

Лабораторная работа № 7

Введение в работу с данными

Герра Гарсия Паола Валентина

Содержание

Цель работы	2
Задание	2
Теоретическое введение	2
Выполнение лабораторной работы	2
Выводы	10
Список литературы.....	10

Список иллюстраций

Считывание данных.....	2
Запись данных в файл	3
Словари.....	3
DataFrames	3
RDatasets	4
Работа с переменными отсутствующего типа (MissingValues).....	4
FileIO.....	4
Кластеризация данных. Метод k-средних	5
Кластеризация данных. Метод k-средних	5
ластеризация данных. Метод k ближайших соседей	5
Обработка данных. Метод главных компонент	6
Обработка данных. Линейная регрессия	6
Обработка данных. Линейная регрессия	6
Кластеризация	7
Кластеризация	7
Регрессия	8
Регрессия	8
Модель ценообразования биномиальных опционов	9
Модель ценообразования биномиальных опционов	10
Модель ценообразования биномиальных опционов	10

Цель работы

Основной целью работы является освоение специализированных пакетов Julia для обработки данных.

Задание

1. Используя JupyterLab, повторите примеры. При этом дополните графики обозначениями осей координат, легендой с названиями траекторий, названиями графиков и т.п.
 2. Выполните задания для самостоятельной работы.

Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [[@julialang](#)]. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia [[@juliadoc](#)].

Выполнение лабораторной работы

Выполним примеры из лабораторной работы (рис. [-@fig:001]-[-@fig:013]).

Считывание данных

```

In [3]: # Выводим определение по названию языка программирования года его создания:
function language_created_year(P::Language::String)
    loc = findfirst(P[1:2], lowercase(language))
    return P[loc,1]
end
# Пример вызова функции и определение даты создания языка Python:
language_created_year("Python")
# Пример вызова функции и определение даты создания языка Julia:
language_created_year("Julia")

```

Out[3]: 2012

```

In [4]: # Выводим пример при вызове функции, в качестве аргумента которой указано
        # языок Julia, написание со строчной буквы:
language_created_year("Julia")
#У меня дана ошибка, так как список не содержит таких данных.

MethodError: no method matching getindex(::DataFrame, ::Nothing, ::Int64)
The function "getindex" exists, but no method is defined for this combination of argument types.

Closest candidates are:
getindex(::DataFrame, ::Colon, ::Union(AbstractString, Signed, Symbol, Unsigned)) at ./dataframe.jl:510
getindex(::DataFrame, ::Colon, ::Vector{Symbol}, ::Vector{Signed}) at ./dataframe.jl:510
getindex(::DataFrame, ::Colon, ::Vector{Signed}) at ./dataframe.jl:516
getindex(::DataFrame, ::InvertedIndex, ::Union(AbstractString, Signed, Symbol, Unsigned)) at ./dataframes.jl:585
...
```

Stacktrace:

```

[1] language_created_year(::DataFrame, language::String)
@ ./Julia/packages/Julia/v1.7.5/src/frame/dataframe.jl:510
[2] top-level scope
@ In[4]:3
```

Запись данных в файл

```

In [8]: # Пример записи данных в текстовый файл с разделителем ',':
writedlm("programming_languages_data.txt", Tx, ",")
# Пример записи данных в текстовый файл с разделителем '-':
writedlm("programming_languages_data2.txt", Tx, "-")

In [9]: # Построчное считывание данных с указанием разделителя:
P_new_delim = readdlm("programming_languages_data2.txt", '-')

```

```

Out[9]: 7x2 Matrix{Any}:
 1958  "LISP"
 1959  "ALGOL 58"
 1959  "COBOL"
 1959  "RPG"
 1962  "APL"
 2008  "Scala"
 2005  "R"
 2006  "PowerShell"
 2007  "Clojure"

```

```

In [10]: # Инициализация словаря:
dict = Dict{Integer,Vector{String}}()
Out[10]: Dict{Integer, Vector{String}}()

In [11]: # Инициализация словаря:
dict2 = Dict{String}()

```

Словари

```

In [5]: # Для того, чтобы убрать из функции зависимости данных от регистра, необходимо изменить исходную функцию следующим образом:
# функция определения по названию языка программирования
# года его создания (без учета регистра):
function language_created_year(P::Language::String)
    loc = findfirst(lowercase.(P[1:2]), lowercase.(language))
    return P[loc,1]
end
# Пример вызова функции и определение даты создания языка Julia:
language_created_year("Julia")

```

Out[5]: 2012

```

In [6]: Tx = readdlm("programminglanguages.csv", ',')

```

```

1958  "LISP"
1959  "ALGOL 58"
1959  "COBOL"
1959  "RPG"
1962  "APL"
2008  "Scala"
2005  "R"
2006  "PowerShell"
2007  "Clojure"
2009  "Go"
2010  "Rust"
2011  "Dart"
2011  "Node.js"
2011  "Redis"
2011  "Elixir"
2012  "Julia"
2014  "Swift"

```

```

In [7]: # Запись данных в CSV-файл:
CSV.write("programming_languages_data2.csv", P)

```

DataFrames

Jupyter Lab статистический анализ Last checkpoint: ayer a las 4:38 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Julia 1.12.0

```
In [22]: # Задание следующий о ценах:
# prices = [0.85, 1.6, 0.8, 0.1]
# DataFrame(item=foods, calories=calories)
# DataFrame(calories = DataFrame(item=foods, calories=calories))
# # Формирование данных о ценах
# DATA_PRICES = DataFrame(item=foods, price=prices)
# # Объединение данных о калориях и ценах
# DF = innerjoin(DataFrame_calories, DataFrame_prices, on = :item)
```

```
Out [22]: 4 rows x 3 columns
```

	Item	Calories	Price
1	apple	missing	0.85
2	cucumber	47	1.6
3	tomato	22	0.8
4	banana	105	0.6

```
In [23]: # Доработанное пакет FileIO:
using FileIO
```

```
In [24]: # Доработанное пакет ImageIO:
import Pkg
Pkg.add("ImageIO")
```

```
Resolving package versions...
Project No packages added or removed from `~/julia/environments/v1.12/Project.toml`
Manifest No packages added or removed from `~/julia/environments/v1.12/Manifest.toml`
```

Precompiling packages...

- x LinearSolve
- x GribCommon
- x GribParser
- x OrdinaryDiffEq
- x DelayedDiffEq
- x DiffEqBase
- x DiffEqFlux

R Datasets

Работа с переменными отсутствующего типа (MissingValues)

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Metac Opt, Fortran, Lust, python, SQL, courses, Reload to update
- URL:** localhost:8888/notebooks/fd67_статистический_анализ.ipynb
- User Information:** Logout, Julia 1.12.0
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Code Cell [In 29]:**

```
# Построение графика:  
wsized = plt.figure(figsize=(500,500),leg=False)  
x = houses[:,sq_ft]  
y = houses[:,price]  
scatter(x,y,markerSize=3)
```
- Output Cell [Out 29]:** A scatter plot showing house price (y-axis) versus square footage (x-axis). The x-axis ranges from approximately 1000 to 5000 sq ft, and the y-axis ranges from approximately \$200,000 to \$800,000. The data points show a positive correlation.
- Code Cell [In 30]:**

```
# Фильтрация данных по заданному условию:  
filter_houses = houses[houses[:,sq_ft]>800,:]  
# Построение графика:  
x = filter_houses[:,sq_ft]  
y = filter_houses[:,price]  
scatter(x,y)
```
- Output Cell [Out 30]:** A scatter plot showing house price (y-axis) versus square footage (x-axis) for houses larger than 800 square feet. The x-axis ranges from approximately 1000 to 5000 sq ft, and the y-axis ranges from approximately \$200,000 to \$800,000. The data points show a positive correlation, similar to the first plot but with fewer points.

FileIO

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** The title bar includes tabs for "Metodo Opt", "Fortran", "Latin", "python", "SQL", "courses", "Help", and "Relaunch to update".
- Toolbar:** Includes icons for File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Traducir, Integrates, Segundo Grado 20..., 8.0.0 miercoles, 2..., and All Bookmarks.
- User Information:** Shows "Luis" as the user, "python" as the language, "Metodo Opt" as the kernel, and "Julia 1.12.0" as the version.
- Code Cells:**
 - In [31]:** A code cell containing Python code to calculate the mean price for different house types. The output shows 3 rows by 2 columns of data.
 - In [32]:** A code cell containing Julia code for clustering. It imports Pkg, adds "Clustering", defines X as a matrix of coordinates, and defines X as a Matrix{X}. The output shows a warning about precompiling errors.
- Output Cells:**
 - Out[31]:** Displays the mean price for Residential, Condo, and Multi-Family categories.
 - Out[32]:** Displays the matrix X with 8142 rows and 4 columns, containing longitude and latitude values.

Кластеризация данных. Метод k -средних

Jupyter Лаб7_статистический_анализ Last Checkpoint: ayer a las 4:38 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

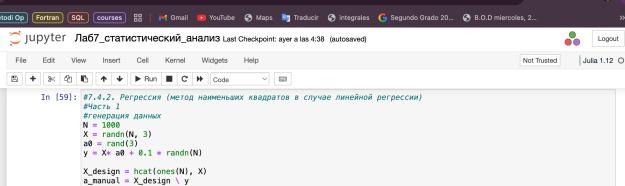
Not Trusted Julia 1.12.0

```
In [37]: clusters_figure = plt(legend = false)
for i = 1:k
    clustered_houses = df[!(df[:,cluster] == i,:)]
    xvals = clustered_houses[:,latitude]
    yvals = clustered_houses[:,longitude]
    scatter!(clusters_figure,xvals,yvals,markersize=4)
end
xlabel!("latitude")
ylabel!("longitude")
title("Houses color-coded by cluster")
display(clusters_figure)
```

Houses color-coded by cluster

```
In [38]: unique_zips = unique(filter_houses[:,zip])
zip_theta = plt(legend = false)
for zip in unique_zips
    subs = filter_houses[filter_houses[:,zip]==zip,:]
```

Кластеризация данных. Метод k -средних



The screenshot shows a Jupyter Notebook interface with several cells of code and their outputs.

In [59]:

```
# #7.4.2. Регрессия (метод наименших квадратов в случае линейной регрессии)
#Число 1
#Генерация данных
N = 1000
X = randn(N, 3)
y = randn(N)
u = X* 80 + B1 * randn(N)

X = np.array(X)
y = np.array(y)
u = np.array(u)
```

Код генерирует 1000 точек для линейной регрессии с небольшим шумом.

In [60]:

```
# #7.4.2. Регрессия (метод наименших квадратов)
#Пакет MultivariateStats
using MultivariateStats

#даты для MultivariateStats
X_ms = X
y_ms = y

a_multivariate = llsq(Matrix(X), y; bias=false)
println("сумма MultivariateStats:", a_multivariate)
```

Код использует пакет MultivariateStats для вычисления коэффициентов линейной регрессии.

In [61]:

```
# #7.4.2. Регрессия (метод наименших квадратов)
#Пакет MultivariateStats
using MultivariateStats

#даты для MultivariateStats
X_ms = X
y_ms = y

a_multivariate = llsq(Matrix(X), y; bias=false)
println("сумма MultivariateStats:", a_multivariate)
```

Код использует пакет MultivariateStats для вычисления коэффициентов линейной регрессии.

лестеризация данных. Метод k ближайших соседей

Обработка данных. Метод главных компонент

Jupyter Лаборатория Python

In [62]:

```
# ЧАСТЬ 2
# Генерация данных
X_simple = randn(100)
y_simple = 2 + X_simple + 0.1 * randn(100)

x_mean = mean(X_simple)
y_mean = mean(y_simple)

numerator = sum((X_simple - x_mean) * (y_simple - y_mean))
denominator = sum((X_simple - x_mean)^2)
b1 = numerator / denominator
b0 = y_mean - b1 * x_mean

print("оценимый наклон (b1):", b1)
print("оценимый intercept (b0):", b0)

оценимый наклон (b1):2.010863255675926
оценимый intercept (b0):0.817333296464038524
```

In [63]:

```
#изначение
scatter(X_simple, y_simple,
        label="Data points",
        marker="o", color="red", fillstyle="circle", color="pink",
        title="Regression plot",
        xlabel="X", ylabel="y")
```

Out [63]:

Regression plot



Обработка данных. Линейная регрессия

The figure shows a Jupyter Notebook interface with a scatter plot titled "Jupyter Lab67_статистический_анализ Last Checkpoint: ayer a las 4:38 (autosaved)". The plot displays data points as red circles with black outlines, showing a positive linear trend. A solid purple line represents the regression fit. The x-axis is labeled "X" and ranges from 0.00 to 1.00. The y-axis ranges from 0.00 to 2.00. The notebook cell below the plot contains the code for generating the regression line:

```
In [64]: #linea de regresion
x_range = np.arange(X_simple), maximum(X_simple)
y_pred = b0 + b1 * x_range
plot(x_range, y_pred, label="Regression line: y = $(round(b0, digits=3)) + $(round(b1, digits=3))x",
      linewidth=3, color="purple")
```

The output cell shows the resulting plot with the regression line equation $y = 0.017 + 2.016x$ displayed.

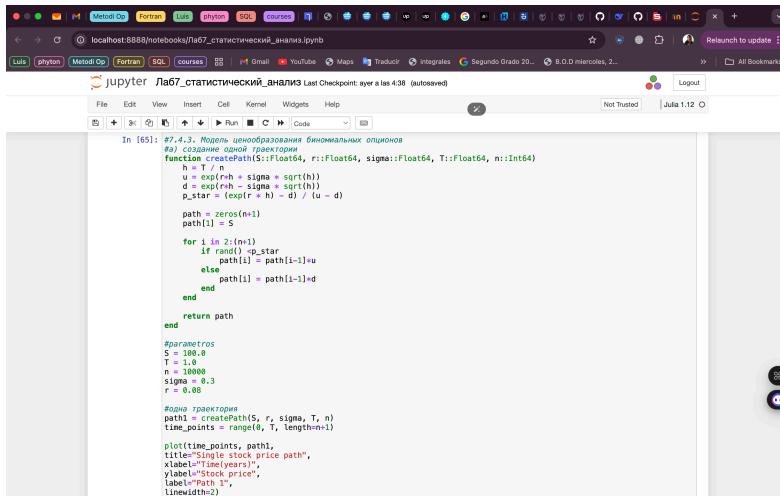
Обработка данных. Линейная регрессия

Теперь выполним задания для самостоятельный работы.

Загрузим

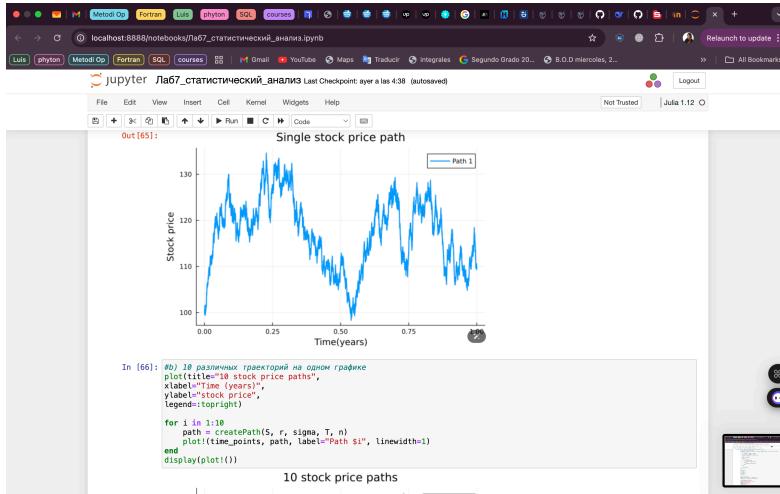
```
using RDatasets  
iris = dataset("datasets", "iris")
```

Используем Clustering.jl для кластеризации на основе k-средних. Сделаем точечную диаграмму полученных кластеров. Для этого проиндексируем фрейм данных, преобразуем его в массив и транспонируем (рис. [-@fig:014]).



```
In [65]: #7.4.3. Модели ценообразования биномиальных опционов  
#а) создание одной траектории  
function createPath(S::Float64, r::Float64, sigma::Float64, T::Float64, n::Int64)  
    u = T^(r+sigma*sqrt(n))  
    d = exp(r*T - sigma*sqrt(n))  
    p_star = (exp(r*T) + d) / (u + d)  
    path = zeros(n+1)  
    path[1] = S  
    for i = 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  
  
#параметры  
S = 100.0  
T = 1.0  
n = 10000  
r = 0.03  
sigma = 0.08  
  
#одна траектория  
path1 = createPath(S, r, sigma, T, n)  
time_points = range(0, T, length=n+1)  
plot(time_points, path1,  
      title="Single stock price path",  
      xlabel="Time(years)",  
      ylabel="Stock price",  
      label="Path 1",  
      linewidth=2)
```

Кластеризация

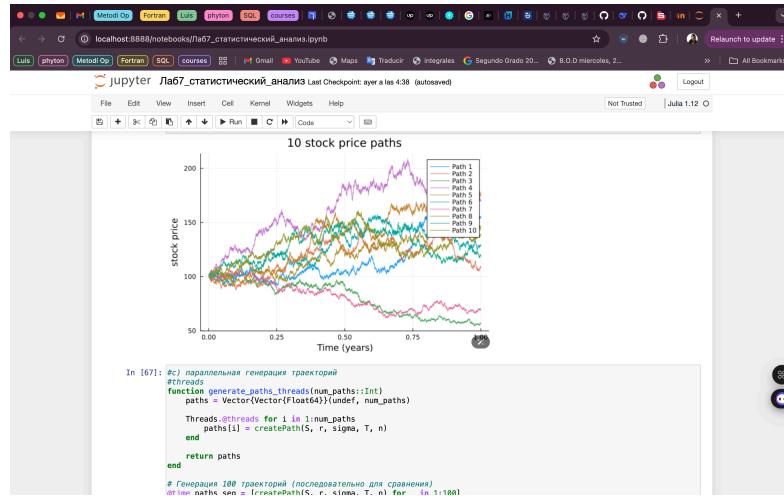


Кластеризация

Пусть регрессионная зависимость является линейной. Матрица наблюдений факторов X имеет размерность $N \times 3$ $\text{randn}(N, 3)$, массив результата в $N \times 1$, регрессионная зависимость является линейной. Найдем МНК-оценку для линейной модели.

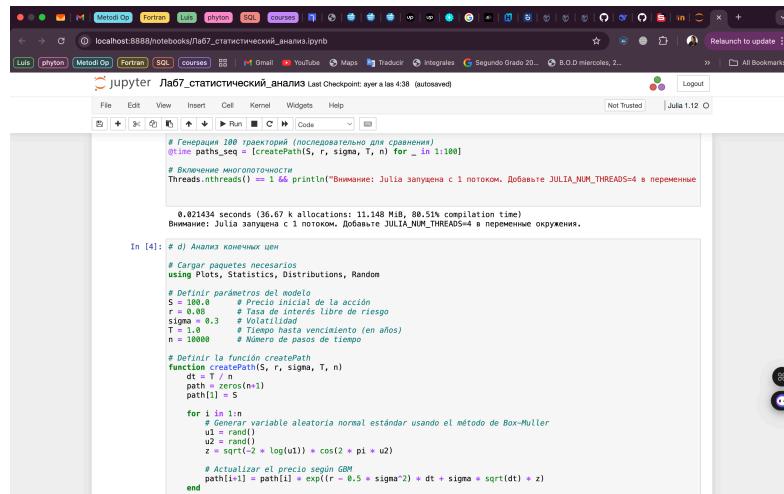
- Сравним свои результаты с результатами использования `llsq` из `MultivariateStats.jl`.
- Сравним свои результаты с результатами спользования регулярной регрессии наименьших квадратов из `GLM.jl`.

Создадим матрицу данных X^2 , которая добавляет столбец единиц в начало матрицы данных, и решим систему линейных уравнений.



Регрессия

Найдем линию регрессии, используя данные (X, y) . Построим график (X, y) , используя точечный график. Добавим линию регрессии, используя `abline!`. Добавим заголовок «График регрессии» и подпишем оси x и y .



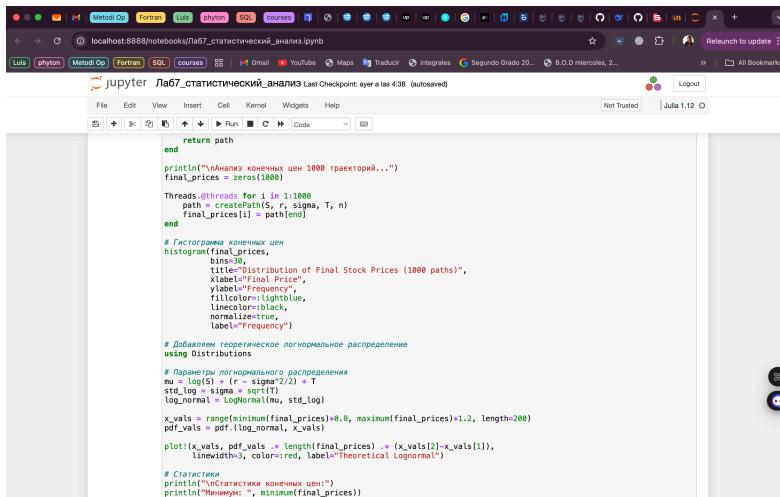
Регрессия

Построим траекторию возможных цен на акции:

- S – начальная цена акции;

- T – длина биномиального дерева в годах;
- n – количество периодов;
- $h = Tn$ – длина одного периода;
- σ – волатильность акции;
- r – годовая процентная ставка;
- $u = \exp(rh + \sigma\sqrt{h})$;
- $d = \exp(rh - \sigma\sqrt{h})$;
- $p^* = \frac{\exp(rh) - d}{u - d}$;

Пусть $S = 100$, $T = 1$, $n = 10000$, $\sigma = 0.3$ и $r = 0.08$. Попробуем построить траекторию курса акций.



```

# Длина пути
Threads = 1000
T = 1
n = 10000
sigma = 0.3
r = 0.08
final_prices = zeros(1000)

Threads.threads_for_3_in_1:T
    final_prices[i] = path[Threads]
end

# Гистограмма конечных цен
histogram(final_prices,
          title="Distribution of Final Stock Prices (1000 paths)",
          xlabel="Final Price",
          ylabel="Frequency",
          fillcolor=lightblue,
          linecolor=black,
          normalize=true,
          label="Frequency")

# Добавляем теоретическое логнормальное распределение
using Distributions

# Параметры логнормального распределения
mu = log(S) + (r - sigma^2/2) * T
std_log = sigma * sqrt(T)
lognormal = LogNormal(mu, std_log)

x_vals = range(minimum(final_prices)+0.8, maximum(final_prices)+1.2, length=200)
pdf_vals = pdf(lognormal, x_vals)

plot!(x_vals, pdf_vals, label="Theoretical Lognormal",
      linewidth=3, color=:red, label="Theoretical Lognormal")

# Статистики
println("Статистики конечных цен:")
println("Минимум: ", minimum(final_prices))

```

Модель ценообразования биномиальных опционов

Создадим функцию `createPath` ($S :: \text{Float64}$, $r :: \text{Float64}$, $\sigma :: \text{Float64}$, $T :: \text{Float64}$, $n :: \text{Int64}$), которая создает траекторию цены акции с учетом начальных параметров. Используем `createPath`, чтобы создать 10 разных траекторий и построим их все на одном графике.

A screenshot of a Jupyter Notebook interface in a web browser. The title bar shows the URL as `localhost:8888/notebooks/jlab7_статистический_анализ.ipynb`. The notebook content displays a Julia script for calculating option prices. The script includes code to calculate statistics of final prices (minimum, maximum, median, standard deviation) and a function to calculate the price of a European call option using 1000 simulations. The output cell shows the results of the analysis for 1000 trajectories, including minimum, maximum, median, and standard deviation of the final prices, and the calculated option price.

```
print("nГрафике конечных цен:")
print("Минимум: ", minimum(final_prices))
print("Максимум: ", maximum(final_prices))
print("Медиана: ", median(final_prices))
print("Стандартное отклонение: ", std(final_prices))

# Дополнительно: вычисление цены опциона как (European Call)
print("nОценка цены европейского опциона копи ===")
function european_call_price(S, K, r, sigma, T, n, num_simulations)
    total_payoff = 0.0

    for i = 1:num_simulations
        S, r, sigma, T, n = path_end(i, S, r, sigma, T, n)
        ST = path(ST -> K, 0)
        payoff = max(ST - K, 0)
        total_payoff += payoff
    end

    average_payoff = total_payoff / num_simulations
    discounted_price = exp(-r * T) * average_payoff
    return discounted_price
end

K = 100.0
num_simulations = 1000
call_price = european_call_price(50, K, 0.05, 0.2, 1.0, 1000)
println("nОцененная цена опциона копи (K=100): ", round(call_price, digits=4))

Анализ конечных цен 1000 траекторий...
Статистика конечных цен:
Минимум: 31.1234463632595
Максимум: 241.8129402785584
Среднее: 104.491215219452
Медиана: 104.491215219452
```

Модель ценообразования биномиальных опционов

Распараллелим генерацию траектории. Можем использовать `Threads.@threads`, `pmap` и `@parallel`.

A screenshot of a Jupyter Notebook interface in a web browser, similar to the previous one. The title bar shows the URL as `localhost:8888/notebooks/jlab7_статистический_анализ.ipynb`. The notebook content displays a modified Julia script for calculating option prices, utilizing parallel processing (`@parallel`) instead of sequential loops. The output cell shows the results of the analysis for 1000 trajectories, including minimum, maximum, median, and standard deviation of the final prices, and the calculated option price.

```
end
average_payoff = total_payoff / num_simulations
discounted_price = exp(-r * T) * average_payoff
return discounted_price
end

K = 100.0
num_simulations = 1000
call_price = european_call_price(50, K, 0.05, 0.2, 1.0, 1000)
println("nОцененная цена опциона копи (K=100): ", round(call_price, digits=4))

Анализ конечных цен 1000 траекторий...
Статистика конечных цен:
Минимум: 31.1234463632595
Максимум: 241.8129402785584
Среднее: 104.491215219452
Медиана: 104.491215219452
Стандартное отклонение: 32.415266621123514
== Оценка цены европейского опциона копи ===
Оцененная цена опциона копи (K=100): 15.9723
```

Модель ценообразования биномиальных опционов

Выводы

В результате выполнения данной лабораторной работы я освоила специализированные пакеты Julia для обработки данных.

Список литературы

:::{#ref}s :::