Лабораторная работа №8

Оптимизация

Легиньких Г.А.

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Легиньких Галина Андреевна
- НФИбд-02-21
- Российский университет дружбы народов
- 1032216447@pfur.ru
- https://github.com/galeginkikh

Основная информация

Цель работы

Основная цель работа — 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).

Выполнение лабораторной работы

1. Повторила пример линейного программирования. Линейное программирование рассматривает решения экстремальных задач на множествах ?-мерного векторного пространства, задаваемых системами линейных уравнений и неравенств.

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

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

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

[15]: x

```
[17]: @variable(model, y >= 0)
```

[17]: y

```
[19]: # Определение ограничений модели:
      @constraint(model, 6x + 8y >= 100)
```

[19]:

$$6x + 8y \geq 100$$

```
[21]: @constraint(model, 7x + 12y >= 120)
```

[21]:

$$7x + 12y \geq 120$$

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

[23]: $12x + 20y$

Рис. 1: Линейное программирование

```
[25]: # Вызов функции оптимизации:
      optimize!(model)

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

[27]: OPTIMAL::TerminationStatusCode = 1

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

      value(x) = 14.999999999999993

[31]: @show value(y);

      value(y) = 1.2500000000000047

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

      objective_value(model) = 205.0
```

Рис. 2: Оптимизация в линейном программировании

2. Далее перешла к примеру “Векторизованные ограничения и целевая функция оптимизации”. Можно добавить ограничения и цель в JuMP, используя векторизованную линейную алгебру.

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

[47]: 4-element Vector{VariableRef}:
       x[1]
       x[2]
       x[3]
       x[4]

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

[49]: 3-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAf
      fineFunction{Float64}, MathOptInterface.EqualTo{Float64}}, ScalarShape}}:
       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

[51]: # Определение целевой функции:
      @objective(vector_model, Min, c' * x)
```

[51]: $x_1 + 3x_2 + 5x_3 + 2x_4$

Рис. 3: Векторы

```
[53]: # Вызов функции оптимизации:
      optimize!(vector_model)

[55]: # Определение причины завершения работы оптимизатора:
      termination_status(vector_model)

[55]: OPTIMAL::TerminationStatusCode = 1

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

      objective_value(vector_model) = 4.9230769230769225
```

Рис. 4: Оптимизация векторов

3. Рассмотрела пример “Оптимизация рациона питания”.

```
[81]: # Вызов функции оптимизации:
      JuMP.optimize!(model)

[83]: term_status = JuMP.termination_status(model)

[83]: OPTIMAL::TerminationStatusCode = 1

[85]: hcat(buy.data,JuMP.value.(buy.data))

[85]: 9×2 Matrix{AffExpr}:
       buy[hamburger]  0.6045138888888888
       buy[chicken]    0
       buy[hot dog]    0
       buy[fries]      0
       buy[macaroni]   0
       buy[pizza]      0
       buy[salad]      0
       buy[milk]       6.9701388888888935
       buy[ice cream]  2.591319444444441
```

Рис. 5: Оптимизация рациона питания

4. Попробовала пример с графиком “Портфельные инвестиции”.

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

[105]:

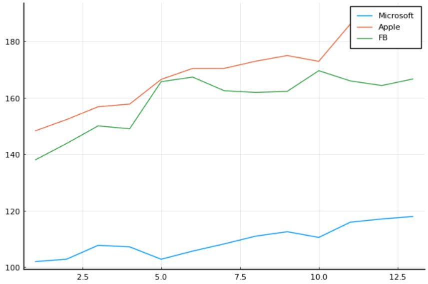

Рис. 6: График

```
[129]: x
```

```
[129]: Variable
       size: (3, 1)
       sign: real
       vexity: affine
       id: 773…887
       value: [0.07740709795031023, 0.11606771003983785, 0.8065276266877958]
```

```
[131]: sum(x.value)
```

```
[131]: 1.000002434677944
```

```
[133]: r'*x.value
```

```
[133]: 1×1 adjoint(::Vector{Float64}) with eltype Float64:
        0.019947233108531883
```

```
[135]: x.value .* 1000
```

```
[135]: 3×1 Matrix{Float64}:
          77.40709795031023
         116.06771003983785
         806.5276266877958
```

Рис. 7: Оптимизация инвистиций

5. И в конце повторила пример по востановлению ихображения. Предположим есть изображение, на котором были изменены некоторые пиксели. Требуется восстановить неизвестные пиксели путём решения задачи оптимизации.

6. Перешла к заданиям для самостоятельной работы.

- Задание 1

```
[157]: using JuMP, GLPK
model = Model(GLPK.Optimizer)
@variable(model, 0 <= x1 <= 10)
@variable(model, x2 >= 0)
@variable(model, x3 >= 0)
@objective(model, Max, x1 + 2 * x2 + 5 * x3)
@constraint(model, -x1 + x2 + 3 * x3 <= -5)
@constraint(model, x1 + 3 * x2 - 7 * x3 <= 10)
optimize!(model)
println("Optimal value: ", objective_value(model))
println("x1: ", value(x1), ", x2: ", value(x2), ", x3: ", value(x3))
```

```
Optimal value: 19.0625
x1: 10.0, x2: 2.1875, x3: 0.9375
```

Рис. 8: Задание 1

- Задание 2

```
[159]: using JuMP, GLPK
model = Model(GLPK.Optimizer)
A = [-1 1 3; 1 3 -7]
b = [-5, 10]
c = [1, 2, 5]
@variable(model, x[1:3] >= 0)
@constraint(model, x[1] <= 10)
@constraint(model, A * x .<= b)
@objective(model, Max, c' * x)
optimize!(model)
println("Optimal value: ", objective_value(model))
println("x: ", value.(x))

Optimal value: 19.0625
x: [10.0, 2.1875, 0.9375]
```

Рис. 9: Задание 2

- Задание 3

```
[161]: using Convex, SCS
n = 5
m = 3
A = rand(m, n)
b = rand(m)
x = Variable(n, nonneg=true)
problem = minimize(sumsquares(A * x - b))
solve!(problem, SCS.Optimizer)
println("Optimal value: ", problem.optval)
println("x: ", evaluate(x))
```

{

#fig:010 width 50% }

- Задание 4

```
[169]: # Количество секций и участников
n_sections = 5
n_participants = 1000

# Ограничения на вместимость залов
min_capacity = 180
max_capacity = 250

# Случайная матрица приоритетов (где 10000 = участник не пойдет на секцию)
priority_matrix = rand(1:3, n_participants, n_sections)
priority_matrix[rand(1:n_participants, Int(n_participants * 0.1)), rand(1:n_sections)] .= 10_000

# Модель оптимизации
model = Model(GLPK.Optimizer)

@variable(model, 0 <= x[1:n_sections] <= max_capacity, Int)  # Количество людей в секциях
@objective(model, Min, sum(x .* sum(priority_matrix, dims=1)'))  # Минимизация приоритетов

# Ограничения
@constraint(model, sum(x) == n_participants)  # Все участники распределены
@constraint(model, x .>= min_capacity)       # Минимальная вместимость
@constraint(model, x[3] == 220)              # Третья секция фиксирована на 220 человек

# Решение
optimize!(model)

# Результаты
println("Оптимальная рассадка по залам: ", value.(x))
println("Оптимальное значение приоритета: ", objective_value(model))

Оптимальная рассадка по залам: [240.0, 180.0, 220.0, 180.0, 180.0]
Оптимальное значение приоритета: 1.747633e8
```

Рис. 10: Задание 4

- Задание 5

```
[173]: using JuMP, GLPK
model = Model(GLPK.Optimizer)
@variable(model, raf >= 0)
@variable(model, cappuccino >= 0)
@objective(model, Max, 400 * raf + 300 * cappuccino)
@constraint(model, 40 * raf + 30 * cappuccino <= 500) # Зерна
@constraint(model, 140 * raf + 120 * cappuccino <= 2000) # Молоко
@constraint(model, 5 * raf == 40) # Ванильный сахар
optimize!(model)
println("Optimal value: ", objective_value(model))
println("Raf: ", value(raf), ", Cappuccino: ", value(cappuccino))
```

```
Optimal value: 5000.0
Raf: 8.0, Cappuccino: 6.0
```

Рис. 11: Задание 5

Вывод

Вывод

Освоила пакеты Julia для решения задач оптимизации.