**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

# ОТЧЕТ

# ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

*дисциплина: Компьютерный практикум*

*по математическому моделированию*

Студент: Абрамян Артём

Группа: НПИбд-01-20

**МОСКВА**

2023 г.

**Постановка задачи**

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

**Выполнение работы**

1. **Используя Jupyter Lab, повторите примеры из раздела 6.2.**

## 1.1 Решение обыкновенных дифференциальных уравнений

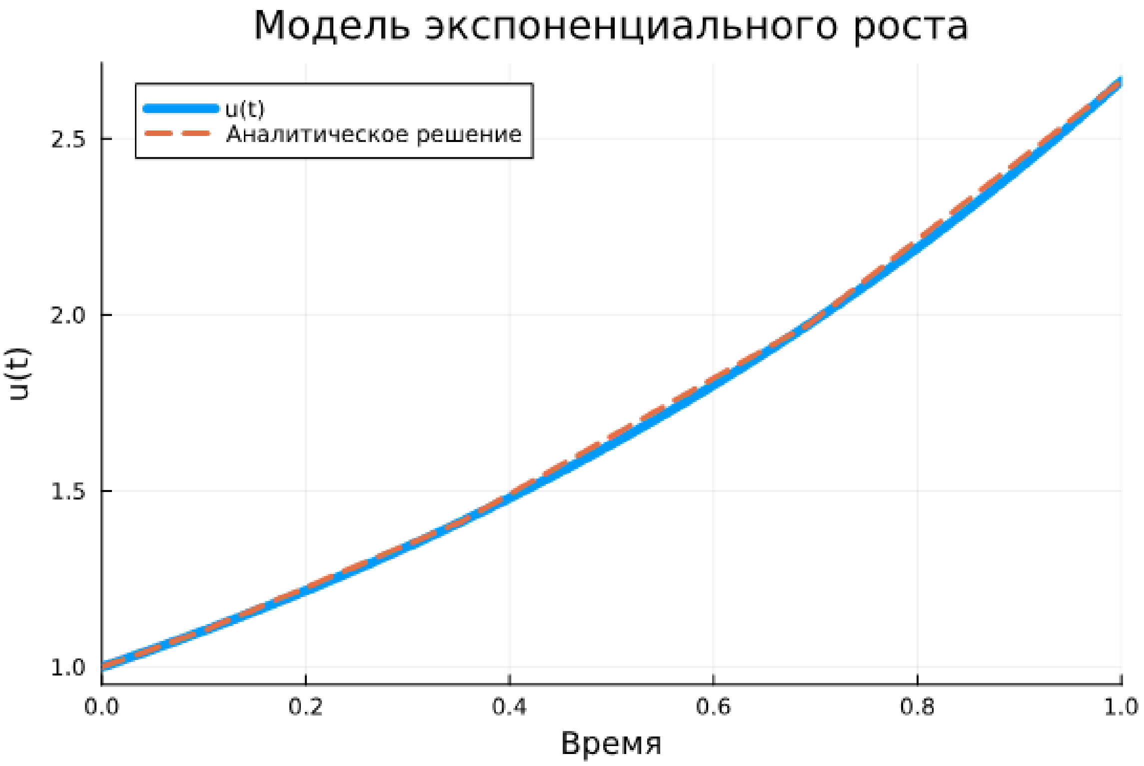
Напомним, что обыкновенное дифференциальное уравнение (ОДУ) описывает изменение некоторой переменной $u$: $u\ '\ (t) = f(u(t), p, t)$, где $f(u(t), p, t)$ — нелинейная модель (функция) изменения $u(t)$ с заданным начальным значением $u(t0\ ) = u0\ $, $p$ — параметры модели,$t$ — время. Для решения обыкновенных дифференциальных уравнений (ОДУ) в Julia можно использовать пакет diffrentialEquations.jl.

## Модель экспоненциального роста

Рассмотрим пример использования этого пакета для решение уравнения модели экспоненциального роста, описываемую уравнением $u\ '\ (t) = au(t)$, $u(0) = u0$. где $a$ — коэффициент роста. Предположим, что заданы следующие начальные данные $a$ = 0, 98, $u(0)$ = 1, 0, $t \in$ [0; 1, 0]. Аналитическое решение модели имеет вид: $u(t) = u0$ exp$(at)u(t)$. Численное решение в Julia будет иметь следующий вид:

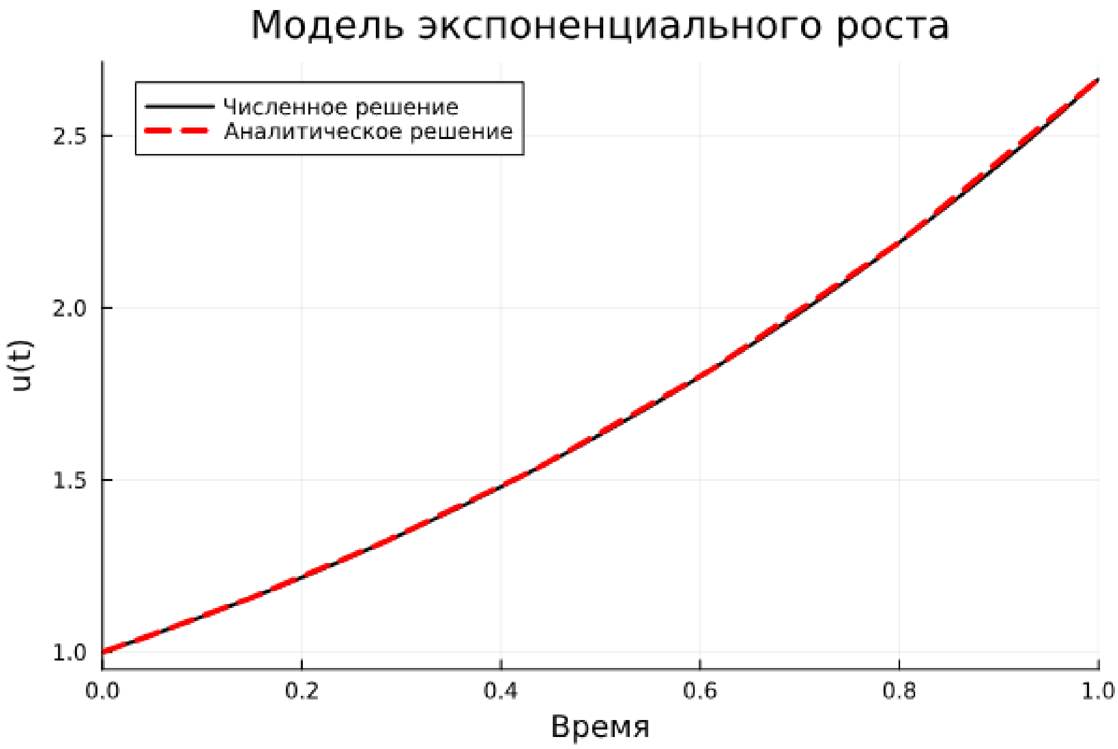
| | | |
|---|---|---|
| 2 | 0.100425 | 1.10342 |
| 3 | 0.352186 | 1.41219 |
| 4 | 0.693444 | 1.97304 |
| 5 | 1.0 | 2.66446 |

```
begin
    # 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)
end
```




```
begin
    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")
end
```

При построении одного из графиков использовался вызов sol.t, чтобы захватить массив моментов времени. Массив решений можно получить, воспользовавшись sol.u. Если требуется задать точность решения, то можно воспользоваться параметрами abstol (задаёт близость к нулю) и reltol (задаёт относительную точность). По умолчанию эти параметры имеют значение abstol = 1e-6 и reltol = 1e-3. Для модели экспоненциального роста:




```
begin
    # 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")
end
```

## 1.2 Система Лоренца

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

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

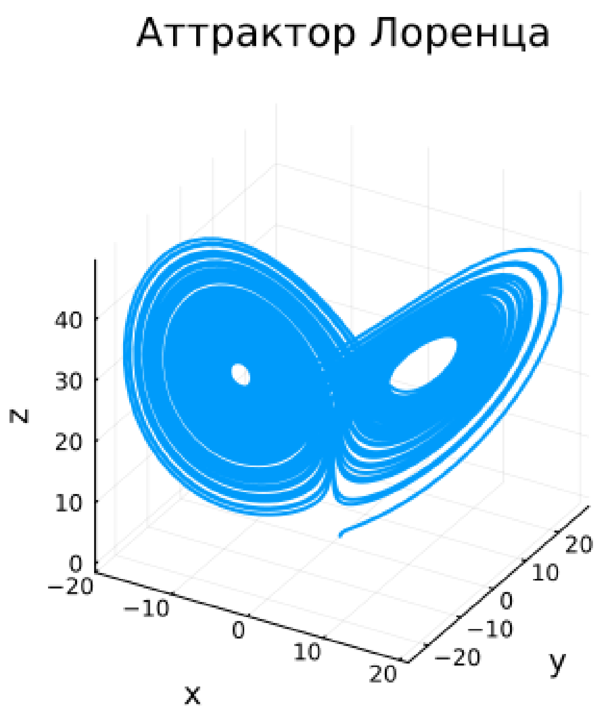
где $\sigma$, $\rho$ и $\beta$ — параметры системы (некоторые положительные числа, обычно указывают $\sigma = 10$, $\rho = 28$ и $\beta = 8/3$). Система (6.2) получена из системы уравнений Навье–Стокса и описывает движение воздушных потоков в плоском слое жидкости постоянной толщины при разложении скорости течения и температуры в двойные ряды Фурье с последующем усечением до первых-вторых гармоник. Решение системы неустойчиво на аттракторе, что не позволяет применять классические численные методы на больших отрезках времени, требуется использовать высокоточные вычисления. Численное решение в Julia будет иметь следующий вид:

```
lorenz! (generic function with 1 method)
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
```

| | timestamp | value1 | value2 | value3 |
|---|---|---|---|---|
| 1 | 0.0 | 1.0 | 0.0 | 0.0 |
| 2 | 3.56786e-5 | 0.999643 | 0.000998805 | 1.78143e-8 |
| 3 | 0.000392465 | 0.996105 | 0.0109654 | 2.14696e-6 |
| 4 | 0.00326241 | 0.969359 | 0.0897706 | 0.000143802 |
| 5 | 0.00905808 | 0.924204 | 0.242289 | 0.00104616 |
| 6 | 0.0169565 | 0.880046 | 0.438736 | 0.00342426 |
| 7 | 0.02769 | 0.848331 | 0.691563 | 0.00848762 |
| 8 | 0.0418564 | 0.849504 | 1.01454 | 0.0182121 |
| 9 | 0.0602404 | 0.913907 | 1.44256 | 0.0366938 |
| 10 | 0.0836854 | 1.08886 | 2.05233 | 0.0740257 |

⋮ more

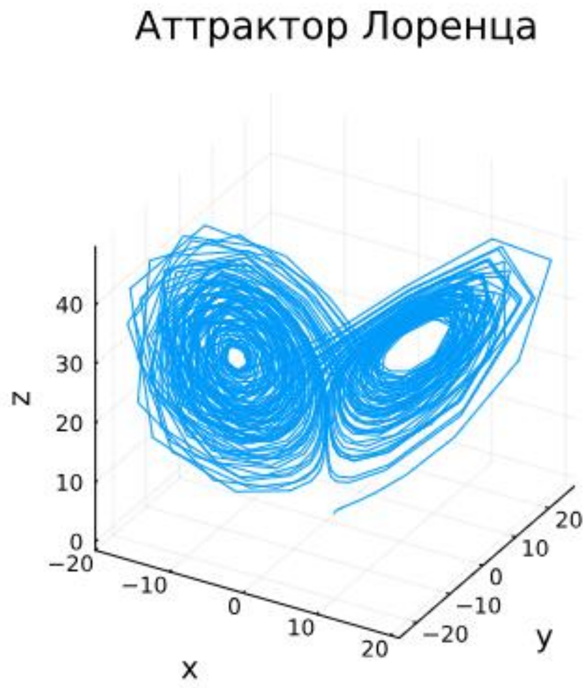
```
begin
    # задаём начальное условие:
    u0 = [1.0,0.0,0.0]
    # задаём значения параметров:
    p = (10,28,8/3)
    # задаём интервал времени:
    tspan = (0.0,100.0)
    # решение:
    prob = ODEProblem(lorenz!,u0,tspan,p)
    sol = solve(prob)
end
```




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

```
⚠ To maintain consistency with solution indexing, keyword argument vars will be rem
oved in a future version. Please use keyword argument idxs instead.
caller: ip:0x0
```

Можно отключить интерполяцию:




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

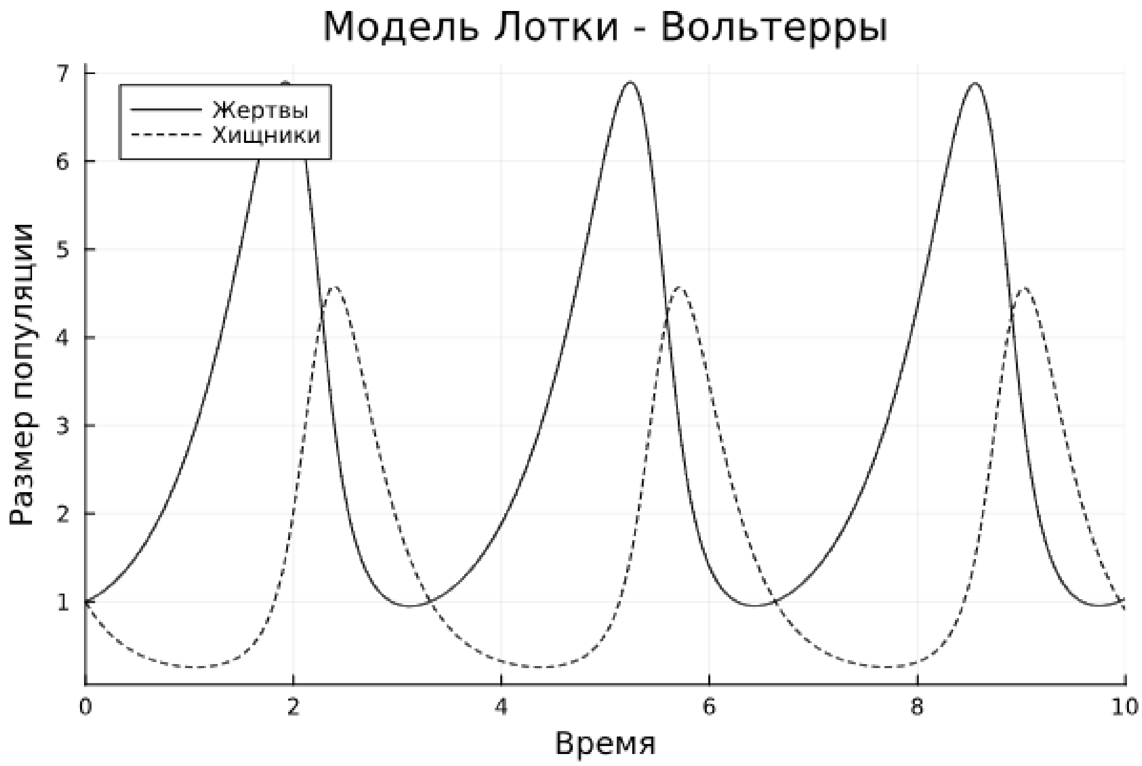
```
⚠ To maintain consistency with solution indexing, keyword argument vars will be rem
oved in a future version. Please use keyword argument idxs instead.
caller: ip:0x0
```

### 1.3 Модель Лотки–Вольтерры

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

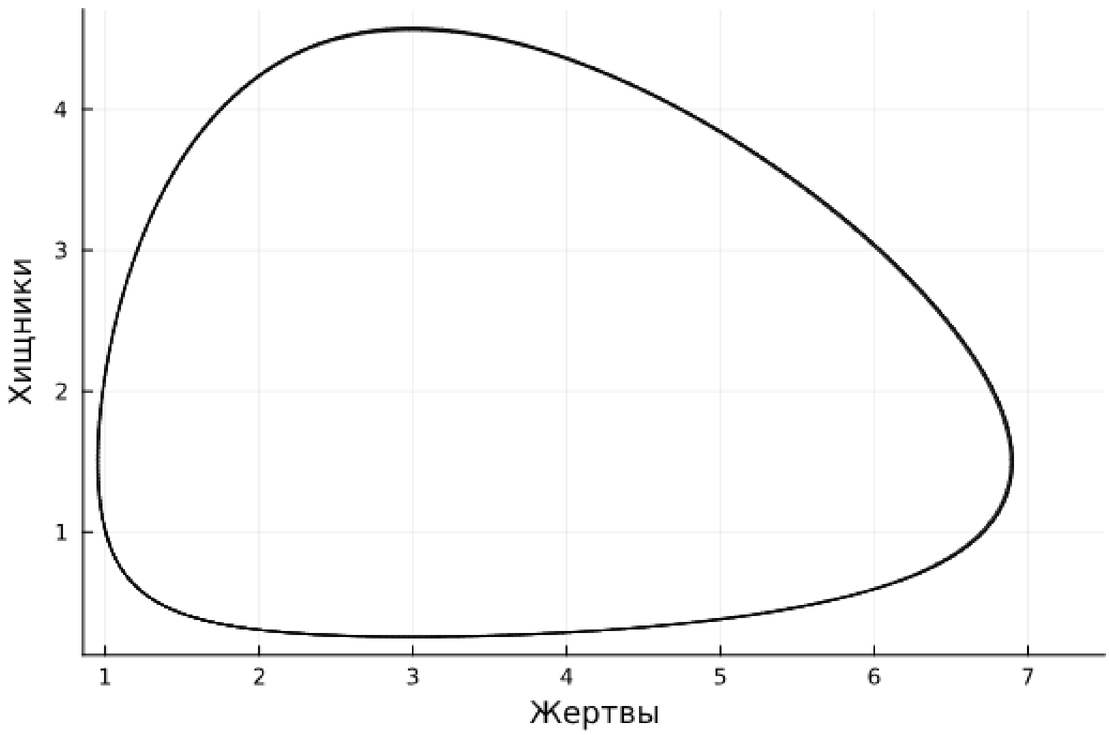
$$\begin{cases} \dot{x} = (\alpha - \beta y)x, \\ \dot{y} = (-\gamma + \delta x)y, \end{cases}$$

где $x$ — количество жертв, $y$ — количество хищников,$t$ — время, $\alpha$, $\beta$, $\gamma$, $\delta$ — коэффициенты, отражающие взаимодействия между видами (в данном случае $\alpha$ — коэффициент рождаемости жертв, $\gamma$ — коэффициент убыли хищников, $\beta$ — коэффициент убыли жертв в результате взаимодействия с хищниками, $\delta$ — коэффициент роста численности хищников). Численное решение в Julia будет иметь следующий вид:




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

Фазовый портрет:




```julia
plot(sol,vars=(1,2), color="black", xaxis="Жертвы",yaxis="Хищники", legend=false)
```

```
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
```

## 2. Задания для самостоятельной работы

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

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

где $x(t)$ — численность изолированной популяции в момент времени $t$, $a$ — коэффициент роста популяции, $b$ — коэффициент рождаемости, $c$ — коэффициент смертности.

Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией).

```
begin
    lv! = @ode_def Malthus begin
    dx = a*x
    end a
    # задаём начальное условие:
    u0 = [2]
    # задаём значения параметров:
    b = 4.0
    c = 1.0
    p = (b - c)
    # задаём интервал времени:
    tspan = (0.0,3.0)
    # решение:
    prob = ODEProblem(lv!,u0,tspan,p)
    sol = solve(prob)
end
```

65.4 μs

Я взял большой коэффициент рождаемости и относительно низкий кэффициент смертности. Вследствии чего ожидаю быстрый рост графика.

```
begin
    lv! = @ode_def Malthus begin
    dx = a*x
    end a
    # задаём начальное условие:
    u0 = [2]
    # задаём значения параметров:
    b = 4.0
    c = 1.0
    p = (b - c)
    # задаём интервал времени:
    tspan = (0.0,3.0)
    # решение:
    prob = ODEProblem(lv!,u0,tspan,p)
    sol = solve(prob)
end
```


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

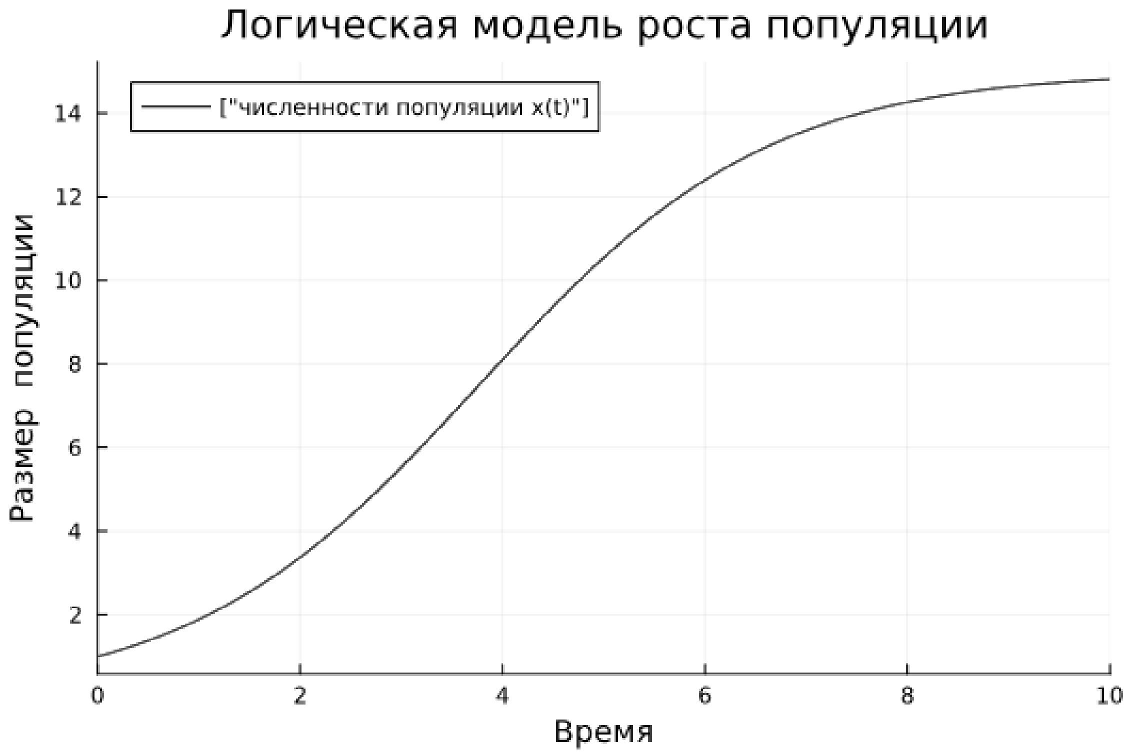
График с анимацией:


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

2. Реализовать и проанализировать логистическую модель роста популяции, заданную уравнением: $\dot{x} = rx\left(1 - \frac{x}{k}\right)$, $r > 0$, $k > 0$, $r$ — коэффициент роста популяции, $k$ — потенциальная ёмкость экологической системы (предельное значение численности популяции). Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией).

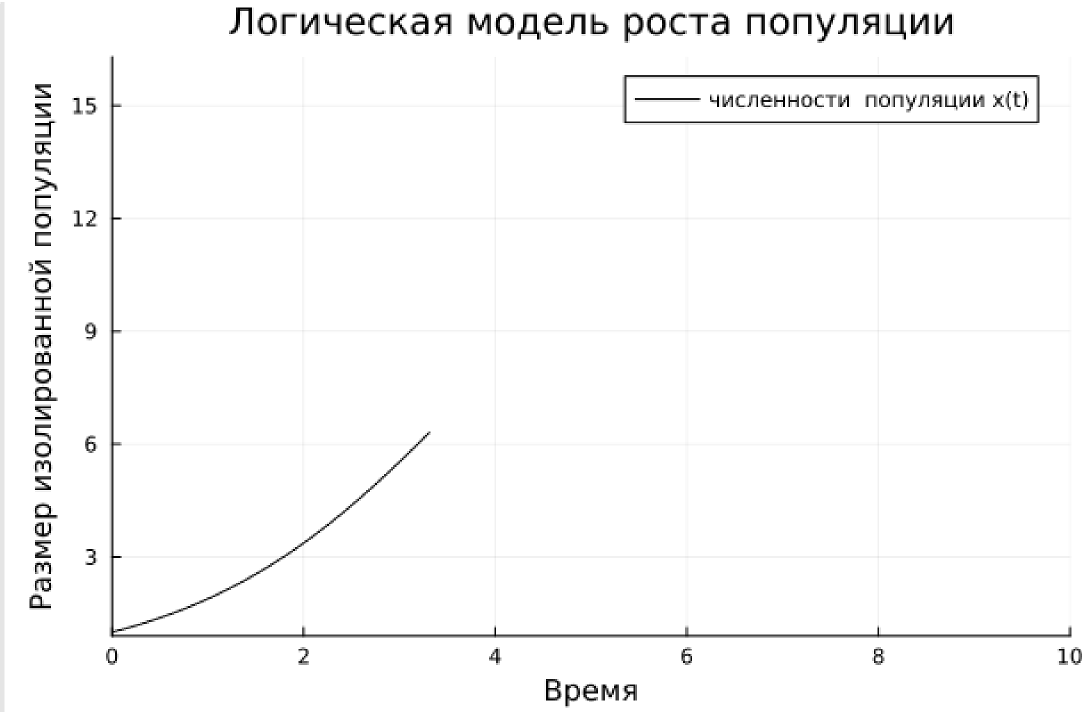
Я задал коэффициент роста r = 0.7 и предельное значение популяции k = 15. График на интервале от 0 до 10 должен достичь предельного значения 15.

```
begin
    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 = (0.7, 15)
    # задаём интервал времени:
    tspan = (0.0,10.0)
    # решение:
    prob = ODEProblem(lv!,u0,tspan,p)
    sol = solve(prob)
end
```




```
plot(sol, label = ["численности популяции x(t)"], color="black", ls=[:solid :dash],
title="Логическая модель роста популяции", xaxis="Время",yaxis="Размер  популяции")
```

График с анимацией:




```
animate(sol, fps=7, "Logistic.gif", label = "численности  популяции x(t)",
color="black", ls=[:solid :dash], title="Логическая модель роста популяции",
xaxis="Время",yaxis="Размер изолированной популяции")
```

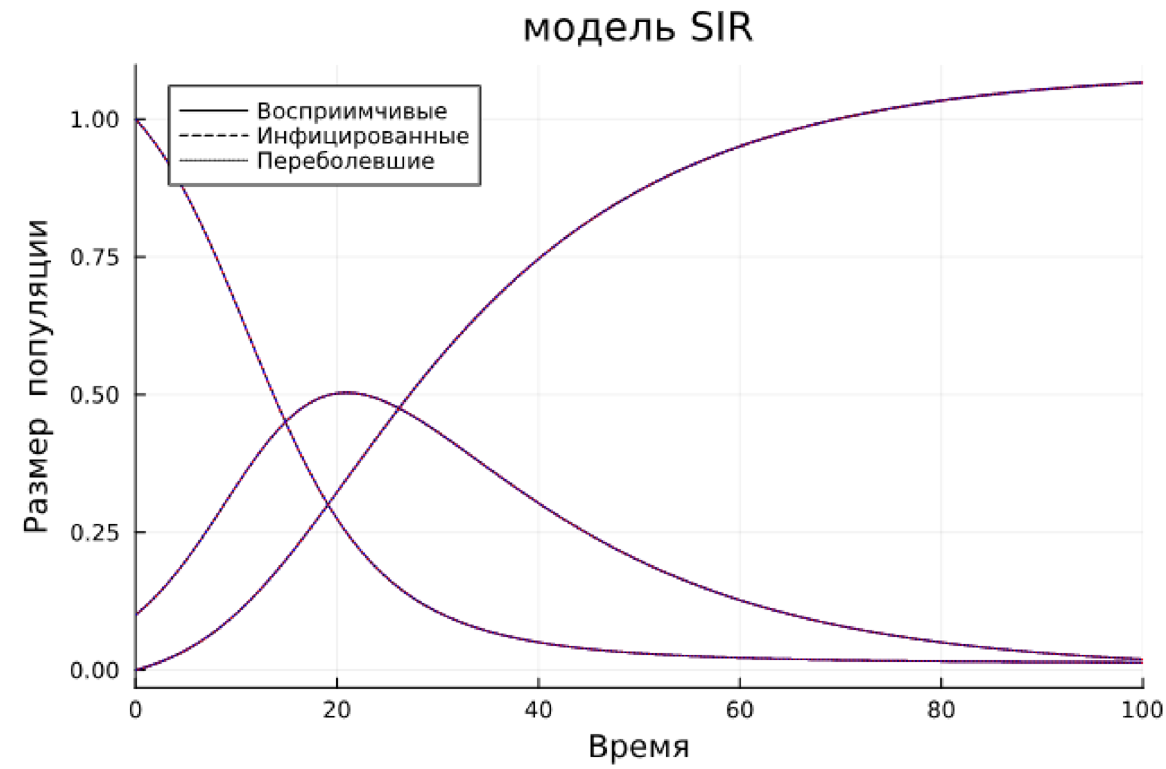
3. Реализовать и проанализировать модель эпидемии Кермака–Маккендрика (SIRмодель):

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

где $s(t)$ — численность восприимчивых к болезни индивидов в момент времени $t$, $i(t)$ — численность инфицированных индивидов в момент времени $t$, $r(t)$ — численность переболевших индивидов в момент времени $t$, $\beta$ — коэффициент интенсивности контактов индивидов с последующим инфицированием, $\nu$ — коэффициент интенсивности выздоровления инфицированных индивидов. Численность популяции считается постоянной, т.е. $\dot{s} + \dot{i} + \dot{r} = 0$. Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией).

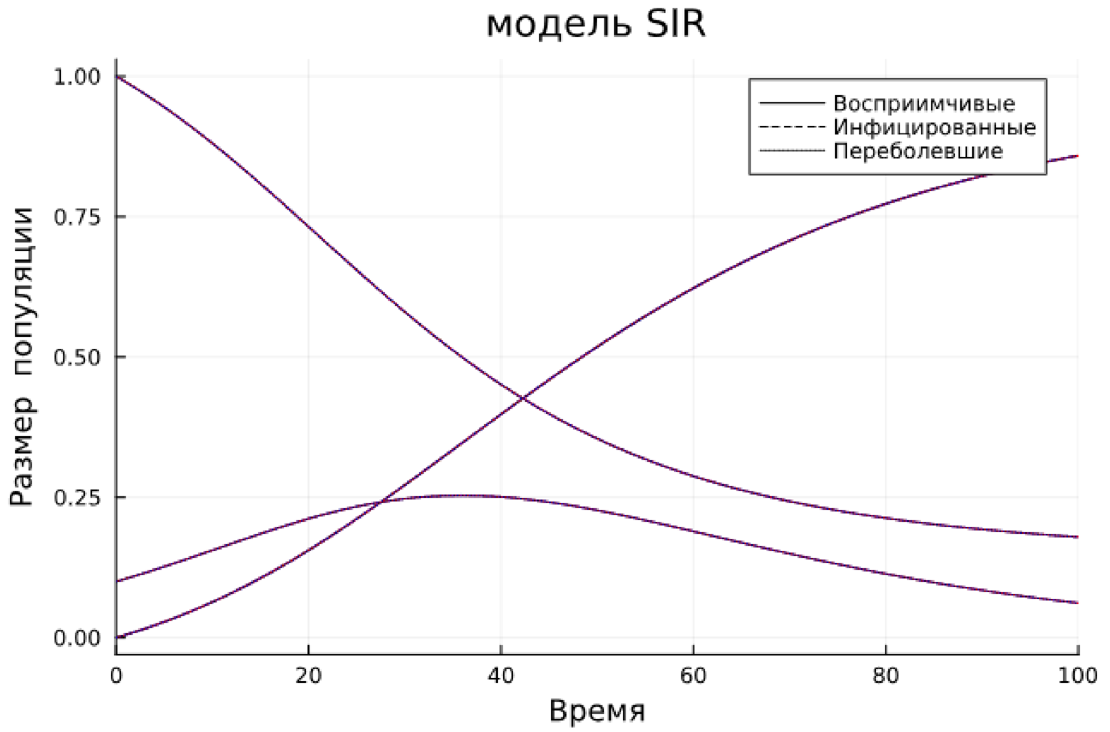
Я задал $\beta$ — коэффициент интенсивности контактов индивидов с последующим инфицированием = 0.2, $\nu$ — коэффициент интенсивности выздоровления инфицированных индивидов = 0.05. Коэффициент интенсивности выздоровлениям мал, а коэффициент интенсивности контактов индивидов с последующим инфицированием наоборот относительно большой.

```
begin
    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 зналчения параметров:
    p = (0.2, 0.05)
    # задаём интервал времени:
    tspan = (0.0,100.0)
    # решение:
    prob = ODEProblem(lv!,u0,tspan,p)
    sol = solve(prob)
end
```




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

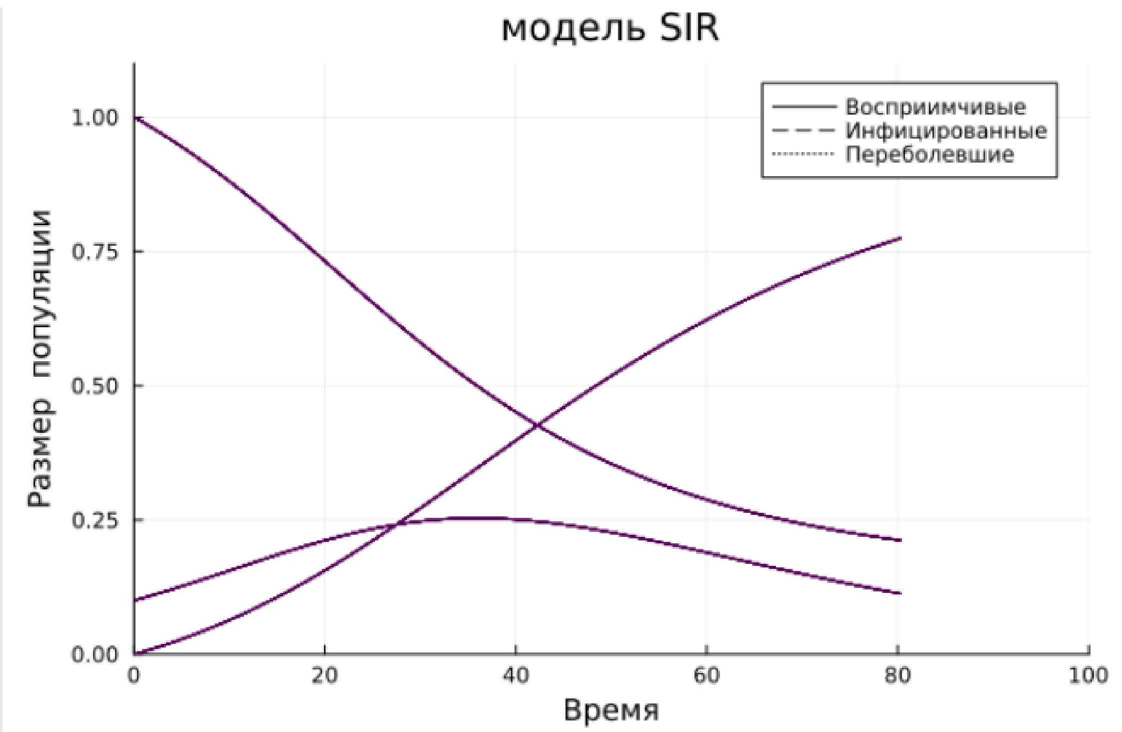
Синий график демонстрирует количество инфицированных. Попробуем уменьшить коэффициен интенсивности до 0.1.




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

Легко заметить, что график роста числа инфецированных растёт медленнее.

График с анимацией:




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

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

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

Размер популяции сохраняется:

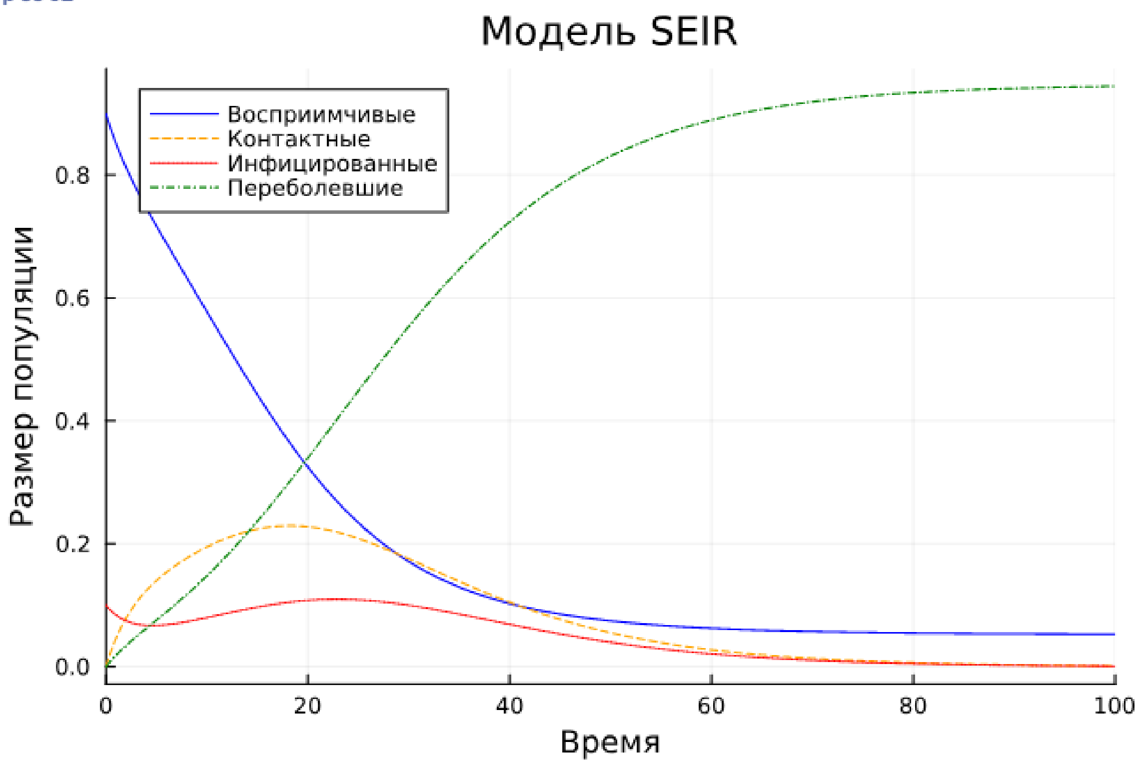
$$s(t) + e(t) + i(t) + r(t) = N.$$

Исследуйте, сравните с SIR.

```
begin

    # 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)
end
```

коэффициенты: β = 0.6, γ = 0.2, δ = 0.1. Коэффициент δ - величина, обратная среднему инкубационному периоду заболевания.

```
plot1 =
```



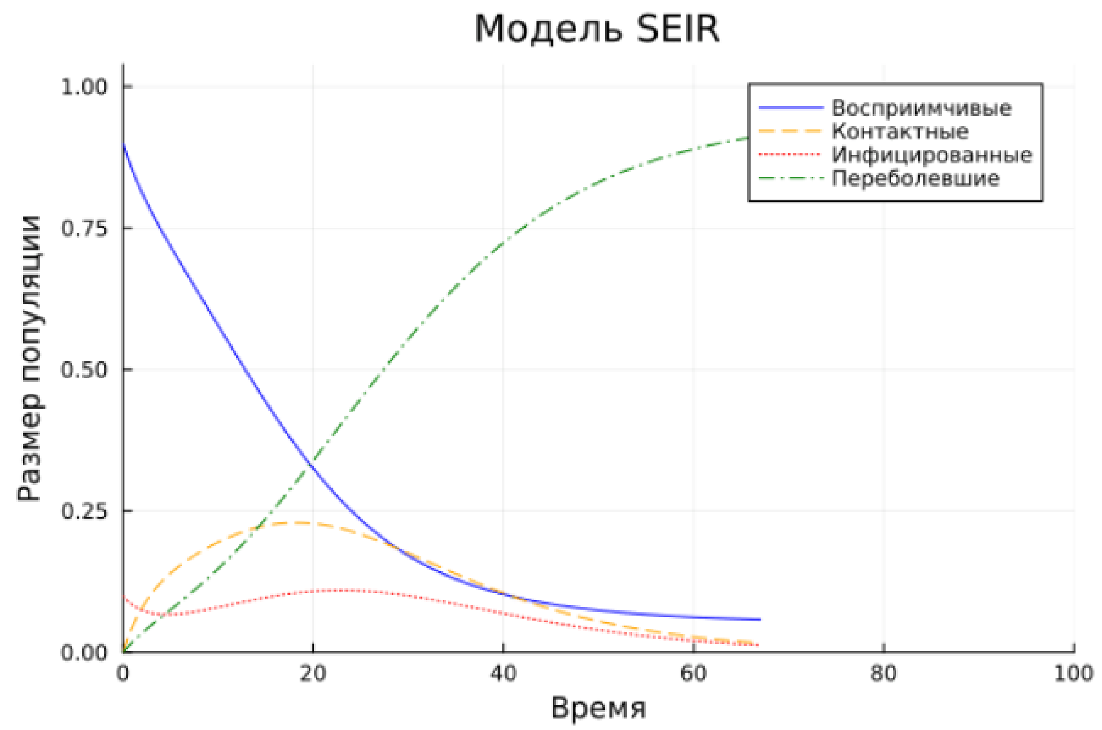

```
plot1 = plot(sol, label = ["Восприимчивые" "Контактные" "Инфицированные"
"Переболевшие"],
    color=["blue" "orange" "red" "green"], ls=[:solid :dash :dot :dashdot],
    title="Модель SEIR",
    xaxis="Время",yaxis="Размер популяции")
```

Различия между моделями SIR и SEIR:

В данных моделях есть 1

различие, касаемое того, что модель SEIR учитывает латентное состояние человека во время болезни. Это такое состояние при котором внутри человека уже есть вирус, однако пока что он не передает его другим индивидуумам. Можно описать это уравне- ние так: оно вносит задержку по времени при переходе из состояния контактного в состояние инфицированного (больного). Это происходит через время, равное инкубационному периоду болезни.

График с анимацией:



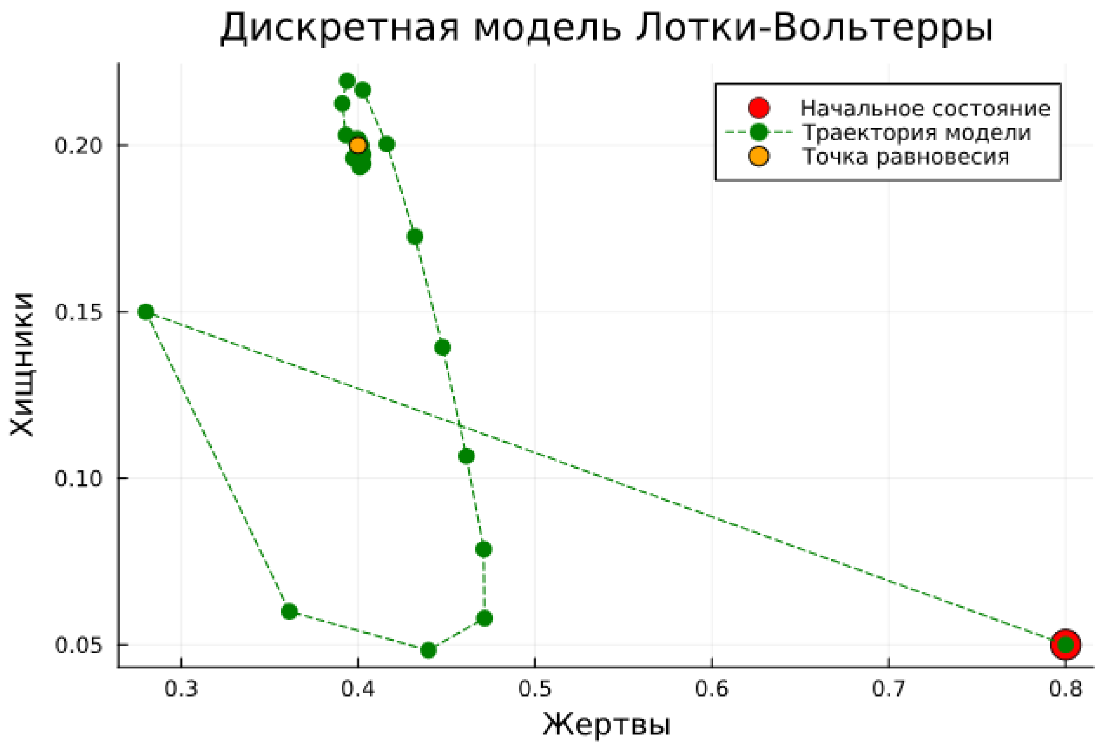

```
animate(sol, fps=7, "SEIR.gif", label = ["Восприимчивые" "Контактные"
"Инфицированные" "Переболевшие"],
    color=["blue" "orange" "red" "green"], ls=[:solid :dash :dot :dashdot],
    title="Модель SEIR",
    xaxis="Время",yaxis="Размер популяции")
```

5. Для дискретной модели Лотки–Вольтерры:

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

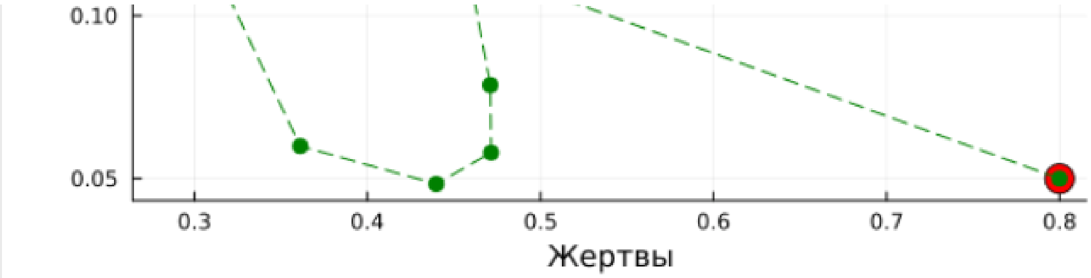
с начальными данными $a = 2$, $c = 1$, $d = 5$ найдите точку равновесия. Получите и сравните аналитическое и численное решения. Численное решение изобразите на фазовом портрете.

```
begin
    # 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
end
```




```
begin
    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")
end
```

График с анимацией:




```julia
begin
    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)
end
```

```
ⓘ Saved animation to D:\2023-2024\Практикум\stat-analys\lab6\LotkaVolterra.gif
```

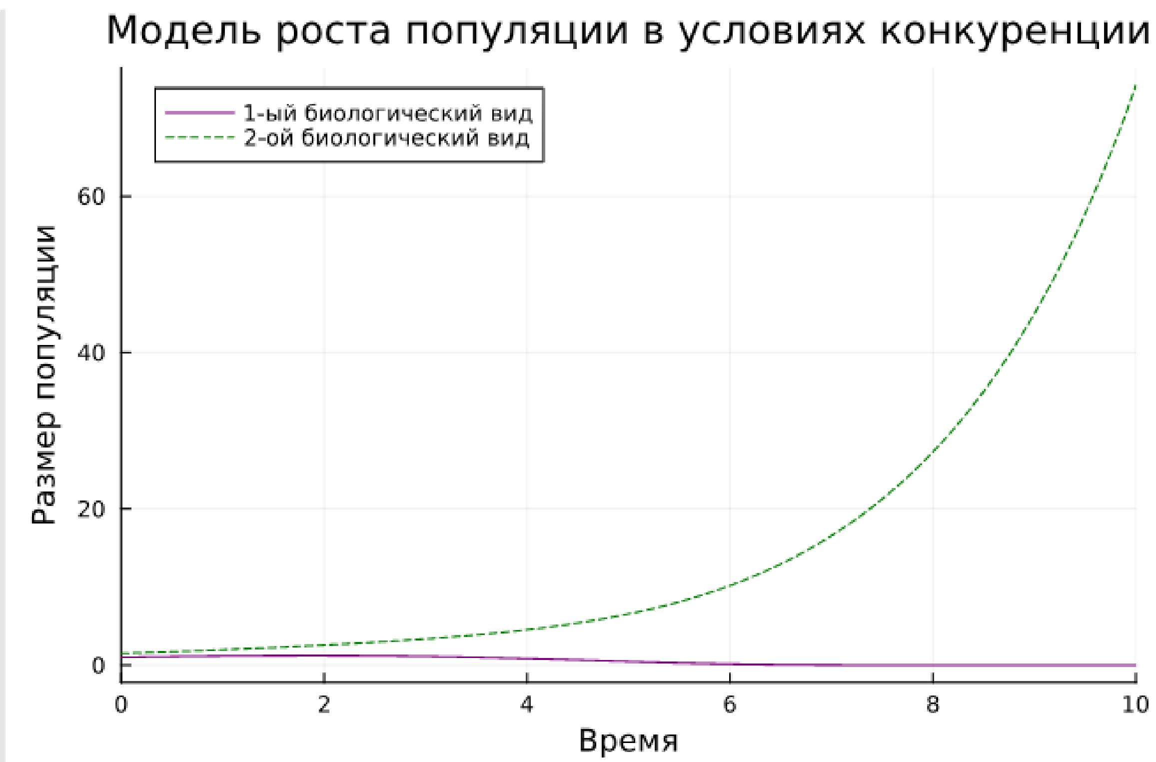
6. Реализовать на языке Julia модель отбора на основе конкурентных отношений:

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

Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет.

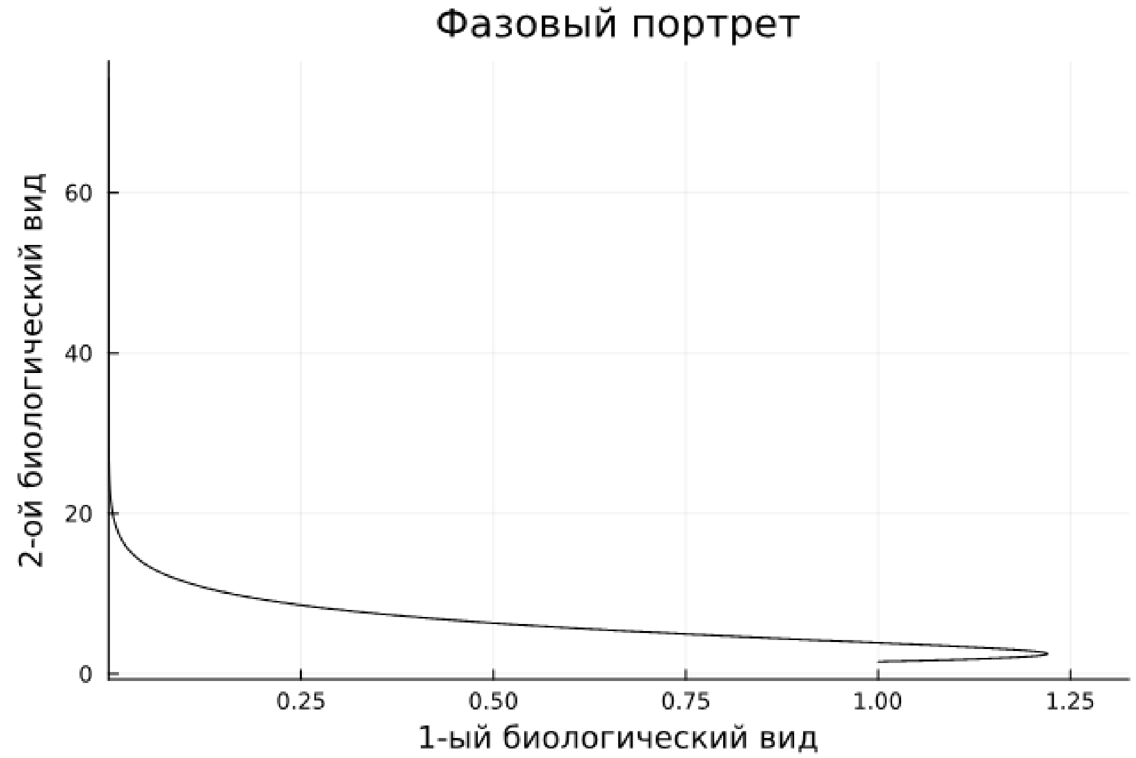
```
# задаём описание модели:
lv! = @ode_def CompetitiveSelectionModel begin
dx = a*x - b*x*y
dy = a*y - b*x*y
    end a b

# 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)
```

В данной модели мы задаем 2 параметра: первый отвечает за рост популяции обеих групп, а второй - коэффициент конкурентности. У первой группы значение 1, у второй группы - 1.5. Следовательно 2-ой биологический вид должен выиграть в данной конкурентной среде.




```
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")
```

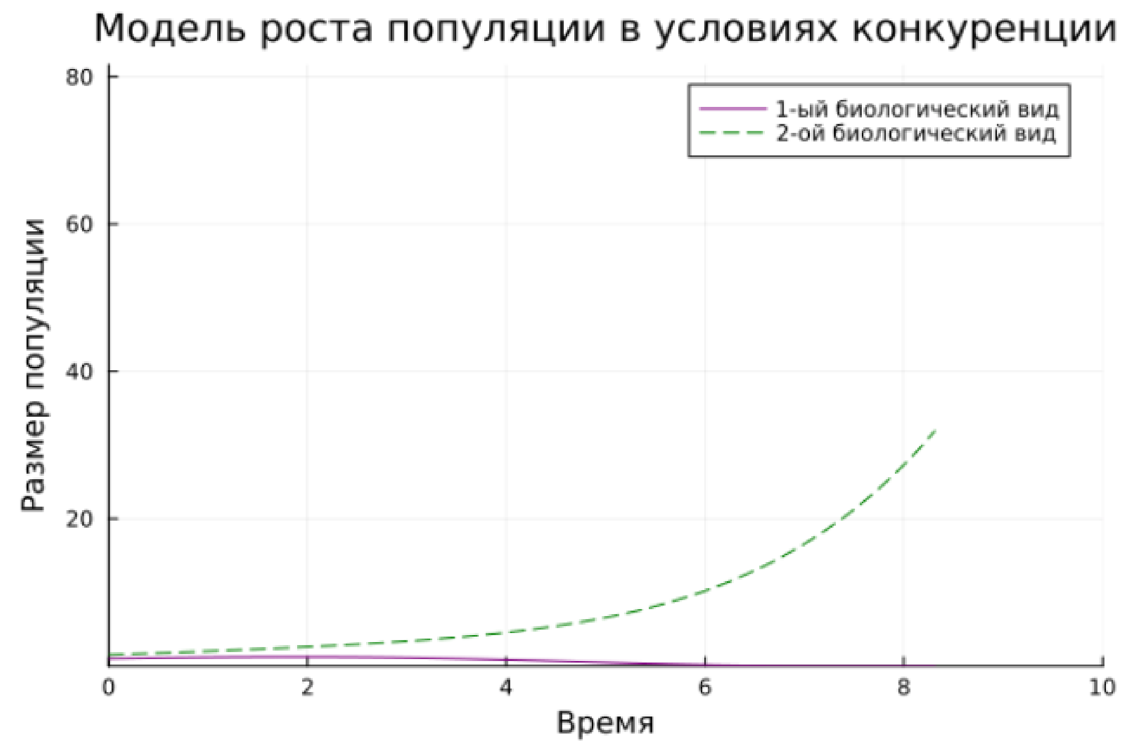
Фазовый портрет с начальными параметрами:




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

```
To maintain consistency with solution indexing, keyword argument vars will be rem
oved in a future version. Please use keyword argument idxs instead.
caller: ip:0x0
```

График с анимацией:




```
animate(sol, fps=7, "Competitive.gif", 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")
```

7. Реализовать на языке Julia модель консервативного гармонического осциллятора

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

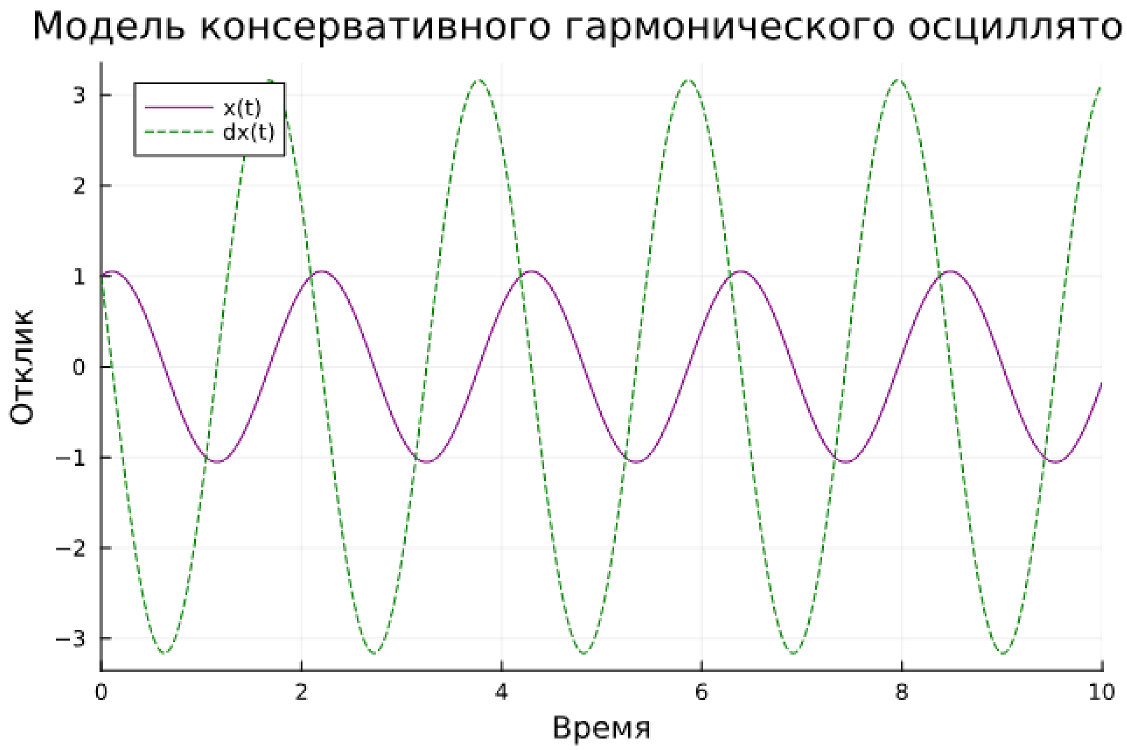
где $\omega 0$ — циклическая частота. Начальные параметры подобрать самостоятельно, выбор пояснить. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет.

Я задал в параметрах только циклическую частоту и чем

больше она будет, тем чаще будут колебания консервативного гармонического

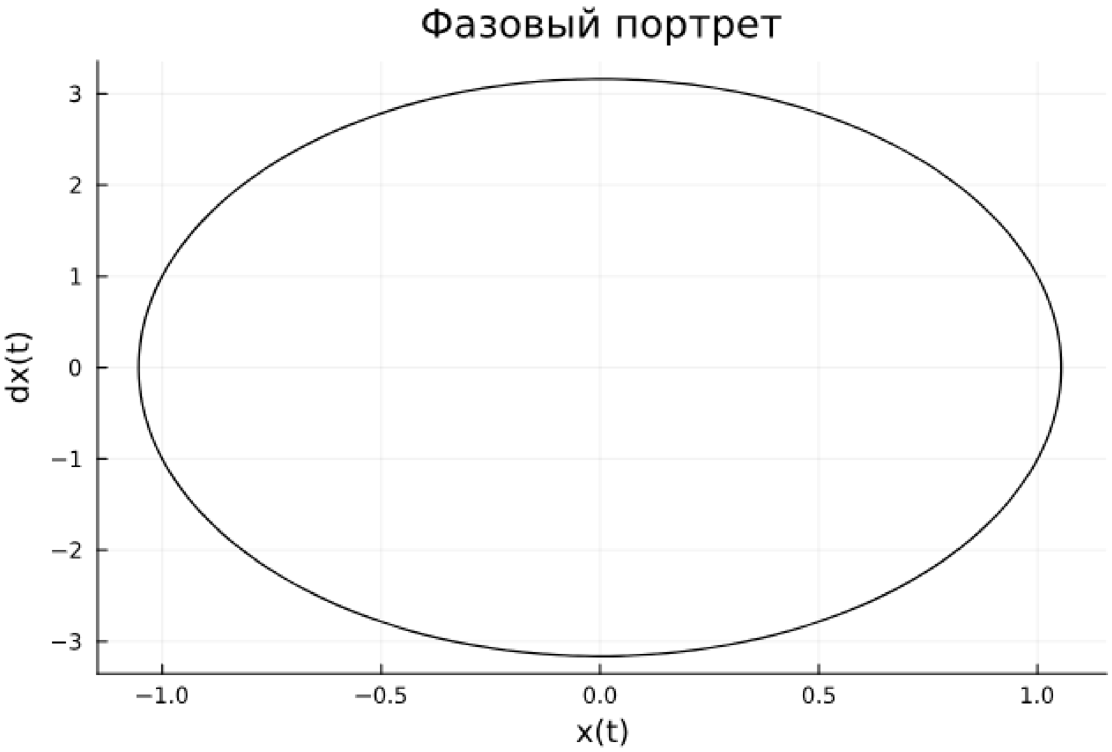
осциллятора. Например, при $\omega 0 = 3$

```
begin
    # 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 = (3.0)
    # задаём интервал времени:
    tspan = (0.0, 10.0)

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




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

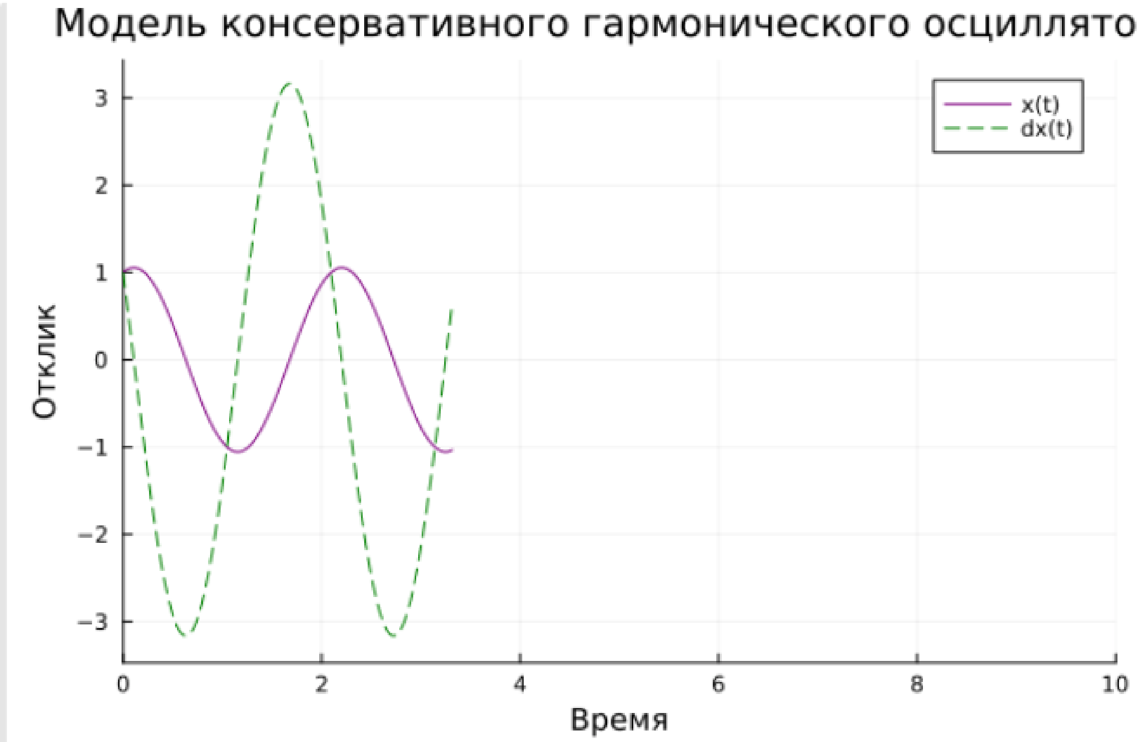
Фазовый портрет:




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

```
To maintain consistency with solution indexing, keyword argument vars will be rem
oved in a future version. Please use keyword argument idxs instead.
caller: ip:0x0
```

График с анимацией:




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

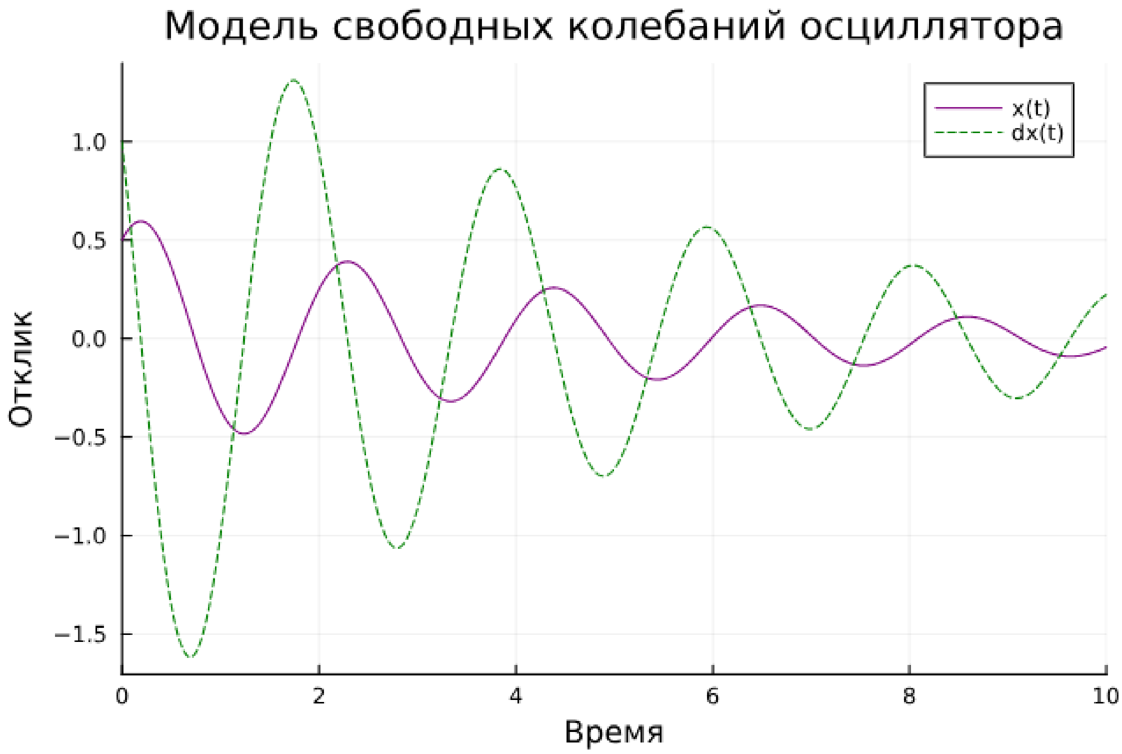
8. Реализовать на языке Julia модель свободных колебаний гармонического осциллятора $\ddot{x} + 2\gamma\dot{x} + \omega_0^2 x = 0$, $x(t_0) = x_0$, $\dot{x}(t_0) = y_0$, где $\omega_0$ — циклическая частота, $\gamma$ — параметр, характеризующий потери энергии. Начальные параметры подобрать самостоятельно, выбор пояснить. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет.

Я задал циклическую частоту равную 3.0, а параметр, характеризующий потери энергии

0.2.

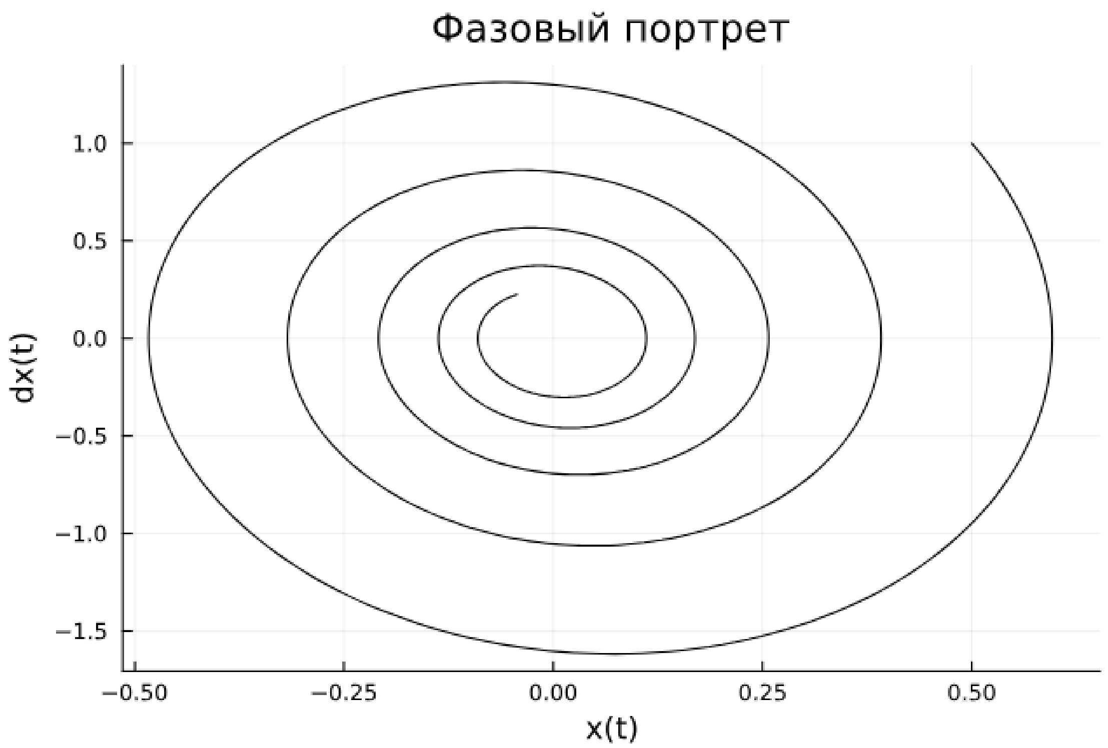
Чем меньше этот коэффициент, тем медленнее будут происходить затухания гармонического осциллятора.

```
begin
    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 = (0.2, 3.0)
    # задаём интервал времени:
    tspan = (0.0, 10.0)

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




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

Фазовый портрет:

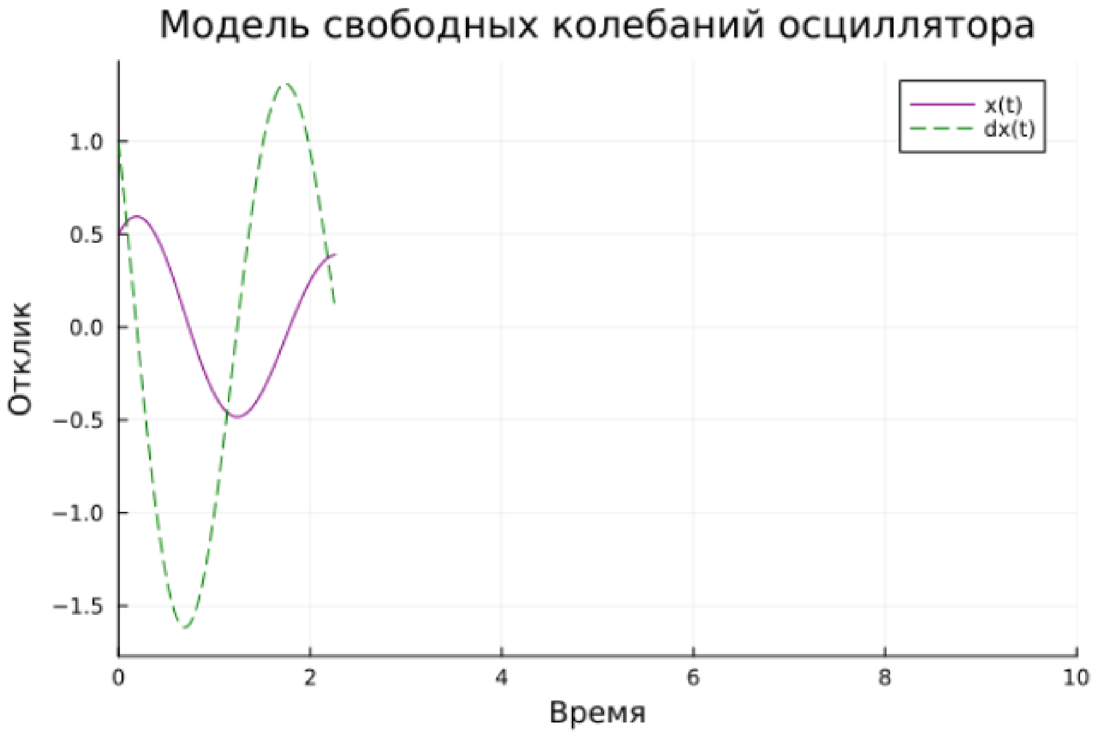



```
plot(sol, vars=(1,2), color="black",  title="Фазовый портрет", xaxis="x(t)",
yaxis="dx(t)", legend=false)
```

```
To maintain consistency with solution indexing, keyword argument vars will be rem
oved in a future version. Please use keyword argument idxs instead.
caller: ip:0x0
```

532 ms

График с анимацией:




```
animate(sol, fps=7, "oscillator_2.gif", label = ["x(t)" "dx(t)"], color=["purple"
"green"], ls=[:solid :dash],
    title="Модель свободных колебаний осциллятора",
    xaxis="Время",yaxis="Отклик")
```

Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной outer_v.

```
outer_v = 5×5 Matrix{Int64}:
           1   2   3   4   5
           2   4   6   8  10
           3   6   9  12  15
           4   8  12  16  20
           5  10  15  20  25
```

```
outer_v = v * v'
```

## 2.1 Системы линейных уравнений

Решить СЛАУ с двумя неизвестными.

a) $\begin{cases} x + y = 2, \\ x - y = 3. \end{cases}$

b) $\begin{cases} x + y = 2, \\ 2x + 2y = 4. \end{cases}$

c) $\begin{cases} x + y = 2, \\ 2x + 2y = 5. \end{cases}$

d) $\begin{cases} x + y = 1, \\ 2x + 2y = 2, \\ 3x + 3y = 3. \end{cases}$

e) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ x - y = 3. \end{cases}$

f) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ 3x + 2y = 3. \end{cases}$

```
▸ [2.5, -0.5]
```

```
begin
    K = [1 1; 1 -1]
    answers = [2; 3]
    K\answers
end
```

```
LinearAlgebra.SingularException(2)

1. checknonsingular @ factorization.jl:19 [inlined]
2. checknonsingular @ factorization.jl:21 [inlined]
3. #lu!#170 @ lu.jl:82 [inlined]
4. #lu#177 @ lu.jl:279 [inlined]
5. lu @ lu.jl:278 [inlined]
6. \(::Matrix{Int64}, ::Vector{Int64}) @ generic.jl:1110
7. top-level scope @ Local: 4 [inlined]
```

```
begin
    K2 = [1 1; 2 2]
    answers2 = [2; 4]
    K2\answers
end
# решений бесконечно
```

```
LinearAlgebra.SingularException(2)

 1. checknonsingular @ factorization.jl:19 [inlined]
 2. checknonsingular @ factorization.jl:21 [inlined]
 3. #lu!#170 @ lu.jl:82 [inlined]
 4. #lu#177 @ lu.jl:279 [inlined]
 5. lu @ lu.jl:278 [inlined]
 6. \(::Matrix{Int64}, ::Vector{Int64}) @ generic.jl:1110
 7. top-level scope @ Local: 4 [inlined]
```

```julia
begin
    K3 = [1 1; 2 2]
    answers3 = [2; 5]
    K3\answers3
end
# пустое множество решений
```

```
[0.5, 0.5]
```

```julia
begin
    K4 = [1 1; 2 2; 3 3]
    answers4 = [1;2;3]
    K4\answers4
end
```

```
[1.5, -1.0]
```

```julia
begin
    K5 = [1 1; 2 1; 1 -1]
    answers5 = [2;1;3]
    K5\answers5
end
# полученные значения неверны
```

99.2 μs

99.2 μs

```
[-1.0, 3.0]
```

```julia
begin
    K6 = [1 1; 2 1; 3 2]
    answers6 = [2;1;3]
    K6\answers6
end
```

Решить СЛАУ с тремя неизвестными

a) $\begin{cases} x+y+z=2, \\ x-y-2z=3. \end{cases}$

b) $\begin{cases} x+y+z=2, \\ 2x+2y-3z=4, \\ 3x+y+z=1. \end{cases}$

c) $\begin{cases} x+y+z=1, \\ x+y+2z=0, \\ 2x+2y+3z=1. \end{cases}$

d) $\begin{cases} x+y+z=1, \\ x+y+2z=0, \\ 2x+2y+3z=0. \end{cases}$

```
[2.21429, 0.357143, -0.571429]
```

```
begin
    K7 = [1 1 1; 1 -1 -2]
    answers7 = [2;3]
    K7\answers7
end
# решения неверны, так как необходимо минимум 3 уравнения для решения СУ с 3 неизвестными
```

123 μs

123 μs

```
[-0.5, 2.5, 0.0]
```

```
begin
    K8 = [1 1 1; 2 2 -3;3 1 1]
    answers8 = [2;4;1]
    K8\answers8
end
# решение верное
```

44.1 μs

44.1 μs

```
LinearAlgebra.SingularException(2)

1. checknonsingular @ factorization.jl:19 [inlined]
2. checknonsingular @ factorization.jl:21 [inlined]
3. #lu!#170 @ lu.jl:82 [inlined]
4. #lu#177 @ lu.jl:279 [inlined]
5. lu @ lu.jl:278 [inlined]
6. \(::Matrix{Int64}, ::Vector{Int64}) @ generic.jl:1110
7. top-level scope @ Local: 4 [inlined]
```

```
begin
    K9 = [1 1 1;1 1 2; 2 2 3]
    answers9 = [1;0;1]
    K9\answers9
end
# пустое множество решений
```

---

```
LinearAlgebra.SingularException(2)

 1. checknonsingular @ factorization.jl:19 [inlined]
 2. checknonsingular @ factorization.jl:21 [inlined]
 3. #lu!#170 @ lu.jl:82 [inlined]
 4. #lu#177 @ lu.jl:279 [inlined]
 5. lu @ lu.jl:278 [inlined]
 6. \(::Matrix{Int64}, ::Vector{Int64}) @ generic.jl:1110
 7. top-level scope @ Local: 4 [inlined]
```

```julia
begin
    K10 = [1 1 1;1 1 2; 2 2 3]
    answers10 = [1;0;0]
    K10\answers10
end
# пустое множество решений
```

### 2.2 Операции с матрицами

Приведите приведённые ниже матрицы к диагональному виду

a) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$

b) $\begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}$

c) $\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix}$

```
2×2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
```

```
begin
    matrix1 = [1 -2; -2 1]
    Asym = matrix1 + matrix1'
    AsymEig = eigen(Asym)
    inv(AsymEig.vectors) * matrix1 * AsymEig.vectors
end
```

```
2×2 Matrix{Float64}:
 -0.236068     3.46945e-16
  4.44089e-16  4.23607
```

```
begin
        matrix = [1 -2; -2 3]
        Asym = matrix + matrix'
        AsymEig = eigen(Asym)
        inv(AsymEig.vectors) * matrix * AsymEig.vectors
    end
```

```
3×3 Matrix{Float64}:
 -2.14134       3.55271e-15  -1.9984e-15
  3.38618e-15   0.515138      1.11022e-16
 -6.66134e-16  -4.44089e-16   3.6262
```

```
begin
        matrix = [1 -2 0; -2 1 2; 0 2 0]
        Asym = matrix + matrix'
        AsymEig = eigen(Asym)
        inv(AsymEig.vectors) * matrix * AsymEig.vectors
    end
```

Вычислите

a) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}^{10}$

b) $\sqrt{\begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}}$

c) $\sqrt[3]{\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}}$

d) $\sqrt{\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}}$

```
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
```

```
([1 -2; -2 1])^10
```

```
2×2 Matrix{Float64}:
  2.1889   -0.45685
 -0.45685   2.1889
```

```
sqrt([5 -2; -2 5])
```

```
2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
  0.971125+0.433013im  -0.471125+0.433013im
 -0.471125+0.433013im   0.971125+0.433013im
```

```
([1 -2; -2 1])^(1/3)
```

```
2×2 Matrix{ComplexF64}:
 0.568864+0.351578im  0.920442-0.217287im
 0.920442-0.217287im   1.48931+0.134291im
```

```
sqrt([1 2; 2 3])
```

## 2.3 Найдите собственные значения матрицы *A*, если

$$A = \begin{pmatrix} 140 & 97 & 74 & 168 & 131 \\ 97 & 106 & 89 & 131 & 36 \\ 74 & 89 & 152 & 144 & 71 \\ 168 & 131 & 144 & 54 & 142 \\ 131 & 36 & 71 & 142 & 36 \end{pmatrix}.$$

```
[-128.493, -55.8878, 42.7522, 87.1611, 542.468]
begin
    A = [140 97 74 168 131
    97 106 89 131 36
    74 89 152 144 71
    168 131 144 54 142
    131 36 71 142 36]
    eigenvalues = eigvals(A)
end
```

Создайте диагональную матрицу из собственных значений матрицы $A$. Создайте нижнедиагональную матрицу из матрица $A$. Оцените эффективность выполняемых операций

```
[-128.493, -55.8878, 42.7522, 87.1611, 542.468]
@btime eigvals(A)
```

```
2.000 μs (10 allocations: 2.59 KiB)
```

2.6 s

```
matrix_new = 5×5 Matrix{Float64}:
              0.0  0.0  0.0  0.0  0.0
              0.0  0.0  0.0  0.0  0.0
              0.0  0.0  0.0  0.0  0.0
              0.0  0.0  0.0  0.0  0.0
              0.0  0.0  0.0  0.0  0.0
matrix_new = zeros(5,5)
```

```
5×5 Matrix{Float64}:
 -128.493     0.0       0.0      0.0       0.0
    0.0     -55.8878    0.0      0.0       0.0
    0.0       0.0      42.7522   0.0       0.0
    0.0       0.0       0.0     87.1611    0.0
    0.0       0.0       0.0      0.0     542.468
begin

        @btime for i in 1:1:5
            matrix_new[i, i] = eigenvalues[i]
        end


    matrix_new
end
```

```
229.695 ns (5 allocations: 80 bytes)
```

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
5×5 Matrix{Float64}:
 1.0        0.0        0.0        0.0       0.0
 0.779762   1.0        0.0        0.0       0.0
 0.440476  -0.47314    1.0        0.0       0.0
 0.833333   0.183929  -0.556312   1.0       0.0
 0.577381  -0.459012  -0.189658   0.897068  1.0
U factor:
5×5 Matrix{Float64}:
 168.0  131.0      144.0       54.0      142.0
   0.0  -66.1488   -41.2857    99.8929   -74.7262
   0.0    0.0       69.0375   167.478    -26.9035
   0.0    0.0        0.0      197.797     11.4442
   0.0    0.0        0.0        0.0      -95.657
```

```
begin
    # Создайте нижнедиагональную матрицу из матрица A.

    Alu = lu(A)
    @btime lu(A)

end
```

```
362.679 ns (3 allocations: 384 bytes)
```

## 2.5 Линейные модели экономики

Линейная модель экономики может быть записана как СЛАУ
$x - Ax = y$,
где элементы матрицы $A$ и столбца $y$ — неотрицательные числа. По своему смыслу в экономике элементы матрицы $A$ и столбцов $x$, $y$ не могут быть отрицательными числами.
1. Матрица $A$ называется продуктивной, если решение $x$ системы при любой неотрицательной правой части $y$ имеет только неотрицательные элементы $xi$. Используя это
определение, проверьте, являются ли матрицы продуктивными.

a) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```
is_productive_matrix (generic function with 1 method)
```

```julia
function is_productive_matrix(matrix, size)
        ans=""
         #единичная матрица
        E = [1 0; 0 1]
        # зададим любые неотрицательные числа
        Y = rand(0:1000, size)
        # По формуле вычислим x - A*x = y
        S = E - matrix
        # найдем значения x
        X = S\Y
        # теперь проверим есть ли среди x отрциательное число
        for i in 1:1:size
            if X[i] < 0
                ans = "Матрица непродуктивная"
                break
            else
                ans = "Матрица продуктивная"
            end
        end
        return ans
    end
```

```julia
begin
    matrix1 = [1 2; 3 4]
    matrix2 = ([1 2; 3 4]) * (1/2)
    matrix3 = ([1 2; 3 4]) * (1/10)
    println(is_productive_matrix(matrix1, 2))
    println(is_productive_matrix(matrix2, 2))
    println(is_productive_matrix(matrix3, 2))
end
```

```
Матрица непродуктивная
Матрица непродуктивная
Матрица продуктивная
```

Критерий продуктивности: матрица $A$ является продуктивной тогда и только тогда, когда все элементы матрица
$(E - A)$−1
являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными.

a) $\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

```
is_productive_matrix_2 (generic function with 1 method)
```

```
function is_productive_matrix_2(matrix, size)
        # единичная матрица
        ans = ""
        E = [1 0; 0 1]
        matrix_new = E - matrix
        inv_matrix_new = inv(matrix_new)
        for i in 1:1:size
            for j in 1:1:size
                if inv_matrix_new[i, j] < 0
                    ans = "Матрица непродуктивная"
                    break
                else
                    ans = "Матрица продуктивная"
                end
            end
        end
        return ans
    end
```

```
begin
    matrix1 = [1 2; 3 1]
    matrix2 = ([1 2; 3 1]) * (1/2)
    matrix3 = ([1 2; 3 1]) * (1/10)

    println(is_productive_matrix_2(matrix1,2))
    println(is_productive_matrix_2(matrix2,2))
    println(is_productive_matrix_2(matrix3,2))
end
```

```
Матрица непродуктивная
Матрица непродуктивная
Матрица продуктивная
```

Спектральный критерий продуктивности: матрица $A$ является продуктивной тогда и только тогда, когда все её собственные значения по модулю меньше 1. Используя этот критерий, проверьте, являются ли матрицы продуктивными.

a) $\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

d) $\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0 & 0.1 & 0.2 \\ 0 & 0.1 & 0.3 \end{pmatrix}$

```
is_productive_matrix_3 (generic function with 1 method)
function is_productive_matrix_3(matrix, size)
        ans=""
        # найдем собственные значения переданной матрицы
        eigenvalues = eigvals(matrix)
        for i in 1:1:size
            if abs(eigenvalues[i]) > 1
                ans = "Матрица непродуктивная"
                break
            else
                ans = "Матрица продуктивная"
            end
        end
        return ans
    end
```

```
begin
    matrix1 = [1 2; 3 1]
    matrix2 = ([1 2; 3 1])*(1/2)
    matrix3 = ([1 2; 3 1])*(1/10)
    matrix4 = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
    println(is_productive_matrix_3(matrix1,2))
    println(is_productive_matrix_3(matrix2,2))
    println(is_productive_matrix_3(matrix3,2))
    println(is_productive_matrix_3(matrix4,2))

end
```

```
Матрица непродуктивная
Матрица непродуктивная
Матрица продуктивная
Матрица продуктивная
```

**Выводы**

В ходе выполнения лабораторной работы успешно удалось освоение специализированных пакетов для решения задач в непрерывном и дискретном времени.