

Лабораторная работа № 8

Оптимизация

Герра Гарсия Паола Валентина

Российский университет дружбы народов, Москва, Россия

Информация

- Герра Гарсия Паола Валентина
- студентка
- Российский университет дружбы народов
- 1032225472@pfur.ru

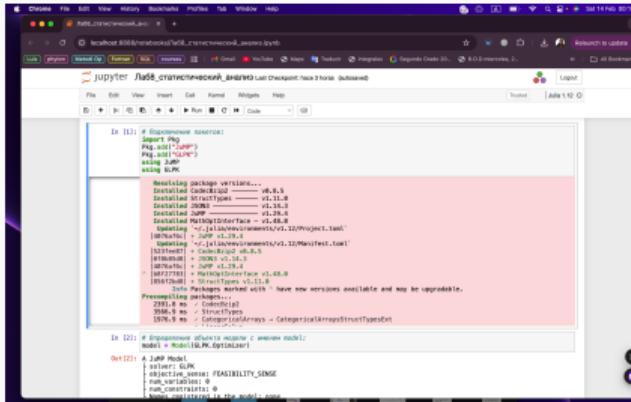
Цель работы

Основная цель работы – освоить пакеты Julia для решения задач оптимизации.

Задание

1. Используя JupyterLab, повторите примеры.
2. Выполните задания для самостоятельной работы.

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook cell with two parts of Python code. The first part imports GLPK and defines a model with variables and constraints. The second part solves the model.

```
In [11]: # Инициализация переменных
import pulp
prob = pulp.LpProblem("LP", pulp.LpMaximize)
x1 = pulp.LpVariable("x1", lowBound=0, cat="Continuous")
x2 = pulp.LpVariable("x2", lowBound=0, cat="Continuous")
prob += x1 + x2, "Z"

# Ограничения
prob += 2*x1 + 3*x2 <= 12, "C1"
prob += 3*x1 + 2*x2 <= 10, "C2"
prob += x1 <= 4, "C3"
prob += x2 <= 3, "C4"

# Целевая функция
prob += x1 + x2, "Z"

print(prob)

In [12]: # Выводим результат модели с помощью метода solve()
prob.solve(pulp.GLPK())

Out[12]: A LpStatusOptimal
A simplex: GLPK
objective sense: FEASIBILITY_SENSE
nonzero elements: 9
nonzero coefficients: 9
non-constraints: 0
linear constraints in the model: none
```

Рис. 1: Линейное программирование

Выполнение лабораторной работы

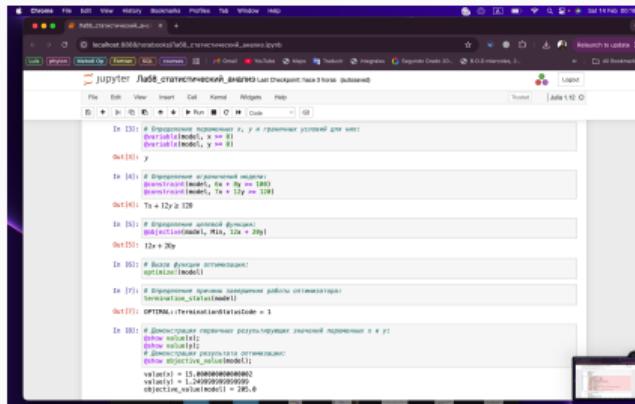


Рис. 2: Векторизованные ограничения и целевая функция оптимизации

Выполнение лабораторной работы

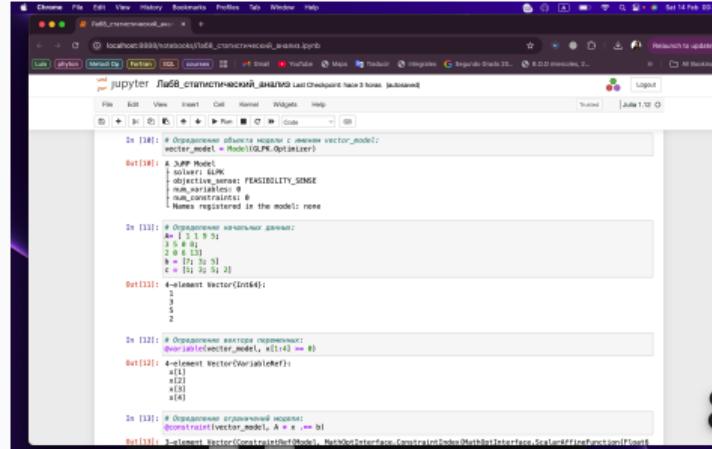


Рис. 3: Оптимизация рациона питания

Выполнение лабораторной работы

The screenshot shows a Jupyter Notebook running in a Chrome browser window. The notebook has three cells:

- In [19]:** A NumPy array named `category_data` is defined, containing values for protein, fat, sodium, and carbohydrates across 12986 items.
- In [20]:** A 2D NumPy array `And data` is defined, showing the relationship between food items and categories.
- In [21]:** A vector of food names is created, including "hamburger", "chicken", "nuggets", "fries", "macaroni", "pizza", "salad", "milk", and "ice cream".
- In [22]:** A cost vector is defined for each food item.

Рис. 4: Оптимизация рациона питания

Выполнение лабораторной работы

The screenshot shows a Jupyter Notebook running in a Chrome browser window. The notebook has several cells:

- Cell [14]:

```
# Определение целевой функции
objective(vector_model, Min, c' * x)
```
- Cell [15]:

```
x1 + 3x2 + 5x3 + 2x4
```
- Cell [16]:

```
# Вызов функции оптимизации
optimize(vector_model)
```
- Cell [17]:

```
# Проверка правильности работы алгоритмов
optimize_stochasticModel
```
- Cell [18]:

```
OPTIMAL::TerminationStatusCode = 1
```
- Cell [19]:

```
# Демонстрация результатов оптимизации
show(objective,vector_model);

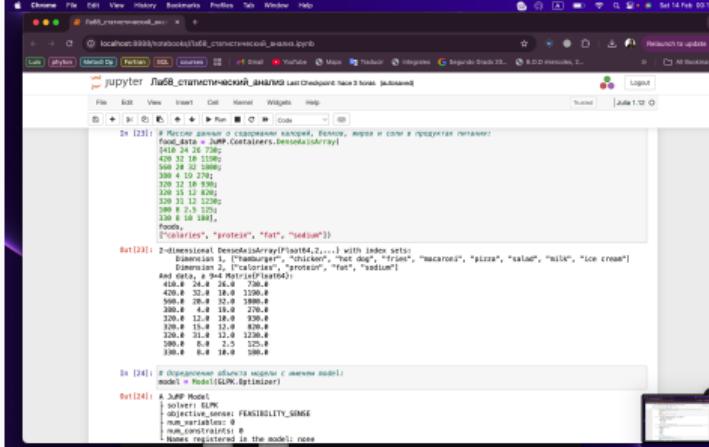
objective_value(vector_model) = 4.5239769230199225
```
- Cell [20]:

```
# Подключение пакетов
import Pkg
Pkg.add("GLPK")
Pkg.add("GLM")
using GLPK
using GLM
```
- Cell [21]:

```
Resolving package versions...
Project No packages added to or removed from `~/julia/environments/v1.12/Project.toml'.
Manifest No packages added to or removed from `~/julia/environments/v1.12/Manifest.toml'.
Precompiling packages...
  • LinearSolve
    ▾ LinearSolve
      ▾ GLPK
        ▾ GLPK
          ▾ GLPK
            ▾ GLPK
```

Рис. 5: Оптимизация рациона питания

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface running in a browser window. The code cell [23] contains Python code to calculate the nutritional values of various food items based on their components. The code uses a dictionary to map food items to their nutritional values and then calculates the total values for each component across all items. The output shows the total nutritional values for each component: carbohydrates (416.8), protein (32.0), fat (11.0), and sodium (1.0). The code cell [24] shows the creation of a GLPK model object.

```
# Матрица значений о содержании калорий, белков, жира и соли в продуктах питания
nutrients = {
    "hamburger": {"carbohydrates": 416.8, "protein": 32.0, "fat": 11.0, "sodium": 1.0},
    "chicken": {"carbohydrates": 420.0, "protein": 32.0, "fat": 11.0, "sodium": 1.0},
    "fries": {"carbohydrates": 380.0, "protein": 10.0, "fat": 19.0, "sodium": 1.0},
    "milk": {"carbohydrates": 320.0, "protein": 8.0, "fat": 8.0, "sodium": 1.0},
    "pizza": {"carbohydrates": 320.0, "protein": 12.0, "fat": 12.0, "sodium": 1.0},
    "salad": {"carbohydrates": 320.0, "protein": 12.0, "fat": 12.0, "sodium": 1.0},
    "ice cream": {"carbohydrates": 320.0, "protein": 12.0, "fat": 12.0, "sodium": 1.0}
}

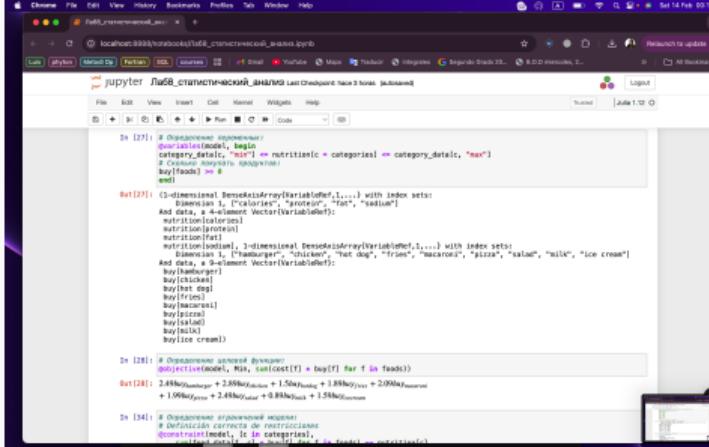
# Вычисление общего количества каждого компонента
total_nutrients = {component: sum(item[component] for item in nutrients.values()) for component in ["carbohydrates", "protein", "fat", "sodium"]}

# Вывод результатов
print(total_nutrients)

# [24]: # Определение объекта модели с именем model
model = Model(GLPK.Optimizer)
```

Рис. 6: Путешествие по миру

Выполнение лабораторной работы

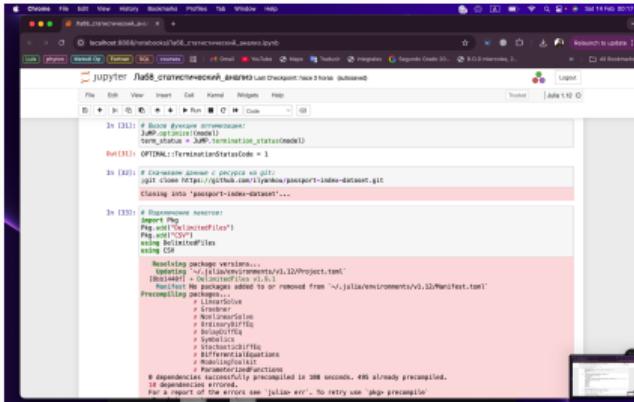


The screenshot shows a Jupyter Notebook interface running in a Chrome browser window. The notebook has three cells:

- Cell 1 [27]:** Contains code for defining a model and creating a constraint for the maximum nutritional value (max). It includes imports for DensesMatrix and Vector from cvxpy, and defines variables for categories and foods.
- Cell 2 [28]:** Contains code for defining a model and creating a constraint for the sum of weights equal to 1. It includes imports for cvxpy and cvxpy.solvers, and defines variables for categories and foods.
- Cell 3 [29]:** Contains code for defining a model, setting the objective to minimize the sum of squared weights, and defining a constraint for the sum of weights equal to 1. It includes imports for cvxpy and cvxpy.solvers, and defines variables for categories and foods.

Рис. 7: Портфельные инвестиции

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface running in a browser window. The title bar reads "jupyter: Лаб8_математической_аналитики синтезом поисковых алгоритмов". The main area displays a terminal session with the following commands and output:

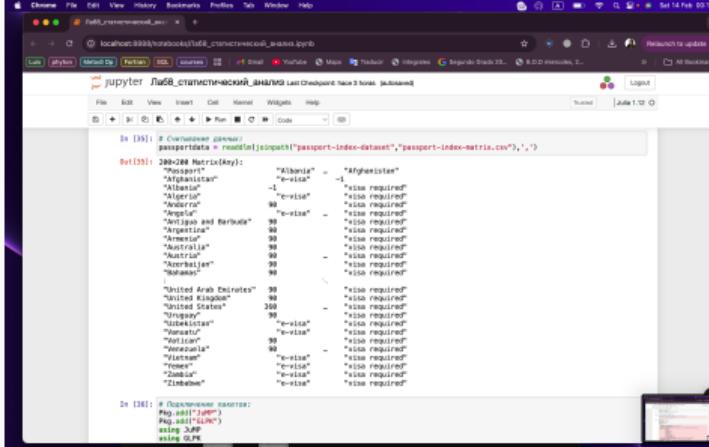
```
In [31]: # Вызовите функцию precompile()
# для отладки визуализации
term_status = XMPTerminationStatusModel()
Out[31]: OPTIMAL(terminationStatusCode = 1)

In [32]: # Создаваем архив с папкой с git2
git clone https://github.com/tylevko/passport-index-dataset.git
cloning into 'passport-index-dataset'...
In [33]: # Возвращение сервера
Pkg.add("YAML")
Pkg.add("JSON")
using Pkg
using CSV
using DataFrames

# Установка пакетов версии...
# Вызывается в /Users/tylevko/.julia/environments/v1.12/Project.toml
# (без @edit) + DelimitedFiles v1.6.1
# Несколько пакетов были добавлены к или удалены из `~/.julia/environments/v1.12/Manifest.toml`
Precompiling packages...
x Arbre
x ArbreAbstraction
x ArbreAlgorithm
x ArbreGraph
x ArbreGraphs
x ArbreMatrix
x ArbreMatrixOperations
x ArbreMatrixUtil
x ArbreMatrixUtilOperations
x ArbreMatrixUtilUtil
0 dependencies successfully precompiled in 208 seconds. 408 already precompiled.
# Для получения информации о ошибках выполните `jlinfo err` - To retry use `jplg-precompile`
```

Рис. 8: Портфельные инвестиции

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface running in a Chrome browser window. The URL is `localhost:8888/notebooks/1668_статистический_анализ.ipynb`. The notebook has one cell containing Python code:

```
# Основное задание:  
passportdata = readfile(juppath("passport-index-dataset","passport-index-matrix.csv"),',')  
  
# In[38]:  
200x200 Matrix[Key:  
"Afghanistan" : "e-visa"  
"Albania" : "-1" : "visa required"  
"Albania" : "1" : "e-visa required"  
"Algeria" : "-1" : "visa required"  
"Andorra" : "99" : "visa required"  
"Angola" : "99" : "e-visa required"  
"Argentina and Bermuda" : "99" : "visa required"  
"Argentina" : "99" : "visa required"  
"Armenia" : "99" : "visa required"  
"Australia" : "99" : "visa required"  
"Australia" : "-1" : "visa required"  
"Azerbaijan" : "99" : "visa required"  
"Bahrain" : "99" : "visa required"  
"United Arab Emirates" : "99" : "visa required"  
"United Kingdom" : "99" : "visa required"  
"United States" : "100" : "visa required"  
"Uruguay" : "99" : "visa required"  
"Uzbekistan" : "99" : "e-visa required"  
"Venezuela" : "99" : "e-visa required"  
"Volvo" : "99" : "visa required"  
"Yemen" : "99" : "visa required"  
"Vietnam" : "99" : "e-visa required"  
"Yemen" : "99" : "visa required"  
"Zambia" : "99" : "e-visa required"  
"Zimbabwe" : "99" : "e-visa required"  
  
# Помощник поиска:  
# подсчитать кол-во  
# подсчитать кол-во  
# ввести 2000  
# ввести 2000
```

Рис. 9: Восстановление изображения

Выполнение лабораторной работы

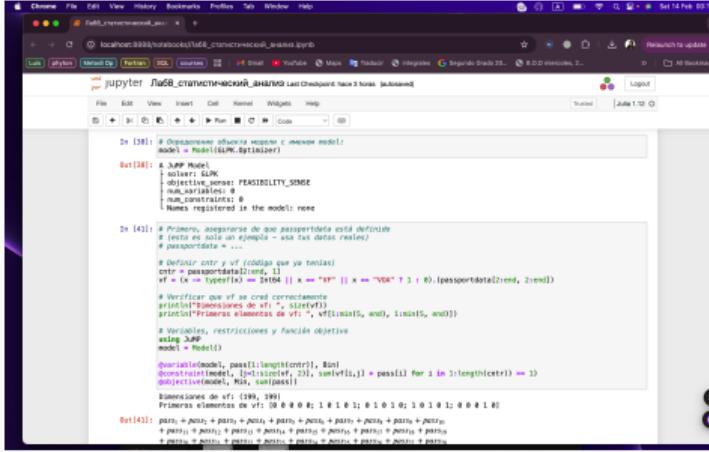
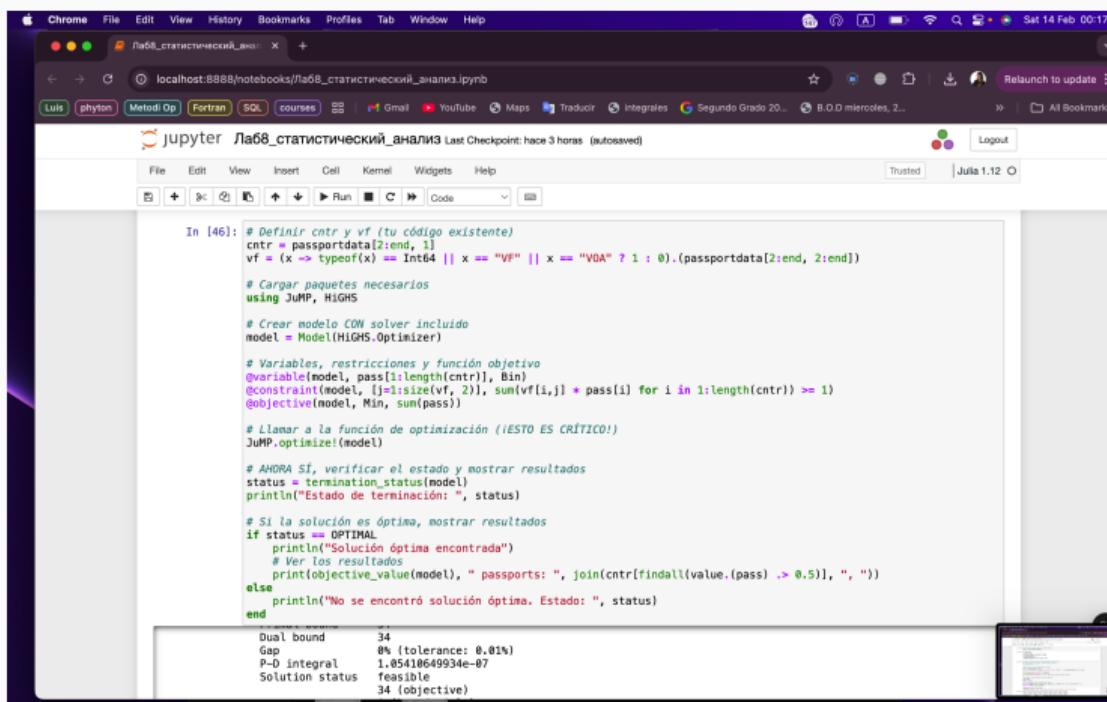


Рис. 10: Восстановление изображения

Выполнение лабораторной работы

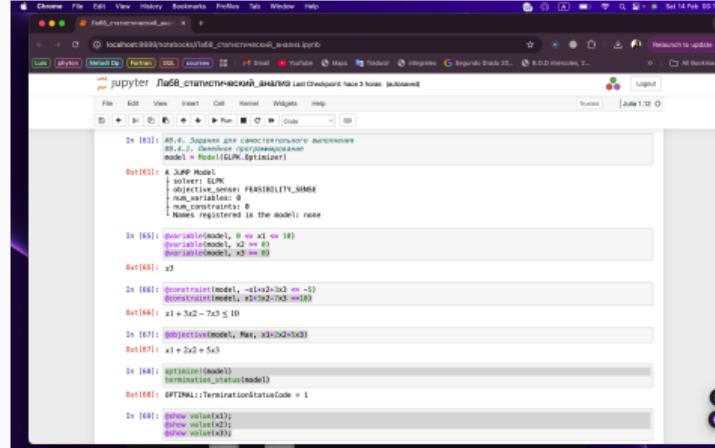


The screenshot shows a Jupyter Notebook interface running on a Mac OS X system. The browser tab is titled "Lab6_статистический_анализ" and the URL is "localhost:8888/notebooks/". The notebook has tabs for "Luis", "python", "Metodo Op", "Fortran", "SQL", and "courses". The current cell is in the "Julia" language, with the kernel version "Julia 1.12" indicated. The code cell contains a Julia script for solving a linear programming problem using the COIN solver included in the Model package. The script defines variables, constraints, and an objective function, then calls the `JUML.optimize!` function. The output pane below the code shows the solver's progress:

```
Dual bound      34
Gap            0% (tolerance: 0.01%)
P-D integral   1.05410649934e-07
Solution status feasible
34 (objective)
```

Рис. 11: Восстановление изображения

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface running in a Chrome browser window. The URL is `localhost:8888/notebooks/168_статистический_анализ.ipynb`. The notebook contains the following code:

```
In [63]: # 6.4. Задача о сверхтекущем выполнении
# 6.4.1. Демонстрация (программирование
# моделирования) в GLPK (optimizer)

Out[63]:
A LP Model
solver: GLPK
| objective sense: FEASIBILITY_SENSE
| max. constraints: 0
| num. constraints: 0
| Names registered in the model: none

In [64]: #quadratic(model, 0 <= x1 <= 10)
#quadratic(model, x2 >= 0)
#quadratic(model, x3 <= 0)

Out[64]: x2

In [65]: #constraint(model, -x1+x2+3x3 <= -1)
#constraint(model, x1+2x2+7x3 <= 10)
#constraint(model, x1+2x2+7x3 <= 10)

Out[65]: x1 + 2x2 + 7x3 <= 10

In [66]: #projection(model, Max, x3>x2>x3)
Out[66]: x1 + 2x2 + 5x3

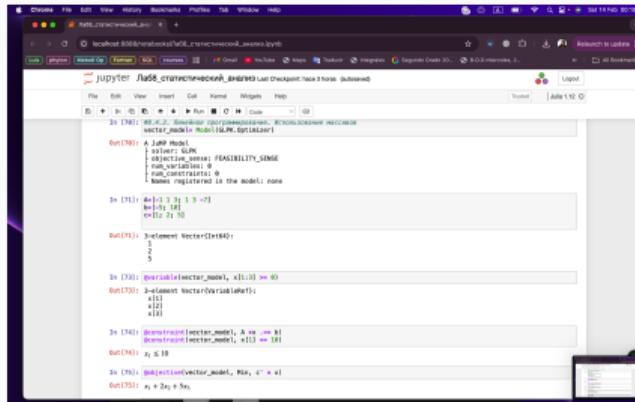
In [67]: #optimize1(model)
termination_status(model)

Out[67]: OPTIMAL;TerminationStatusCode = 1

In [68]: #new_value(x1);
#new_value(x2);
#new_value(x3);
```

Рис. 12: Задание 1. Линейное программирование

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface running on a Mac OS X desktop. The browser tab is titled "jupyter: Линейное программирование с использованием массивов". The notebook cell contains the following Python code:

```
In [1]: from pulp import *
In [2]: prob = LpProblem("Simplex", LpMinimize)
        prob += LpVariable("x1", lowBound=0), LpVariable("x2", lowBound=0)
        prob += LpConstraint(1, 2, 3, 3, 1.5, ">=")
        prob += LpConstraint(1, 2, 3, 3, 2, "<=")
        prob += LpConstraint(1, 2, 3, 3, 3, "<=")

In [3]: prob.writeLP("vector_model")

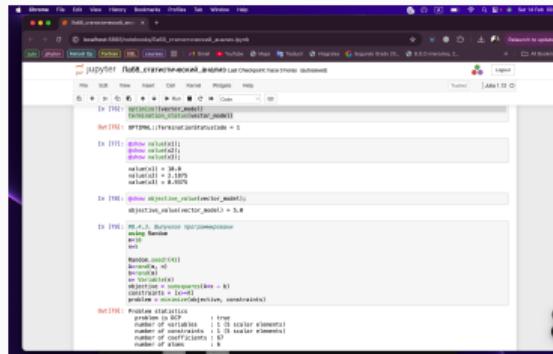
In [4]: prob.solve()
Out[4]: 0
In [5]: print(prob)
Out[5]:
LpProblem('Simplex', sense=LpMinimize)
Variables:
x1 (Type: Continuous, Lower bound: 0.0, Upper bound: None)
x2 (Type: Continuous, Lower bound: 0.0, Upper bound: None)

Constraints:
C1: (1, x1) + (2, x2) - 3.0 >= 1.5
C2: (1, x1) + (2, x2) - 3.0 <= 2.0
C3: (1, x1) + (2, x2) - 3.0 <= 3.0

Objective function:
0.0*x1 + 0.0*x2
```

Рис. 13: Задание 2. Линейное программирование. Использование массивов

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook window with several code cells. The code is related to convex optimization, specifically using the CVXPY library. It includes defining variables, setting up constraints, and solving the problem.

```
In [1]: import cvxpy as cp
x = cp.Variable(3)
y = cp.Variable(3)
z = cp.Variable(3)

# Objective function
obj = cp.Minimize(cp.sum_squares(x - y) + cp.sum_squares(y - z))

# Constraints
constraints = [cp.sum(x) == 1,
               cp.sum(y) == 1,
               cp.sum(z) == 1,
               x >= 0,
               y >= 0,
               z >= 0]

# Problem
prob = cp.Problem(obj, constraints)

# Solve
prob.solve(cp.SCS)
print("Optimal value = " + str(prob.value))
print("Optimal vector = " + str(prob.variables()))
```

In [2]: prob

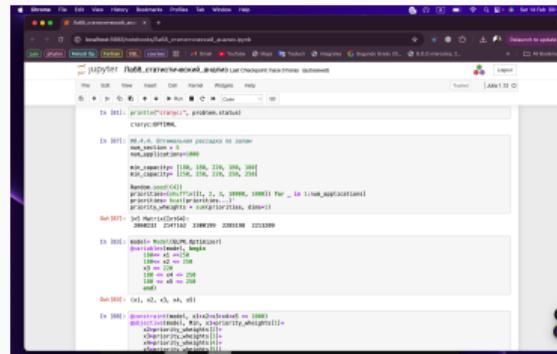
```
Optimization problem
variables: 9
constraints: 6
problem_size: 6x6
KKT matrix: 6x6
problems = 1
constraints = 6
```

In [3]: prob.stats

```
number of variables : 9
number of scalar variables : 3
number of linear constraints : 6
number of nonlinear constraints : 0
number of coefficients : 87
number of atoms : 9
```

Рис. 14: Задание 3. Выпуклое программирование

Выполнение лабораторной работы



```
curl -X POST http://127.0.0.1:5000/seating安排座位
{
    "seats": [
        {
            "x": 100,
            "y": 100
        },
        {
            "x": 200,
            "y": 100
        },
        {
            "x": 300,
            "y": 100
        },
        {
            "x": 400,
            "y": 100
        },
        {
            "x": 500,
            "y": 100
        },
        {
            "x": 600,
            "y": 100
        },
        {
            "x": 700,
            "y": 100
        },
        {
            "x": 800,
            "y": 100
        },
        {
            "x": 900,
            "y": 100
        }
    ],
    "people": [
        {
            "name": "Alice",
            "x": 100,
            "y": 100
        },
        {
            "name": "Bob",
            "x": 200,
            "y": 100
        },
        {
            "name": "Charlie",
            "x": 300,
            "y": 100
        },
        {
            "name": "David",
            "x": 400,
            "y": 100
        },
        {
            "name": "Eve",
            "x": 500,
            "y": 100
        },
        {
            "name": "Frank",
            "x": 600,
            "y": 100
        },
        {
            "name": "Grace",
            "x": 700,
            "y": 100
        },
        {
            "name": "Hank",
            "x": 800,
            "y": 100
        },
        {
            "name": "Ivy",
            "x": 900,
            "y": 100
        }
    ],
    "rows": [
        {
            "x": 100,
            "y": 100
        },
        {
            "x": 200,
            "y": 100
        },
        {
            "x": 300,
            "y": 100
        },
        {
            "x": 400,
            "y": 100
        },
        {
            "x": 500,
            "y": 100
        },
        {
            "x": 600,
            "y": 100
        },
        {
            "x": 700,
            "y": 100
        },
        {
            "x": 800,
            "y": 100
        },
        {
            "x": 900,
            "y": 100
        }
    ]
}
```

```
def seating安排座位(seats, people):
    # Initialize a matrix of seats
    seat_matrix = [[None] * len(seats) for _ in range(len(seats))]

    # Assign people to seats
    for person in people:
        seat_index = person['x'] // 100
        seat_matrix[seat_index][seat_index] = person['name']

    # Print the seating arrangement
    for row in seat_matrix:
        print(row)
```

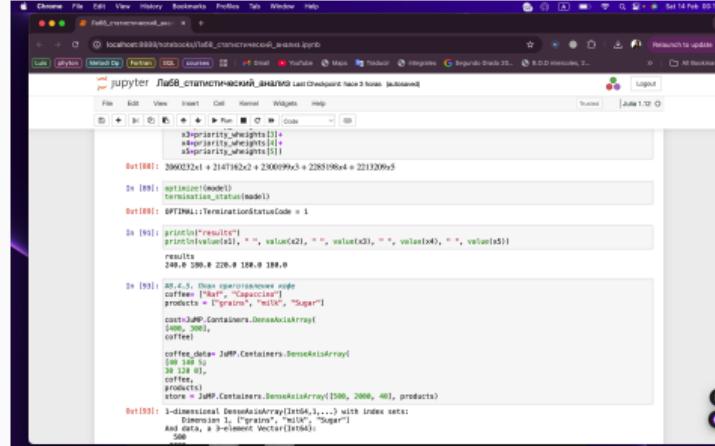
```
def main():
    # Load data from JSON file
    with open('seating安排座位.json') as f:
        data = json.load(f)

    # Call the seating function
    seating安排座位(data['seats'], data['people'])

if __name__ == '__main__':
    main()
```

Рис. 15: Задание 4. Оптимальная рассадка по залам

Выполнение лабораторной работы



The screenshot shows a Jupyter Notebook interface running in a browser window. The URL is `localhost:8888/notebooks/1688_статистический_анализ.ipynb`. The notebook has three cells:

- Cell 1:** Contains the definition of a function `slopeParity` which takes a list of weights and calculates a weighted sum of values x_1, x_2, x_3, x_4, x_5 .
- Cell 2:** Shows the result of running Cell 1, outputting the expression $300x_2 + 2147162x_2 + 230099x_3 + 2385198x_4 + 2213209x_5$.
- Cell 3:** Prints the results of the calculation, showing the total value as 248.0.

Cell 4 contains code to define a class `OptimalModel` that inherits from `TerminationStatusCode` and has a constructor that takes a string `model` and initializes `termination_status` to 1. It also includes a method `printValue` that prints the values of variables `x1` through `x5`.

Cell 5 contains code to create a JuMP container `coffee` with a 3D matrix of type `DenseMatrix{Int64}` of size 3000x2000x40, initialized with zeros. It then stores products like "Grains", "Milk", and "Sugar" into the container.

Cell 6 shows the resulting JuMP container `coffee` with dimensions `3000 2000 40`.

Cell 7 displays the structure of the container `coffee`, showing it is a 3-dimensional `DenseMatrix{Int64}` with index sets.

Рис. 16: Задание 5. План приготовления кофе

Выводы

В результате выполнения данной лабораторной работы я освоила пакеты Julia для решения задач оптимизации.

Список литературы

1. JuliaLang [Электронный ресурс]. 2024 JuliaLang.org contributors.
URL:<https://julialang.org/>(дата обращения: 11.10.2024).
2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.orgcontributors.
URL:<https://docs.julialang.org/en/v1/>(дата обращения:11.10.2024).