Компьютерный практикум по статистическому анализу данных

Лабораторная работа №6: Решение моделей в непрерывном и дискретном времени

Кармацкий Никита Сергеевич

Российский университет дружбы народов, Москва, Россия

Цель лабораторной работы

Основной целью работы освоить синтаксис языка Julia для построения графиков.

Выполнение лабораторной работы: Решение обыкновенных дифференциальных уравнений

Вспомним, что обыкновенное дифференциальное уравнение (ОДУ) описывает изменение некоторой переменной u.

Для решения обыкновенных дифференциальных уравнений (ОДУ) в Julia можно использовать пакет diffrentialEquations.jl.

1. Модель экспоненциального роста

[4]:
```
# 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)
# 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")
```

[4]:

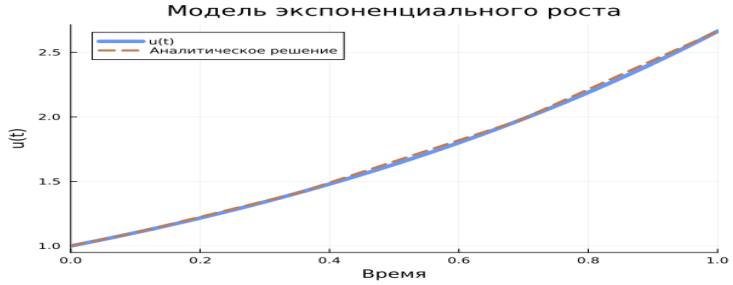

Рис. 1: График модели экспоненциального роста

1. Модель экспоненциального роста

[10]:
```
# задаём точность решения:
sol = solve(prob,abstol=1e-8,reltol=1e-8)
# 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")
```

[10]:

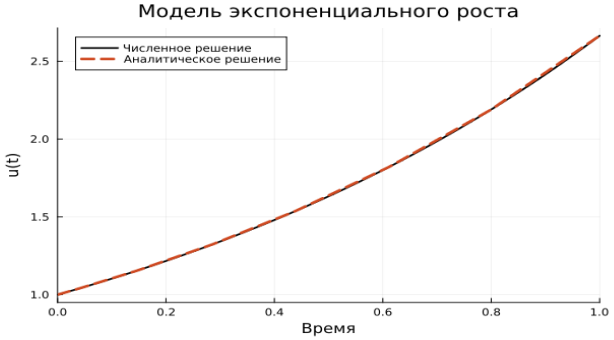

Рис. 2: График модели экспоненциального роста(задана точность решения)

1.2. Система Лоренца

[13]:
```
# 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, Tsit5())
# Строим график
plot(sol, idxs=(1, 2, 3), lw=1, title="Аттрактор Лоренца", xaxis="x", yaxis="y", zaxis="z", legend=false)
```

[13]:

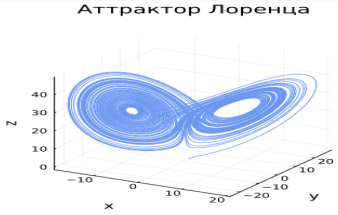

Рис. 3: Аттрактор Лоренца

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

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

[15]:

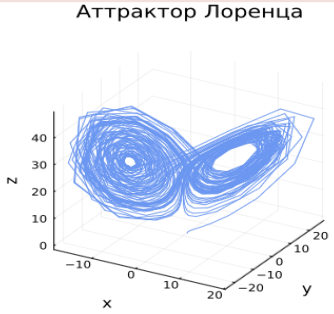

Рис. 4: Аттрактор Лоренца(интерполяция отключена)

2. Модель Лотки–Вольтерры

[18]:
```
# Определяем модель Лотки–Вольтерры
function lotka_volterra!(du, u, p, t)
    x, y = u
    a, b, c, d = p
    du[1] = a * x - b * x * y   # Уравнение для жертв
    du[2] = -c * y + d * x * y   # Уравнение для хищников
end
# Начальные условия
u0 = [1.0, 1.0]  # начальные популяции жертв и хищников
# Параметры модели
p = [1.5, 1.0, 3.0, 1.0]  # a, b, c, d
# Интервал времени
tspan = (0.0, 10.0)
# Создаём задачу
prob = ODEProblem(lotka_volterra!, u0, tspan, p)
# Решаем задачу
sol = solve(prob, Tsit5())
# Построение графика
plot(sol, label=["Жертвы" "Хищники"], color="black",
     linestyle = [:solid :dash], title="Модель Лотки–Вольтерры",
     xlabel="Время", ylabel="Размер популяции")
```

[18]:

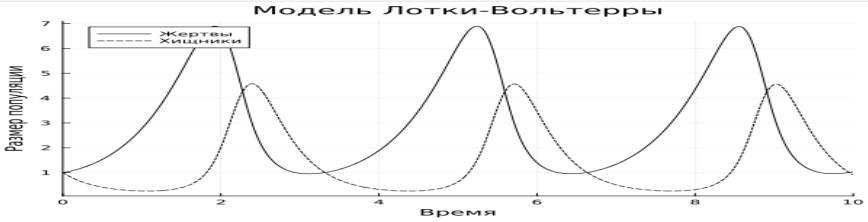

Рис. 5: Модель Лотки–Вольтерры: динамика изменения численности популяций

Время

[20]:
```
# фазовый портрет:
plot(sol,vars=(1,2), color="black", xaxis="Жертвы",yaxis="Хищники", legend=false)
```

[20]:

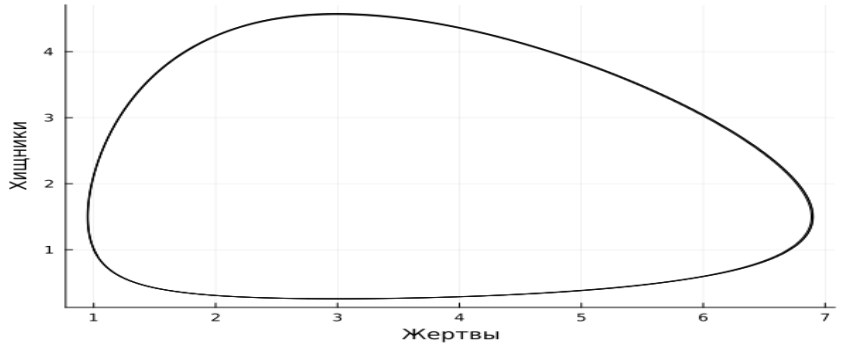

Рис. 6: Модель Лотки–Вольтерры: фазовый портрет

4. Самостоятельное выполнение

1) Реализовать и проанализировать модель роста численности изолированной популяции (модель Мальтуса). Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией):

[24]:
```
# Определение параметров
b = 1.0
c = 0.2
a = b - c
u0 = 1.0  # Начальная численность популяции
tspan = (0.0, 10.0)  # Интервал времени
# Описание модели
f(u, p, t) = a * u  # Уравнение роста популяции
# Задаём задачу
prob = ODEProblem(f, u0, tspan)
# Решение задачи
sol = solve(prob)
# Настройка анимации
anim = @animate for i in 1:length(sol.t)
    plot(sol.t[1:i], sol.u[1:i], linewidth=3, title="Модель Мальтуса", xlabel="Время", ylabel="Численность популяции", legend=false)
end
# Сохранение анимации в файл
gif(anim, "malthus_population_growth.gif", fps=15)
```

```
[ Info: Saved animation to /Users/nikitakarm/malthus_population_growth.gif
```

[24]:

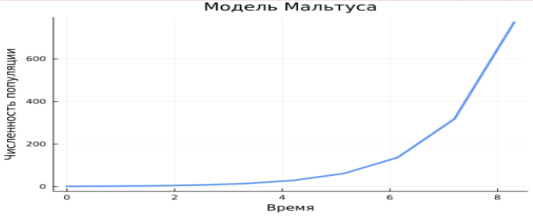

Рис. 7: Выполнение задания №1

4. Самостоятельное выполнение

2) Реализовать и проанализировать логистическую модель роста популяции. Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией):

[27]:
```
# Определение модели
function logistic!(du, u, p, t)
    r, k = p
    du[1] = r * u[1] * (1 - u[1] / k)
end
# Начальные условия
u0 = [1.0]  # Начальная численность популяции
# Параметры
r = 0.5  # Коэффициент роста
k = 10.0  # Емкость экосистемы
p = (r, k)
# Интервал времени
tspan = (0.0, 20.0)
# Решение
prob = ODEProblem(logistic!, u0, tspan, p)
sol = solve(prob, Tsit5())  # Используем Tsit5 для не-жёсткой задачи
# Создание объекта анимации
anim = Animation()
# Построение анимации
for t in 0:0.5:20
    plot(sol, vars=(0, 1), linewidth=2, color=:blue, title="Логистическая модель роста",
         xlabel="Время", ylabel="Популяция", legend=false)
    scatter!([t], [sol(t)[1]], color=:red, label="", markersize=5)  # Добавляем текущую точку
    frame(anim)  # Добавляем кадр в анимацию
end
# Сохранение анимации
gif(anim, "logistic_growth.gif", fps=10)  # Сохраняем анимацию в файл
```

```
[ Info: Saved animation to /Users/nikitakarm/logistic_growth.gif
```

[27]:

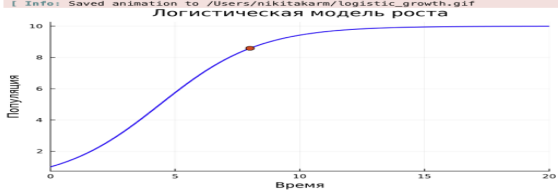

Рис. 8: Выполнение задания №2

4. Самостоятельное выполнение

3) Реализовать и проанализировать модель эпидемии Кермака–Маккендрика (SIRмодель). Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией):

[30]:
```
# Определение модели
function sir!(du, u, p, t)
    β, ν = p
    s, i, r = u
    du[1] = -β * s * i           # Восприимчивые
    du[2] = β * s * i - ν * i    # Инфицированные
    du[3] = ν * i                # Выздоровевшие
end
# Начальные условия
u0 = [0.99, 0.01, 0.0]  # 99% восприимчивых, 1% инфицированных
# Параметры
β = 0.3  # Коэффициент передачи инфекции
ν = 0.1  # Коэффициент выздоровления
p = (β, ν)
# Интервал времени
tspan = (0.0, 100.0)
# Решение
prob = ODEProblem(sir!, u0, tspan, p)
sol = solve(prob, Tsit5())  # Используем Tsit5 для универсального решения
# Создание анимации
anim = @animate for t in 1:length(sol.t)
    plot(sol.t[1:t], [u[1] for u in sol.u[1:t]], label="Восприимчивые", color=:blue, linewidth=2)
    plot!(sol.t[1:t], [u[2] for u in sol.u[1:t]], label="Инфицированные", color=:red, linewidth=2)
    plot!(sol.t[1:t], [u[3] for u in sol.u[1:t]], label="Выздоровевшие", color=:green, linewidth=2)
    title!("Модель эпидемии SIR")
    xlabel!("Время")
    ylabel!("Численность")
end
# Сохранение анимации
gif(anim, "sir_model_animation.gif", fps=120)
```

```
[ Info: Saved animation to /Users/nikitakarm/sir_model_animation.gif
```

[30]:

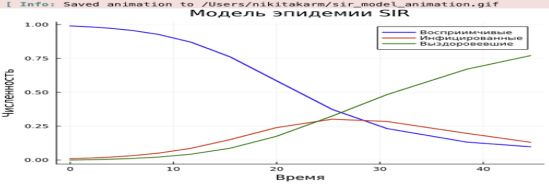

Рис. 9: Выполнение задания №3

4. Самостоятельное выполнение

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

[32]:

```
# Определение модели
function seir!(du, u, p, t)
    β, δ, γ, N = p
    s, e, i, r = u
    du[1] = -β * s * i / N           # Восприимчивые
    du[2] = β * s * i / N - δ * e # Экспонированные
    du[3] = δ * e - γ * i          # Инфицированные
    du[4] = γ * i                  # Выздоровевшие
end
# Начальные условия
u0 = [0.99, 0.0, 0.01, 0.0]  # 99% восприимчивых, 1% инфицированных
# Параметры
β = 0.3  # Коэффициент передачи инфекции
δ = 0.1  # Параметр инкубационного периода
γ = 0.1  # Коэффициент выздоровления
N = 1.0  # Общая численность населения
p = (β, δ, γ, N)
# Интервал времени
tspan = (0.0, 100.0)
# Решение
prob = ODEProblem(seir!, u0, tspan, p)
sol = solve(prob, Tsit5())  # Используем Tsit5 для не-жёстких задач
# Построение графика
plot(sol, label=["Восприимчивые" "Экспонированные" "Инфицированные" "Выздоровевшие"],
     title="Модель эпидемии SEIR", xlabel="Время", ylabel="Численность", linewidth=2)
```

[32]:

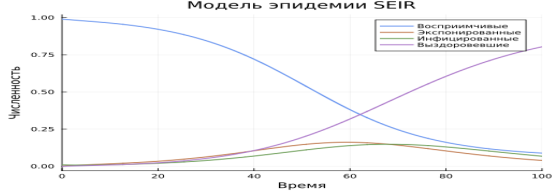

Рис. 10: Выполнение задания №4

5) Для дискретной модели Лотки–Вольтерры найдите точку равновесия. Получите и сравните аналитическое и численное решения. Численное решение изобразите на фазовом портрете:

[35]:
```
# Параметры модели
a = 2
c = 1
d = 5
# Численные методы для дискретной модели
function lotka_volterra!(X, p)
    a, c, d = p
    x1, x2 = X
    dx1 = a * x1 * (1 - x1) - x1 * x2
    dx2 = -c * x2 + d * x1 * x2
    return [dx1, dx2]
end
# Начальные условия
X0 = [0.1, 0.1]  # начальные значения X1 и X2
# Время и шаг
T = 100
dt = 0.1
times = 0:dt:T
# Массивы для численного решения
X = zeros(length(times), 2)
X[1, :] = X0
# Численное решение
for i in 2:length(times)
    X[i, :] = X[i-1, :] + dt * lotka_volterra!(X[i-1, :], (a, c, d))
end
# Построение фазового портрета
plot(X[:, 1], X[:, 2], label="Фазовый портрет", xlabel="X1", ylabel="X2")
```

[35]:

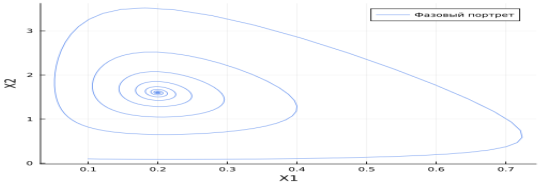

Рис. 11: Выполнение задания №5.

6) Реализовать на языке Julia модель отбора на основе конкурентных отношений. Начальные данные и параметры задать самостоятельно и пояснить их выбор. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет:

[70]:

```
# Модель конкуренции
function competition!(du, u, p, t)
    α, β = p
    x, y = u
    du[1] = α * x - β * x * y  # Изменение популяции x
    du[2] = α * y - β * x * y  # Изменение популяции y
end
# Начальные условия
u0 = [10.0, 5.0]  # Популяции x и y
# Параметры модели
α = 0.1  # Скорость роста
β = 0.01  # Коэффициент конкуренции
p = (α, β)
# Интервал времени
tspan = (0.0, 100.0)
# Решение ОДУ
prob = ODEProblem(competition!, u0, tspan, p)
sol = solve(prob, Tsit5())
# Построение анимации временных рядов
anim = @animate for t in range(tspan[1], tspan[2], length=200)
    current_time = t
    current_state = sol(t)  # Получение текущего состояния [x, y]
    plot(sol.t, sol[1, :], label="Популяция x", color=:blue, linewidth=2,
         xlabel="Время", ylabel="Популяция", title="Динамика популяций")
    plot!(sol.t, sol[2, :], label="Популяция y", color=:green, linewidth=2)
    scatter!([current_time], [current_state[1]], color=:blue, label="")
    scatter!([current_time], [current_state[2]], color=:green, label="")
end
gif(anim, "time_series_animation.gif", fps=30)
```

```
[ Info: Saved animation to /Users/nikitakarm/time_series_animation.gif
```

[70]:

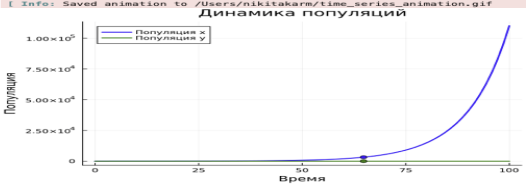

Рис. 12: Выполнение задания №6

4. Самостоятельное выполнение

7) Реализовать на языке Julia модель консервативного гармонического осциллятора. Начальные параметры подобрать самостоятельно, выбор пояснить. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет:

[81]:
```
# Модель гармонического осциллятора
function harmonic_oscillator!(du, u, p, t)
    ω0 = p[1]  # Циклическая частота
    du[1] = u[2]  # x' = v
    du[2] = -ω0^2 * u[1]  # v' = -ω0^2 * x
end
# Начальные условия
x0 = 1.0     # Начальное положение (м)
v0 = 0.0     # Начальная скорость (м/с)
u0 = [x0, v0]  # Начальные значения: x0 и v0
# Параметры
ω0 = 1.0     # Циклическая частота (рад/с)
p = [ω0]     # Параметры системы
# Интервал времени
tspan = (0.0, 50.0)
# Создание задачи для решения
prob = ODEProblem(harmonic_oscillator!, u0, tspan, p)
# Решение задачи
sol = solve(prob, Tsit5())
# Анимация
anim = @animate for t in range(tspan[1], tspan[2], length=200)
    current_time = t
    current_state = sol(t)  # Текущее состояние [x, v]
    # График положения и скорости
    plot(sol.t, sol[1, :], label="Положение x(t)", color=:blue, linewidth=2,
         xlabel="Время (t)", ylabel="Значение", title="Гармонический осциллятор")
    plot!(sol.t, sol[2, :], label="Скорость v(t)", color=:red, linewidth=2)
    scatter!([current_time], [current_state[1]], color=:blue, label="")
    scatter!([current_time], [current_state[2]], color=:red, label="")
end
gif(anim, "harmonic_oscillator_animation.gif", fps=30)
```

[81]:
```
[ Info: Saved animation to /Users/nikitakarm/harmonic_oscillator_animation.gif
```

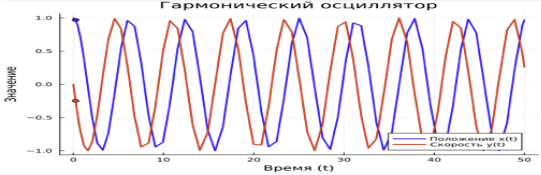

Рис. 13: Выполнение задания №7

4. Самостоятельное выполнение

8) . Реализовать на языке Julia модель свободных колебаний гармонического осциллятора. Начальные параметры подобрать самостоятельно, выбор пояснить. Построить соответствующие графики (в том числе с анимацией) и фазовый портрет:

[44]:
```
# Модель свободных колебаний с потерями
function damped_oscillator!(du, u, p, t)
    γ, ω0 = p
    du[1] = u[2]
    du[2] = -2 * γ * u[2] - ω0^2 * u[1]
end
# Начальные условия
u0 = [1.0, 0.0]  # Начальное положение и скорость
# Параметры
γ = 0.1
ω0 = 1.0
p = (γ, ω0)
# Интервал времени
tspan = (0.0, 50.0)
# Решение
prob = ODEProblem(damped_oscillator!, u0, tspan, p)
# Используем более подходящий алгоритм, например, Tsit5
sol = solve(prob, Tsit5())
# Построение графиков
plot(sol, label=["Положение x" "Скорость x'"], title="Свободные колебания с потерями энергии", xlabel="Время", ylabel="Значение", linewidth=2)
```

[44]:

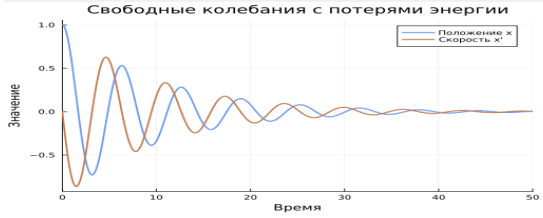

Рис. 14: Выполнение задания №8

Вывод

В ходе выполнения лабораторной работы были освоены специализированные пакеты для решения задач в непрерывном и дискретном времени.

Список литературы. Библиография

[1] Julia Documentation: https://docs.julialang.org/en/v1/