

Лабораторная работа № 8

Оптимизация

Герра Гарсия Паола Валентина

Содержание

Цель работы	1
Задание	2
Теоретическое введение	2
Выполнение лабораторной работы	2
Выводы	8
Список литературы.....	8

Список иллюстраций

Линейное программирование	2
Векторизованные ограничения и целевая функция оптимизации	3
Оптимизация рациона питания	3
Оптимизация рациона питания.....	3
Оптимизация рациона питания.....	4
Путешествие по миру	4
Портфельные инвестиции	4
Портфельные инвестиции	5
Восстановление изображения	5
Восстановление изображения	5
Восстановление изображения	6
Задание 1. Линейное программирование	6
Задание 2. Линейное программирование. Использование массивов	7
Задание 3. Выпуклое программирование.....	7
Задание 4. Оптимальная рассадка по залам	7
Задание 5. План приготовления кофе.....	8

Цель работы

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

Задание

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

Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [[@julialang](#)]. 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 [[@juliadoc](#)].

Выполнение лабораторной работы

Выполним примеры из лабораторной работы (рис. [-@fig:001]-[-@fig:011]).

In [1]:

```
# Импортируем пакеты:
import Pkg
Pkg.add("GLPK")
Pkg.add("GLPKC")
using JuMP
using GLPK

Resolving package versions...
 Installed Codec2j2p2 v0.8.5
 Installed StructTypes v1.11.0
 Installed JSON v1.14.3
 Installed JSON3 v1.29.4
 Installed MathOptInterface v1.1.0
 Updating `~/.julia/environments/v1.12/Project.toml'
[407a5fc0] + JuMP v0.29.0
Updating `~/.julia/environments/v1.12/Manifest.toml'
[523f6e87] + Codec2j2p2 v0.8.5
[0f8b85d5] + JSON v1.14.3
[49d4c84d] + JSON3 v1.29.4
^ [b92f7283] + MathOptInterface v1.48.0
[8567d2b0] + StructTypes v1.11.0
[1f3a3723] + CategoricalArrays v1.3.0
[313a40ab] + SparseArrays v0.12.0
Precompiling packages...
2391.8 ms ✓ Codec2j2p2
3565.8 ms ✓ StructTypes
1576.9 ms ✓ CategoricalArrays - CategoricalArraysStructTypesExt
```

In [2]:

```
# Определяем объекта модели с внешними параметрами:
model = Model(GLPK.optimizer)
```

Out[2]:

```
A JuMP Model
  * Status: GLPK
  * Objective sense: FEASIBILITY_SENSE
  * Number of variables: 0
  * Number of constraints: 0
  * Names registered in the model: none
```

Линейное программирование

```

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

Out[3]: y

In [4]: # Определение ограничений модели:
@constraint(model, 6x + 8y >= 160)
@constraint(model, 7x + 12y == 120)

Out[4]: 7x + 12y ≥ 120

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

Out[5]: 12x + 20y

In [6]: # Вызов функции оптимизации:
optimize!(model)

In [7]: # Определение причин завершения работы оптимизатора:
termination_status(model)

Out[7]: OPTIMAL:terminationStatusCode = 1

In [8]: # Демонстрация первоначальных результатирующих значений переменных x и у:
@show value(x);
@show value(y);

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

value(x) = 15.00000000000002
value(y) = 1.249999999999999
objective_value(model) = 205.0

```

Векторизованные ограничения и целевая функция оптимизации

```

In [10]: # Определение объекта модели с именем vector_model:
vector_model = Model(GLPK.Optimizer)

Out[10]: A JuMP Model
  - solver: GLPK
  - sense: MAX_SENSE
  - num_variables: 0
  - num_constraints: 0
  - Names registered in the model: none

In [11]: # Определение начальных данных:
A = [ 1 1 9 5;
      3 1 0 6;
      2 0 6 13];
b = [7; 3; 5];
c = [1; 3; 5; 2];

Out[11]: 4-element Vector{Int64}:
 1
 3
 5
 2

In [12]: # Определение вектора переменных:
@variable(vector_model, x[1:4] >= 0)

Out[12]: 4-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
 x[4]

In [13]: # Определение ограничений модели:
@constraint(vector_model, A * x == b)

Out[13]: 3-element Vector{ConstraintRef{Model, MathDataInterface, ConstraintIndex{MathDataInterface}, ScalarAffineFunction{Float64}}}

```

Оптимизация рациона питания

```

In [19]: # Контейнер для хранения данных по ограничениям на количество потребляемых калорий, белков, жиров и соли:
category_data = JuMP.Containers.DenseAxisArray(
    [180, 2200;
     91.16, 65.0;
     0.0, 1779.0],
    ["calories", "protein", "fat", "sodium"],
    ["min", "max"])

Out[19]: 2-dimensional DenseAxisArray{Float64,2,...} with index sets:
 Dimension 1, ["calories", "protein", "fat", "sodium"]
 Dimension 2, [min, max]
 Data: 4x2 Matrix{Float64}:
 180.0  2200.0
 91.16  65.0
 0.0   1779.0

In [20]: # список данных с наименованиями продуктов:
foods = ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]

Out[20]: 9-element Vector{String}:
 "hamburger"
 "chicken"
 "hot dog"
 "fries"
 "macaroni"
 "pizza"
 "salad"
 "milk"
 "ice cream"

In [21]: # Масив стоимостей продуктов:
cost = JuMP.Containers.DenseAxisArray(
    [2.49, 2.89, 1.56, 1.89, 2.89, 1.99, 2.49, 0.89, 1.59],
    foods)

```

Оптимизация рациона питания

```

In [14]: # Определение целевой функции:
#objective!(vector_model, Min, c' * x)
Out[14]: x₁ + 3x₂ + 5x₃ + 2x₄

In [15]: # Вызов функции оптимизации:
optimize!(vector_model)

In [16]: # Определение причин завершения работы оптимизатора:
termination_status!(vector_model)

Out[16]: OPTIMALT::TerminationStatusCode = 1

In [17]: # Демонстрация результата оптимизации:
@show objective_value!(vector_model);
objective_value!(vector_model) = 4.9238769230769225

In [18]: # Подключение пакетов:
import Pkg
Pkg.add("JuMP")
Pkg.add("GLPK")
using JuMP
using GLPK

Resolving package versions...
Project No packages added or removed from ~/julia/environments/v1.12/Project.toml
Manifest No packages added or removed from ~/julia/environments/v1.12/Manifest.toml
Precompiling packages...
  x LinearSolve
    x Sparse
      click to unroll output: double click to hide glpk
        x CPLEX
        x GLPK
        x DICOPTer
        x StochasticDICOPT

```

Оптимизация рациона питания

```

In [23]: # Массив данных о содержании калорий, белков, жиров и соли в продуктах питания:
food_data = Dict{String, Vector{Float64}}()
for file in readdir("./data")
    if file == "FoodData.csv"
        food_data = CSV.read(file, Dict)
    end
end
food_data["calories"], food_data["protein"], food_data["fat"], food_data["sodium"]

Out[23]: 2-dimensional DenseAxisArray{Float64,1,...} with index sets:
Dimension 1: ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]
Dimensions 2: [100.0, 24.0, 26.0, 738.0, 428.0, 32.0, 18.0, 568.0, 20.0, 32.0, 1800.0, 388.0, 4.0, 19.0, 278.0, 328.0, 15.0, 12.0, 820.0, 328.0, 31.0, 12.0, 820.0, 108.0, 2.5, 125.0, 338.0, 8.0, 10.0, 180.0]
食物营养素: ["calories", "protein", "fat", "sodium"]

In [24]: # Определение объекта модели с именем model:
model = Model(GLPK.Optimizer)

Out[24]: A JuMP Model
  - solver: GLPK
  - objective_sense: FEASIBILITY_SENSE
  - name: none
  - num_variables: 0
  - num_constraints: 0
  - Names registered in the model: none

```

Путешествие по миру

```

In [21]: # Определение переменных:
buy_food = VariableRef()
category_data["calories"] => nutrition[c == categories] == category_data[c, "max"]
# Сколько покупать продуктов:
buyFood0s => 0
buyFood1s => 0

Out[21]: (1-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
Dimension 1: ["calories", "protein", "fat", "sodium"]
And data, a 4-element Vector{VariableRef}:
  buyFood0s
  buyFood1s
  nutrition[protein]
  nutrition[fat]
nutrition[calories], 1-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
Dimension 1: ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]
And data, a 9-element Vector{VariableRef}:
  buyFood0s
  buyFood1s
  buy(chicken)
  buy(hot dog)
  buy(fries)
  buy(macaroni)
  buy(pizza)
  buy(salad)
  buy(milk)
  buy(ice cream))

In [28]: # Определение целевой функции:
#objective!(model, Min, sum(cost[f] * buy[f] for f in foods))

Out[28]: 2.49*buy_chicken + 2.89*buy_hotdog + 1.58*buy_fries + 1.89*buy_macaroni + 2.09*buy_pizza + 1.99*buy_salad + 2.49*buy_milk + 0.89*buy_icecream + 1.59*buy_cream

In [34]: # Определение ограничений модели:
#Definicion correcta de restricciones
@constraint!(model, [c in categories],
  sum(food_data[c, f] * buy[f] for f in foods) == nutrition[c])

```

Портфельные инвестиции

The screenshot shows a Jupyter Notebook running on a Mac OS X system. The browser window title is "jupyter La68_статистический_анализ Last Checkpoint: hace 3 horas (autoassessed)". The URL is localhost:8888/notebooks/la68_статистический_анализ.ipynb. The notebook contains the following code:

```
In [31]: # Базовы функции оптимизации:  
JUMP.optimize!(model)  
term_status = JUMP.termination_status(model)  
  
Out[31]: OPTIMAL:terminationStatusCode = 1  
  
In [32]: # Скачиваем данные с ресурса на git:  
git clone https://github.com/ilyankou/passport-index-dataset.git  
Cloning into 'passport-index-dataset'...  
  
In [33]: # Использование пакета:  
import Pkg  
Pkg.add("DelimitedFiles")  
Pkg.add("CSV")  
using DelimitedFiles  
using CSV  
  
    Resolving package versions...  
    Updating `~/.julia/environments/v1.12/Project.toml'  
    [B翊] DelimitedFiles v1.9.1  
Manifest: No packages added to or removed from `~/.julia/environments/v1.12/Manifest.toml'  
Precompiling packages...  
    ↗ LinearSolve  
    ↗ Optimistic  
    ↗ NonlinearSolve  
    ↗ OrdinaryDiffEq  
    ↗ DiffEqBase  
    ↗ Symbolics  
    ↗ SymbolicDiffEq  
    ↗ DifferentialEquations  
    ↗ ModelingToolkit  
    ↗ ParameterizedFunctions  
0 dependencies successfully precompiled in 108 seconds. 495 already precompiled.  
18 dependencies errored.  
For a report of the errors see 'Julia-err'. To retry use 'pkp-precompile'
```

Портфельные инвестиции

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Sat 14 Feb 00:17

localhost:8888/notebooks/la68_статистический_анализ.ipynb

Luisa python Method Of Fortran SQL courses Gmail YouTube Maps Traductor integrates Segundo Grado 30... B.O.D miércoles, 2... Relaunch to update

jupyter La68_статистический_анализ Last Checkpoint: hace 3 horas (autosaved)

Logout Trusted Julia 1.12.0

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

Out [35]: 280x280 Matrix{Any}:

	"Passport"	"Albania"	"Afghanistan"
"Passport"	-1	"e-visa"	"visa required"
"Albania"	-1	"e-visa"	"visa required"
"Algeria"	98	"e-visa"	"visa required"
"Angola"	98	"e-visa"	"visa required"
"Antigua and Barbuda"	98	"e-visa"	"visa required"
"Argentina"	98	"e-visa"	"visa required"
"Armenia"	98	"e-visa"	"visa required"
"Australia"	98	"e-visa"	"visa required"
"Austria"	98	"e-visa"	"visa required"
"Azerbaijan"	98	"e-visa"	"visa required"
"Bahamas"	98	"e-visa"	"visa required"
"United Arab Emirates"	98	"e-visa"	"visa required"
"United Kingdom"	98	"e-visa"	"visa required"
"United States"	380	"e-visa"	"visa required"
"Uruguay"	98	"e-visa"	"visa required"
"Uzbekistan"	98	"e-visa"	"visa required"
"Vanuatu"	98	"e-visa"	"visa required"
"Venezuela"	98	"e-visa"	"visa required"
"Vietnam"	98	"e-visa"	"visa required"
"Yemen"	98	"e-visa"	"visa required"
"Zambia"	98	"e-visa"	"visa required"
"Zimbabwe"	98	"e-visa"	"visa required"

In [36]: # Доработка пакетов:
Pkg.add("JMP")
Pkg.add("RMP")
using JMP
using GLPK

Восстановление изображения

```
Chrome File Edit View History Bookmarks Profiles Tab Window Help
localhost:8888/notebooks/la68_статистический_анализ.ipynb
Relaunch to update

File Edit View Insert Cell Kernel Widgets Help
Logout
File Edit View Insert Cell Kernel Widgets Help
In [38]: # Определяем объект response с именем model:
model = Model(GLPK.Optimizer)

Out[38]: A JuMP Model
├─ solver: GLPK
└─ objective_sense: FEASIBILITY_SENSE
    └─ name: 
    └─ num_variables: 0
    └─ num_constraints: 0
    └─ Names registered in the model: none

In [41]: # Primero, asegurarse de que passportdata está definido
# (esto es solo un ejemplo - usa tus datos reales)
# passportdata = ...

# Definir entr y vft (codigo que ya tenias)
cntr = passportdata[2:end, 1]
vft = (x => typeof(x) == Int64 || x == "VF" || x == "VGA"? 1 : 0).(passportdata[2:end, 2:end])

# Verificar que vft se creó correctamente
print(cntr) # Dimensiones de cntr: 1x size(cntr)
print(vft) # Primeros elementos de vft: , vft[1:min(5, end), 1:min(5, end)]

# Variables, restricciones y función objetivo
using JuMP
model = Model()

@variable(model, pass[i].length(cntr), Bin)
@constraint(model, [j] >= 1 <= size(vft, 2)), sum(vft[i,j] == pass[i] for i in 1:length(cntr)) >= 1
@objective(model, Min, sum(pass))

Primeros elementos de vft: [199, 199]
Primeros elementos de cntr: [1 0 1 0 1; 0 1 0 1 0; 1 0 1 0 1; 0 0 0 1 0]

Out[41]: pass + pass * pass + pass * pass = pass + pass * pass + pass * pass + pass * pass
+ pass * pass + pass * pass + pass * pass + pass * pass + pass * pass
+ pass * pass + pass * pass + pass * pass + pass * pass + pass * pass
```

Восстановление изображения

The screenshot shows a Jupyter Notebook interface with a single cell containing Python code. The code defines a function to find a password given its length and type (VPA or VFB). It uses a GLPK solver to iterate through possible combinations of characters (A-Z, 0-9) until it finds a match. The output shows the search space and the final solution found.

```
In [46]: # Definir entr y vf (tu código existe)
ctrn = passportdata[2:end, 1]
vf = (x => typeof(x) == Int64 || x == "VF" || x == "VPA" ? 1 : 0).(passportdata[2:end, 2:end])

# Cargar paquetes necesarios
using JuMP, GLPK
# Crear modelo CON solver incluido
model = Model(GLPK.Optimizer)

# Variables, restricciones y función objetivo
@variable(model, pass[i] length(ctrn), Bin)
@constraint(model, [i] size(vf, 2)), sum(vf[i, j] * pass[j] for i in 1:length(ctrn)) ≥ 1
@objective(model, Min, sum(pass))

# Llamar a la función de optimización (ESTO ES CRITICO)
JuMP.optimize!(model)

# AHORA SI, verificar el estado y mostrar resultados
status = termination_status(model)
println("Estado: ", status)

# Si la solución es óptima, mostrar resultados
if status == OPTIMAL
    println("Solución óptima encontrada")
    v = value.(pass)
    print("passwords: ", join(ctrn.findall(value.(pass) .> 0.5)), ", ")
    print(objective_value(model), " passports: ", sum(pass))
else
    println("No se encontró solución óptima. Estado: ", status)
end

Dual bound      34
Gap             0% (tolerance: 0.81%)
P-O integral   0.0000009934e-07
Solution status feasible
34 (objective)
```

Восстановление изображения

Теперь выполним задания для самостоятельный работы (рис. [-@fig:012]-[-@fig:016]).

The screenshot shows a Jupyter Notebook interface with several cells of Python code related to a linear programming assignment. The code includes importing packages, defining variables and constraints for a linear program involving three variables (x1, x2, x3), and solving the problem using GLPK. The output shows the solved model and the optimal values for the variables.

```
In [61]: # Задание для самостоятельного выполнения
# 4.1. Линейное программирование
model = Model(GLPK.Optimizer)

Out[61]: A JuMP Model
| solved by GLPK
| objective sense: FEASIBILITY_SENSE
| num_variables: 0
| num_constraints: 0
| Names registered in the model: none

In [65]: @variable(model, 0 ≤ x1 ≤ 10)
@variable(model, x2 ≥ 0)
@variable(model, x3 ≤ 10)

Out[65]: x3

In [66]: @constraint(model, -x1+x2-x3 ≤ -5)
@constraint(model, x1+3x2-7x3 ≤ 10)

Out[66]: x1 + 3x2 - 7x3 ≤ 10

In [67]: @objective(model, Max, x1+2x2+5x3)

Out[67]: x1 + 2x2 + 5x3

In [68]: optimize!(model)
termination_status(model)

Out[68]: OPTIMAL:terminationStatusCode = 1

In [69]: @show value(x1);
@show value(x2);
@show value(x3);
```

Задание 1. Линейное программирование

```

In [70]: A=([-1, 1, 3; 1, 3, -7])
         b=[-5; 1; 5]
         c=[1, 2, 5]

Out[70]: 3-element Vector{Int64}:
 1
 2
 5

In [71]: @variable(vector_model, x[1:3] >= 0)

Out[71]: 3-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]

In [72]: @constraint(vector_model, A*x .≤ b)
         @constraint(vector_model, x[1] ≤ 10)

Out[72]: x₁ ≤ 10

In [73]: @objective(vector_model, Min, c' * x)

Out[73]: x₁ + 2x₂ + 5x₃

```

Задание 2. Линейное программирование. Использование массивов

```

In [76]: optimize!(vector_model)
termination_status!(vector_model)

Out[76]: OPTIMAL:terminationStatusCode = 1

In [77]: @show value(x1);
@show value(x2);
@show value(x3);

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

In [78]: @show objective_value!(vector_model);
objective_value!(vector_model) = 5.0

In [79]: # 4.3. Выпуклое программирование
using Random
m=10
n=5

Random.seed!(42)
A=rand(m,n)
b=rand(n)
x= Variable(n)
objective=objective_function(A*x - b)
constraints = [x==0]
problem = minimize!(objective, constraints)

Out[79]: Problem statistic:
problem is LP
number of variables : 1 (5 scalar elements)
number of constraints : 1 (5 scalar elements)
number of coefficients : 67
number of degens : 0

```

Задание 3. Выпуклое программирование

```

In [81]: println("Статус:", problem.status)
статус:OPTIMAL

In [82]: # 4.4. Оптимальная рассадка по залам
num_section = 5
num_applications=1000
min_capacity= [100, 100, 200, 100, 100]
min_capacity= [250, 250, 250, 250, 250]

Random.seed!(42)
priorities=[shuffle(1, 2, 3, 10000, 1000)] for _ in 1:num_applications
priorities=heatPriorities_...
priorities=heatPriorities_d...
priorities=heatPriorities_dims=1

Out[82]: 1x5 Matrix{Int64}:
 206023 2147102 2300199 2285198 2213209

In [83]: model = Model(GLPK.Optimizer)
@variable(model, begin
    180≤ x1 ≤250
    180≤ x2 ≤250
    20≤ x3 ≤250
    180≤ x4 ≤250
    180≤ x5 ≤250
end)

Out[83]: (x1, x2, x3, x4, x5)

In [88]: @constraint(model, 1*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])

```

Задание 4. Оптимальная рассадка по залам

The screenshot shows a Jupyter Notebook interface running in a Chrome browser. The notebook has three cells:

- In [88]:** Contains optimization code using JuMP and Ipopt. It defines variables x_3 , x_4 , and x_5 with priority weights, and calculates a cost function.
- In [89]:** Prints the termination status of the optimization model.
- In [91]:** Prints the results of the optimization, showing values for x_2 , x_3 , x_4 , and x_5 .

In [93]: Shows code for preparing coffee. It defines products like "Latte", "Cappuccino", "Grains", "Milk", and "Sugar". It creates a JuMP model, sets constraints, and solves it. The resulting data is stored in a JuMP DenseAxisArray.

Out [93]: Prints the JuMP DenseAxisArray containing the solved data for the coffee preparation problem.

Задание 5. План приготовления кофе

Выводы

В результате выполнения данной лабораторной работы я освоила пакеты Julia для решения задач оптимизации.

Список литературы