Лабораторная работа №7

Леснухин Даниил Дмитриевич Российский университет дружбы народов Москва

Цель работы

Основной целью работы является освоение специализированных пакетов Julia для обработки данных

Задание

1. Используя JupyterLab, повторите примеры, дополнив графики обозначениями осей координат, легендой и названиями графиков

2. Выполните задания для самостоятельной работы

Теоретическое введение

Julia — высокоуровневый язык программирования с динамической типизацией для математических вычислений
Эффективен для программ общего назначения
Использовалась официальная документация Julia

Задания для самостоятельной работы

Задание №1 — Кластеризация

Загрузим данные Iris и построим диаграммы кластеров

Задание №2 — Линейная регрессия

Пусть регрессионная зависимость является линейной. Матрица наблюдений факторов имеет размер N на 3, массив результатов размер N на 1. Найдем МНК-оценку для линейной модели

- Сравним свои результаты с результатами использования функции llsq из MultivariateStats

- Сравним с результатами использования регулярной регрессии наименьших квадратов из GLM

Создадим матрицу данных с добавленным столбцом единиц и решим систему линейных уравнений

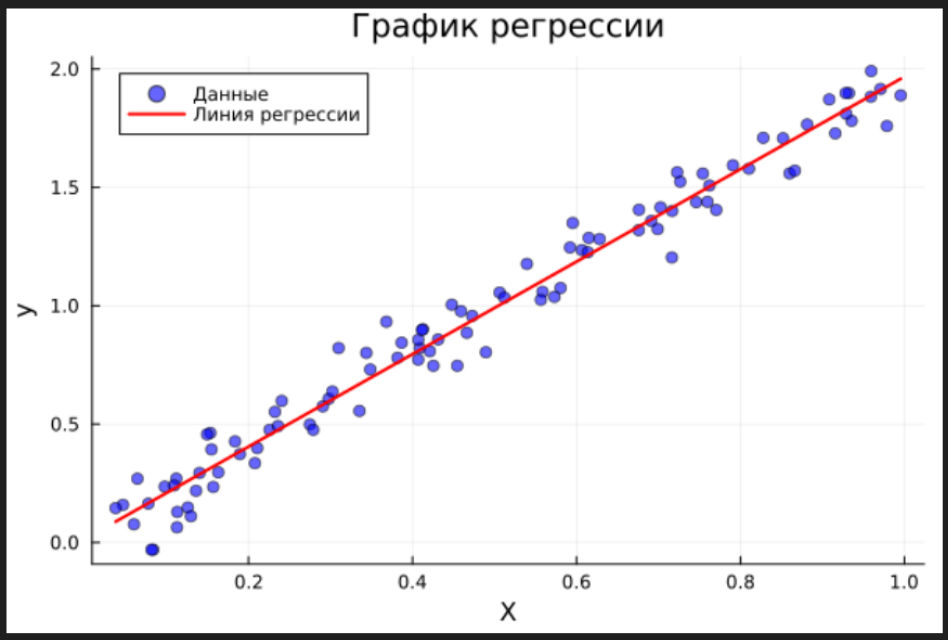
```julia
using Pkg
Pkg.update()

# Установка пакетов (если не установлены)
for p in ["CSV", "DataFrames", "RDatasets", "FileIO",
          "DelimitedFiles", "Clustering", "Plots", "Statistics"]
    Pkg.add(p)
end

using RDatasets
using Clustering
using Plots
using DataFrames
using Statistics

Pkg.add("GLM")

using GLM

N = 1000
X = rand(N, 3)
a0 = rand(3)

y = X * a0 + 0.1 * randn(N)
```

```julia
y = X * a0 + 0.1 * randn(N)

X_with_intercept = hcat(ones(N), X)
B_manual = (X_with_intercept' * X_with_intercept) \ (X_with_intercept' * y)

df = DataFrame(X1 = X[:, 1], X2 = X[:, 2], X3 = X[:, 3], Y = y)
model = lm(@formula(Y ~ X1 + X2 + X3), df)
B_glm = coef(model)

println("Коэффициенты, найденные вручную: ", B_manual)
println("Коэффициенты, найденные с помощью GLM: ", B_glm)
println("Разница между коэффициентами: ", B_glm[2:end])

y_pred_manual = X_with_intercept * B_manual
mse_manual = mean((y - y_pred_manual).^2)
println("MSE для ручного решения: ", mse_manual)
```

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Коэффициенты, найденные вручную: [0.016039051641520956, 0.2151320191113325, 0.13257289225109858, 0.18689555171543937]
Коэффициенты, найденные с помощью GLM: [0.01603905164152073, 0.2151320191113327, 0.1325728922510988, 0.18689555171543934]
Разница между коэффициентами: [0.2151320191113327, 0.1325728922510988, 0.18689555171543934]
MSE для ручного решения: 0.010556585871961083
```

Найдем линию регрессии, используя данные X и y. Построим график точек и добавим линию регрессии с заголовком и подписями осей

```
Коэффициенты регрессии: a = 1.9536027139429344, b = 0.013864760907581819
```



Задание №3 — Модель ценообразования опционов

Построим траекторию возможных цен на акции с учетом начальных параметров

- S — начальная цена акции

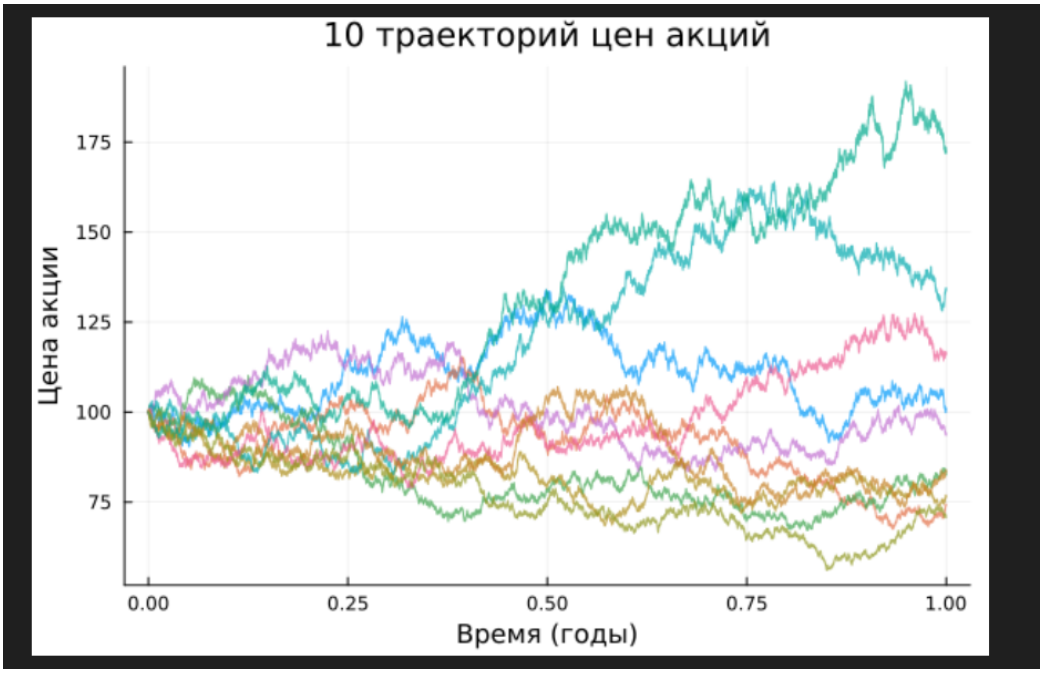
- T — длина биномиального дерева в годах
- n — количество периодов
- h — длина одного периода
- sigma — волатильность акции
- r — годовая процентная ставка
- u — верхнее значение для дерева
- d — нижнее значение для дерева
- p — вероятность вверх

```julia
using Plots
using Random
using Distributed


function createPath(S::Float64, r::Float64, sigma::Float64, T::Float64
    h = T / n
    u = exp(r * h + sigma * sqrt(h))
    d = exp(r * h - sigma * sqrt(h))
    p_star = (exp(r * h) - d) / (u - d)

    path = zeros(n + 1)
    path[1] = S

    for i in 2:(n + 1)
        if rand() < p_star
            path[i] = path[i-1] * u
        else
            path[i] = path[i-1] * d
        end
    end

    return path
end


println("Задание а: Построение одной траектории")
S = 100.0
```

```
println("Задание a: Построение одной траектории")
S = 100.0
T = 1.0
n = 10000
sigma = 0.3
r = 0.08

path = createPath(S, r, sigma, T, n)
time_points = range(0, T, length = n + 1)

plot(time_points, path,
     title = "Биномиальная модель: траектория цены акции",
     xlabel = "Время (годы)",
     ylabel = "Цена акции",
     label = "Траектория цены",
     linewidth = 2,
     legend = :topleft)

savefig("single_trajectory.png")
println("График сохранен как 'single_trajectory.png'")


println("\nЗадание b: Построение 10 траекторий")
time_points = range(0, T, length = n + 1)
plt = plot(title = "10 траекторий цен акций",
```

7.5s

Создадим функцию createPath, которая создает траекторию цены акции с учетом начальных параметров. Построим 10 разных траекторий на одном графике



Распараллелим генерацию траектории с использованием потоков и параллельных вычислений

```
Задание c: Тестирование параллельных методов
Доступное количество потоков: 1
Последовательная версия (50 траекторий):
  0.035322 seconds (31.95 k allocations: 1.969 MiB, 94.51% compilation time)
Параллельная версия с Threads.@threads:
  0.027088 seconds (12.13 k allocations: 1.002 MiB, 98.01% compilation time)

Задание d: Демонстрация модели
Демонстрационный график сохранен как 'demo_trajectory.png'

Статистика по сгенерированным траекториям:
Средняя конечная цена: 98.80017230409572
Минимальная конечная цена: 43.57284868796773
Максимальная конечная цена: 167.59737122187457
Стандартное отклонение: 32.06805411484571

Все задания выполнены успешно!
```

Выводы

В результате выполнения лабораторной работы я освоил специализированные пакеты Julia для обработки данных