Лабораторная работа №8

Леснухин Даниил Дмитриевич Российский университет дружбы народов Москва

Цель работы

Основная цель работы — освоить пакеты Julia для решения задач оптимизации

Задание

1. Используя Jupyter Lab, повторите примеры из раздела 8.2
2. Выполните задания для самостоятельной работы (раздел 8.4)

Лабораторная работа

Линейное программирование

```
using JuMP, GLPK

model = Model(GLPK.Optimizer)
@variable(model, 0 <= x1 <= 10)
@variable(model, x2 >= 0)
@variable(model, x3 >= 0)

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

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

optimize!(model)

println("Статус решения: ", termination_status(model))
println("Значение целевой функции: ", objective_value(model))
println("оптимальное значение переменных: ", value(x1), " ", value(x2), " ", value(x3))
```

✓ 3.1s

```
Статус решения: OPTIMAL
Значение целевой функции: 19.0625
оптимальное значение переменных: 10.0 2.1875 0.9375
```

Линейное программирование с массивами

```julia
using JuMP, GLPK

model = Model(GLPK.Optimizer)

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

lb = [0, 0, 0]
ub = [10, Inf, Inf]

@variable(model, lb[i] <= x[i=1:3] <= ub[i])

@constraint(model, A * x .<= b)

@objective(model, Max, c' * x)

optimize!(model)

println("Статус решения: ", termination_status(model))
println("Значение целевой функции: ", objective_value(model))
println("оптимальное значение переменных: ", [value(x[i]) for i in 1:3])
```

✓ 0.1s

```
Статус решения: OPTIMAL
Значение целевой функции: 19.0625
оптимальное значение переменных: [10.0, 2.1875, 0.9375]
```

Выпуклое программирование

```
using Convex, SCS, LinearAlgebra

m, n = 5, 3

A = randn(m, n)
b = randn(m)

x = Variable(n)

problem = minimize(sumsquares(A * x - b))
problem.constraints += x >= 0


solve!(problem, () -> SCS.Optimizer())

println("Статус решения: ", problem.status)
println("Значение целевой функции: ", problem.optval)
println("Оптимальное значение переменных: ", x.value)


x_values = Convex.evaluate(x)
println("Проверка ограничений: ", all(x_values .>= 0))
println("Все элементы неотрицательны: ", all(x_values .>= 0))
```

```
-----------------------------------------------------------------
problem:  variables n: 4, constraints m: 10
cones:    l: linear vars: 3
          q: soc vars: 7, 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): 20, nnz(P): 0
-----------------------------------------------------------------
 iter | pri res | dua res |   gap   |   obj   |  scale  | time (s)
-----------------------------------------------------------------
     0| 2.25e+01  1.00e+00  3.16e+01 -1.55e+01  1.00e-01  3.78e-03
   150| 1.26e-05  9.84e-07  2.61e-05  3.89e-01  6.64e-01  4.39e-03
-----------------------------------------------------------------
status:  solved
timings: total: 4.40e-03s = setup: 6.17e-04s + solve: 3.78e-03s
         lin-sys: 3.02e-05s, cones: 1.69e-05s, accel: 6.60e-06s
-----------------------------------------------------------------
objective = 0.389185
```

```
----------------------------------------------------------------------
Статус решения:
[ Info: [Convex.jl] Compilation finished: 4.44 seconds, 743.927 MiB of memory allocated
OPTIMAL
Значение целевой функции: 0.3891716695824046
Оптимальное значение переменных: [-3.6837871138445625e-6; 0.759672421747426; 1.1293849442386812;;]
Проверка ограничений: false
все элементы неотрицательны: false
```

Оптимальная рассадка по залам

```julia
using JuMP, GLPK

# Параметры задачи
num_participants = 1000
num_sections = 5

# Вместимость залов (сделаем более реалистичными)
min_capacity = 150  # Уменьшим минимальную вместимость
max_capacity = 250
section3_capacity = 220

# Генерируем случайные приоритеты (более сбалансированно)
# Уменьшим количество "непойду" (10000)
priorities = rand([1, 2, 3, 10000], num_participants, num_sections)
# Сделаем так, чтобы в среднем только 20% участников отказывались от секции
for i in 1:num_participants, j in 1:num_sections
    if priorities[i,j] == 10000 && rand() < 0.8
        priorities[i,j] = rand([1, 2, 3])
    end
end

# Создаем модель
model = Model(GLPK.Optimizer)

# Переменная: x[i,j] = 1, если участник i идет на секцию j
@variable(model, x[1:num_participants, 1:num_sections], Bin)

# Ограничения на вместимость залов (сделаем более мягкими)
@constraint(model, [j in 1:num_sections; j != 3],
    min_capacity <= sum(x[i,j] for i in 1:num_participants) <= max_capacity)
@constraint(model, sum(x[i,3] for i in 1:num_participants) == section3_capacity)

# Каждый участник идет ровно на одну секцию
@constraint(model, [i in 1:num_participants],
    sum(x[i,j] for j in 1:num_sections) == 1)

# Участник не может идти на секцию с приоритетом 10000
@constraint(model, [i in 1:num_participants, j in 1:num_sections; priorities[i,j] == 10000],
    x[i,j] == 0)

# Целевая функция: минимизация суммы приоритетов
@objective(model, Min, sum(priorities[i,j] * x[i,j] for i in 1:num_participants, j in 1:num_sections))

# Решаем задачу
optimize!(model)

# Проверяем статус решения
status = termination_status(model)
println("Статус решения: ", status)

if status == :Optimal
    # Выводим результаты
    println("Общая сумма приоритетов: ", objective_value(model))

    # Статистика по секциям
```

```
    # Проверяем ограничения
    println("\nПроверка ограничений:")
    for j in 1:num_sections
        if j != 3
            count = sum(value.(x[:,j]))
            println("Секция $j: $count (должно быть между $min_capacity и $max_capacity)")
        else
            count = sum(value.(x[:,3]))
            println("Секция 3: $count (должно быть равно $section3_capacity)")
        end
    end

    # Рекомендация для нового участника
    new_priorities = rand([1, 2, 3, 10000], num_sections)
    available_sections = findall(new_priorities .!= 10000)
    if !isempty(available_sections)
        recommended_section = available_sections[argmin(new_priorities[available_sections])]
        println("\nРекомендация для нового участника: секция $recommended_section")
    else
        println("\nНет доступных секций для нового участника")
    end

else
    println("Задача не имеет допустимого решения!")
    println("Попробуем ослабить ограничения...")

    # Альтернативный вариант с более мягкими ограничениями
    println("\nАнализ проблемы:")

    # Проверим, сколько участников готовы идти на каждую секцию
    println("Доступность секций:")
    for j in 1:num_sections
        available_count = sum(priorities[:,j] .!= 10000)
        println("Секция $j: $available_count участников готовы посетить")
    end

    # Проверим общее количество доступных вариантов
    total_available = sum(priorities .!= 10000)
    println("Всего доступных вариантов распределения: $total_available")
end
```

```
Статус решения: OPTIMAL
Задача не имеет допустимого решения!
Попробуем ослабить ограничения...

Анализ проблемы:
Доступность секций:
Секция 1: 944 участников готовы посетить
Секция 2: 957 участников готовы посетить
Секция 3: 956 участников готовы посетить
Секция 4: 947 участников готовы посетить
Секция 5: 948 участников готовы посетить
Всего доступных вариантов распределения: 4752
```

План приготовления кофе

```
using JuMP, GLPK

model = Model(GLPK.Optimizer)

@variable(model, raf >= 0, Int)
@variable(model, cap >= 0, Int)


@constraint(model, 40raf + 30cap <= 500)
@constraint(model, 140raf + 120cap <= 2000)
@constraint(model, 5raf == 40)

@objective(model, Max, 400raf + 300cap)



o-ptimize!(model)


println("Статус решения: ", termination_status(model))
println("Максимальная прибыль: ", objective_value(model))
println("Количество Raf: ", value(raf))
println("Количество Cap: ", value(cap))
```

```
Статус решения: OPTIMAL
Максимальная прибыль: 5000.0
Количество Raf: 8.0
Количество Car: 6.0
```