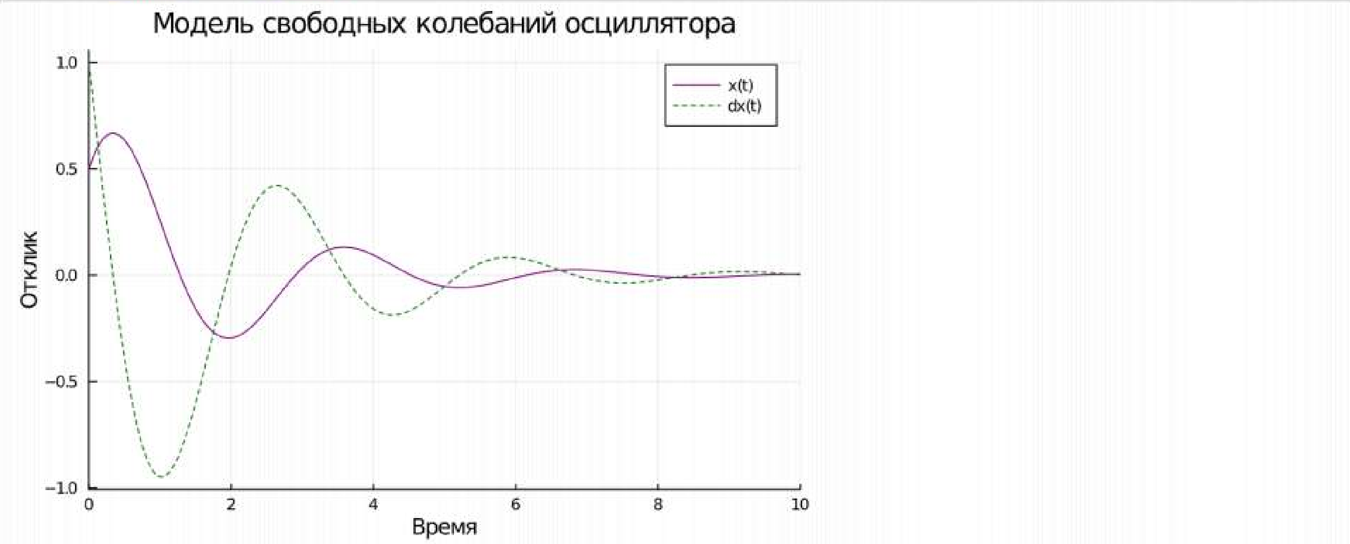

Рисунок 8.2 Код и результат Задания 8-2

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

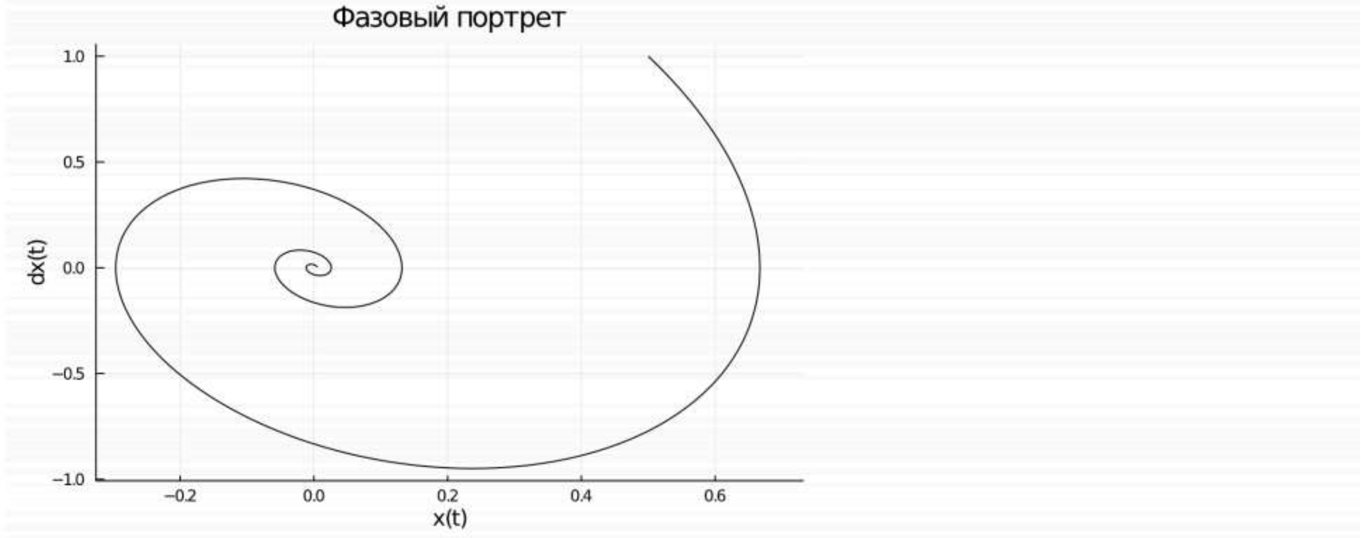

Рисунок 8.3 Код и результат Задания 8-3

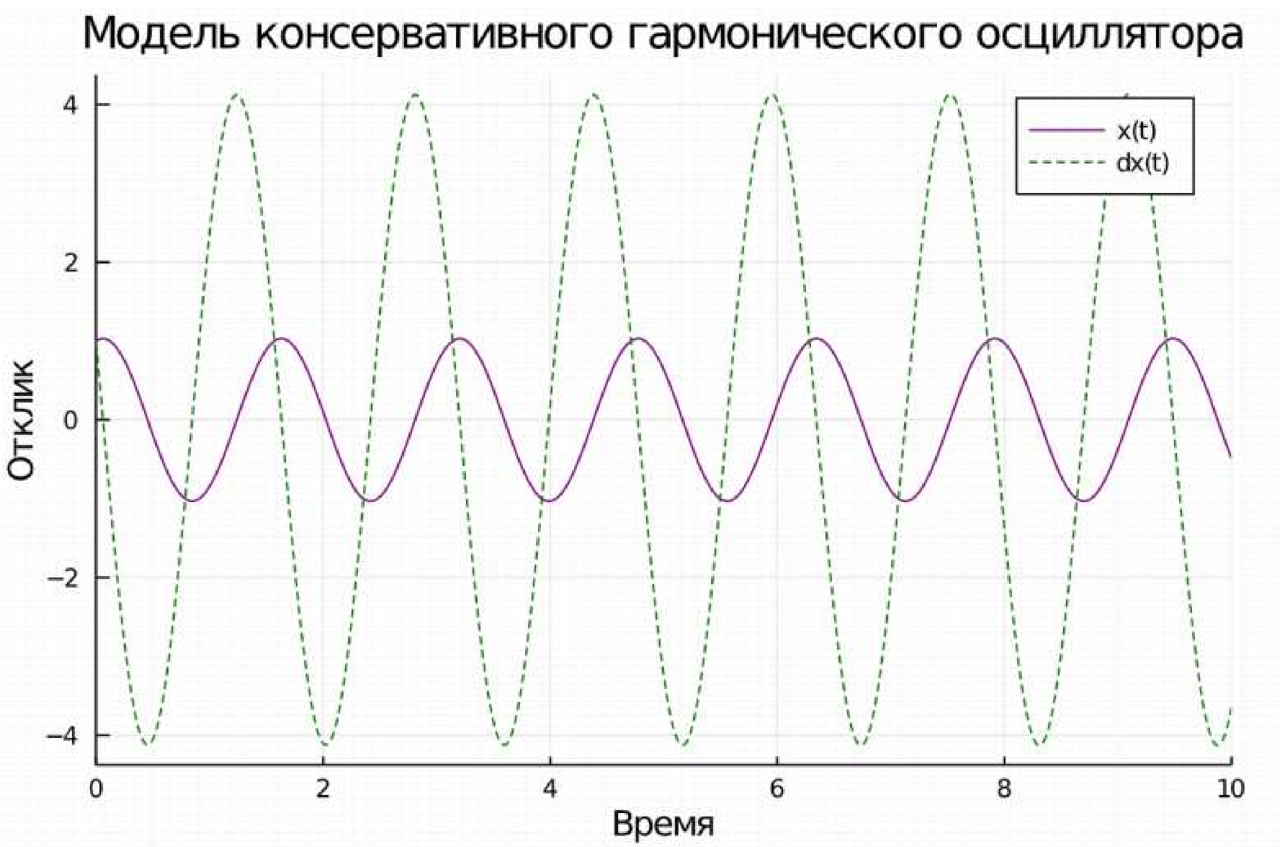

Рисунок 8.4 Код и результат Задания 8-4

1.Выбор параметров и начальных условий

- Параметр gamma = 0.5 задал для иллюстрации умеренного затухания; omega_0 = 2.0 — для определённой «скорости» колебаний. - Начальное смещение x(0) = 0.5 и начальная скорость dot{x}(0) = 1.0 выбраны, чтобы наблюдать свободные затухающие колебания.

2.Численное решение

- Сформировал задачу `ODEProblem` (из пакета `DifferentialEquations.jl`), передал описание системы, вектор начальных условий и параметры.
- Функция `solve(prob)` вернула решение, по которому можно строить графики.

3.Построение графиков и анимации - Построил во времени функции x(t) и dot{x}(t), чтобы увидеть колебания и их затухание.
- На фазовом портрете ось x, ось dot{x} затухающие колебания визуально проявляются в виде спиральной траектории, «закручивающейся» к равновесию.
- С помощью `animate` создал GIF-файл, на котором видно, как решение эволюционирует по времени.

Таким образом, получилась модель свободных (но затухающих) колебаний гармонического осциллятора, позволяющая наглядно проследить динамику как в координате времени, так и на фазовом портрете.

Вывод

Я освоил специализированные пакеты для решения задач в непрерывном и дискретном времени