Лабораторная работа № 8

Оптимизация

Беличева Дарья Михайловна

Содержание

Список иллюстраций

1 Цель работы

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

2 Задание

1. Используя JupyterLab, повторите примеры.

2. Выполните задания для самостоятельной работы.

3 Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [1]. 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 [2].

4 Выполнение лабораторной работы

Выполним примеры из лабораторной работы (рис. 4.1-4.11).

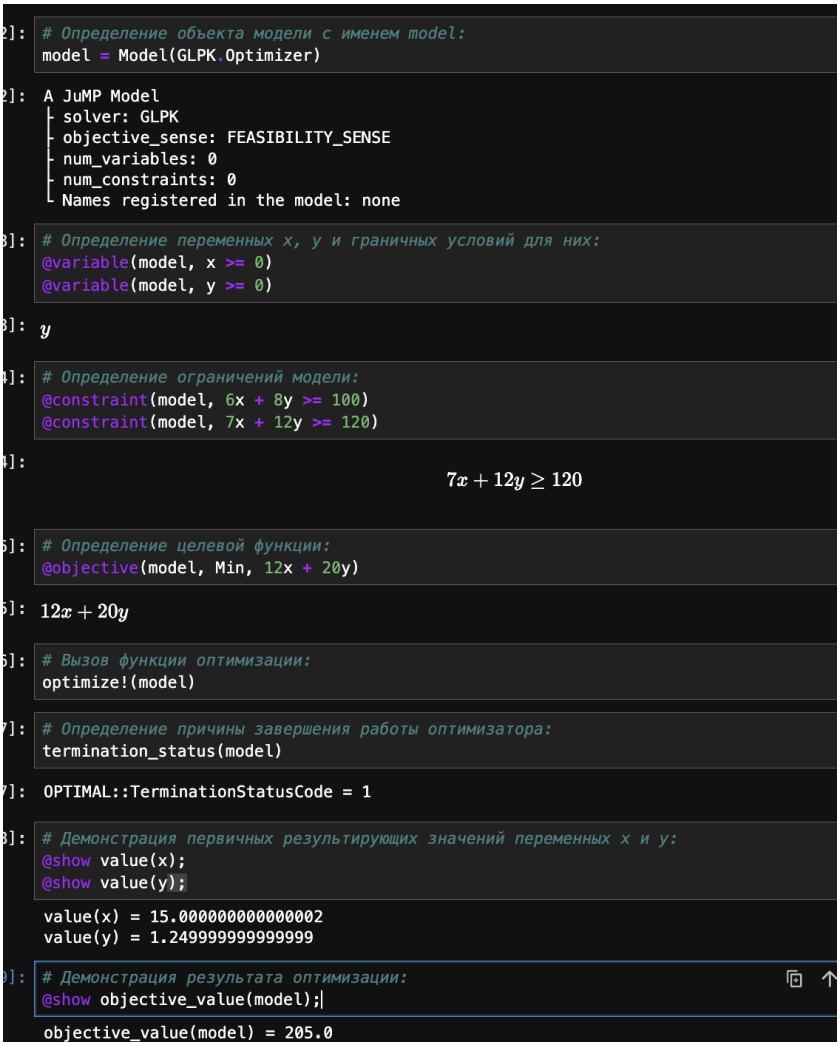


Рис. 4.1: Линейное программирование

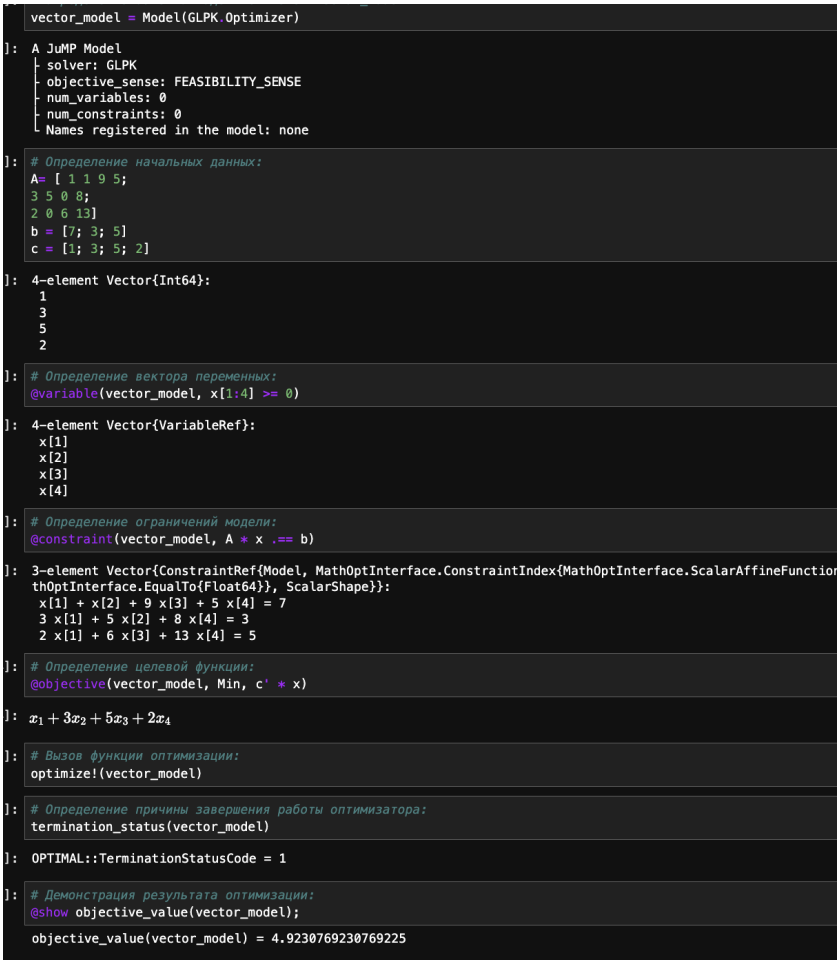
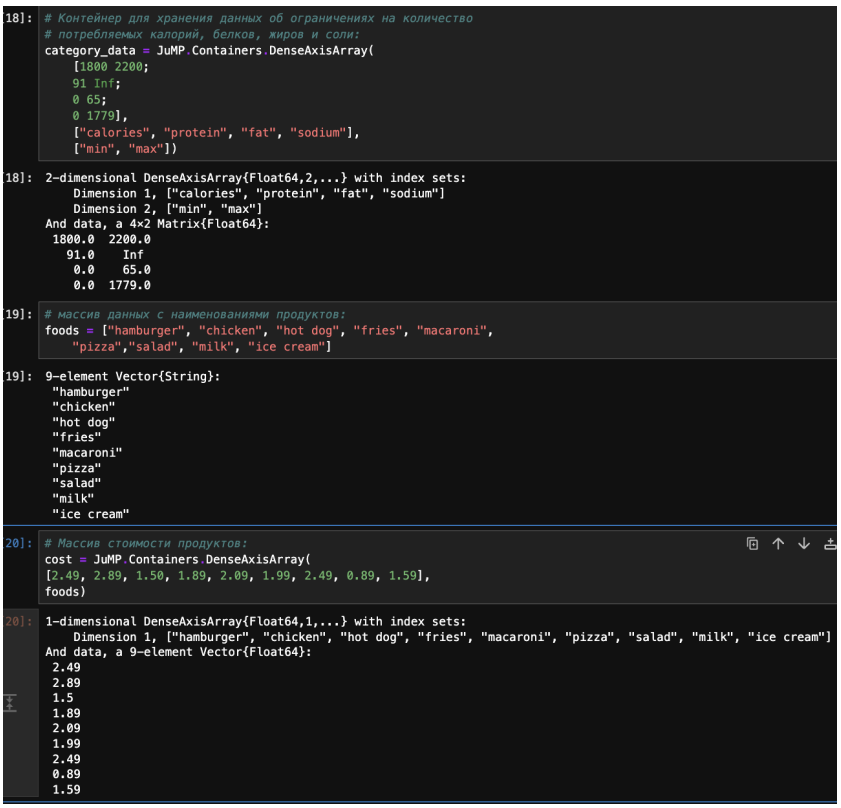
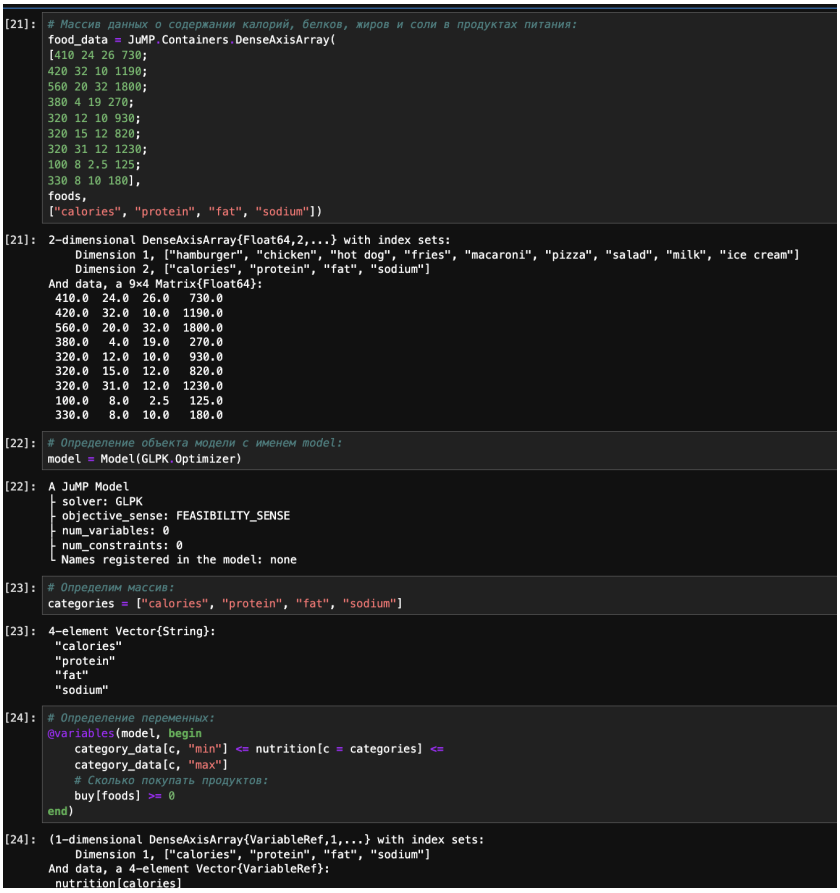


Рис. 4.2: Векторизованные ограничения и целевая функция оптимизации



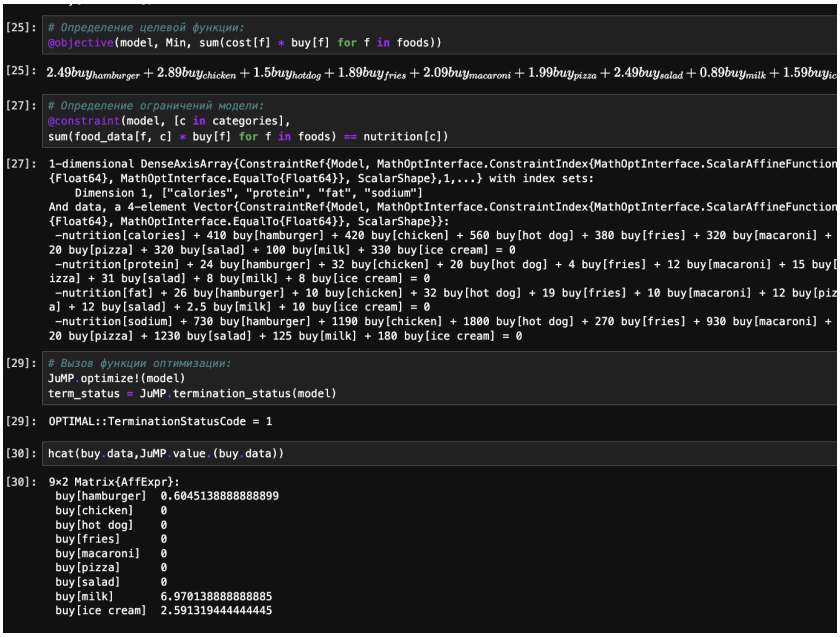
```
[18]: # Контейнер для хранения данных об ограничениях на количество
      # потребляемых калорий, белков, жиров и соли:
      category_data = JuMP.Containers.DenseAxisArray(
          [1800 2200;
          91 Inf;
          0 65;
          0 1779],
          ["calories", "protein", "fat", "sodium"],
          ["min", "max"])

[18]: 2-dimensional DenseAxisArray{Float64,2,...} with index sets:
          Dimension 1, ["calories", "protein", "fat", "sodium"]
          Dimension 2, ["min", "max"]
      And data, a 4×2 Matrix{Float64}:
       1800.0  2200.0
         91.0     Inf
          0.0    65.0
          0.0  1779.0

[19]: # массив данных с наименованиями продуктов:
      foods = ["hamburger", "chicken", "hot dog", "fries", "macaroni",
          "pizza","salad", "milk", "ice cream"]

[19]: 9-element Vector{String}:
       "hamburger"
       "chicken"
       "hot dog"
       "fries"
       "macaroni"
       "pizza"
       "salad"
       "milk"
       "ice cream"

[20]: # Массив стоимости продуктов:
      cost = JuMP.Containers.DenseAxisArray(
      [2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59],
      foods)

[20]: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
          Dimension 1, ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]
      And data, a 9-element Vector{Float64}:
       2.49
       2.89
       1.5
       1.89
       2.09
       1.99
       2.49
       0.89
       1.59
```

Рис. 4.3: Оптимизация рациона питания



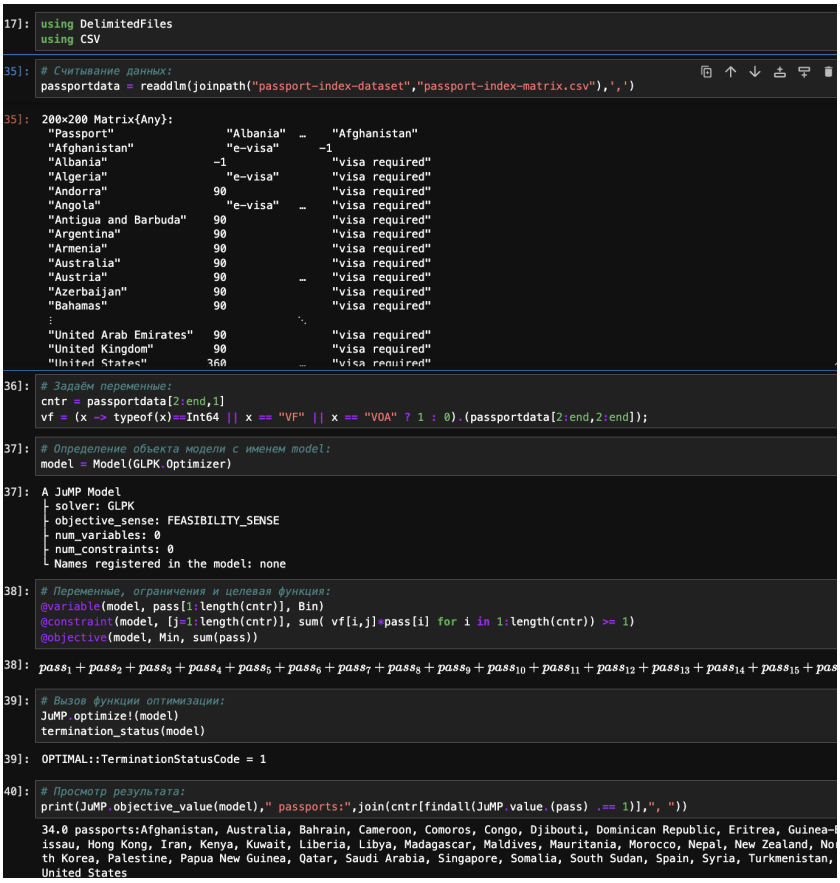
```
[21]: # Массив данных о содержании калорий, белков, жиров и соли в продуктах питания:
food_data = JuMP.Containers.DenseAxisArray(
[410 24 26 730;
420 32 10 1190;
560 20 32 1800;
380 4 19 270;
320 12 10 930;
320 15 12 820;
320 31 12 1230;
100 8 2.5 125;
330 8 10 180],
foods,
["calories", "protein", "fat", "sodium"])

[21]: 2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]
    Dimension 2, ["calories", "protein", "fat", "sodium"]
And data, a 9×4 Matrix{Float64}:
 410.0  24.0  26.0   730.0
 420.0  32.0  10.0  1190.0
 560.0  20.0  32.0  1800.0
 380.0   4.0  19.0   270.0
 320.0  12.0  10.0   930.0
 320.0  15.0  12.0   820.0
 320.0  31.0  12.0  1230.0
 100.0   8.0   2.5   125.0
 330.0   8.0  10.0   180.0

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

[22]: A JuMP Model
├ solver: GLPK
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none

[23]: # Определим массив:
categories = ["calories", "protein", "fat", "sodium"]

[23]: 4-element Vector{String}:
 "calories"
 "protein"
 "fat"
 "sodium"

[24]: # Определение переменных:
@variables(model, begin
    category_data[c, "min"] <= nutrition[c = categories] <=
    category_data[c, "max"]
    # Сколько покупать продуктов:
    buy[foods] >= 0
end)

[24]: (1-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
    Dimension 1, ["calories", "protein", "fat", "sodium"]
And data, a 4-element Vector{VariableRef}:
 nutrition[calories]
```

Рис. 4.4: Оптимизация рациона питания



```
[25]: # Определение целевой функции:
      @objective(model, Min, sum(cost[f] * buy[f] for f in foods))

[25]: 2.49buy_hamburger + 2.89buy_chicken + 1.5buy_hotdog + 1.89buy_fries + 2.09buy_macaroni + 1.99buy_pizza + 2.49buy_salad + 0.89buy_milk + 1.59buy_ic

[27]: # Определение ограничений модели:
      @constraint(model, [c in categories],
      sum(food_data[f, c] * buy[f] for f in foods) == nutrition[c])

[27]: 1-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction
      {Float64}, MathOptInterface.EqualTo{Float64}}, ScalarShape},1,...} with index sets:
          Dimension 1, ["calories", "protein", "fat", "sodium"]
      And data, a 4-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction
      {Float64}, MathOptInterface.EqualTo{Float64}}, ScalarShape}}:
       -nutrition[calories] + 410 buy[hamburger] + 420 buy[chicken] + 560 buy[hot dog] + 380 buy[fries] + 320 buy[macaroni] +
      20 buy[pizza] + 320 buy[salad] + 100 buy[milk] + 330 buy[ice cream] = 0
       -nutrition[protein] + 24 buy[hamburger] + 32 buy[chicken] + 20 buy[hot dog] + 4 buy[fries] + 12 buy[macaroni] + 15 buy[
      izza] + 31 buy[salad] + 8 buy[milk] + 8 buy[ice cream] = 0
       -nutrition[fat] + 26 buy[hamburger] + 10 buy[chicken] + 32 buy[hot dog] + 19 buy[fries] + 10 buy[macaroni] + 12 buy[piz
      a] + 12 buy[salad] + 2.5 buy[milk] + 10 buy[ice cream] = 0
       -nutrition[sodium] + 730 buy[hamburger] + 1190 buy[chicken] + 1800 buy[hot dog] + 270 buy[fries] + 930 buy[macaroni] +
      20 buy[pizza] + 1230 buy[salad] + 125 buy[milk] + 180 buy[ice cream] = 0

[29]: # Вызов функции оптимизации:
      JuMP.optimize!(model)
      term_status = JuMP.termination_status(model)

[29]: OPTIMAL::TerminationStatusCode = 1

[30]: hcat(buy.data,JuMP.value.(buy.data))

[30]: 9×2 Matrix{AffExpr}:
       buy[hamburger]  0.6045138888888899
       buy[chicken]    0
       buy[hot dog]    0
       buy[fries]      0
       buy[macaroni]   0
       buy[pizza]      0
       buy[salad]      0
       buy[milk]       6.970138888888885
       buy[ice cream]  2.591319444444445
```

Рис. 4.5: Оптимизация рациона питания



```
17]: using DelimitedFiles
     using CSV

35]: # Считывание данных:
     passportdata = readdlm(joinpath("passport-index-dataset","passport-index-matrix.csv"),',')

35]: 200×200 Matrix{Any}:
      "Passport"                "Albania"  …    "Afghanistan"
      "Afghanistan"             "e-visa"        -1
      "Albania"                -1                "visa required"
      "Algeria"                 "e-visa"         "visa required"
      "Andorra"                90                "visa required"
      "Angola"                  "e-visa"   …     "visa required"
      "Antigua and Barbuda"    90                "visa required"
      "Argentina"              90                "visa required"
      "Armenia"                90                "visa required"
      "Australia"              90                "visa required"
      "Austria"                90           …    "visa required"
      "Azerbaijan"             90                "visa required"
      "Bahamas"                90                "visa required"
      ⋮                                     ⋱
      "United Arab Emirates"   90                "visa required"
      "United Kingdom"         90                "visa required"
      "United States"          360          …    "visa required"

36]: # Задаём переменные:
     cntr = passportdata[2:end,1]
     vf = (x -> typeof(x)==Int64 || x == "VF" || x == "VOA" ? 1 : 0).(passportdata[2:end,2:end]);

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

37]: A JuMP Model
     ├ solver: GLPK
     ├ objective_sense: FEASIBILITY_SENSE
     ├ num_variables: 0
     ├ num_constraints: 0
     └ Names registered in the model: none

38]: # Переменные, ограничения и целевая функция:
     @variable(model, pass[1:length(cntr)], Bin)
     @constraint(model, [j=1:length(cntr)], sum( vf[i,j]*pass[i] for i in 1:length(cntr)) >= 1)
     @objective(model, Min, sum(pass))

38]: pass_1 + pass_2 + pass_3 + pass_4 + pass_5 + pass_6 + pass_7 + pass_8 + pass_9 + pass_10 + pass_11 + pass_12 + pass_13 + pass_14 + pass_15 + pas

39]: # Вызов функции оптимизации:
     JuMP.optimize!(model)
     termination_status(model)

39]: OPTIMAL::TerminationStatusCode = 1

40]: # Просмотр результата:
     print(JuMP.objective_value(model)," passports:",join(cntr[findall(JuMP.value.(pass) .== 1)],", "))

     34.0 passports:Afghanistan, Australia, Bahrain, Cameroon, Comoros, Congo, Djibouti, Dominican Republic, Eritrea, Guinea-B
     issau, Hong Kong, Iran, Kenya, Kuwait, Liberia, Libya, Madagascar, Maldives, Mauritania, Morocco, Nepal, New Zealand, Nor
     th Korea, Palestine, Papua New Guinea, Qatar, Saudi Arabia, Singapore, Somalia, South Sudan, Spain, Syria, Turkmenistan,
     United States
```

Рис. 4.6: Путешествие по миру

```
data_table = XLSX.readtable("data/stock_prices.xlsx", "Sheet2")
```

```
XLSX.DataTable(Any[Any[101.93, 102.8, 107.71, 107.17, 102.78, 105.67, 108.22, 110.97, 112.53, 110.51, 115.91,
7.94], Any[137.95, 143.8, 150.04, 149.01, 165.71, 167.33, 162.5, 161.89, 162.28, 169.6, 165.98, 164.34, 166.69
26, 152.29, 156.82, 157.76, 166.52, 170.41, 170.42, 172.97, 174.97, 172.91, 186.12, 191.05, 189.95]], [:MSFT,
L], Dict(:FB => 2, :MSFT => 1, :AAPL => 3))
```

```
df = DataFrame(data_table)
```

2	102.8	143.8	152.29
3	107.71	150.04	156.82
4	107.17	149.01	157.76
5	102.78	165.71	166.52
6	105.67	167.33	170.41
7	108.22	162.5	170.42
8	110.97	161.89	172.97
9	112.53	162.28	174.97
10	110.51	169.6	172.91
11	115.91	165.98	186.12
12	117.05	164.34	191.05

```
# Построение графика:
plot(df[!,:MSFT],label="Microsoft")
plot!(df[!,:AAPL],label="Apple")
plot!(df[!,:FB],label="FB")
```

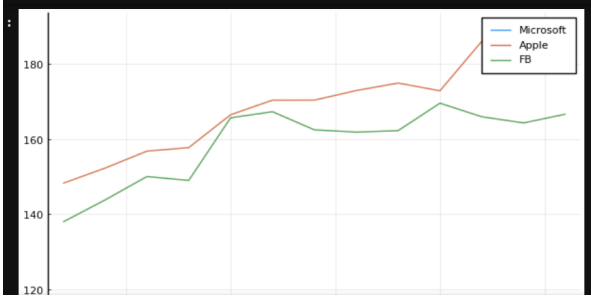


Рис. 4.7: Портфельные инвестиции

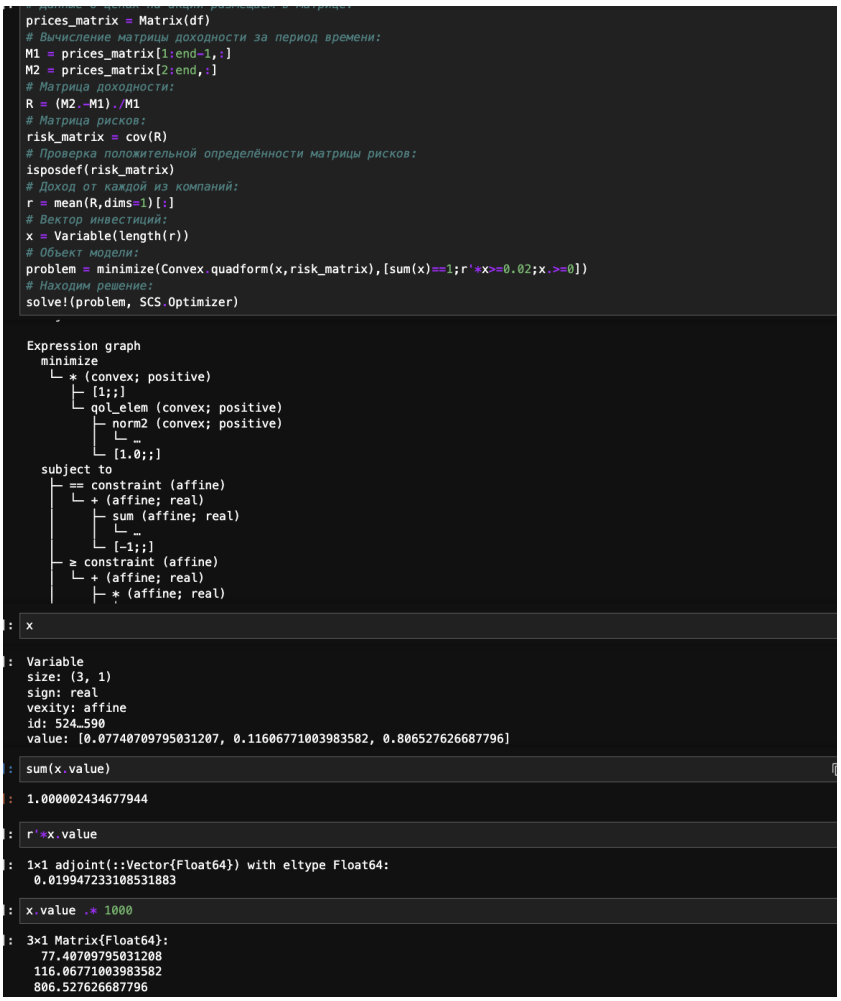


Рис. 4.8: Портфельные инвестиции

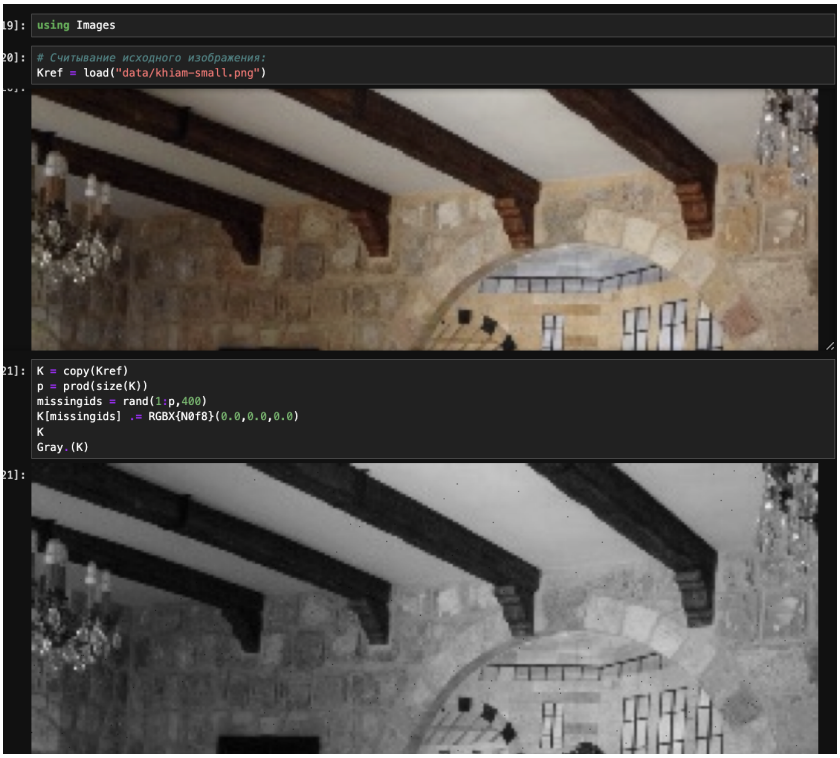


Рис. 4.9: Восстановление изображения

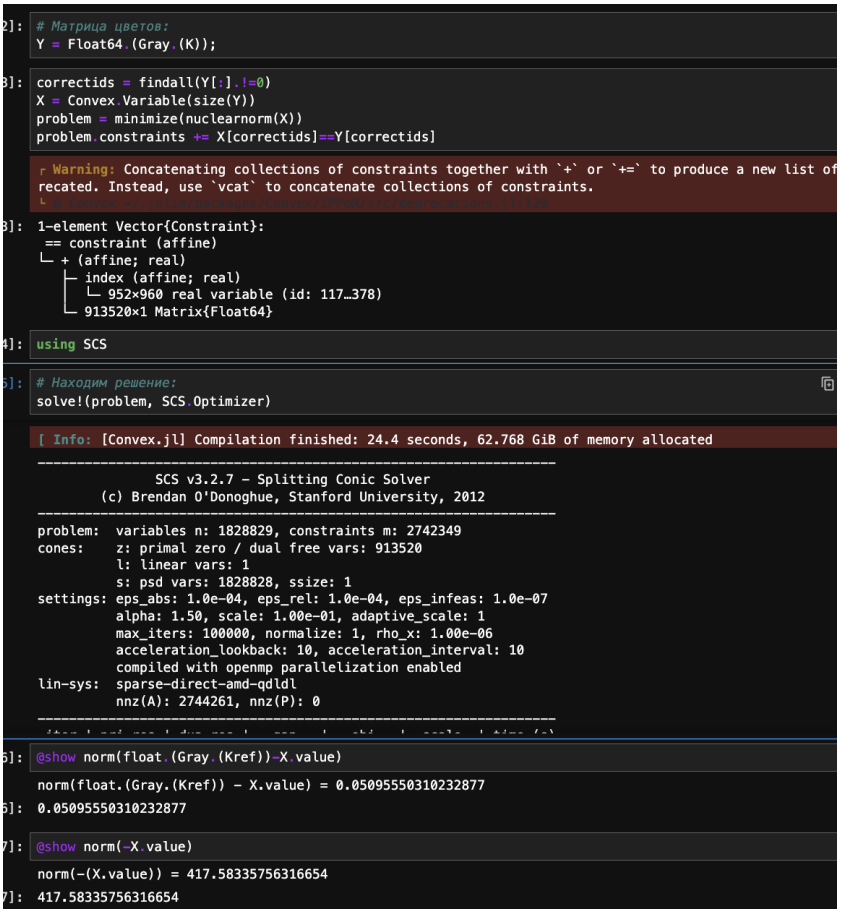


Рис. 4.10: Восстановление изображения

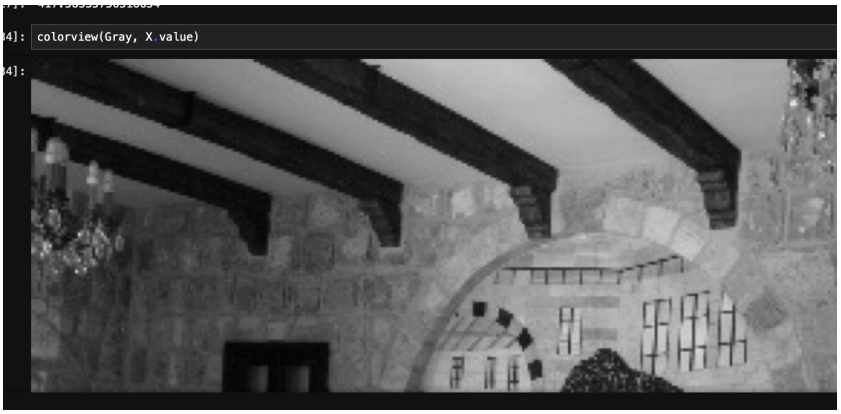


Рис. 4.11: Восстановление изображения

Теперь выполним задания для самостоятельный работы (рис. 4.12-4.16).

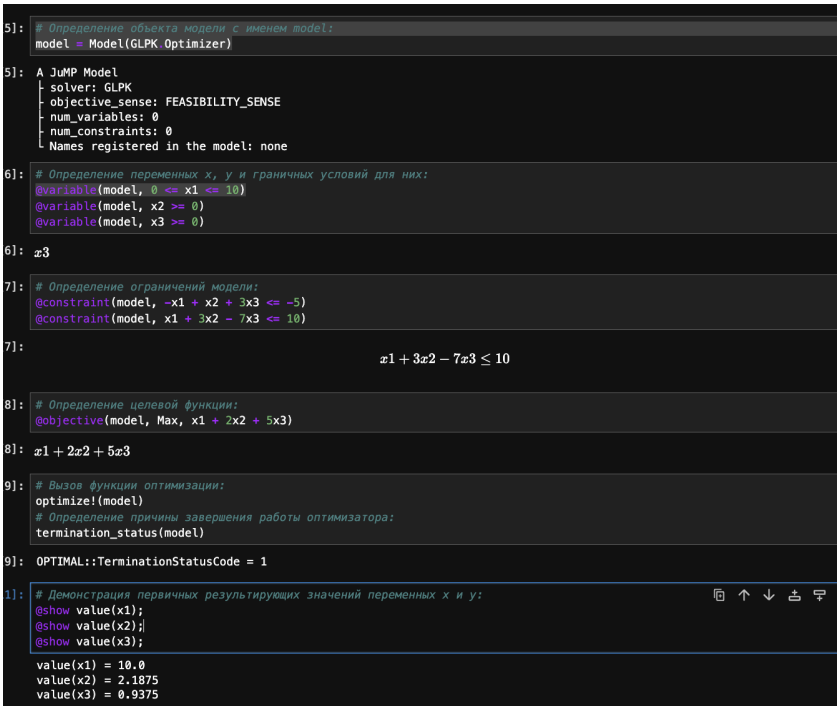


Рис. 4.12: Задание 1. Линейное программирование

Рис. 4.13: Задание 2. Линейное программирование. Использование массивов

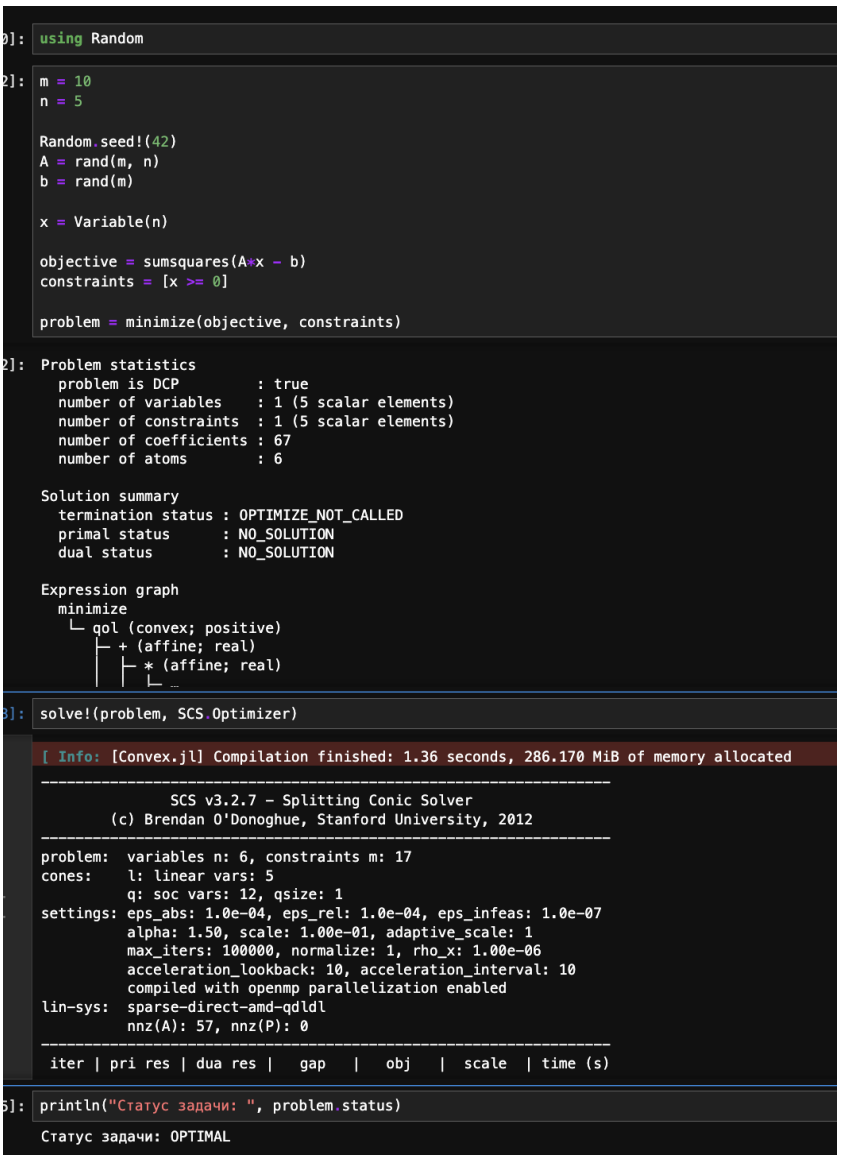


Рис. 4.14: Задание 3. Выпуклое программирование

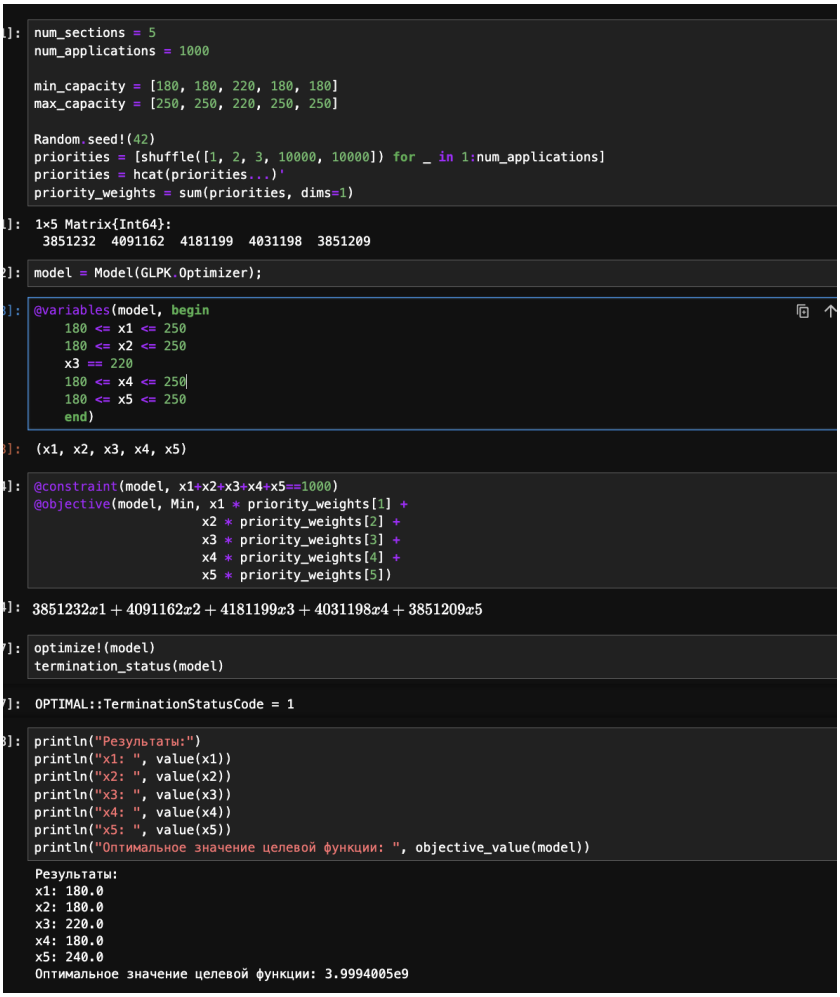


Рис. 4.15: Задание 4. Оптимальная рассадка по залам

```
]: coffee = ["Raf", "Cappuccino"]
   products = ["grains", "milk", "sugar"]

   cost = JuMP.Containers.DenseAxisArray(
       [400, 300],
       coffee
   )
   coffee_data = JuMP.Containers.DenseAxisArray(
       [40 140 5;
       30 120 0],
       coffee,
       products
   )
   store = JuMP.Containers.DenseAxisArray([500, 2000, 40],products)

]: 1-dimensional DenseAxisArray{Int64,1,...} with index sets:
       Dimension 1, ["grains", "milk", "sugar"]
   And data, a 3-element Vector{Int64}:
     500
    2000
      40

]: model = Model(GLPK.Optimizer)
   @variables(model, begin
       buy[coffee] >= 0
   end)
   @objective(model, Max, sum(cost[c] * buy[c] for c in coffee))
   @constraint(model, [p in products],
   sum(coffee_data[c, p] * buy[c] for c in coffee) <= store[p])
   @constraint(model, coffee_data["Raf", "sugar"] * buy["Raf"] == 40)

]:
```

$$5buy_{Raf} = 40$$

```
]: JuMP.optimize!(model)
   term_status = JuMP.termination_status(model)
   hcat(buy.data, JuMP.value.(buy.data))

]: 2×2 Matrix{AffExpr}:
    buy[Raf]         8
    buy[Cappuccino]  6
```

Рис. 4.16: Задание 5. План приготовления кофе

5 Выводы

В резцльтате выполнения данной лабораторной работы я оосвоила 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.org contributors. URL: https://docs.julialang.org/en/v1/ (дата обращения: 11.10.2024).