

Компьютерный практикум по статистическому анализу данных

Лабораторная работа № 8. Оптимизация

Демидова Екатерина Алексеевна

Содержание

1	Введение	4
2	Теоретическое введение	5
3	Выполнение лабораторной работы	6
4	Выводы	16
	Список литературы	17

Список иллюстраций

3.1	Примеры	6
3.2	Примеры	7
3.3	Примеры	8
3.4	Примеры	9
3.5	Примеры	9
3.6	Примеры	10
3.7	Примеры	11
3.8	Задание 1	12
3.9	Задание 2	12
3.10	Задание 3	13
3.11	Задания 4	14
3.12	Задание 5	15

1 Введение

Цель работы

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

Задачи

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

2 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].

3 Выполнение лабораторной работы

Выполним примеры из лабораторной работы для изучения циклов и функций(рис. 3.1 - 3.7)

```
Лабораторная работа № 8. Оптимизация

using JuMP
using GLPK

[1] ✓ 1.8s

model = Model{GLPK.Optimizer}
# Определение переменных x, y и граничных условий для них:
@variable(model, x_example >= 0)
@variable(model, y_example >= 0)
# Определение ограничений модели:
@constraint(model, 6x_example + 8y_example >= 100)
@constraint(model, 7x_example + 12y_example >= 120)
# Определение целевой функции:
@objective(model, Min, 12x_example + 20y_example)

[5] ✓ 1.8s
... 12x_example + 20y_example

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

[6] ✓ 1.6s
... OPTIMAL::TerminationStatusCode = 1

# Демонстрация первичных результирующих значений переменных x и y:
@show value(x_example);
@show value(y_example);
# Демонстрация результата оптимизации:
@show objective_value(model);

[7] ✓ 0.5s
... value(x_example) = 14.999999999999993
value(y_example) = 1.25000000000000047
objective_value(model) = 205.0
```

Рис. 3.1: Примеры

```
# Определение объекта модели с именем vector_model:
vector_model = Model(GLPK.Optimizer)
# Определение начальных данных:
A = [1 1 9 5;
     3 5 0 0;
     2 0 6 13]
b = [7; 3; 5]
c = [1; 3; 5; 2]

[0] ✓ 1.5s

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

▷ # Определение вектора переменных:
  @variable(vector_model, x_example[1:4] >= 0)
# Определение ограничений модели:
  @constraint(vector_model, A * x_example .== b)
# Определение целевой функции:
  @objective(vector_model, Min, c' * x_example)

[1] ✓ 0.8s

... x_example1 + 3x_example2 + 5x_example3 + 2x_example4

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

[10] ✓ 0.0s

... OPTIMAL::TerminationStatusCode = 1

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

[11] ✓ 0.0s

... objective_value(vector_model) = 4.9230769230769225
```

Рис. 3.2: Примеры

Оптимизация рациона питания

```
# Контейнер для хранения данных об ограничениях на количество потребляемых калорий, белков, жиров и соли
category_data = JuMP.Containers.DenseAxisArray{
    [1800 2200;
     0 100;
     0 65;
     0 1779],
    ["calories", "protein", "fat", "sodium"],
    ["min", "max"]}

# массив данных с наименованиями продуктов:
foods = ["hamburger", "chicken", "hot dog", "fries", "macaroni",
         "pizza", "salad", "milk", "ice cream"]
# Массив стоимости продуктов:
cost = JuMP.Containers.DenseAxisArray{
    [2.49, 2.89, 1.58, 1.89, 2.09, 1.99, 2.49, 0.89, 1.99],
    foods}

food_data = JuMP.Containers.DenseAxisArray{
    [419 24 26 738;
     426 32 18 1190;
     560 30 32 1800;
     380 4 19 270;
     320 12 18 830;
     320 15 12 820;
     320 31 12 1230;
     160 8 2 5 125;
     320 8 19 180],
    foods,
    ["calories", "protein", "fat", "sodium"]};
```

```
# Определение объекта модели с именем model:
model = Model(GLPK.Optimizer)
# Определение массива:
categories = ["calories", "protein", "fat", "sodium"]
# Определение переменных:
@variables(model, begin
    category_data[c, "min"] <= nutrition[c = categories] <=
        category_data[c, "max"]
    # Сколько покупать продуктов:
    buy[foods] >= 0
end;
```

```
# Определение целевой функции:
(objective(model, Min, sum(cost[f] * buy[f] for f in foods))
# Определение ограничений модели:
@constraint(model, [c in categories],
sum(food_data[f, c] * buy[f] for f in foods) == nutrition[c])
```

```
# Вызов функции оптимизации:
JMP.optimize!(model)
term_status = JMP.termination_status(model)

[17]  ✓ 0.1s

... OPTIMAL::TerminationStatusCode = 1
```

```
9x2 Matrix(AffExpr):
buy[hamburger] 0.6045138888888889
buy[chicken] 0
buy[hot dog] 0
buy[fries] 0
buy[macaroni] 0
buy[pizza] 0
buy[salad] 0
buy[milk] 6.970138888888889
buy[ice cream] 2.591319444444444
```

8


```

using DelimitedFiles
using CSV

(13)  ✓ 0.2s

passportdata = readlm(joinpath("passport-index-dataset", "passport-index-matrix.csv"), '.');

(14)  # Задаём переменные:
      #tr = passportdata[2:end,1]
      vf = {x -> typeof(x) == Int64 || x == "VF" || x == "VGA" ? 1 :
            0} (passportdata[2:end,2:end]);

(15)  # Определение объекта модели с именем model:
      #del = Model{GLPK.Optimizer}

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

(16)  ...
      # Выход функции оптимизации:
      #MP.optimize!(model)
      termination_status(model)

(17)  ... OPTIMAL::TerminationStatusCode = 1

      # Печатаем результаты:
      #int(JuMP.objective_value(model), " passports:", join(cnr, findall(JuMP.value.(pass) .== 1)), ", ")

(18)  ... 34.0 passports:Afghanistan, Australia, Bahrain, Cameroon, Canada, Comoros, Congo, Denmark, Djibouti, Eritrea, Guinea-Bissau, Hong Kong, Iran, Kenya, Kuwait, Liberia,

```

Рис. 3.4: Примеры

```

Портфельные инвестиции

using DataFrames
using XLSX
using Plots
pyplot()
using Convex
using SCS
using Statistics

(19)  ... [ Info: Precompiling DataFrames [a93c6f08-e37d-5684-b7b6-d8103f3e46c0]
      [ Info: Precompiling Plots [91a5bcdd-55d7-5caf-9e0b-5208899ce888]
      [ Info: Precompiling JuliaExt [2f4321a4-3b3a-5ce6-9c5e-1f2673ce168a]
      [ Info: Precompiling Convex [f6b330ba-76f8-5f13-ba0b-18010c17839a]
      [ Info: Precompiling SCS [c946c3f1-8d1f-5ce8-9dea-7daa1762d13]

      # Считываем данные и размещаем их во фрейме:
      T = DataFrame{XLSX.readtable("data/stock_prices.xlsx", "Sheet2")...}

      # Построение графика:
      plot(T[1,:MSFT], label="Microsoft")
      plot(T[1,:AAPL], label="Apple")
      plot(T[1,:FB], label="FB")

      # Данные о ценах на акции размещаем в матрице:
      prices_matrix = Matrix{T}

      # 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)[:3]

      # 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)

      r'*x.value
      x.value .* 1000

```

Рис. 3.5: Примеры

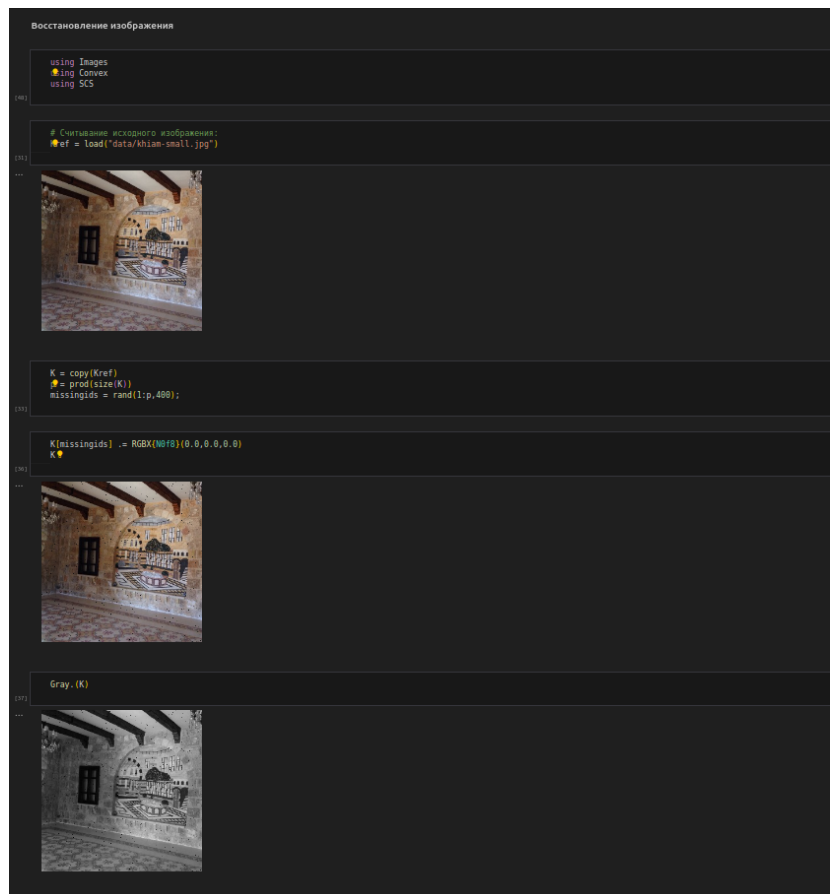


Рис. 3.6: Примеры

```

# Матрица весов:
# Float64 (Gray, K);

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

1-element Vector{Constraint}:
= constraint (affine)
├─ index (affine; real)
├─ 283x283 real variable (id: 242_387)
└─ 79008-element Vector{Float64}


solve!(problem, SCS.Optimizer())

-----
SCS v3.2.4 - Splitting Conic Solver
(c) Brendan O'Donoghue, Stanford University, 2012
-----
problem: variables n: 248268, constraints m: 488047
cones: z: primal zero / dual free vars: 239586
s: psd vars: 168481, ssize: 1
settings: eps_rho: 1.0e-04, eps_rel: 1.0e-04, eps_infcons: 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
lin-sys: sparse-direct-mmd-qldl
nnz(A): 480330, nnz(P): 0
-----
iter | pri res | dua res | gap | obj | scale | time (s)
-----
0 | 1.50e+01 | 9.06e-01 | 0.34e+03 | 1.77e+02 | 1.00e-01 | 6.68e-01
250 | 1.97e-04 | 1.91e-05 | 8.72e-06 | 4.46e+02 | 3.29e-01 | 6.87e-01
275 | 2.85e-04 | 1.57e-05 | 3.86e-06 | 4.46e+02 | 3.29e-01 | 7.57e-01
-----
status: solved
timings: total: 7.57e+01s = setup: 3.88e-01s + solve: 7.53e+01s
lin-sys: 4.22e+00s, cones: 6.88e+01s, accel: 4.35e-01s
-----
objective = 445.786048
norm(float, Gray, {Kref}) - X.value = 1.0436917853159704
norm(-(X.value)) = 124.33509422854655

124.33509422854655

@show norm(float, Gray, {Kref})-X.value
@show norm(X.value)
colorview(Gray, X.value)

norm(float, Gray, {Kref}) - X.value = 1.0436917853159704
norm(-(X.value)) = 124.33509422854655



```

Рис. 3.7: Примеры

Затем выполним задания(рис. 3.8 - 3.12)

```

Задания

№ 1

D>
using JuMP
import GLPK
model = Model{GLPK.Optimizer}

[195]

... A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK

@variable(model, 0<=x1<=10)
@variable(model, x2<=0)
@variable(model, x3<=0)

@constraint(model, -x1+x2+2x3 <= -5)
@constraint(model, x1+3x2-7x3 <= 10)

[197]
@objective(model, Max, x1+2x2+5x3)

... x1 + 2x2 + 5x3

# Базис функции оптимизации:
optimize!(model)
# Описание причины завершения работы оптимизатора:
termination_status(model)

[198]

OPTIMAL::TerminationStatusCode = 1

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

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

[199]

... value(x1) = 10.0
value(x2) = 2.1875
value(x3) = 0.9375
objective_value(model) = 19.0625

```

Рис. 3.8: Задание 1

```

№ 2

vector_model = Model{GLPK.Optimizer}
A = [-1 1 3; 1 3 -7]
b = [-5; 10]
c = [1; 2; 5]

[202]

... 3-element Vector{Int64}:
 1
 2
 5

@variables(vector_model, begin
    X[1:3]>=0
end)

[203]

... (VariableRef{X[1], X[2], X[3]}.)

@constraint(vector_model, A * X .<= b)
@constraint(vector_model, X[1] <= 10)
# Определение целевой функции:
@objective(vector_model, Min, c' * X)

[204]

...  $X_1 + 2X_2 + 5X_3$ 

optimize!(vector_model)
termination_status(model)

[205]

... OPTIMAL::TerminationStatusCode = 1

@show objective_value(vector_model);

[206]

... objective_value(vector_model) = 5.0

```

Рис. 3.9: Задание 2

```

Выпуклое программирование

using Convex
using CSV
n = 3
m = 4
A = rand(1:100, m,n);
b = rand(1:100,m);

x = Variable(n)
⚡objval = minimize(sumsquares(A*x-b), [x>=0])
solve!(problem, SCS.Optimizer)

SCS v3.2.4 - Splitting Conic Solver
(c) Brendan O'Donoghue, Stanford University, 2012
problem: variables n: 6, constraints m: 13
cones: z: primal zero / dual free vars: 1
      l: linear vars: 4
      q: sec vars: 8, qsize: 2
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
lin-sys: sparse-direct-and-qrldl
         nnz(A): 21, nnz(P): 0

iter | pri res | dua res | gap | obj | scale | time (s)
-----|-----|-----|-----|-----|-----|-----
0 | 5.72e+01 | 3.83e+00 | 1.95e+01 | -6.40e+00 | 1.00e-01 | 1.55e-04
250 | 1.31e+00 | 3.89e-01 | 7.54e+00 | 1.91e+02 | 2.88e-02 | 5.24e-04
500 | 2.77e+00 | 4.68e-04 | 1.33e-01 | 3.30e+02 | 1.41e-02 | 8.53e-04
750 | 1.97e+01 | 1.03e+00 | 2.06e+00 | 4.14e+02 | 5.97e-02 | 1.21e-03
1000 | 5.69e-02 | 7.99e-05 | 1.29e-02 | 3.71e+02 | 5.97e-02 | 1.58e-03
1250 | 5.14e-02 | 2.07e-05 | 3.58e-03 | 3.71e+02 | 5.97e-02 | 1.89e-03
1500 | 4.66e-02 | 1.37e-05 | 3.74e-04 | 3.71e+02 | 5.97e-02 | 2.24e-03
1750 | 4.21e-02 | 1.04e-05 | 4.91e-04 | 3.71e+02 | 5.97e-02 | 2.55e-03

lin-sys: 1.05e-03s, cones: 5.09e-04s, accel: 4.67e-04s
objective = 371.474812

problem.optval
371.47513696483946

x
Variable
size: (3, 1)
sign: real
vexity: affine
id: 597_220
value: [5.5677525181955666e-6, 0.8721825960167477, -5.759830635024598e-8]

```

Рис. 3.10: Задание 3

```

Оптимальная рассадка по залам

using GLPK, Convex, DataFrames, Random

[228]

participants_df = DataFrame()
#participants_df[1,:id] = 1:1000
participants_df[1,:priority] = [shuffle([1 2 3 10000 10000]) for i=1:1000];
priority = shuffle([1 2 3 10000 10000])
for i=1:999
    priority = vcat(priority, shuffle([1 2 3 10000 10000]))
end
[243]

[1 1 1 1 1]*priority'

[249]
... 1*1000 Matrix{Int64}:
20006 20006 20006 20006 20006 20006 20006 20006 20006 20006 20006

model = Model{GLPK.Optimizer};

[248]

@variables(model, begin
    #
    100<=x1<=250
    100<=x2<=250
    100<=x4<=250
    100<=x5<=250
end)
x3 = 220

@constraint(model, x1+x2+x3+x4+x5 == 1000)
@objective(model, Min, sum([x1, x2, x3, x4, x5]*priority'))

[ 3]

▷ participants_df[1,:2]

[228]
... 1000-element Vector{Matrix{Int64}}:
[10000 2 - 1 10000]
[2 10000 - 3 1]
[10000 10000 - 3 1]
[1 10000 - 10000 3]
[1 10000 - 3 10000]
[10000 1 - 10000 3]
[3 1 - 10000 10000]
[2 10000 - 3 1]
[3 10000 - 10000 1]
[3 1 - 10000 2]
[3 1 - 10000 10000]
[1 10000 - 10000 3]
[3 10000 - 1 10000]
1
[10000 2 - 1 10000]
[10000 3 - 1 10000]
[10000 1 - 2 3]
[3 10000 - 10000 2]

```

Рис. 3.11: Задания 4

```
using JuMP
using GLPK

coffee = ["Raf coffee", "Cappuccino"]
products = ["grains", "milk", "sugar"]
costs = 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}
coffee_data["Raf coffee","grains"]
store["grains"]

[27]
... 500

▷
model = Model{GLPK.Optimizer}
@variables(model, begin
    buy[coffee] >= 0
end)

# определение целевой функции:
@objective(model, Max, sum(costs[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])
# вызов функции оптимизации:
JuMP.optimize!(model)
term_status = JuMP.termination_status(model)
hcat(buy.data, JuMP.value.(buy.data))

[33]
... 2x2 Matrix{AffExpr}:
buy[Raf coffee] 8
buy[Cappuccino] 6
```

Рис. 3.12: Задание 5

4 Выводы

В результате выполнения работы освоили использование специализированных пакетов 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).