

Лабораторная работа № 8

Оптимизация

Беличева Дарья Михайловна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Линейное программирование	7
4.2	Векторизованные ограничения и целевая функция оптимизации .	8
4.3	Оптимизация рациона питания	9
4.4	Оптимизация рациона питания	10
4.5	Оптимизация рациона питания	11
4.6	Путешествие по миру	11
4.7	Портфельные инвестиции	12
4.8	Портфельные инвестиции	13
4.9	Восстановление изображения	14
4.10	Восстановление изображения	15
4.11	Восстановление изображения	15
4.12	Задание 1. Линейное программирование	16
4.13	Задание 2. Линейное программирование. Использование массивов	17
4.14	Задание 3. Выпуклое программирование	18
4.15	Задание 4. Оптимальная рассадка по залам	19
4.16	Задание 5. План приготовления кофе	20

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).

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

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

3]: # Определение переменных x, y и граничных условий для них:
@variable(model, x >= 0)
@variable(model, y >= 0)

3]: y

4]: # Определение ограничений модели:
@constraint(model, 6x + 8y >= 100)
@constraint(model, 7x + 12y >= 120)

4]:

$$7x + 12y \geq 120$$


5]: # Определение целевой функции:
@objective(model, Min, 12x + 20y)

5]: 12x + 20y

5]: # Вызов функции оптимизации:
optimize!(model)

7]: # Определение причины завершения работы оптимизатора:
termination_status(model)

7]: OPTIMAL::TerminationStatusCode = 1

8]: # Демонстрация первичных результирующих значений переменных x и y:
@show value(x);
@show value(y);

value(x) = 15.000000000000002
value(y) = 1.2499999999999999

9]: # Демонстрация результата оптимизации:
@show objective_value(model);

objective_value(model) = 205.0
```

Рис. 4.1: Линейное программирование

```

vector_model = Model(GLPK.Optimizer)

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

]: # Определение начальных данных:
A = [ 1 1 9 5;
      3 5 0 8;
      2 0 6 13]
b = [7; 3; 5]
c = [1; 3; 5; 2]

]: 4-element Vector{Int64}:
 1
 3
 5
 2

]: # Определение вектора переменных:
@variable(vector_model, x[1:4] >= 0)

]: 4-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
 x[4]

]: # Определение ограничений модели:
@constraint(vector_model, A * x .== b)

]: 3-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{MathOptInterface.EqualTo{Float64}}}, ScalarShape}}:
 x[1] + x[2] + 9 x[3] + 5 x[4] = 7
 3 x[1] + 5 x[2] + 8 x[3] + 13 x[4] = 3
 2 x[1] + 6 x[3] + 13 x[4] = 5

]: # Определение целевой функции:
@objective(vector_model, Min, c' * x)

]:  $x_1 + 3x_2 + 5x_3 + 2x_4$ 

]: # Вызов функции оптимизации:
optimize!(vector_model)

]: # Определение причины завершения работы оптимизатора:
termination_status(vector_model)

]: OPTIMAL::TerminationStatusCode = 1

]: # Демонстрация результата оптимизации:
@show objective_value(vector_model);
objective_value(vector_model) = 4.9230769230769225

```

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 4x2 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 9x4 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]: # Определена область 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 + 1.59buy_hotdog + 1.89buy_fries + 2.09buy_macaroni + 1.99buy_pizza + 2.49buy_salad + 0.89buy_milk + 1.59buy_ice_cream

[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{1,...} with index sets:
Dimension 1, ["calories", "protein", "fat", "sodium"]}}:
-nutrition[calories] + 410 buy[hamburger] + 420 buy[chicken] + 560 buy[hot dog] + 380 buy[fries] + 320 buy[macaroni] + 20 buy[pizza] + 320 buy[salad] + 180 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[pizza] + 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[pizza] + 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]: # Вывод функции оптимизации:
JuMP.optimize!(model)
term_status = JuMP.termination_status(model)

[29]: OPTIMAL::TerminationStatusCode = 1

[30]: hcat(buy_data, JuMP.value.(buy_data))

[30]: 9x2 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]  2.5913194444444445

```

Рис. 4.5: Оптимизация рациона питания

```

[17]: using DelimitedFiles
using CSV

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

[35]: 200x200 Matrix{Any}:
"Passport"      "Albania"  -   "Afghanistan"
"Afghanistan"   "e-visa"   -1
"Albania"       -1         "visa required"
"Algeria"       90         "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 + pass_16 + pass_17 + pass_18 + pass_19 + pass_20 + pass_21 + pass_22 + pass_23 + pass_24 + pass_25 + pass_26 + pass_27 + pass_28 + pass_29 + pass_30 + pass_31 + pass_32 + pass_33 + pass_34 + pass_35 + pass_36 + pass_37 + pass_38 + pass_39 + pass_40 + pass_41 + pass_42 + pass_43 + pass_44 + pass_45 + pass_46 + pass_47 + pass_48 + pass_49 + pass_50 + pass_51 + pass_52 + pass_53 + pass_54 + pass_55 + pass_56 + pass_57 + pass_58 + pass_59 + pass_60 + pass_61 + pass_62 + pass_63 + pass_64 + pass_65 + pass_66 + pass_67 + pass_68 + pass_69 + pass_70 + pass_71 + pass_72 + pass_73 + pass_74 + pass_75 + pass_76 + pass_77 + pass_78 + pass_79 + pass_80 + pass_81 + pass_82 + pass_83 + pass_84 + pass_85 + pass_86 + pass_87 + pass_88 + pass_89 + pass_90 + pass_91 + pass_92 + pass_93 + pass_94 + pass_95 + pass_96 + pass_97 + pass_98 + pass_99 + pass_100 + pass_101 + pass_102 + pass_103 + pass_104 + pass_105 + pass_106 + pass_107 + pass_108 + pass_109 + pass_110 + pass_111 + pass_112 + pass_113 + pass_114 + pass_115 + pass_116 + pass_117 + pass_118 + pass_119 + pass_120 + pass_121 + pass_122 + pass_123 + pass_124 + pass_125 + pass_126 + pass_127 + pass_128 + pass_129 + pass_130 + pass_131 + pass_132 + pass_133 + pass_134 + pass_135 + pass_136 + pass_137 + pass_138 + pass_139 + pass_140 + pass_141 + pass_142 + pass_143 + pass_144 + pass_145 + pass_146 + pass_147 + pass_148 + pass_149 + pass_150 + pass_151 + pass_152 + pass_153 + pass_154 + pass_155 + pass_156 + pass_157 + pass_158 + pass_159 + pass_160 + pass_161 + pass_162 + pass_163 + pass_164 + pass_165 + pass_166 + pass_167 + pass_168 + pass_169 + pass_170 + pass_171 + pass_172 + pass_173 + pass_174 + pass_175 + pass_176 + pass_177 + pass_178 + pass_179 + pass_180 + pass_181 + pass_182 + pass_183 + pass_184 + pass_185 + pass_186 + pass_187 + pass_188 + pass_189 + pass_190 + pass_191 + pass_192 + pass_193 + pass_194 + pass_195 + pass_196 + pass_197 + pass_198 + pass_199 + pass_200

[39]: # Вывод 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)], ", "))

[40]: 34.0 passports:Afghanistan, Australia, Bahrain, Cameroon, Comoros, Congo, Djibouti, Dominican Republic, Eritrea, Guinea-Bissau, Hong Kong, Iran, Kenya, Kuwait, Liberia, Libya, Madagascar, Maldives, Mauritania, Morocco, Nepal, New Zealand, North Korea, Palestine, Papua New Guinea, Qatar, Saudi Arabia, Singapore, Somalia, South Sudan, Spain, Syria, Turkmenistan, United States

```

Рис. 4.6: Путешествие по миру

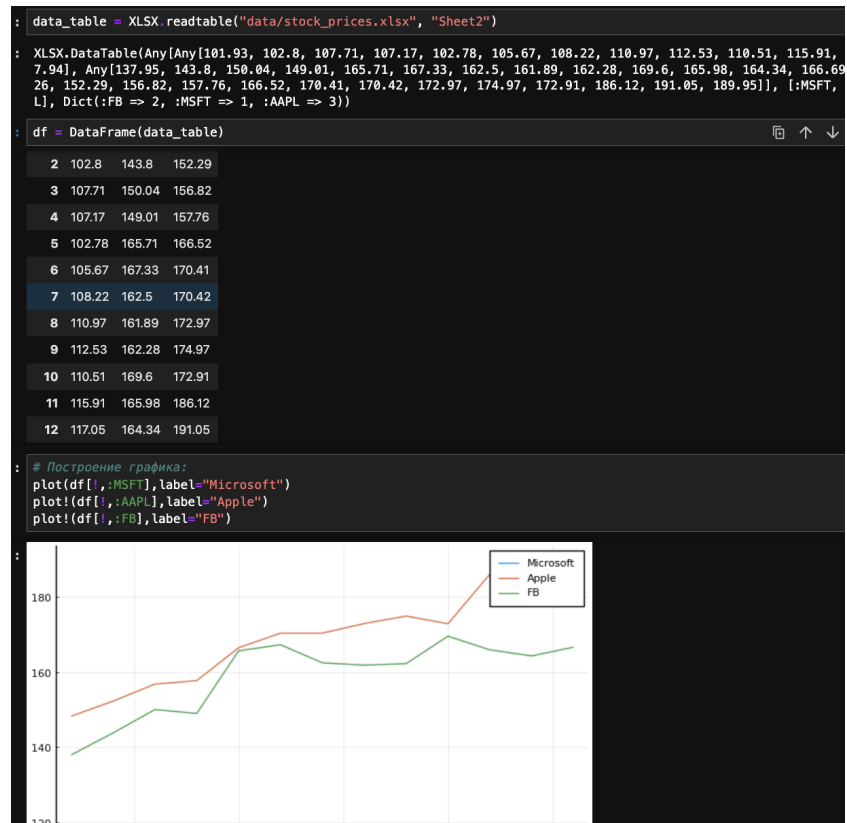


Рис. 4.7: Портфельные инвестиции

```

prices_matrix = Matrix(df)
# Вычисление матрицы доходности за период времени:
M1 = prices_matrix[1:end-1,:]
M2 = prices_matrix[2:end,:]
# Матрица доходности:
R = (M2 - M1) ./ M1
# Матрица рисков:
risk_matrix = cov(R)
# Проверка положительной определённости матрицы рисков:
isposdef(risk_matrix)
# Доход от каждой из компаний:
r = mean(R,dims=1)[:]
# Вектор инвестиций:
x = Variable(length(r))
# Объект модели:
problem = minimize(Convex.quadform(x,risk_matrix),[sum(x)==1;r'*x>=0.02;x.>=0])
# Находим решение:
solve!(problem, SCS.Optimizer)

Expression graph
minimize
└─ * (convex; positive)
   └─ [1;]
      └─ qol_elem (convex; positive)
         └─ norm2 (convex; positive)
            └─ [1.0;]
subject to
└─ == constraint (affine)
   └─ + (affine; real)
      └─ sum (affine; real)
         └─ [-1;]
└─ ≥ constraint (affine)
   └─ + (affine; real)
      └─ * (affine; real)
         └─ r' * x

x
Variable
size: (3, 1)
sign: real
vexity: affine
id: 524_590
value: [0.07740709795031207, 0.11606771003983582, 0.806527626687796]

sum(x.value)
1.000002434677944

r'*x.value
1x1 adjoint(::Vector{Float64}) with eltype Float64:
 0.019947233108531883

x.value .* 1000
3x1 Matrix{Float64}:
 77.40709795031208
116.06771003983582
806.527626687796

```

Рис. 4.8: Портфельные инвестиции

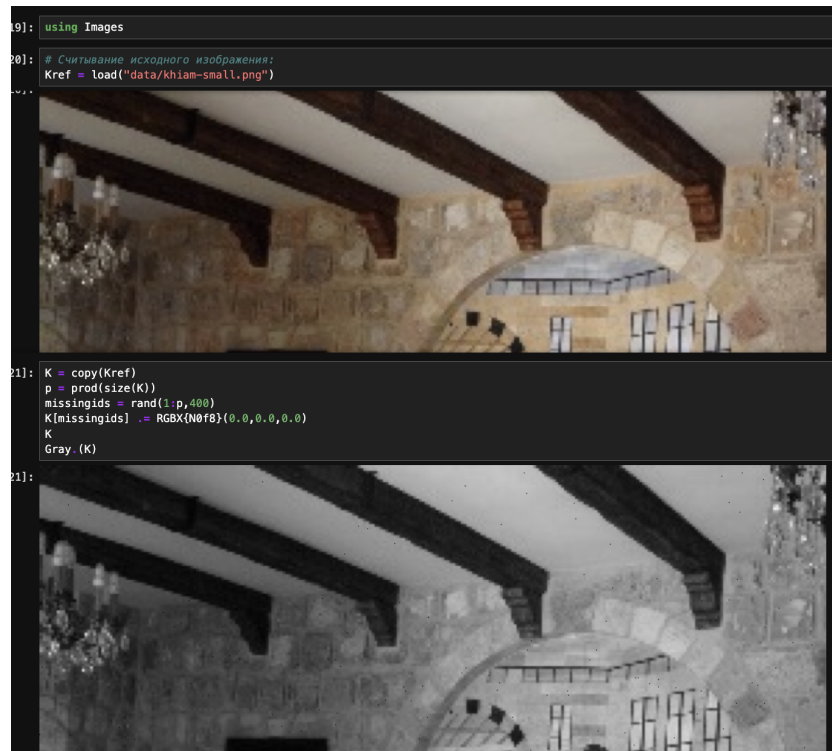


Рис. 4.9: Восстановление изображения

```

2]: # Матрица цветов:
   Y = Float64.(Gray.(K));

3]: correctids = findall(Y[:,!]=0)
   X = Convex.Variable(size(Y))
   problem = minimize(nuclearnorm(X))
   problem.constraints += X[correctids]==Y[correctids]

Warning: Concatenating collections of constraints together with `+` or `+=` to produce a new list of
recated. Instead, use `vcat` to concatenate collections of constraints.

3]: 1-element Vector{Constraint}:
  == constraint (affine)
    + (affine; real)
      index (affine; real)
        952x960 real variable (id: 117_378)
        913520x1 Matrix{Float64}

4]: using SCS

5]: # Находим решение:
   solve!(problem, SCS.Optimizer)

[ Info: [Convex.jl] Compilation finished: 24.4 seconds, 62.768 GiB of memory allocated

=====
SCS v3.2.7 - Splitting Conic Solver
(c) Brendan O'Donoghue, Stanford University, 2012

problem: variables n: 1828829, constraints m: 2742349
cones:    z: primal zero / dual free vars: 913520
          l: linear vars: 1
          s: psd vars: 1828828, ssize: 1
settings: eps_abs: 1.0e-04, eps_rel: 1.0e-04, eps_infeas: 1.0e-07
          alpha: 1.50, scale: 1.00e-01, adaptive_scale: 1
          max_iters: 100000, normalize: 1, rho_x: 1.00e-06
          acceleration_lookback: 10, acceleration_interval: 10
          compiled with openmp parallelization enabled
          lin-sys: sparse-direct-amd-qdldl
          nnz(A): 2744261, nnz(P): 0
=====

6]: @show norm(float.(Gray.(Kref))-X.value)
   norm(float.(Gray.(Kref)) - X.value) = 0.05095550310232877

6]: 0.05095550310232877

7]: @show norm(-(X.value))
   norm(-(X.value)) = 417.58335756316654

7]: 417.58335756316654

```

Рис. 4.10: Восстановление изображения

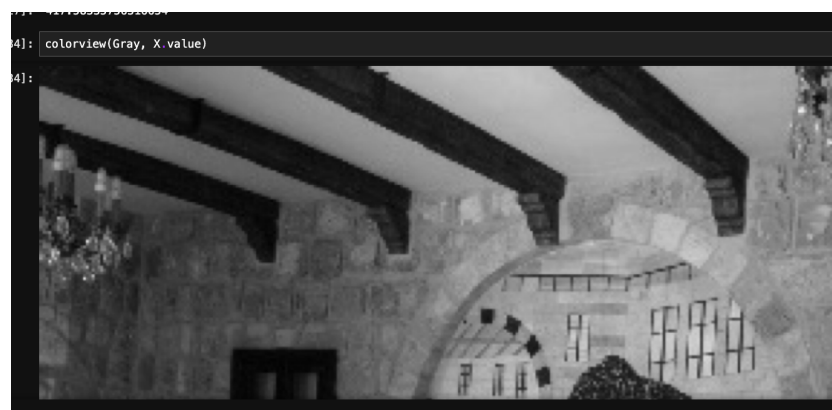


Рис. 4.11: Восстановление изображения

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

```

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

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

61: # Определение переменных x, y и граничных условий для них:
@variable(model, 0 <= x1 <= 10)
@variable(model, x2 >= 0)
@variable(model, x3 >= 0)

61: x3

71: # Определение ограничений модели:
@constraint(model, -x1 + x2 + 3x3 <= -5)
@constraint(model, x1 + 3x2 - 7x3 <= 10)

71:

$$x1 + 3x2 - 7x3 \leq 10$$


81: # Определение целевой функции:
@objective(model, Max, x1 + 2x2 + 5x3)

81:  $x1 + 2x2 + 5x3$ 

91: # Вызов функции оптимизации:
optimize!(model)
# Определение причины завершения работы оптимизатора:
termination_status(model)

91: OPTIMAL::TerminationStatusCode = 1

11: # Демонстрация первичных результирующих значений переменных x и y:
@show value(x1);
@show value(x2);
@show value(x3);

value(x1) = 10.0
value(x2) = 2.1875
value(x3) = 0.9375

```

Рис. 4.12: Задание 1. Линейное программирование


```

1): # Определение объекта модели с именем model:
   vector_model = Model(GLPK.Optimizer)

2): A JuMP Model
   | solver: GLPK
   | objective_sense: FEASIBILITY_SENSE
   | num_variables: 0
   | num_constraints: 0
   | Names registered in the model: none

3): A = [-1 1 3; 1 3 -7]
   b = [-5; 10]
   c = [1; 2; 5]

4): 3-element Vector{Int64}:
   1
   2
   5

5): @variable(vector_model, x[1:3] >= 0)

6): 3-element Vector{VariableRef}:
   x[1]
   x[2]
   x[3]

7): # Определение ограничений модели:
   @constraint(vector_model, A * x .<= b)
   @constraint(vector_model, x[1] <= 10)

8):

$$x_1 \leq 10$$


9): # Определение целевой функции:
   @objective(vector_model, Min, c' * x)

10):  $x_1 + 2x_2 + 5x_3$ 

11): # Вызов функции оптимизации:
   optimize!(vector_model)
   # Определение причины завершения работы оптимизатора:
   termination_status(vector_model)

12): OPTIMAL::TerminationStatusCode = 1

13): # Демонстрация первичных результирующих значений переменных x и y:
   @show value(x1);
   @show value(x2);
   @show value(x3);

   value(x1) = 10.0
   value(x2) = 2.1875
   value(x3) = 0.9375

14): # Демонстрация результата оптимизации:
   @show objective_value(vector_model);
   objective_value(vector_model) = 5.0

```

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

```

1]: using Random

2]: m = 10
   n = 5

   Random.seed!(42)
   A = rand(m, n)
   b = rand(m)

   x = Variable(n)

   objective = sumsquares(A*x - b)
   constraints = [x >= 0]

   problem = minimize(objective, constraints)

3]: Problem statistics
   problem is DCP      : true
   number of variables : 1 (5 scalar elements)
   number of constraints : 1 (5 scalar elements)
   number of coefficients : 67
   number of atoms      : 6

   Solution summary
   termination status : OPTIMIZE_NOT_CALLED
   primal status      : NO_SOLUTION
   dual status        : NO_SOLUTION

   Expression graph
   minimize
   └─ qol (convex; positive)
      └─ + (affine; real)
         └─ * (affine; real)
            └─ _

4]: solve!(problem, SCS.Optimizer)

[ Info: [Convex.jl] Compilation finished: 1.36 seconds, 286.170 MiB of memory allocated

=====
SCS v3.2.7 - Splitting Conic Solver
(c) Brendan O'Donoghue, Stanford University, 2012
=====
problem: variables n: 6, constraints m: 17
cones:   l: linear vars: 5
         q: soc vars: 12, qsize: 1
settings: eps_abs: 1.0e-04, eps_rel: 1.0e-04, eps_infeas: 1.0e-07
          alpha: 1.50, scale: 1.00e-01, adaptive_scale: 1
          max_iters: 100000, normalize: 1, rho_x: 1.00e-06
          acceleration_lookback: 10, acceleration_interval: 10
          compiled with openmp parallelization enabled
lin-sys: sparse-direct-amd-qdldl
          nnz(A): 57, nnz(P): 0
=====
iter | pri res | dua res | gap  | obj  | scale | time (s)
=====

5]: println("Статус задачи: ", problem.status)

Статус задачи: OPTIMAL

```

Рис. 4.14: Задание 3. Выпуклое программирование

```

1]: num_sections = 5
   num_applications = 1000

   min_capacity = [180, 180, 220, 180, 180]
   max_capacity = [250, 250, 220, 250, 250]

   Random.seed!(42)
   priorities = [shuffle([1, 2, 3, 10000, 10000]) for _ in 1:num_applications]
   priorities = hcat(priorities...)
   priority_weights = sum(priorities, dims=1)

2]: 1x5 Matrix{Int64}:
   3851232  4091162  4181199  4031198  3851209

3]: model = Model(GLPK.Optimizer);

4]: @variables(model, begin
   180 <= x1 <= 250
   180 <= x2 <= 250
   x3 == 220
   180 <= x4 <= 250
   180 <= x5 <= 250
   end)

5]: (x1, x2, x3, x4, x5)

6]: @constraint(model, x1+x2+x3+x4+x5==1000)
   @objective(model, Min, x1 * priority_weights[1] +
   x2 * priority_weights[2] +
   x3 * priority_weights[3] +
   x4 * priority_weights[4] +
   x5 * priority_weights[5])

7]: 3851232x1 + 4091162x2 + 4181199x3 + 4031198x4 + 3851209x5

8]: optimize!(model)
   termination_status(model)

9]: OPTIMAL::TerminationStatusCode = 1

10]: println("Результаты:")
   println("x1: ", value(x1))
   println("x2: ", value(x2))
   println("x3: ", value(x3))
   println("x4: ", value(x4))
   println("x5: ", value(x5))
   println("Оптимальное значение целевой функции: ", objective_value(model))

Результаты:
x1: 180.0
x2: 180.0
x3: 220.0
x4: 180.0
x5: 240.0
Оптимальное значение целевой функции: 3.9994005e9

```

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 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.org contributors. URL: <https://docs.julialang.org/en/v1/> (дата обращения: 11.10.2024).