

Лабораторная работа № 8

Оптимизация

Беличева Д. М.

Российский университет дружбы народов, Москва, Россия

Информация

- Беличева Дарья Михайловна
- студентка
- Российский университет дружбы народов
- 1032216453@pfur.ru
- <https://dmbelicheva.github.io/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

```
]: # Определение объекта модели с именем model;
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 переменных x, y и граничных условий для них:
@variable(model, x >= 0)
@variable(model, y >= 0)

]: y

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

]: 
    7x + 12y ≥ 120

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

]: 12x + 20y

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

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

]: OPTIMAL::TerminationStatusCode = 1

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

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

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

objective_value(model) = 205.0
```

Рис. 1: Линейное программирование

Выполнение лабораторной работы

```
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 8 8;
      2 0 6 13]
b = [1; 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{Float64}}, MathOptInterface.ConstraintFunction{MathOptInterface.ScalarAffineFunction{Float64}}}}:
x[1] + x[2] + 9 * x[3] + 5 * x[4] == 7
3 * x[1] + 5 * x[2] + 8 * x[4] == 3
2 * x[1] + 6 * x[3] + 13 * x[4] == 5

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

#: x₁ + 3x₂ + 5x₃ + 2x₄

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

#: # Определение причин завершения работы оптимизатора:
termination_status!(vector_model)

#: OPTIMAL:TerminationStatusCode = 1

#: # Демонстрация результата оптимизации:
@show objective_value!(vector_model)
objective_value!(vector_model) = 4.9238769238769225
```

Рис. 2: Векторизованные ограничения и целевая функция оптимизации

Выполнение лабораторной работы

```
18]: # Контейнер для хранения данных об ограничениях на количество
# потребляемых калорий, белков, жиров и соли:
category_data = JuMP.Containers.DenseAxisArray(
    [1886, 2208;
     91.0;
     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}:
1886.0  2208.0
  91.0   Inf
  0.65   65.0
  0.0    1779.0

19]: # массив продуктов 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. 3: Оптимизация рациона питания

Выполнение лабораторной работы

```
[21]: # Импорт данных о содержании калорий, белков, жиров и соли в продуктах питания
food_data = JuMP.Containers.DenseAxisArray(
    [410 24 26 730;
     420 32 18 1190;
     560 20 32 1880;
     380 4 19 270;
     320 12 18 930;
     320 15 12 820;
     320 31 12 1230;
     180 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 18.0 1190.0
 560.0 20.0 32.0 1880.0
 380.0 4.0 19.0 270.0
 320.0 12.0 18.0 930.0
 320.0 15.0 12.0 820.0
 320.0 31.0 12.0 1230.0
 180.0 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]: # Определение массива
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]: 3-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
  Dimension 1, ["calories", "protein", "fat", "sodium"]
  And data, a 4-element Vector{VariableRef}:
  nutrition[calories]
```

Рис. 4: Оптимизация рациона питания

Выполнение лабораторной работы

```
[25]: # Определение целевой функции:  
@objective(model, Min, sum[cost[f] * buy[f] for f in foods])  
2.49*buy[hamburger] + 2.89*buy[chicken] + 1.59*buy[hotdog] + 1.89*buy[fries] + 2.09*buy[macaroni] + 1.99*buy[pizza] + 2.49*buy[milk] + 0.89*buy[nuts] + 1.59*buy[  
[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, Vector{Float64}}, ScalarShape}),  
And data, 4-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{  
{Float64}}, MathOptInterface.EqualTo{Float64, Vector{Float64}}, ScalarShape}}:  
-nutriton(calories) = 418 buy[hamburger] + 420 buy[chicken] + 568 buy[hot dog] + 380 buy[fries] + 320 buy[macaroni] +  
28 buy[pizza] + 320 buy[salad] = 100 buy[milk] + 330 buy[ice cream] = 0  
-nutriton(protein) = 24 buy[hamburger] + 32 buy[chicken] + 28 buy[hot dog] + 4 buy[fries] + 12 buy[macaroni] + 15 buy[  
pizza] + 31 buy[salad] = 0 buy[milk] + 8 buy[ice cream] = 0  
-nutriton(fat) = 26 buy[hamburger] + 10 buy[chicken] + 32 buy[hot dog] + 19 buy[fries] + 10 buy[macaroni] + 12 buy[pi  
za] + 12 buy[salad] + 2.5 buy[milk] + 10 buy[ice cream] = 0  
-nutriton(sodium) = 738 buy[hamburger] + 1198 buy[chicken] + 1880 buy[hot dog] + 270 buy[fries] + 930 buy[macaroni] +  
20 buy[pizza] + 1238 buy[salad] + 125 buy[milk] + 188 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] 6.978138888888885  
buy[ice cream] 2.5913594444444445
```

Рис. 5: Оптимизация рациона питания

Выполнение лабораторной работы

```
using DelimitedFiles
using CSV

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

# 200-200 Matrix[Any]:
"Passport" "Albania" - "Afghanistan"
"Albania" -1 "e-visa" - "Visa required"
"Algeria" "e-visa" "Visa required"
"Angola" "e-visa" - "Visa required"
"Antigua and Barbuda" "e-visa" "Visa required"
"Argentina" "e-visa" "Visa required"
"Armenia" "e-visa" "Visa required"
"Aruba" "e-visa" "Visa required"
"Austria" "e-visa" "Visa required"
"Azerbaijan" "e-visa" "Visa required"
"Bahamas" "e-visa" "Visa required"
:
"United Arab Emirates" "98" "Visa required"
"United Kingdom" "98" "Visa required"
"United States" "98" "Visa required"

# Задачи переменных:
ctrn = passportdata[2:end,:]
vt = (x > typeof(x)==Int64 || x == "NP" || x == "VOA" ? 1 : 0) (passportdata[2:end,2:end]);
```

Определение объекта модели с именем model:

```
model = Model(GLPK.Optimizer)
```

A JuMP Model

```
solvers = GLPK()
status = GLPK.OPTIMAL_STATUS; FEASIBILITY_SENSE
num_variables = 0
num_constraints = 0
Names registered in the model: none
```

Параметры, ограничения и целевая функция:

```
variable!(model, pass[1:length(ctrn)], Bin)
@constraint(model, [j=1:length(ctrn)], sum( vt[i,j]*pass[i] for i in 1:length(ctrn)) >= 1)
@objective!(model, Min, sum(pass))
```

pass₁ + pass₂ + pass₃ + pass₄ + pass₅ + pass₆ + pass₇ + pass₈ + pass₉ + pass₁₀ + pass₁₁ + pass₁₂ + pass₁₃ + pass₁₄ + pass₁₅ + pass₁₆ + pass₁₇ + pass₁₈ + pass₁₉ + pass₂₀ + pass₂₁ + pass₂₂ + pass₂₃ + pass₂₄ + pass₂₅ + pass₂₆ + pass₂₇ + pass₂₈ + pass₂₉ + pass₃₀ + pass₃₁ + pass₃₂ + pass₃₃ + pass₃₄ + pass₃₅ + pass₃₆ + pass₃₇ + pass₃₈ + pass₃₉ + pass₄₀ + pass₄₁ + pass₄₂ + pass₄₃ + pass₄₄ + pass₄₅ + pass₄₆ + pass₄₇ + pass₄₈ + pass₄₉ + pass₅₀ + pass₅₁ + pass₅₂ + pass₅₃ + pass₅₄ + pass₅₅ + pass₅₆ + pass₅₇ + pass₅₈ + pass₅₉ + pass₆₀ + pass₆₁ + pass₆₂ + pass₆₃ + pass₆₄ + pass₆₅ + pass₆₆ + pass₆₇ + pass₆₈ + pass₆₉ + pass₇₀ + pass₇₁ + pass₇₂ + pass₇₃ + pass₇₄ + pass₇₅ + pass₇₆ + pass₇₇ + pass₇₈ + pass₇₉ + pass₈₀ + pass₈₁ + pass₈₂ + pass₈₃ + pass₈₄ + pass₈₅ + pass₈₆ + pass₈₇ + pass₈₈ + pass₈₉ + pass₉₀ + pass₉₁ + pass₉₂ + pass₉₃ + pass₉₄ + pass₉₅ + pass₉₆ + pass₉₇ + pass₉₈ + pass₉₉ + pass₁₀₀ + pass₁₀₁ + pass₁₀₂ + pass₁₀₃ + pass₁₀₄ + pass₁₀₅ + pass₁₀₆ + pass₁₀₇ + pass₁₀₈ + pass₁₀₉ + pass₁₁₀ + pass₁₁₁ + pass₁₁₂ + pass₁₁₃ + pass₁₁₄ + pass₁₁₅ + pass₁₁₆ + pass₁₁₇ + pass₁₁₈ + pass₁₁₉ + pass₁₂₀ + pass₁₂₁ + pass₁₂₂ + pass₁₂₃ + pass₁₂₄ + pass₁₂₅ + pass₁₂₆ + pass₁₂₇ + pass₁₂₈ + pass₁₂₉ + pass₁₃₀ + pass₁₃₁ + pass₁₃₂ + pass₁₃₃ + pass₁₃₄ + pass₁₃₅ + pass₁₃₆ + pass₁₃₇ + pass₁₃₈ + pass₁₃₉ + pass₁₄₀ + pass₁₄₁ + pass₁₄₂ + pass₁₄₃ + pass₁₄₄ + pass₁₄₅ + pass₁₄₆ + pass₁₄₇ + pass₁₄₈ + pass₁₄₉ + pass₁₅₀ + pass₁₅₁ + pass₁₅₂ + pass₁₅₃ + pass₁₅₄ + pass₁₅₅ + pass₁₅₆ + pass₁₅₇ + pass₁₅₈ + pass₁₅₉ + pass₁₆₀ + pass₁₆₁ + pass₁₆₂ + pass₁₆₃ + pass₁₆₄ + pass₁₆₅ + pass₁₆₆ + pass₁₆₇ + pass₁₆₈ + pass₁₆₉ + pass₁₇₀ + pass₁₇₁ + pass₁₇₂ + pass₁₇₃ + pass₁₇₄ + pass₁₇₅ + pass₁₇₆ + pass₁₇₇ + pass₁₇₈ + pass₁₇₉ + pass₁₈₀ + pass₁₈₁ + pass₁₈₂ + pass₁₈₃ + pass₁₈₄ + pass₁₈₅ + pass₁₈₆ + pass₁₈₇ + pass₁₈₈ + pass₁₈₉ + pass₁₉₀ + pass₁₉₁ + pass₁₉₂ + pass₁₉₃ + pass₁₉₄ + pass₁₉₅ + pass₁₉₆ + pass₁₉₇ + pass₁₉₈ + pass₁₉₉ + pass₂₀₀

Виды функций оптимизации:

```
JuMP.optimize!(model)
termination_status(model)
```

OPTIMAL::TerminationStatusCode = 1

Прострой результаты:
print(JuMP.objective_value(model)," passports:",join(mtrc[findall(JuMP.value.(pass) .== 1)]),";")

34.0 passports:Afghanistan, Australia, Bahrain, Cameroon, Comoros, Congo, Djibouti, Dominican Republic, Eritrea, Guinea, Iraq, Israel, Jordan, Kenya, Kuwait, Liberia, Libya, Madagascar, Maldives, Mauritania, Morocco, Nepal, New Zealand, Oman, Palestine, Papua New Guinea, Qatar, Saudi Arabia, Singapore, Somalia, South Sudan, Spain, Syria, Turkmenistan, United States, Venezuela, Yemen

Рис. 6: Путешествие по миру

Выполнение лабораторной работы

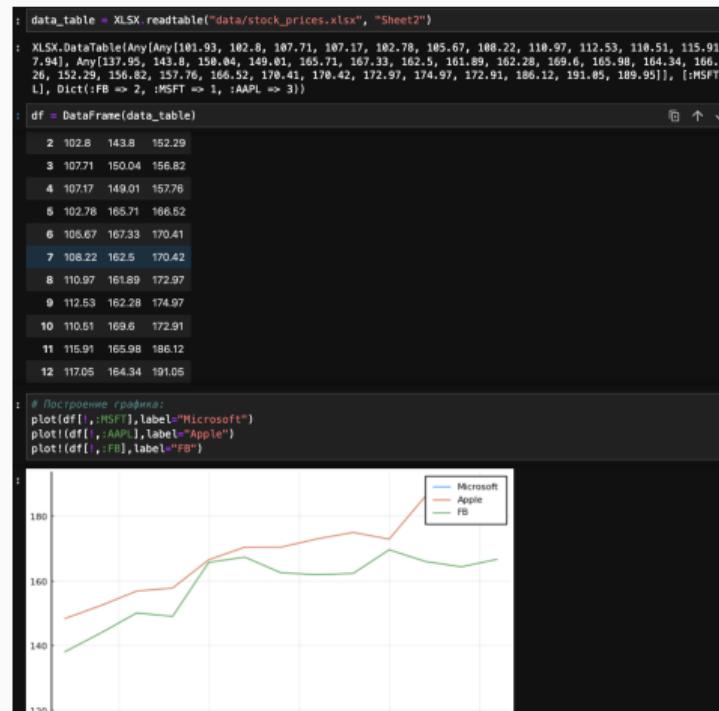


Рис. 7: Портфельные инвестиции

Выполнение лабораторной работы

```
prices_matrix = Matrix(df)
# Выделенная матрица доходности за период времени:
M1 = prices_matrix[1:end-1,:]
M2 = prices_matrix[2:end,:]
# Матрица доходности:
R = (M2-M1)/M1
# Матрица риска:
risk_matrix = cov(R)
# Проверка положительной определенности матрицы риска:
isposdef(risk_matrix)
# Длина от квадратов из компаний:
r = mean(R, dims=1)[1]
# Генерация случайных чисел:
x = Variable(length(r))
# Объект решения:
problem = minimize(Convex.quadform(x, risk_matrix), [sum(x)==1|r*x==0.02;x,>=0])
# Нахождение решения:
solve!(problem, SCS.Optimizer)

Expression graph
minimize
└─ (convex; positive)
  └─ [1,:]
    └─ qo_elem (convex; positive)
      └─ norm2 (convex; positive)
        └─ [1,:]
subject to
└─ == constraint (affine)
  └─ + (affine; real)
    └─ sum (affine; real)
      └─ [1,:]
└─ == constraint (affine)
  └─ + (affine; real)
    └─ + (affine; real)
x
:
x: Variable
size: (3, 1)
sign: real
value: affine
id: 524.598
value: [0.07740789795831287, 0.11606771003983582, 0.886527626687796]
:
sum(x.value)
:
1.000002434677944
:
r*x.value
:
1x1 adjoint(::Vector{Float64}) with eltype Float64:
0.019947233108531883
:
x.value .≈ 1000
:
3x1 Matrix{Float64}:
77.46709795831288
116.86771003983582
886.527626687796
```

Рис. 8: Портфельные инвестиции

Выполнение лабораторной работы

```
0]: using Images  
0]: # Считывание исходного изображения:  
Kref = load("data/khiam-small.png")  
...  
  
1]: K = copy(Kref)  
p = prod(size(K))  
missingids = rand(1:p,400)  
K[missingids] .= RGB(NaN)(0.0,0.0,0.0)  
K  
Gray.(K)  
1]:  

```

Рис. 9: Восстановление изображения

Выполнение лабораторной работы

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

2]: correctids = findall(Y[:,!]==0)
X = Convex.Variables{size(Y)}
problem = minimize(nuclearnorm(X))
problem.constraints += X[correctids]==Y[correctids]

r Warning: Concatenating collections of constraints together with '+' or '+=' to produce a new list of
repeated. Instead, use 'vcat' to concatenate collections of constraints.
└
3]: 1-element Vector{Constraint}:
  == constraint (affine)
  └ + (affine; real)
    └ index (affine; real)
      └ 952×968 real variable (id: 117_378)
      913520×1 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'Donnoghue, 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, sszie: 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-and-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. 10: Восстановление изображения

Выполнение лабораторной работы

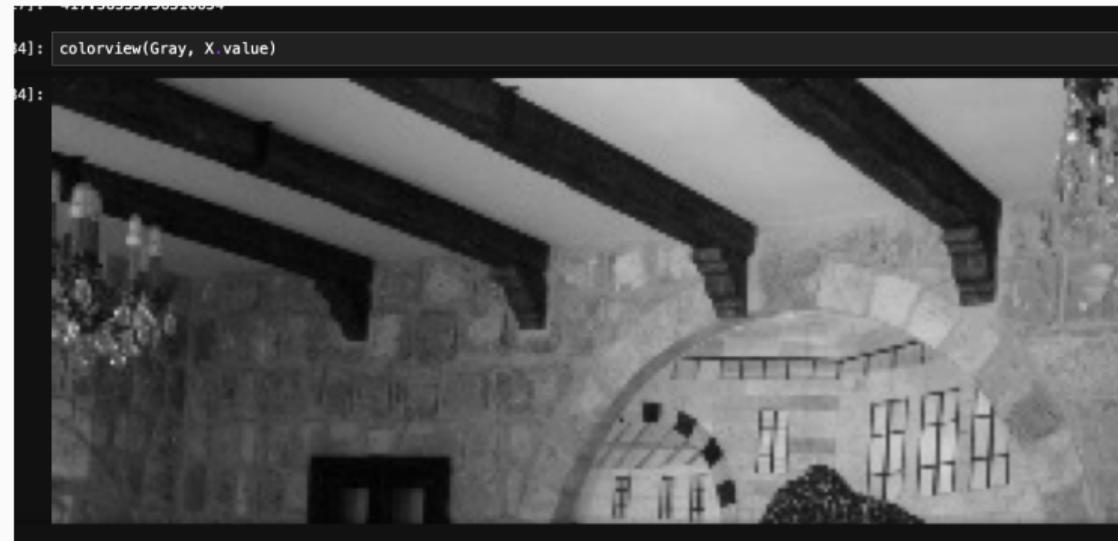


Рис. 11: Восстановление изображения

Выполнение лабораторной работы

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

5]: A JuMP Model
+ solver: GLPK
+ objective_sense: FEASIBILITY_SENSE
+ num_variables: 0
+ num_constraints: 0
| Names registered in the model: none

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

6]: z3

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

7]: 
x1 + 3x2 - 7x3 ≤ 10

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

8]: z1 + 2z2 + 5z3

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

9]: OPTIMAL::TerminationStatusCode = 1

10]: # Демонстрация первичных результатирующих значений переменных x и y:
@show value(x1);
@show value(x2);
@show value(x3);

value(x1) = 10.0
value(x2) = 2.1075
value(x3) = 0.9375
```

Рис. 12: Задание 1. Линейное программирование

Выполнение лабораторной работы

```
; # Определение объекта модели с именем model;
vector_model = Model(GLPK_Optimizer)

; A LP Model
; | solver: GLPK
; | objective_sense: FEASIBILITY_SENSE
; | num_variables: 0
; | num_constraints: 0
; | Names registered in the model: none

;| A = {-1 1 3; 1 3 -7}
;| B = {-5; 10}
;| c = {1; 2; 5}

;| 3-element Vector{Int64}:
;|   1
;|   2
;|   5

;| @variable(vector_model, x[1:3] >= 0)
;| 3-element Vector{VariableRef}:
;|   x[1]
;|   x[2]
;|   x[3]

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

;|           x1 ≤ 10

;| # Определение целевой функции:
;| @objective(vector_model, Min, c' * x)
;| x1 + 2x2 + 5x3

;| # Вызов функции оптимизации:
;| optimize!(vector_model)
;| # Определение причин завершения работы оптимизатора:
;| termination_status(vector_model)
;| OPTIMAL::TerminationStatusCode = 1

;| # Демонстрация первичных результатирующих значений переменных x и у:
;| @show value(x1);
;| @show value(x2);
;| @show value(x3);

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

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

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

Выполнение лабораторной работы

```
[1]: using Random

[2]: m = 10
n = 5

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

x = Variable(n)

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

problem = minimize(objective, constraints)

[3]: Problem statistics
      problem is DCP
      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
      └─ g0l (convex; positive)
          └─ + (affine; real)
              └─ x (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: 5, qzsize: 1
settings: sparsen: 1.0e-04, qpz_rel: 1.0e-04, eps_imfeas: 1.0e-07
          algmax: 1.59, scale: 1.00e-01, adaptive.scale: 1
          max_iters: 100000, normalize: 1, rho_xi: 1.00e-06
          acceleration_lookbacks: 10, acceleration_interval: 10
          compiled with openmp parallelization enabled
lin-sys: sparse-direct-mnd-qdld
        nnz(A): 57, nnz(P): 8
-----  
iter | pri res | dnu res |  gap |  obj  | scale | time (s)
```

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

```
Статус задачи: OPTIMAL
```

Рис. 14: Задание 3. Выпуклое программирование

Выполнение лабораторной работы

```
1: num_sections = 5
2: num_applications = 1000
3: min_capacity = [100, 100, 220, 100, 100]
4: max_capacity = [250, 250, 220, 250, 250]
5: Random.seed!(42)
6: priorities = [shuffle!(1, 2, 3, 10000, 10000) for _ in 1:num_applications]
7: priorities = hcat(priorities...)
8: priority_weights = sum(priorities, dims=1)
9: 
10: 1x5 Matrix{Int64}:
11:     3851232 4091162 4181198 4031198 3851209
12: 
13: model = Model(GLPK.Optimizer)
14: 
15: @variables(model, begin
16:     100 <= x1 <= 250
17:     100 <= x2 <= 250
18:     x3 == 220
19:     100 <= x4 <= 250
20:     100 <= x5 <= 250
21: end)
22: 
23: (x1, x2, x3, x4, x5)
24: 
25: #constraint(model, x1+x2+x3+x4+x5==100)
26: #objective(model, Min, x1 * priority_weights[] +
27:             x2 * priority_weights[] +
28:             x3 * priority_weights[] +
29:             x4 * priority_weights[] +
30:             x5 * priority_weights[])
31: 
32: 3851232x1 + 4091162x2 + 4181198x3 + 4031198x4 + 3851209x5
33: 
34: optimize!(model)
35: termination_status(model)
36: 
37: @PTIMAL::TerminationStatusCode = 1
38: 
39: println("Результаты:")
40: println("x1: ", value(x1))
41: println("x2: ", value(x2))
42: println("x3: ", value(x3))
43: println("x4: ", value(x4))
44: println("x5: ", value(x5))
45: println("Оптимальное значение целевой функции: ", objective_value(model))
46: 
47: Результаты:
48: x1: 100.0
49: x2: 100.0
50: x3: 220.0
51: x4: 100.0
52: x5: 240.0
53: Оптимальное значение целевой функции: 2.999495e5
```

Рис. 15: Задание 4. Оптимальная рассадка по залам

Выполнение лабораторной работы

```
|: 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)
|: 
|:      5buyRaf = 40

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

|: 2x2 Matrix{AffExpr}:
|:   buy[Raf]      0
|:   buy[Cappuccino] 6
```

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).