

Лабораторная работа № 8. Оптимизация

Компьютерный практикум по статистическому анализу данных

Демидова Е. А.

24 ноября 2024

Российский университет дружбы народов, Москва, Россия

Информация

- Демидова Екатерина Алексеевна
- студентка группы НКНбд-01-21
- Российский университет дружбы народов
- <https://github.com/eademidova>



Введение

Цель работы

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

Задачи

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

Выполнение лабораторной работы

Лабораторная работа № 8. Оптимизация

```
using JuMP
using GLPK

131 ✓ 1.8s

D> 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)

131 ✓ 1.8s

132 12x_example + 20y_example

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

131 ✓ 1.8s

OPTIMAL::TerminationStatusCode = 1

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

131 ✓ 0.5s

132 value(x_example) = 14.999999999999993
value(y_example) = 1.25000000000000047
objective_value(model) = 205.0
```

Рис. 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]

[1] ✓ 15s

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

D
# 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)

[2] ✓ 0.0s

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

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

[3] ✓ 0.0s

--- OPTIMAL::TerminationStatusCode = 1

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

[4] ✓ 0.0s

--- objective_value(vector_model) = 4.9230769230769225
```

Рис. 2: Примеры

Выполнение примеров

[illegible]

Рис. 3: Примеры

Выполнение примеров

```
using DelimitedFiles
using CSV

1001 ✓ 0.2s

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

1002

1003 # Inplace normalization:
1004 # ctr = passportdata[2:end, 1]
1005 # v = {x -> typemax(x) - Int64 || x == "VF" || x == "VDA" ? 1 :
1006 | 0}.[passportdata[2:end, 2:end]];

1007

1008 # Suppress some of the solver's messages:
1009 #del = Model{GLPK.Optimizer}

1010 # Suppress some of the solver's messages:
1011 @variable(model, pass[1:length(ctr)], Bin)
1012 @constraint(model, [1:length(ctr)], sum(v[i,j]*pass[j] for j in 1:length(ctr)) >= 1)
1013 @objective(model, Min, sum(pass))

1014

1015 # Solve the problem:
1016 #MP.optimize!(model)
1017 termination_status(model)

1018

1019 OPTIMAL::TerminationStatusCode = 1

1020

1021 # Print the results:
1022 # set{JMP.objective_value(model), "passports": join(ctr[findall(JMP.value.(pass) .> 1)], ", ")}

1023

1024 31.0 passports:Afghanistan, Australia, Bahrain, Cameroon, Canada, Comoros, Congo, Denmark, Djibouti, Eritrea, Guinea-Bissau, Hong Kong, Iran, Kenya, Kuwait, Liberia,
```

Рис. 4: Примеры

Выполнение примеров

[illegible]

Рис. 5: Примеры

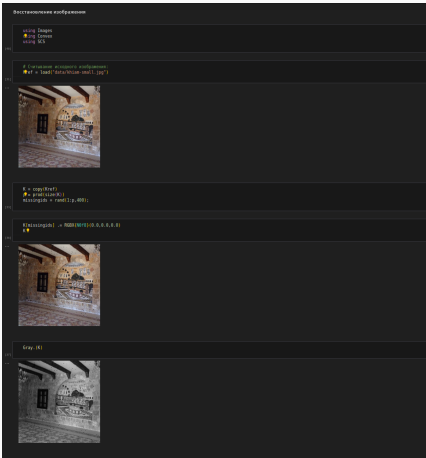


Рис. 6: Примеры

Выполнение примеров

```
# Runtime context
# • FlatM4 library (C++)

correctids = findall(Y)[1,140]
# • Given NeuralNetwork()
problem = NeuralNetworkProblem(K)
problem.constraints = X[correctids][0][correctids]

# Element Vector (constraints)
# constraints (affine)
# index (affine, real)
# - 2000x2000 real, variable (id: 202,203)
# - 2000x1 element Vector (FlatM4)

solve(problem, GCS.Optimizer())

=====
MCS v3.2.0 - Splitting Conic Solver
(c) Brandon L. Bonet, Stanford University, 2012
=====
problem: variables = 200000, constraints = none/0
constraints: 0 primal, none / dual free vars: 200000
# pos vars: 20000, neg: 0
settings: eps_obj: 1.0e-04, eps_feas: 1.0e-04, eps_sol: 1.0e-07
         alpha: 1.5, scale: 1.0e+01, adaptive_scale: 1
         max_iters: 10000, max_subpro: 1, rho: 0.1, rho0: 0
         acceleration_lookback: 10, acceleration_interval: 10
         max_iter_direct_and_split:
         max_obj: 400000, min_obj: 0

iter | pri_val | dual_val | gap | obj | scale | time (s)
-----|-----|-----|-----|-----|-----|-----
0 | 1.5e+02 | 8.0e+01 | 8.3e+03 | 1.7e+02 | 1.0e+01 | 8.0e-01
100 | 3.0e+04 | 1.1e+03 | 8.7e+06 | 4.0e+02 | 3.0e+01 | 6.0e-01
200 | 2.0e+04 | 1.1e+03 | 8.0e+06 | 4.0e+02 | 3.0e+01 | 7.5e-01

status: solved
         final: 1.0e+01 = primal, 8.0e+01 = dual, 7.0e-01
         obj_val: 4.22e+04, cone: 8.0e+01, accel: 4.0e-01
         objective = 445.700000

=====
new(float, <rap, <rap>) - 8, value = 1.0420007012125070
new(<Z, value>) = 124.330942103403

124.330942103403

# show new(float, <rap, <rap>): 8, value
# show new(<Z, value>)
# show new(float, <rap, <rap>): 8, value
# show new(<Z, value>) = 124.330942103403



```

Рис. 7: Примеры

Выполнение заданий для самостоятельной работы

Задания

```
int 1

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

[1m]

--- A JuMP Model
Feasibility problem with:
Variables: 3
Model name: AUTOMATIC
CachingOptimizer state: BOMTtr_OPTIMIZER
Solver name: GLPK

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

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

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

[1m]

--- z1 + 2z2 + 5z3

# Basic function optimization:
@optimize!(model)
# Возвращаем терминус завершения работы оптимизатора:
termination_status(model)

[1m]

--- OPTIMAL::TerminationStatusCode = 1

# Демонстрация переменных результатов оптимизации: значения переменных x и y:
@show value(x1);
@show value(x2);
@show value(x3);

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

[1m]

--- value(x1) = 10.0
value(x2) = 2.1875
value(x3) = 0.8375
objective_value(model) = 19.8625
```

Рис. 8: Задание 1

Выполнение заданий для самостоятельной работы

```
№ 2

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

[42] 3-element Vector{Int64}:
      1
      2
      5

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

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

@constraint(vector_model, A * X .<= b)
@constraint(vector_model, X[1] <= 10)
# @dispatch does not know how to dispatch:
@objective(vector_model, Min, c' * X)

[44]  $X_1 + 2X_2 + 5X_3$ 

optimize!(vector_model)
termination_status(model)

[45] OPTIMAL::TerminationStatusCode = 1

@show objective_value(vector_model);

[46] objective_value(vector_model) = 5.0
```

Рис. 9: Задание 2

Выполнение заданий для самостоятельной работы

```
Выполнение программирования

using Convex
using CSV
n = 5
m = 4
A = rand(1:100, m,n);
b = rand(1:100, m);

[100]

D
x = Variable{n}
minimize(sum_squares(A*x-b), [x==0])
solve!(problem, GCS.Optimizer)

[101]

=====
KS v0.2.4 - Splitting Conic Solver
(c) Brendan O'Donoghue, Stanford University, 2012
=====
problem: variables n: 6, constraints n: 33
cones: 2: primal zero / dual free vars: 1
1: linear vars: 4
q: soc vars: 8, sz: 2
settings: eps abs: 1.0e-04, eps rel: 1.0e-04, eps infoss: 1.0e-07
alpha: 1.50, scale: 1.0e-01, adaptive scale: 1
max iters: 100000, normalize: 1, rho x: 1.0e-06
acceleration (epochs): 10, acceleration_interval: 10
lin-eps: 1e-08, direct-and-opt: 0
nuc(A): 21, nuc(P): 0
=====
iter | pri res | dual res | gap | obj | scale | time (s)
=====
0 | 5.79e+01 | 5.03e+00 | 1.95e+01 | 6.48e+00 | 1.00e-01 | 1.55e-04
250 | 2.31e+00 | 3.09e-01 | 7.54e+00 | 1.51e+02 | 2.08e-02 | 5.24e-04
500 | 2.77e+00 | 4.60e-04 | 1.12e-01 | 3.06e+02 | 1.41e-02 | 6.53e-04
750 | 1.97e+01 | 1.03e+00 | 2.06e+00 | 4.14e+02 | 5.97e-02 | 1.23e-03
1000 | 1.09e+02 | 7.99e-05 | 1.29e-02 | 3.72e+02 | 5.97e-02 | 1.58e-03
1250 | 1.14e+02 | 2.47e-05 | 3.15e-03 | 3.72e+02 | 5.97e-02 | 1.19e-03
1500 | 4.66e+02 | 1.37e-05 | 3.74e-04 | 3.72e+02 | 5.97e-02 | 2.24e-03
1750 | 4.21e+02 | 1.04e-05 | 4.01e-04 | 3.72e+02 | 5.97e-02 | 2.51e-03
...
lin-eps: 1.05e-03, cones: 5.09e-04, accval: 4.63e-04
=====
objective = 371.474012
=====
Output is truncated. View on a public server or open in a text editor. Adjust cell output settings.

problem.optval

[102]

-- 371.47513094603946

x

[103]

-- Variable
size: (3, 1)
shape: real
weights: affine
id: 593.220
value: [2.5677732518195466e-04, 0.8721825968157477, -5.759838635824550e-01]
```

Рис. 10: Задание 3

Выполнение заданий для самостоятельной работы

```

Оптимизация процесса по заказам

using GLPK, Convex, DataFrames, Random

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

[1 1 1 1 1]"priority"

1=1000 Matrix{Int64}:
2005 2006 2007 2008 2009 2006 2006 2005 2006 2006 2006

model = Model{GLPK.Optimizer}()

@variables(model, begin
    x1=0x1=258
    x2=0x2=258
    x3=0x3=258
    x4=0x4=258
end)
x5 = 220

@constraint(model, c1=x2+x3+x4+x5 == 1000)
@objective(model, Min, sum([x1, x2, x3, x4, x5]"priority"))

participants_df[:,1,2]

1000-element Vector{Matrix{Int64}}:
[10000 2 1 10000]
[2 10000 2 3]
[10000 30000 3 3]
[1 10000 10000 3]
[1 10000 3 10000]
[10000 1 10000 3]
[3 1 30000 10000]
[2 10000 3 3]
[3 10000 10000 1]
[3 1 30000 2]
[3 1 30000 10000]
[1 10000 10000 2]
[3 10000 1 10000]

[10000 2 1 10000]
[10000 3 1 10000]
[10000 2 2 3]
[3 10000 10000 2]

```

Рис. 11: Задания 4

Выполнение заданий для самостоятельной работы

```
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. 12: Задание 5

Выводы

В результате выполнения работы освоили использование специализированных пакетов Julia для решения задач оптимизации.

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).