Лабораторная работа №6.

Евдокимов Иван Андреевич. НФИбд-01-20

6 декабря, 2023, Москва, Россия

Российский Университет Дружбы Народов

Цель лабораторной работы

Цель лабораторной работы

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

Процесс выполнения лабораторной работы

Пункт 1

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

```
using Plots

function malthus_model(x0, b, c, T)
    # Функция, представляющая модель Мальтуса
    a = b - c
    x = [x0]
    for t in 2:T
        push!(x, a * x[end])
    end
    return x
end

# Начальные данные и параметры
x0 = 100.0  # начальная численность популяции
b = 1.02  # коэффициент рождаемости
c = 0.01  # коэффициент смертности
T = 50.0    # количество шагов моделирования

# Моделирование
population = malthus_model(x0, b, c, T)

# Построение графика
plot(1:T, population, label="Численность популяции", xlabel="Время", ylabel="Численность")
```

Рис. 1: Код пункт 1

1.1.

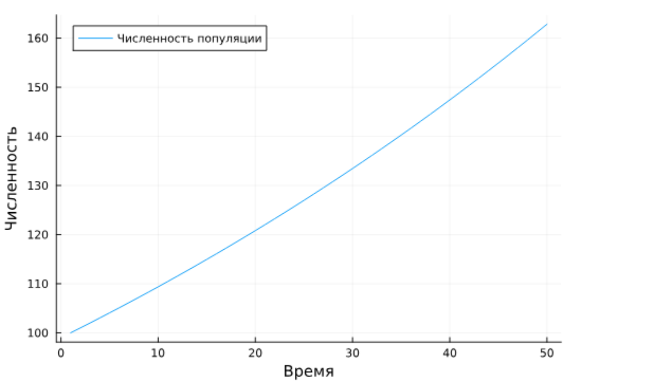

Рис. 2: Пункт 1.1

Пункт 2

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).

2.0.

```
using Plots

function logistic_model(x0, r, k, T, dt)
    # Функция, представляющая логистическую модель роста популяции
    x = [x0]
    for t in 2:T
        deltax = r * x[end] * (1 - x[end] / k) * dt
        push!(x, x[end] + deltax)
    end
    return x
end

# Начальные данные и параметры
x0 = 10.0  # начальная численность популяции
r = 0.1    # коэффициент роста популяции
k = 100.0  # потенциальная ёмкость экологической системы
T = 200    # количество шагов моделирования
dt = 0.1   # временной шаг

# Моделирование
population = logistic_model(x0, r, k, T, dt)

# Построение графика
plot(1:T, population, label="Численность популяции", xlabel="Время", ylabel="Численность")

# Анимация
anim = @animate for t in 1:T
    plot(1:t, population[1:t], label="Численность популяции", xlabel="Время", ylabel="Численность")
end
gif(anim, "logistic_animation.gif", fps = 10)
```

Рис. 3: Код пункт 2

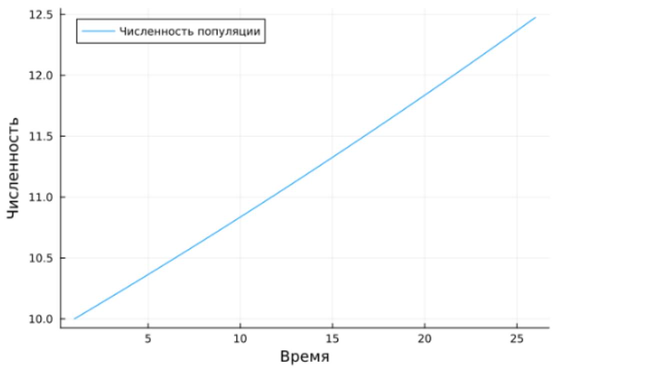

Рис. 4: Пункт 2.1

Пункт 3

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

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

Рис. 5: Система пункт 3

где $s\left(t\right)$ — численность восприимчивых к болезни индивидов в момент времени t, $i\left(t\right)$ — численность инфицированных индивидов в момент времени t, $r\left(t\right)$ — численность переболевших индивидов в момент времени t, β — коэффициент интенсивности контактов индивидов с последующим инфицированием, v — коэффициент интенсивности выздоровления инфицированных индивидов. Численность популяции считается постоянной,

```
using DifferentialEquations
using Plots

function sir_model(du, u, p, t)
    betta, v = p
    du[1] = -betta * u[1] * u[2]
    du[2] = betta * u[1] * u[2] - v * u[2]
    du[3] = v * u[2]
end

# Начальные данные и параметры
betta = 0.3   # коэффициент интенсивности контактов с последующим инфицированием
v = 0.1   # коэффициент интенсивности выздоровления
s0 = 0.9  # начальная доля восприимчивых к болезни
i0 = 0.1  # начальная доля инфицированных
r0 = 0.0  # начальная доля выздоровевших
u0 = [s0, i0, r0]

T = 200   # общее время моделирования
tspan = (0.0, T)
prob = ODEProblem(sir_model, u0, tspan, [betta, v])

# Решение уравнений модели SIR
sol = solve(prob)

# Построение графиков
plot(sol, label=["S" "I" "R"], xlabel="Время", ylabel="Доля популяции", title="Модель SIR")
```

Рис. 6: Код пункт 3

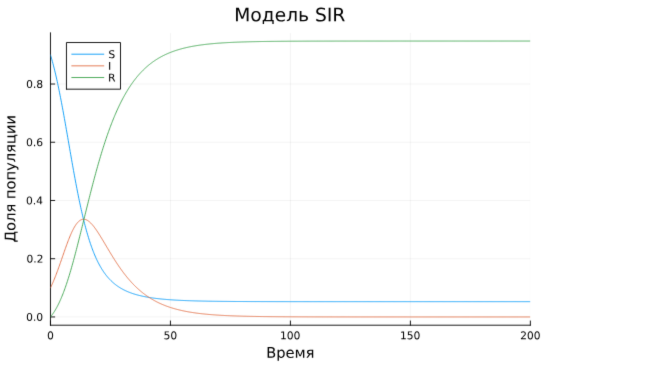

Рис. 7: Пункт 3.1

Пункт 4

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

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

Рис. 8: Система пункт 4

Размер популяции сохраняется: $s(t) + e(t) + i(t) + r(t) = N$. Исследуйте, сравните с SIR.

```julia
using DifferentialEquations
using Plots

function seir!(du, u, p, t)
    betta, delta, gamma, N = p
    s, e, i, r = u
    du[1] = -betta / N * s * i
    du[2] = betta / N * s * i - delta * e
    du[3] = delta * e - gamma * i
    du[4] = gamma * i
end

# Начальные условия
u0 = [0.99, 0.01, 0.0, 0.0]  # Начальные доли S, E, I, R
betta = 0.3
delta = 0.1
gamma = 0.05
N = sum(u0)
p = [betta, delta, gamma, N]

# Временной интервал и параметры интегрирования
tspan = (0.0, 200.0)
prob = ODEProblem(seir!, u0, tspan, p)

# Решение системы уравнений с использованием метода Эйлера
sol = solve(prob, Euler(), dt=0.1)

# Визуализация результатов
plot(sol, labels=["S" "E" "I" "R"], xlabel="Время", ylabel="Доля популяции", title="Модель SEIR")
```

Рис. 9: Код пункт 4

4.1.

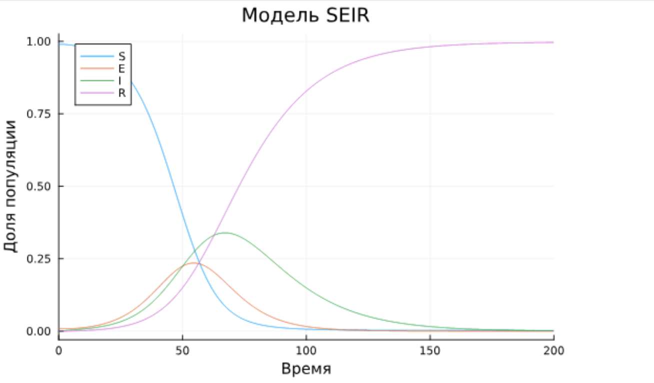

Рис. 10: Пункт 4.1

Пункт 5

5. Для дискретной модели Лотки–Вольтерры:

$$\begin{cases} X_1(t+1) = aX_1(t)(1 - X_1(t)) - X_1(t)X_2(t) \\ X_2(t+1) = -cX_2 + dX_1X_1(t) \end{cases}$$

Рис. 11: Система пункт 5

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.

```
using Plots
using NLsolve

function lotka_volterra(a, c, d, x1_0, x2_0, dt, num_steps)
    x1 = x1_0
    x2 = x2_0
    results = [(x1, x2)]

    for _ in 1:num_steps
        x1_new = x1 + dt * (a * x1 * (1 - x1) - x1 * x2)
        x2_new = x2 + dt * (-c * x2 + d * x1 * x2)
        x1, x2 = x1_new, x2_new
        push!(results, (x1, x2))
    end

    return results
end

# Параметры модели
a = 2.0
c = 1.0
d = 5.0
x1_0 = 0.1
x2_0 = 0.1
dt = 0.01
num_steps = 1000

# Получение численного решения
numerical_solution = lotka_volterra(a, c, d, x1_0, x2_0, dt, num_steps)

function find_equilibrium(a, c, d)
    function system!(F, x)
        F[1] = a * x[1] * (1 - x[1]) - x[1] * x[2]
        F[2] = -c * x[2] + d * x[1] * x[2]
    end

    initial_guess = [0.5, 0.5]  # Начальное предположение для точки равновесия
    result = nlsolve(system!, initial_guess)

    equilibrium_point = result.zero
    return equilibrium_point
end

equilibrium = find_equilibrium(a, c, d)
println("Точка равновесия: ", equilibrium)
x1_values = [x[1] for x in numerical_solution]
x2_values = [x[2] for x in numerical_solution]

plot(x1_values, x2_values, xlabel="X1", ylabel="X2", label="Численное решение", title="Лотки-Вольтерра Фазовый портрет")
scatter!([equilibrium[1]], [equilibrium[2]], color="red", label="Точка равновесия")
```

Рис. 12: Код пункт 5

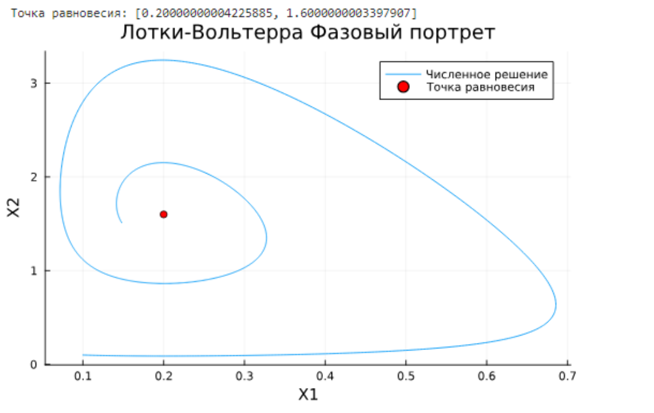

Рис. 13: Пункт 5.1

Пункт 6

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

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

Рис. 14: Система пункт 6

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

6.0.

```julia
using DifferentialEquations
using Plots

# Определение функции для системы дифференциальных уравнений
function competitive_selection!(du, u, p, t)
    alfa, betta = p
    du[1] = alfa * u[1] - betta * u[1] * u[2]
    du[2] = -alfa * u[2] + betta * u[1] * u[2]
end

# Начальные параметры
alfa = 0.1  # Параметр роста
betta = 0.02  # Параметр взаимодействия

# Начальные условия
x0 = 15.0  # Начальное количество популяции вида x
y0 = 2.0  # Начальное количество популяции вида y

# Параметры системы
u0 = [x0, y0]
tspan = (0.0, 200.0)
p = [alfa, betta]

# Решаем дифференциальное уравнение
prob = ODEProblem(competitive_selection!, u0, tspan, p)
sol = solve(prob)

# Строим графики
plot1 = plot(sol, label=["x(t)" "y(t)"], xlabel="Время", ylabel="Популяция", title="Конкурсный отбор")

# Строим фазовый портрет
plot2 = plot(sol, vars=(1,2), xlabel="Популяция x", ylabel="Популяция y", title="Фазовый портрет", label="")

# Анимация
anim1 = @animate for i in 1:length(sol)
    plot(sol[1:i], label=["x(t)" "v(t)"], xlabel="Время", ylabel="Популяция", title="Гармонический осциллятор свободных колебаний")
end

anim2 = @animate for i in 1:length(sol)
    plot(sol[1:i], vars=(1,2), xlabel="Популяция x", ylabel="Популяция y", title="Фазовый портрет", label="")
end

plot(plot1, plot2, layout=(2,1), legend=:bottomright, size=(800, 600))
display(plot!())

display(gif(anim1, "Competitive_relations1.gif", fps = 10))
display(gif(anim2, "Competitive_relations2.gif", fps = 100))
```

Рис. 15: Код пункт 6

6.1.

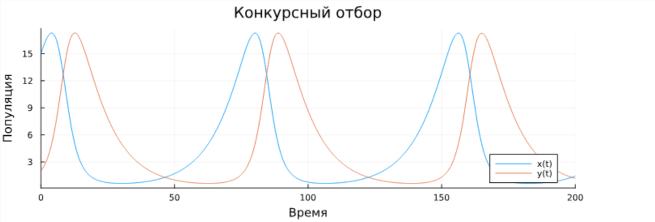

Рис. 16: Пункт 6.1

6.2.

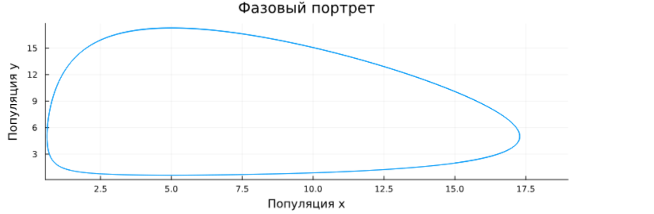

Рис. 17: Пункт 6.2

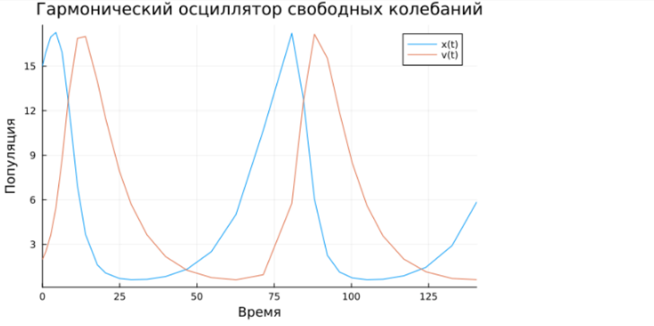

Рис. 18: Пункт 6.3

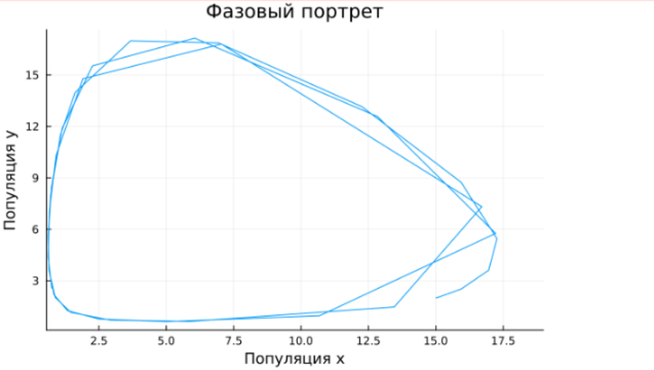

Рис. 19: Пункт 6.4

Пункт 7

7. Реализовать на языке Julia модель консервативного гармонического осциллятора
   $\ddot{x} + \omega_0^2 x = 0, x(t_0) = x_0, \dot{x}(t_0) = y_0$
   где ω_0 — циклическая частота. Начальные параметры подобрать самостоятельно, выбор пояснить. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет.

```
using DifferentialEquations
using Plots

# Определение функции для системы дифференциальных уравнений
function harmonic_oscillator!(du, u, p, t)
    du[1] = u[2]
    du[2] = -p[1]^2 * u[1]
end

# Начальные параметры
w0 = 2.0  # Циклическая частота
tₒ = 0.0  # Начальное время
x0 = 1.0  # Начальное положение
y0 = 0.0  # Начальная скорость

# Задаем начальные условия
u0 = [x0, y0]

# Временной интервал
tspan = (tₒ, 10.0)

# Параметры системы
p = [w0]

# Решаем дифференциальное уравнение
prob = ODEProblem(harmonic_oscillator!, u0, tspan, p)
sol = solve(prob)

# Строим графики
plot1 = plot(sol, label=["x(t)" "v(t)"], xlabel="Время", ylabel="Значение", title="Гармонического консервативный осциллятор")

# Строим фазовый портрет
plot2 = plot(sol, vars=(1,2), xlabel="Позиция (x)", ylabel="Скорость (v)", title="Фазовый портрет", label="")

# Анимация
anim1 = @animate for i in 1:length(sol)
    plot(sol[1:i], label=["x(t)" "v(t)"], xlabel="Время", ylabel="Значение", title="Гармонический осциллятор свободных колебаний")
end

anim2 = @animate for i in 1:length(sol)
    plot(sol[1:i], vars=(1,2), xlabel="Позиция (x)", ylabel="Скорость (v)", title="Фазовый портрет", label="")
end

plot(plot1, plot2, layout=(2,1), legend=:bottomright, size=(800, 600))
display(plot!())

display(gif(anim1, "Harmonic_conservative_oscillator1.gif", fps = 10))
display(gif(anim2, "Harmonic_conservative_oscillator2.gif", fps = 100))
```

Рис. 20: Код пункт 7

7.1.

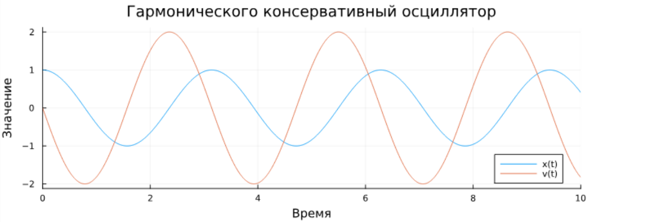

Рис. 21: Пункт 7.1

7.2.

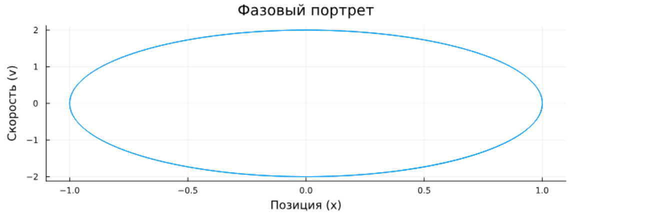

Рис. 22: Пункт 7.1

7.3.

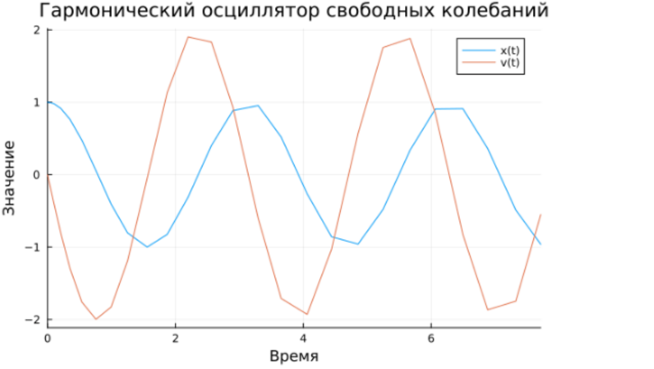

Рис. 23: Пункт 7.1

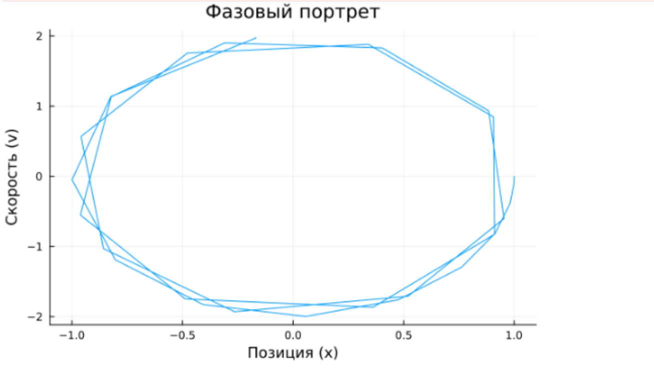

Рис. 24: Пункт 7.1

Пункт 8

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 ω_0 — циклическая частота, γ — параметр, характеризующий потери энергии. Начальные параметры подобрать самостоятельно, выбор пояснить. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет.

8.0.

```
using DifferentialEquations
using Plots

# Определение функции для системы дифференциальных уравнений
function damped_harmonic_oscillator!(du, u, p, t)
    du[1] = u[2]
    du[2] = -2.0 * p[2] * u[2] - p[1]^2 * u[1]
end

# Начальные параметры
w0 = 2.0  # Циклическая частота
γ = 0.1   # Параметр затухания
t₀ = 0.0  # Начальное время
x0 = 1.0  # Начальное положение
y0 = 0.5  # Начальная скорость

# Задаем начальные условия
u0 = [x0, y0]

# Временной интервал
tspan = (t₀, 20.0)

# Параметры системы
p = [w0, γ]

# Решаем дифференциальное уравнение
prob = ODEProblem(damped_harmonic_oscillator!, u0, tspan, p)
sol = solve(prob)

# Строим графики
plot1 = plot(sol, label=["x(t)" "v(t)"], xlabel="Время", ylabel="Значение", title="Гармонический осциллятор свободных колебаний")

# Строим фазовый портрет
plot2 = plot(sol, vars=(1,2), xlabel="Позиция (x)", ylabel="Скорость (v)", title="Фазовый портрет", label="")

# Анимация
anim1 = @animate for i in 1:length(sol)
    plot(sol[1:i], label=["x(t)" "v(t)"], xlabel="Время", ylabel="Значение", title="Гармонический осциллятор свободных колебаний")
end

anim2 = @animate for i in 1:length(sol)
    plot(sol[1:i], vars=(1,2), xlabel="Позиция (x)", ylabel="Скорость (v)", title="Фазовый портрет", label="")
end

plot(plot1, plot2, layout=(2,1), legend=:bottomright, size=(800, 600))
display(plot!())

display(gif(anim1, "Free_Oscillator_Harmonic_Oscillator1.gif", fps = 10))
display(gif(anim2, "Free_Oscillator_Harmonic_Oscillator2.gif", fps = 100))
```

Рис. 25: Код пункт 8

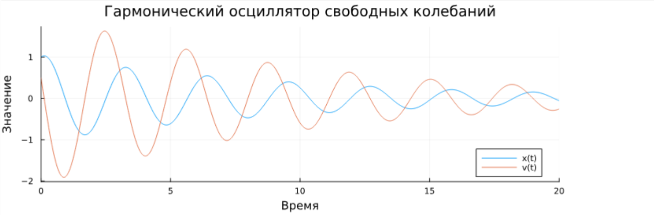

Рис. 26: Пункт 8.1

8.2.

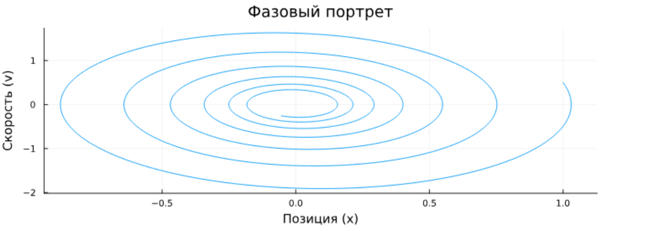

Рис. 27: Пункт 8.2

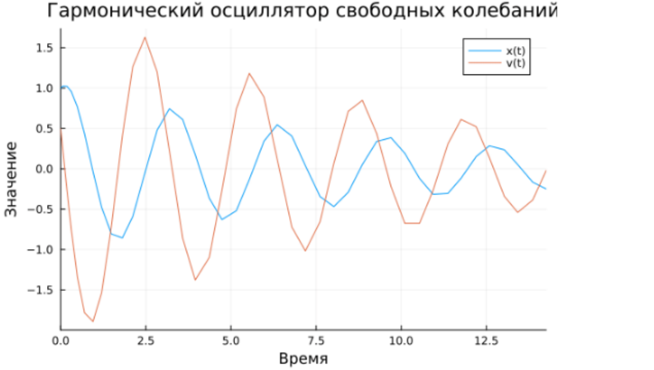

Рис. 28: Пункт 8.3

8.4.

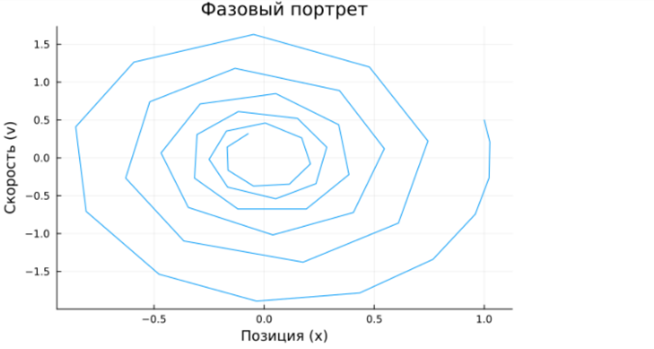

Рис. 29: Пункт 8.4

Выводы

Выводы

Мною были освоены специализированные пакеты для решения задач в непрерывном и дискретном времени.