Лабораторная работа №8. Оптимизация

Выполнила: Лебедева Ольга Андреевна

2025

Российский университет дружбы народов, Москва, Россия

Цель работы

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

Задачи

- Изучить методы и инструменты для решения задач оптимизации, включая линейное и выпуклое программирование.
- Овладеть навыками использования языка Julia и пакетов JuMP, Convex.jl и других для моделирования и решения задач.
- Применить теоретические знания на практике через выполнение различных примеров, демонстрирующих реальные приложения оптимизации.

Термины и условные обозначения

- Оптимизация - процесс нахождения экстремумов целевой функции при заданных ограничениях.
- JuMP - язык моделирования для описания задач математической оптимизации на Julia.
- Целевая функция - функция, значение которой минимизируется или максимизируется.
- Ограничения - линейные или нелинейные условия, которые должны быть выполнены в задаче оптимизации.

Объект и предмет исследования

Объект исследования: Задачи оптимизации в различных практических приложениях.

Предмет исследования: Методы и инструменты, обеспечивающие эффективное решение оптимизационных задач.

Техническое оснащение

1. Среда программирования Julia.
2. Пакеты: JuMP, Convex.jl, GLPK, SCS, Statistics.jl.
3. Инструменты для работы с табличными данными и графиками, такие как DataFrames и Plots.

Теоретическое введение

Под оптимизацией в математике и информатике понимается решение задачи нахождения экстремума (минимума или максимума) целевой функции в некоторой области конечномерного векторного пространства, ограниченной набором линейных и/или нелинейных равенств и/или неравенств.

Оптимизационной задачей называется задача определения наилучших с точки зрения структуры или значений параметров объектов.

Задание лабораторной работы №7

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

Задание лабораторной работы №7

Задание См. рис. 1, См. рис. 2.

8.4.1. Линейное программирование

Решите задачу линейного программирования:

$$x_1 + 2x_2 + 5x_3 \to \max,$$

при заданных ограничениях:

$$-x_1 + x_2 + 3x_3 \leqslant -5, \quad x_1 + 3x_2 - 7x_3 \leqslant 10, \quad 0 \leqslant x_1 \leqslant 10, \quad x_2 \geqslant 0, \quad x_3 \geqslant 0.$$

8.4.2. Линейное программирование. Использование массивов

Решите предыдущее задание, используя массивы вместо скалярных переменных.

Рекомендация. Запишите систему ограничений в виде $A\vec{x} = \vec{b}$, а целевую функцию как $\vec{c}^T\vec{x}$.

8.4.3. Выпуклое программирование

Решите задачу оптимизации:

$$\|A\vec{x} - \vec{b}\|_2^2 \to \min$$

при заданных ограничениях:

$$\vec{x} \succeq 0,$$

где $\vec{x} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $\vec{b} \in \mathbb{R}^m$.

Матрицу A и вектор $\vec{b}$ задайте случайным образом.

Для решения задачи используйте пакет Convex и решатель SCS.

Рис. 1: Задание_1

Задание лабораторной работы №7

8.4.4. Оптимальная рассадка по залам

Проводится конференция с 5 разными секциями. Забронировано 5 залов различной вместимости: в каждом зале не должно быть меньше 180 и больше 250 человек, а на третьей секции активность подразумевает, что должно быть точно 220 человек.

В заявке участник указывает приоритет посещения секции: 1 — максимальный приоритет, 3 — минимальный, а значение 10000 означает, что человек не пойдёт на эту секцию.

Организаторам удалось собрать 1000 заявок с указанием приоритета посещения трёх секций. Необходимо дать рекомендацию слушателю, на какую же секцию ему пойти, чтобы хватило места всем.

Для решения задачи используйте пакет `Convex` и решатель `GLPK`.

Приоритеты по слушателям распределите случайным образом.

8.4.5. План приготовления кофе

Кофейня готовит два вида кофе «Раф кофе» за 400 рублей и «Капучино» за 300. Чтобы сварить 1 чашку «Раф кофе» необходимо: 40 гр. зёрен, 140 гр. молока и 5 гр. ванильного сахара. Для того чтобы получить одну чашку «Капучино» необходимо потратить: 30 гр. зёрен, 120 гр. молока. На складе есть: 500 гр. зёрен, 2000 гр. молока и 40 гр. ванильного сахара.

Необходимо найти план варки кофе, обеспечивающий максимальную выручку от их реализации. При этом необходимо потратить весь ванильный сахар.

Для решения задачи используйте пакет `JuMP` и решатель `GLPK`.

Рис. 2: Задание_2

Выполнение лабораторной работы. Повторение примеров

С полным файлом лабораторной работы можно ознакомиться по ссылке ниже: https://github.com/OlgaLeb21/-/tree/main/lab08

Ниже приведена часть повторов примеров, данных в лабораторной работе для ознакомления: См. рис. 3, См. рис. 4, См. рис. 5.

```
[3]:  using JuMP
      using GLPK

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

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

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

[23]: y

```
[25]: # Определение ограничений модели:
      @constraint(model, 6x + 8y >= 100)
      @constraint(model, 7x + 12y >= 120)
```

[25]:

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

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

[27]: $12x + 20y$

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

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

[31]: OPTIMAL::TerminationStatusCode = 1

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

      value(x) = 14.999999999999993
      value(y) = 1.2500000000000047

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

      objective_value(model) = 205.0
```

Рис. 3: Линейное программирование_1

Выполнение лабораторной работы. Повторение примеров

```
[145]: # Считывание исходного изображения:
Kref = load("khiam-small.jpg")
```

[145]:

```
[163]: using Images
using Convex
using SCS

K = copy(Kref)
p = prod(size(K))
missingids = rand(1:p,400)
```

```
[163]: 400-element Vector{Int64}:
 42377
 50744
```

```
[165]: K[missingids] .= RGBX{N0f8}(0.0,0.0,0.0)
K
Gray.(K)
```

[165]:

Рис. 4: Восстановление изображения_2

Выполнение лабораторной работы. Повторение примеров

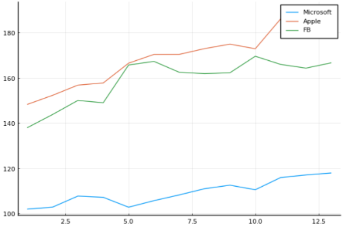
[105]:
```
# Считываем данные и размещаем их во фрейм:
T = DataFrame(XLSX.readtable("stock_prices.xlsx","Sheet2"))
```

[105]: 13×3 DataFrame

Row	MSFT	FB	AAPL
	Any	Any	Any
1	101.93	137.95	148.26
2	102.8	143.8	152.29
3	107.71	150.04	156.82

[107]:
```
# Построение графика:
plot(T[!,:MSFT],label="Microsoft")
plot!(T[!,:AAPL],label="Apple")
plot!(T[!,:FB],label="FB")
```

[107]:



[109]:
```
# Данные о ценах на акции размещаем в матрице:
prices_matrix = Matrix(T)
```

[109]:
```
13×3 Matrix{Any}:
 101.93  137.95  148.26
 102.8   143.8   152.29
 107.71  150.04  156.82
 107.17  149.01  157.76
 102.78  165.71  166.52
 105.67  167.33  170.41
 108.22  162.5   170.42
 110.97  161.89  172.97
```

Рис. 5: Портфельные инвестиции_3

Самостоятельная работа

Выполним задания 8.4.1. Линейное программирование и 8.4.2. Линейное программирование. Использование массивов: См. рис. 6.

[185]:
```
c = [1, 2, 5]
A = [-1 1 3; 1 3 -7]
b = [-5, 10]
display(c); display(A); b
```

```
3-element Vector{Int64}:
 1
 2
 5
2×3 Matrix{Int64}:
 -1  1   3
  1  3  -7
```

[185]:
```
2-element Vector{Int64}:
 -5
 10
```

[187]:
```
model = Model(GLPK.Optimizer)
@variable(model, x[1:3]>=0)
```

[187]:
```
3-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
```

[189]:
```
@constraint(model, 0<=x[1]<=10)
```

[189]:

$$x_1 \in [0, 10]$$

[193]:
```
@objective(model, Max, transpose(c)*x)
```

[193]: $x_1 + 2x_2 + 5x_3$

[195]:
```
@constraint(model, A*x .<=b)
```

[195]:
```
2-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterf
ace.LessThan{Float64}}, ScalarShape}}:
 -x[1] + x[2] + 3 x[3] <= -5
 x[1] + 3 x[2] - 7 x[3] <= 10
```

[199]:
```
optimize!(model)
```

[201]:
```
println("Оптимальное значение функции: ", objective_value(model))
println("Оптимальное значение переменных :", value.(x))
```

```
Оптимальное значение функции: 19.0625
Оптимальное значение переменных :[10.0, 2.1875, 0.9375]
```

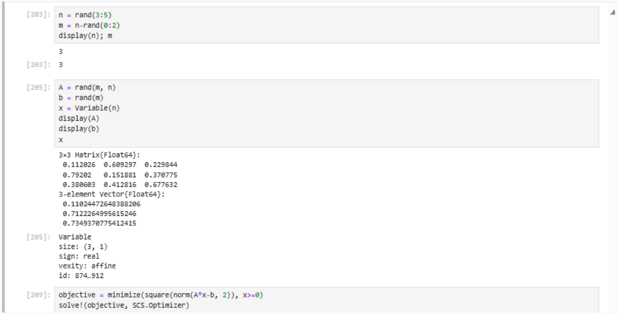
Рис. 6: Линейное программирование + использование массивов

Самостоятельная работа

Решение выполнено с использованием пакета JuMP и решателя GLPK. После построения модели и оптимизации найдено решение: целевая функция достигает максимального значения 19.025, при этом переменные принимают значения x1 = 10.0, x2 = 2.1875, x3 = 0.9375.

Тот же пример решен с использованием массивов вместо скалярных переменных. Матрица ограничений A вектор правых частей b, а также вектор коэффициентов целевой функции c были заданы в векторизованной форме. Векторы переменных и ограничения также были записаны в матричном виде. Оптимизация с использованием пакета JuMP дала тот же результат, что и в первом случае.

8.4.3. Выпуклое программирование: См. рис. 7, См. рис. 8



```
[203]: n = rand(3:5)
       m = n-rand(0:2)
       display(n); m

       3
[203]: 3

[205]: A = rand(m, n)
       b = rand(m)
       x = Variable(n)
       display(A)
       display(b)
       x

       3×3 Matrix{Float64}:
        0.112026  0.609297  0.229844
        0.79202   0.151881  0.370775
        0.380603  0.412816  0.677632
       3-element Vector{Float64}:
        0.11024472648388206
        0.7122264995615246
        0.7349370775412415
[205]: Variable
       size: (3, 1)
       sign: real
       vexity: affine
       id: 874…912

[209]: objective = minimize(square(norm(A*x-b, 2)), x>=0)
       solve!(objective, SCS.Optimizer)
```

Рис. 7: Выпуклое программирование_1

Самостоятельная работа

```
------------------------------------------------------------------
problem:  variables n: 5, constraints m: 10
cones:    l: linear vars: 3
          q: soc vars: 7, qsize: 2
settings: eps_abs: 1.0e-004, eps_rel: 1.0e-004, eps_infeas: 1.0e-007
          alpha: 1.50, scale: 1.00e-001, adaptive_scale: 1
          max_iters: 100000, normalize: 1, rho_x: 1.00e-006
          acceleration_lookback: 10, acceleration_interval: 10
          compiled with openmp parallelization enabled
lin-sys:  sparse-direct-amd-qdldl
          nnz(A): 16, nnz(P): 0
------------------------------------------------------------------
 iter | pri res | dua res |   gap   |   obj   |  scale  | time (s)
------------------------------------------------------------------
     0|1.45e+001 1.00e+000 2.00e+001 -9.98e+000 1.00e-001 7.57e-004
   125|7.12e-006 3.93e-006 4.26e-006 1.50e-002 8.98e-001 3.18e-003
------------------------------------------------------------------
status:  solved
timings: total: 3.19e-003s = setup: 2.06e-004s + solve: 2.98e-003s
         lin-sys: 1.10e-004s, cones: 6.38e-005s, accel: 2.27e-005s
------------------------------------------------------------------
objective = 0.014986
------------------------------------------------------------------
```

```
[209]: Problem statistics
  problem is DCP         : true
  number of variables    : 1 (3 scalar elements)
  number of constraints  : 1 (3 scalar elements)
  number of coefficients : 17
  number of atoms        : 7

Solution summary
  termination status : OPTIMAL
  primal status      : FEASIBLE_POINT
  dual status        : FEASIBLE_POINT
  objective value    : 0.015

Expression graph
  minimize
   └─ qol_elem (convex; positive)
      ├─ norm2 (convex; positive)
      │  └─ + (affine; real)
      │     ├─ …
      │     └─ …
      └─ [1.0;;]
  subject to
   └─ ≥ constraint (affine)
      └─ + (affine; real)
         ├─ 3-element real variable (id: 874…912)
         └─ Convex.NegateAtom (constant; negative)
            └─ …
```

```
[211]: println("Оптимальное значение: ", objective.optval)
       println("Оптимальное решение:", Convex.evaluate(x))
```

```
Оптимальное значение: 0.014984067573087215
Оптимальное решение:[0.5753648379975222, -2.5734995249755774e-6, 0.7009592329541373]
```

Рис. 8: Выпуклое программирование_2

Самостоятельная работа

Здесь решалась задача выпуклого программирования с минимизацией функции. Матрица A и вектор b были сгенерированы случайным образом. Для построения модели использовался пакет Convex, а решателем выступил SCS. Задача включала ограничение, чтобы все значения вектора x были неотрицательными.

После оптимизации модель нашла минимальное значение целевой функции. Алгоритм корректно справился с задачей, предоставив минимальное значение отклонения и решение, которое лежит в допустимой области. Данный подход может быть полезен для задач регрессии, обработки данных или других оптимизационных задач, связанных с минимизацией ошибок.

8.4.4. Оптимальная рассадка по залам: См. рис. 9, См. рис. 10, См. рис. 11.

```
[217]: using Random

       zals_str = collect(1:5)
       zals_data = JuMP.Containers. DenseAxisArray(
           [180 250;
               180 250
               220 220;
               180 250
               180 250],
           zals_str,
           ["min", "max"])

[217]: 2-dimensional DenseAxisArray{Int64,2,...} with index sets:
           Dimension 1, [1, 2, 3, 4, 5]
           Dimension 2, ["min", "max"]
       And data, a 5×2 Matrix{Int64}:
        180  250
        180  250
        220  220
        180  250
        180  250

[219]: N = 1000
       people = collect(1:N)
       people_pref = copy(hcat([shuffle([1,2,3,10000, 10000]) for i in people]...))

[219]: 5×1000 Matrix{Int64}:
            3      3      1      1      1  …  10000  10000      2      1      3
        10000      2      3  10000      3         3      2      1      3  10000
            2  10000  10000      3      2     10000      3      3  10000      1
        10000  10000      2  10000  10000         2      1  10000  10000      2
            1      1  10000      2  10000         1  10000  10000      2  10000

[221]: model_zal = Model(GLPK.Optimizer)

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

[227]: @variable(model_zal, answ[people, zals_str], Bin)

[227]: 2-dimensional DenseAxisArray{VariableRef,2,...} with index sets:
           Dimension 1, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10  …  991, 992, 993, 994, 995, 996, 997, 998, 999, 1000]
           Dimension 2, [1, 2, 3, 4, 5]
       And data, a 1000×5 Matrix{VariableRef}:
        answ[1,1]     answ[1,2]     answ[1,3]     answ[1,4]     answ[1,5]
        answ[2,1]     answ[2,2]     answ[2,3]     answ[2,4]     answ[2,5]
        answ[3,1]     answ[3,2]     answ[3,3]     answ[3,4]     answ[3,5]
        answ[4,1]     answ[4,2]     answ[4,3]     answ[4,4]     answ[4,5]
        answ[5,1]     answ[5,2]     answ[5,3]     answ[5,4]     answ[5,5]
        answ[6,1]     answ[6,2]     answ[6,3]     answ[6,4]     answ[6,5]
        answ[7,1]     answ[7,2]     answ[7,3]     answ[7,4]     answ[7,5]
        answ[8,1]     answ[8,2]     answ[8,3]     answ[8,4]     answ[8,5]
```

Рис. 9: Оптимальная рассадка по залам. 1

Самостоятельная работа

```
[231]: for i in people
           @constraint(model_zal, sum(answ[i,:]) ==1)
       end
       for i in zals_str
           @constraint(model_zal, zals_data[i, "min"] <= sum(answ[:,i]) <= zals_data[i, "max"])
       end
```

```
[235]: @objective(model_zal, Min, sum([sum([answ[t, c]*people_pref[c,t] for c in zals_str]) for t in people]))
```

[235]: $3answ_{1,1} + 10000answ_{1,2} + 2answ_{1,3} + 10000answ_{1,4} + answ_{1,5} + 3answ_{2,1} + 2answ_{2,2} + 10000answ_{2,3} + 10000answ_{2,4} + answ_{2,5} + answ_{3,1}$

```
[237]: optimize!(model_zal)
```

```
[239]: termination_status(model_zal)
```

```
[239]: OPTIMAL::TerminationStatusCode = 1
```

```
[241]: res = value.(answ)
```

```
[241]: 2-dimensional DenseAxisArray{Float64,2,...} with index sets:
           Dimension 1, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10  …  991, 992, 993, 994, 995, 996, 997, 998, 999, 1000]
           Dimension 2, [1, 2, 3, 4, 5]
       And data, a 1000×5 Matrix{Float64}:
        0.0  0.0  0.0  0.0  1.0
        0.0  0.0  0.0  0.0  1.0
        1.0  0.0  0.0  0.0  0.0
        1.0  0.0  0.0  0.0  0.0
        1.0  0.0  0.0  0.0  0.0
        0.0  0.0  1.0  0.0  0.0
        0.0  0.0  0.0  0.0  1.0
        0.0  0.0  0.0  0.0  1.0
        0.0  0.0  0.0  1.0  0.0
        0.0  0.0  0.0  1.0  0.0
        1.0  0.0  0.0  0.0  0.0
        1.0  0.0  0.0  0.0  0.0
        0.0  1.0  0.0  0.0  0.0
        ⋮
```

```
[245]: zals_filling = zeros(5)
       recomendations = zeros(N)
       for i in people
           for j in zals_str
               zals_filling[j] += res[i,j]
               if res[i,j] ==1
                   recomendations[i] = j
               end
           end
       end
```

```
[247]: zals_filling
```

Рис. 10: Оптимальная рассадка по залам_2

Самостоятельная работа

```
[247]: zals_filling

[247]: 5-element Vector{Float64}:
        198.0
        193.0
        220.0
        209.0
        180.0

[249]: recomendations

[249]: 1000-element Vector{Float64}:
        5.0
        5.0
        1.0
        1.0
        1.0
        3.0
        5.0
        5.0
        4.0
        4.0
        1.0
        1.0
        2.0
        ⋮
        4.0
        1.0
        5.0
        1.0
        4.0
        2.0
        2.0
        5.0
        4.0
        2.0
        1.0
        3.0
```

Рис. 11: Оптимальная рассадка по залам_3

Самостоятельная работа

Заданы пять залов с ограничениями по вместимости, с фиксацией числа участников для третьего зала. Сформирован массив из 1000 участников и их предпочтений по трём секциям. Построена бинарная матрица переменных answ[i,j], где i — участник, а j — секция, с условием, что каждый участник в выбирает ровно одну секцию. Добавлены ограничения на заполняемость залов (не меньше минимальной и не больше максимальной). Целевая функция минимизировала суммарные предпочтения участников, обеспечивая максимальное соответствие их желаниям.

Самостоятельная работа

Использовался пакет JuMP и решатель GLPK. После запуска оптимизатора статус задачи показал, что найдено оптимальное решение.

Итоговая заполняемость залов: [198,193,220,209,180], что соответствует ограничениям. Участникам предложены секции, которые в наибольшей степени удовлетворяют их предпочтения. Решение выполнено корректно: учтены все ограничения и предпочтения участников, и распределение получилось оптимальным.

Самостоятельная работа

8.4.5. План приготовления кофе: См. рис. 12.

```
[259]: model = Model(GLPK.Optimizer)
       @variable(model, raf >=0)
       @variable(model, cappuccino >=0)
```

[259]: $cappuccino$

```
[261]: const grain_limit = 500
       @constraint(model, raf*40 + cappuccino * 30 <= grain_limit)
```

[261]:

$$40raf + 30cappuccino \leq 500$$

```
[263]: const milk_limit = 2000
       @constraint(model, raf*140 + cappuccino* 120 <= milk_limit)
```

[263]:

$$140raf + 120cappuccino \leq 2000$$

```
[265]: const sugar_limit = 40
       @constraint(model, raf*5 == sugar_limit)
```

[265]:

$$5raf = 40$$

```
[271]: objective = 400*raf + 300* cappuccino
       @objective(model, Max, objective)
```

[271]: $400raf + 300cappuccino$

```
[273]: optimize!(model)
```

```
[275]: println("Раф: ", round(value(raf)))
       println("Капучино: ", round(value(cappuccino)))
       println("Прибыль: ", value(objective))

Раф: 8.0
Капучино: 6.0
Прибыль: 5000.0
```

Рис. 12: План приготовления кофе

Самостоятельная работа

Решение было получено с помощью пакета JuMP и решателя GLPK. Оптимизация показала статус OPTIMAL, что указывает на нахождение лучшего возможного результата.

Полностью использованы все ограничения, включая фиксированное использование сахара. Найдена комбинация, обеспечивающая максимальный доход. Все ресурсы (зёрна, молоко) использованы рационально, что подтверждает корректную постановку и решение задачи.

Оптимальное количество: «Раф»: 8 чашек. «Капучино»: 6 чашек. Максимальная прибыль: 5000 рублей

Полученные результаты

В ходе выполнения лабораторной работы были изучены и применены подходы к решению задач оптимизации с использованием языка программирования Julia и библиотек JuMP и Convex. Реализованы модели для линейного и выпуклого программирования, а также для оптимизации с конкретными практическими ограничениями.

Вывод

Освоили пакеты Julia для решения задач оптимизации.

Список литературы

[1] Julia: https://ru.wikipedia.org/wiki/Julia