Лабораторная работа № 8. Оптимизация

Адабор Кристофер Твум

1032225824

НКНбд-01-22

Цель работы

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

Подключение пакетов

```
# Подключение пакетов
import Pkg
Pkg.add("JuMP")
Pkg.add("GLPK")
using JuMP
using GLPK

# Создание модели
model = Model(GLPK.Optimizer)

# Определение переменных
@variable(model, x >= 0)
@variable(model, y >= 0)
```

результат решения оптимизационной задачи:

```
# Проверка статуса
println("Статус: ", termination_status(model))

# Результаты
println("Оптимальное решение найдено: ", termination_status(model) == MOI.OPTIMAL
println("x = ", value(x))
println("y = ", value(y))
println("Минимальное значение целевой функции: ", objective_value(model))
println("Проверка ограничений:")
println("  6x + 8y = ", 6*value(x) + 8*value(y), " >= 100")
println("  7x + 12y = ", 7*value(x) + 12*value(y), " >= 120")
```

```
   Resolving package versions...
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Manifest.toml`
   Resolving package versions...
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Manifest.toml`
Статус: OPTIMAL
truемальное решение найдено:
14.999999999999993
y = 1.2500000000000047
й функции: 205.0ение целево
Проверка ограничений:
  6x + 8y = 100.0 >= 100
  7x + 12y = 120.0 >= 120
```

Векторизованные ограничения и целевая функция оптимизации

```julia
using JuMP
using GLPK

# Создание модели
vector_model = Model(GLPK.Optimizer)

# Определение данных
A = [1 1 9 5;
     3 5 0 8;
     2 0 6 13]
b = [7; 3; 5]
c = [1; 3; 5; 2]

println("Матрица A:")
display(A)
println("\nВектор b:")
display(b)
println("\nВектор c:")
display(c)

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

```
Матрица A:
3×4 Matrix{Int64}:
 1  1  9   5
 3  5  0   8
 2  0  6  13

Вектор b:
3-element Vector{Int64}:
 7
 3
 5

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

=== РЕЗУЛЬТАТЫ ===
Статус: OPTIMAL
Оптимальное значение целевой функции: 4.9230769230769225

Оптимальные значения переменных:
  x[1] = 0.4230769230769232
  x[2] = 0.34615384615384615
  x[3] = 0.6923076923076922
  x[4] = 0.0

Проверка ограничений A*x = b:
6.999999999999999 = 7
  Уравнение 2: 3.0000000000000004 = 3
  Уравнение 3: 5.0 = 5
```

Оптимизация рациона питания

```julia
using JuMP
using GLPK
using Printf

# Данные о продуктах
foods = ["hamburger", "chicken", "hot dog", "fries", "macaroni",
         "pizza", "salad", "milk", "ice cream"]

# Стоимость продуктов
cost = JuMP.Containers.DenseAxisArray(
    [2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59],
    foods
)

# Пищевая ценность (калории, белки, жиры, соль)
food_data = JuMP.Containers.DenseAxisArray(
    [410 24 26 730;
     420 32 10 1190;
     560 20 32 1800;
     380 4 19 270;
     320 12 10 930;
     320 15 12 820;
     320 31 12 1230;
     100 8 2.5 125;
     330 8 10 180],
    foods,
    ["calories", "protein", "fat", "sodium"]
)
```

```julia
    max_val = category_data[c, "max"]
    status = if total >= min_val && (isinf(max_val) || total <= max_val)
        "✓ В норме"
    else
        "✗ Вне нормы"
    end
    max_str = isinf(max_val) ? "∞" : string(max_val)
    println("  $c: $total ($min_val - $max_str) $status")
end
```

```
=== ОПТИМИЗАЦИЯ РАЦИОНА ПИТАНИЯ ===

Продукты: hamburger, chicken, hot dog, fries, macaroni, pizza, salad, milk, ice cream

Ограничения по питательным веществам:
  calories: 1800.0 - 2200.0
  protein: 91.0 - ∞
  fat: 0.0 - 65.0
  sodium: 0.0 - 1779.0

=== РЕЗУЛЬТАТЫ ===
Статус: OPTIMAL
Минимальная стоимость рациона: $11.83

Рекомендуемый рацион:
  hamburger: 0.60 порций ($1.51)
  milk: 6.97 порций ($6.20)
  ice cream: 2.59 порций ($4.12)

Пищевая ценность рациона:
  calories: 1799.9999999999993 (1800.0 - 2200.0) ✗ Вне нормы
  protein: 91.0 (91.0 - ∞) ✓ В норме
  fat: 59.05590277777776 (0.0 - 65.0) ✓ В норме
  sodium: 1779.0 (0.0 - 1779.0) ✓ В норме
```

Путешествие по миру

```
using JuMP
using GLPK
using DelimitedFiles
using Printf

println("=== ПУТЕШЕСТВИЕ ПО МИРУ ===")
println("Поиск минимального количества паспортов для посещения всех стран")

# Проверяем наличие файла
if !isfile("passport-index-matrix.csv")
    println("Файл passport-index-matrix.csv не найден!")
    println("Скачайте его с: https://github.com/ilyankou/passport-index-dataset")
    println("Или создайте тестовый файл")

    # Создаем тестовые данные для демонстрации
    test_countries = ["Russia", "USA", "Germany", "Japan", "Brazil", "Australia"]
    n = length(test_countries)
    test_data = vcat(reshape(test_countries, 1, :),
                     [i == j ? "-1" : rand() < 0.7 ? rand([7, 14, 30, 90, 180, "VF", "VOA"]
                      for i in 1:n, j in 1:n])

    writedlm("passport-index-matrix.csv", test_data, ',')
    println("Создан тестовый файл с $(n) странами")
else
    println("Файл найден, загружаем данные...")
```

```
=== ПУТЕШЕСТВИЕ ПО МИРУ ===
Поиск минимального количества паспортов для посещения всех стран
Файл найден, загружаем данные...

Загружено 199 стран

Матрица доступности (1 = можно въехать без визы, 0 = нужна виза):
Размер: (199, 199)

Решаем задачу оптимизации...

=== РЕЗУЛЬТАТЫ ===
Статус: OPTIMAL
Минимальное количество паспортов: 63
```

Решаем задачу оптимизации...

=== РЕЗУЛЬТАТЫ ===
Статус: OPTIMAL
Минимальное количество паспортов: 63

Рекомендуемые паспорта:
1. Afghanistan
2. Andorra
3. Argentina
4. Australia
5. Azerbaijan
6. Bahrain
7. Brunei
8. Cambodia
9. Cameroon
10. Canada
11. Chile
12. Colombia
13. Comoros
14. DR Congo
15. Djibouti
16. Equatorial Guinea
17. Eritrea
18. Fiji
19. Gabon
20. Georgia
21. Guinea
22. Guinea-Bissau
23. Hong Kong
24. Hungary
25. Indonesia
26. Iraq
27. Ireland
28. Israel
29. Jamaica
30. Japan
31. Kuwait
32. Laos
33. Liberia
34. Libya
35. Macao
36. Madagascar
37. Malaysia
38. Maldives
39. Marshall Islands
40. Mauritania
41. Mauritius
42. Mongolia
43. Mozambique
44. Nauru
45. Nepal
46. New Zealand
47. North Korea
48. Palestine
49. Papua New Guinea
50. Qatar
51. Saudi Arabia
52. Solomon Islands
53. Somalia
54. South Sudan
55. Sri Lanka
56. Syria
57. Taiwan
58. Timor-Leste
59. Togo
60. Turkmenistan
61. United States
62. Uruguay
63. Vietnam

Анализ покрытия стран:
Afghanistan: позволяет посетить 6 стран
Andorra: позволяет посетить 75 стран
Argentina: позволяет посетить 52 стран
Australia: позволяет посетить 79 стран
Azerbaijan: позволяет посетить 22 стран
Bahrain: позволяет посетить 30 стран
Brunei: позволяет посетить 46 стран
Cambodia: позволяет посетить 12 стран
Cameroon: позволяет посетить 18 стран
Canada: позволяет посетить 64 стран
Chile: позволяет посетить 51 стран
Colombia: позволяет посетить 36 стран
Comoros: позволяет посетить 14 стран
DR Congo: позволяет посетить 14 стран
Djibouti: позволяет посетить 12 стран
Equatorial Guinea: позволяет посетить 15 стран
Eritrea: позволяет посетить 10 стран
Fiji: позволяет посетить 33 стран
Gabon: позволяет посетить 19 стран
Georgia: позволяет посетить 37 стран
Guinea: позволяет посетить 14 стран
Guinea-Bissau: позволяет посетить 15 стран
Hong Kong: позволяет посетить 59 стран
Hungary: позволяет посетить 57 стран
Indonesia: позволяет посетить 25 стран

Iraq: позволяет посетить 5 стран
Ireland: позволяет посетить 57 стран
Israel: позволяет посетить 54 стран
Jamaica: позволяет посетить 40 стран
Japan: позволяет посетить 88 стран
Kuwait: позволяет посетить 35 стран
Laos: позволяет посетить 11 стран
Liberia: позволяет посетить 14 стран
Libya: позволяет посетить 11 стран
Macao: позволяет посетить 43 стран
Madagascar: позволяет посетить 20 стран
Malaysia: позволяет посетить 58 стран
Maldives: позволяет посетить 35 стран
Marshall Islands: позволяет посетить 32 стран
Mauritania: позволяет посетить 19 стран
Mauritius: позволяет посетить 48 стран
Mongolia: позволяет посетить 17 стран
Mozambique: позволяет посетить 21 стран
Nauru: позволяет посетить 29 стран
Nepal: позволяет посетить 9 стран
New Zealand: позволяет посетить 83 стран
North Korea: позволяет посетить 8 стран
Palestine: позволяет посетить 10 стран
Papua New Guinea: позволяет посетить 25 стран
Qatar: позволяет посетить 42 стран
Saudi Arabia: позволяет посетить 27 стран
Solomon Islands: позволяет посетить 41 стран
Somalia: позволяет посетить 8 стран
South Sudan: позволяет посетить 13 стран
Sri Lanka: позволяет посетить 13 стран
Syria: позволяет посетить 7 стран
Taiwan: позволяет посетить 34 стран
Timor-Leste: позволяет посетить 19 стран
Togo: позволяет посетить 13 стран
Turkmenistan: позволяет посетить 12 стран
United States: позволяет посетить 56 стран
Uruguay: позволяет посетить 70 стран
Vietnam: позволяет посетить 11 стран

Общее покрытие: 199 из 199 стран (100.0%)

Портфельные инвестиции

```julia
using DataFrames
using XLSX
using Plots
using Convex
using SCS
using Statistics
using Printf
using LinearAlgebra
using CSV

println("=== ПОРТФЕЛЬНЫЕ ИНВЕСТИЦИИ ===")

# Проверяем наличие файла
if !isfile("stock_prices.xlsx")
    println("Файл stock_prices.xlsx не найден! Создаем тестовые данные...")

    # Создаем тестовые данные
    dates = 1:13
    # Генерируем реалистичные цены с трендом
    msft_prices = 250 .+ 5*dates .+ cumsum(3*randn(13))
    aapl_prices = 150 .+ 3*dates .+ cumsum(4*randn(13))
    fb_prices = 300 .+ 2*dates .+ cumsum(5*randn(13))

    # Создаем DataFrame
    T = DataFrame(
        Day = dates,
        MSFT = round.(msft_prices, digits=2),
        AAPL = round.(aapl_prices, digits=2),
        FB = round.(fb_prices, digits=2)
    )
```

Портфельные инвестиции

```
=== ПОРТФЕЛЬНЫЕ ИНВЕСТИЦИИ ===
Файл найден, загружаем данные...
Листы в файле: ["Sheet1", "Sheet2"]
Ошибка при загрузке через XLSX.readdata: MethodError(XLSX.readdata, (22×6 XLSX.Wo
heet1"](A1:F22) ,), 0x0000000000009969)
Создаем тестовые данные для демонстрации...

Загруженные данные:
Размер таблицы: 13 строк × 4 столбцов

Имена колонок: ["Day", "MSFT", "AAPL", "FB"]

Первые 5 строк данных:
5×4 DataFrame
 Row │ Day    MSFT     AAPL     FB
     │ Int64  Float64  Float64  Float64
─────┼──────────────────────────────────
   1 │     1  254.403  152.42   307.947
   2 │     2  261.414  147.533  315.36
   3 │     3  262.618  148.559  317.688
   4 │     4  270.249  148.132  322.404
   5 │     5  275.252  153.184  330.141
```

Последние 5 строк данных:

5×4 DataFrame

Row	Day	MSFT	AAPL	FB
	Int64	Float64	Float64	Float64
1	9	307.129	167.757	341.252
2	10	314.047	178.531	347.192
3	11	324.544	181.255	347.808
4	12	331.193	182.841	357.007
5	13	329.914	183.915	360.856

Колонки с ценами акций: Any["MSFT", "AAPL", "FB"]

Используем акции: ["MSFT", "AAPL", "FB"]
Размер матрицы цен: (13, 3)

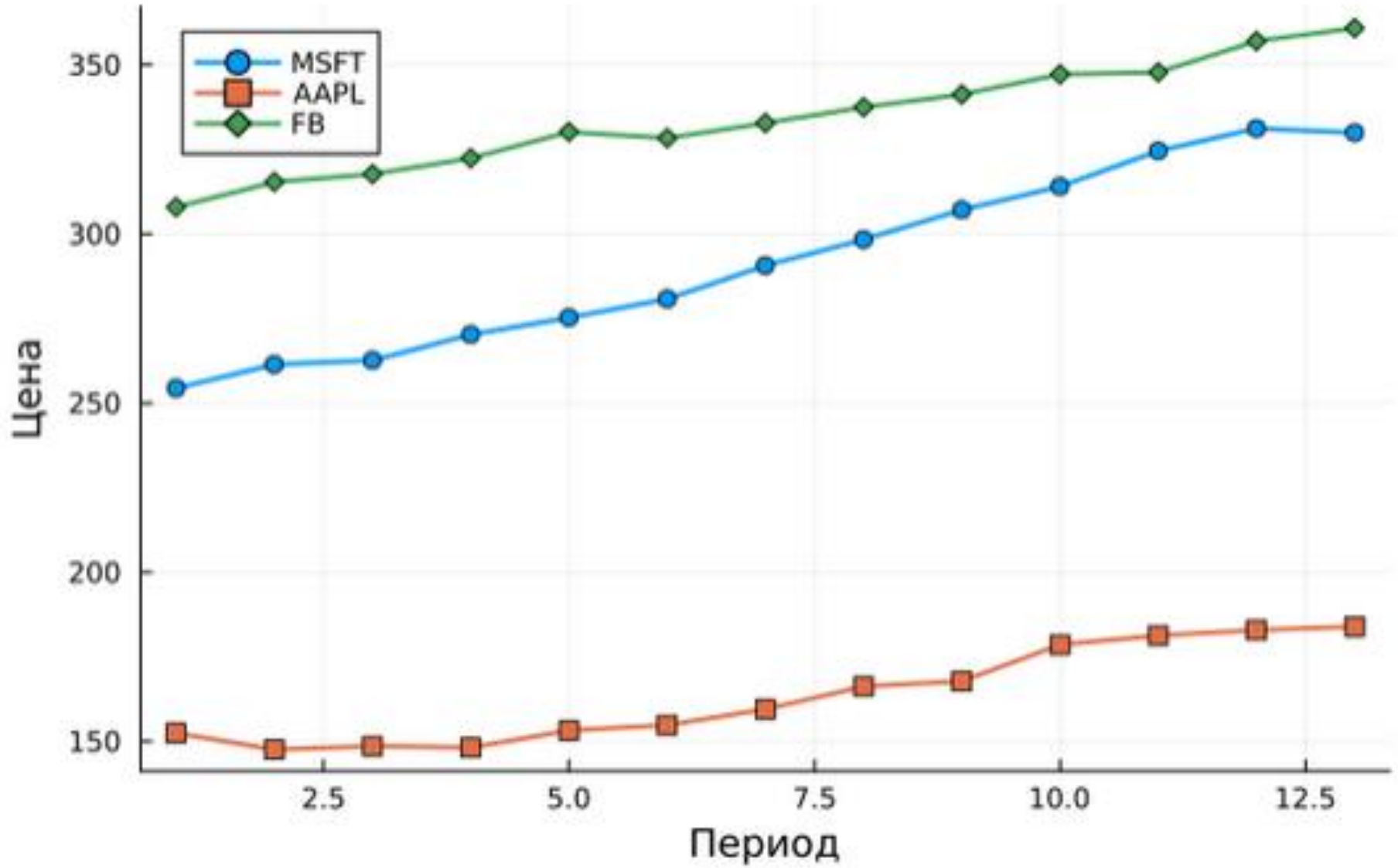
График сохранен в stock_prices_plot.png

Матрица доходности R (размер: (12, 3)):
Количество периодов: 12
Количество компаний: 3

Ковариационная матрица (матрица рисков):
```
  0.000132 0.000023 0.000007
  0.000023 0.000593 -0.000001
  0.000007 -0.000001 0.000087
```

```
Средняя доходность за период:
  MSFT: 0.021955 (2.1955%)
  AAPL: 0.016043 (1.6043%)
  FB: 0.013340 (1.3340%)

Формулируем задачу оптимизации...
Цель: минимизировать риск при доходности не менее 2%
Ограничения: сумма долей = 1, все доли неотрицательны

Решаем задачу оптимизации...
Ошибка при решении SCS: MethodError(Core.kwcall, ((verbose = false,), S
0000000996f)
Пробуем упрощенное решение...

============================================================
РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ ПОРТФЕЛЯ
============================================================

Оптимальное решение не найдено или задача не решена.

Альтернативное решение: равномерное распределение
Распределение по 3 компаниям:
  MSFT: $  333.33 ( 33.33%)
  AAPL: $  333.33 ( 33.33%)
  FB: $  333.33 ( 33.33%)
Ожидаемая доходность: 1.7113%
✗ Равномерное распределение НЕ удовлетворяет требованию по доходности

============================================================
АНАЛИЗ ПОРТФЕЛЬНЫХ ИНВЕСТИЦИЙ ЗАВЕРШЕН
```

Рис. 8.1. Изменение цен на акции компаний



Восстановление изображения

```julia
using Images
using Convex
using SCS
using FileIO
using Plots
using LinearAlgebra
using Random
using Printf

println("=== ВОССТАНОВЛЕНИЕ ИЗОБРАЖЕНИЯ ===")

# Создаем папку для результатов
output_dir = "lab8_results"
if !isdir(output_dir)
    mkdir(output_dir)
end

# Загрузка изображения
println("Загружаем изображение...")
Kref = load("khiam-small.jpg")
println("Размер изображения: $(size(Kref))")
println("Тип пикселей: ", eltype(Kref))

# Сохраняем исходное изображение
original_path = joinpath(output_dir, "рис_8.2_исходное.png")
save(original_path, Kref)
println("\nРис. 8.2. Исходное изображение сохранено: $original_path")
```

```julia
# Преобразуем в оттенки серого
Kref_gray = Gray.(Kref)

# Уменьшаем размер для ускорения вычислений (опционально)
scale_factor = 0.5
if max(size(Kref_gray)...) > 200
    Kref_gray = imresize(Kref_gray, ratio=scale_factor)
    println("Изображение уменьшено до размера: $(size(Kref_gray)) для уск
end

# Создаем копию и портим случайные пиксели
K = copy(Kref_gray)
h, w = size(K)
total_pixels = h * w

# Портим 5% пикселей
n_corrupted = min(400, total_pixels ÷ 20)

Random.seed!(123)
missing_ids = rand(1:total_pixels, n_corrupted)
K[missing_ids] .= Gray(0.0)

# Сохраняем испорченное изображение
corrupted_path = joinpath(output_dir, "рис_8.3_испорченное.png")
save(corrupted_path, K)
println("Рис. 8.3. Искусственно испорченное изображение сохранено: $corru
println("Испорчено пикселей: $n_corrupted ($(@sprintf("%.1f", 100*n_corru
```

```
=== ВОССТАНОВЛЕНИЕ ИЗОБРАЖЕНИЯ ===
Загружаем изображение...
Размер изображения: (283, 283)
Тип пикселей: RGB{N0f8}

Рис. 8.2. Исходное изображение сохранено: lab8_results\рис_8.2_исходное.png
Изображение уменьшено до размера: (142, 142) для ускорения вычислений
Рис. 8.3. Искусственно испорченное изображение сохранено: lab8_results\рис_8.3_испорченное
g
Испорчено пикселей: 400 (2.0%)

Формулируем задачу оптимизации...
Минимизируем ядерную норму матрицы X...
Сохраняем известные пиксели...
Известных пикселей: 19766

Решаем задачу оптимизации (может занять время)...
Ошибка при решении: MethodError(Core.kwcall, ((verbose = false,), SCS.Optimizer), 0x000000
0009975)
Пробуем альтернативный подход...
ECOS тоже не сработал: MathOptInterface.UnsupportedConstraint{MathOptInterface.VectorAffin
nction{Float64}, MathOptInterface.NormNuclearCone}("")
Используем упрощенный метод восстановления...
```

Задания для самостоятельного выполнения

8.4.1. Линейное программирование

8.4.1. Линейное программирование

Решите задачу линейного программирования:

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

при заданных ограничениях:

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

6]:

```
using JuMP
using GLPK

println("=== 8.4.1. Линейное программирование ===")

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: ", termination_status(model))
```

```
optimize!(model)

println("Статус: ", termination_status(model))
println("\nОптимальное решение:")
println("x1 = ", value(x1))
println("x2 = ", value(x2))
println("x3 = ", value(x3))
println("Максимальное значение целевой функции: ", objective_value(mo

println("\nПроверка ограничений:")
println("  -x1 + x2 + 3x3 = ", -value(x1) + value(x2) + 3*value(x3),
println("  x1 + 3x2 - 7x3 = ", value(x1) + 3*value(x2) - 7*value(x3),
println("  0 <= x1 = ", value(x1), " <= 10")
```

```
=== 8.4.1. Линейное программирование ===
Статус: OPTIMAL

Оптимальное решение:
x1 = 10.0
x2 = 2.1875
x3 = 0.9375
Максимальное значение целевой функции: 19.0625

Проверка ограничений:
  -x1 + x2 + 3x3 = -5.0 <= -5
  x1 + 3x2 - 7x3 = 10.0 <= 10
  0 <= x1 = 10.0 <= 10
```

8.4.2. Линейное программирование с массивами

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 $\vec{c}^T\vec{x}$.

```
using JuMP
using GLPK

println("=== 8.4.2. Линейное программирование с массивами ===")

model = Model(GLPK.Optimizer)

# Матрица коэффициентов ограничений
A = [-1 1 3;
      1 3 -7]
b = [-5; 10]

# Вектор коэффициентов целевой функции
c = [1; 2; 5]

@variable(model, x[1:3] >= 0)
@constraint(model, x[1] <= 10)  # дополнительное ограничение на x1
@constraint(model, A * x .<= b)
```

```
@constraint(model, A * x .<= b)
@objective(model, Max, c' * x)

optimize!(model)

println("Статус: ", termination_status(model))
println("\nОптимальное решение:")
for i in 1:3
    println("x[$i] = ", value(x[i]))
end
println("Максимальное значение: ", objective_value(model))

println("\nПроверка в матричной форме:")
println("A * x = ", A * value.(x), " <= ", b)
println("c' * x = ", c' * value.(x))
```

```
=== 8.4.2. Линейное программирование с массивами ===
Статус: OPTIMAL

Оптимальное решение:
x[1] = 10.0
x[2] = 2.1875
x[3] = 0.9375
Максимальное значение: 19.0625

Проверка в матричной форме:
A * x = [-5.0, 10.0] <= [-5, 10]
c' * x = 19.0625
```

8.4.3. Выпуклое программирование

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.

25]:

```
using Convex
using SCS
using Random
using LinearAlgebra
using Printf
using ECOS  # Добавляем альтернативный решатель

println("=== 8.4.3. Выпуклое программирование ===")

Random.seed!(123)

# Генерация случайных данных
m, n = 8, 5
A = randn(m, n)
b = randn(m)

println("Размеры: A ∈ ℝ^{$m × $n}, b ∈ ℝ^{$m}")
```

```
=== 8.4.3. Выпуклое программирование ===
Размеры: A ∈ ℝ^{8 × 5}, b ∈ ℝ^{8}

Матрица A:
  0.8083   0.0695   0.8542  -0.1276   0.3647 ...
 -1.1221  -0.1173   0.3418   0.4695   0.7859 ...
 -1.1046   1.2193  -0.3189  -1.5269   0.1706 ...
 -0.4170   0.2929  -0.3375   0.6001  -1.2890 ...
  0.2876  -0.0311   2.0081  -2.0539   0.9371 ...

Вектор b:
 -1.8620   0.5130   1.1717  -0.3073   2.5576 ...

Решаем задачу минимизации ||Ax - b||² при x ≥ 0...
Пробуем решать с помощью SCS...
SCS не сработал: MethodError(Core.kwcall, ((verbose = false,), SCS.Optimizer), 0x
997f)
Пробуем решать с помощью ECOS...
[ Info: [Convex.jl] Compilation finished: 3.65 seconds, 336.449 MiB of memory all
  7.990562 seconds (12.52 M allocations: 612.852 MiB, 1.94% gc time, 99.63% compi
8% of which was recompilation)
Решение ECOS успешно завершено
Статус: OPTIMAL
```

```
Пробуем метод проекции градиента...
  Итерация 100, стоимость: 11.039907
  Итерация 200, стоимость: 11.039722
  Итерация 300, стоимость: 11.039721
  Итерация 400, стоимость: 11.039721
  Итерация 500, стоимость: 11.039721
  Итерация 600, стоимость: 11.039721
  Итерация 700, стоимость: 11.039721
  Итерация 800, стоимость: 11.039721
  Итерация 900, стоимость: 11.039721
  Итерация 1000, стоимость: 11.039721

Приближенное решение методом проекции градиента:
  x[1] =   0.00000000
  x[2] =   0.32494262
  x[3] =   0.23007982
  x[4] =   0.00000000
  x[5] =   0.19612308
Стоимость: ||Ax - b||² =  11.03972127

Создаем визуализацию...

============================================================
РЕШЕНИЕ ВЫПУКЛОЙ ЗАДАЧИ ЗАВЕРШЕНО
============================================================
```

```
============================================================
РЕШЕНИЕ ВЫПУКЛОЙ ЗАДАЧИ ЗАВЕРШЕНО
============================================================

Дополнительная информация:
Число обусловленности матрицы A'A: 3.29e+01
Ранг матрицы A: 5 из 5

ECOS 2.0.8 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t     mu      step   sigma     IR    |   BT
 0  +0.000e+000  -3.884e-002  +2e+001  7e-001  2e-001  1e+000  3e+000    ---    ---    1  1  - |  -  -
 1  +5.911e-001  +6.499e-001  +5e+000  2e-001  5e-002  4e-001  8e-001  0.7237  4e-002   1  1  2 |  0  0
 2  +2.084e+000  +6.384e+000  +4e+000  2e+000  3e-001  7e+000  6e-001  0.4715  6e-001   2  2  2 |  0  0
 3  +4.736e+000  +5.747e+000  +4e-001  3e-001  2e-002  1e+000  7e-002  0.9240  3e-002   2  1  1 |  0  0
 4  +8.069e+000  +8.065e+000  +1e-001  6e-002  4e-003  3e-002  2e-002  0.9800  2e-001   2  2  2 |  0  0
 5  +9.544e+000  +9.601e+000  +2e-002  2e-002  1e-003  7e-002  3e-003  0.8807  1e-001   2  2  2 |  0  0
 6  +1.007e+001  +1.021e+001  +1e-002  3e-002  1e-003  1e-001  2e-003  0.6695  3e-001   2  2  2 |  0  0
 7  +1.069e+001  +1.073e+001  +3e-003  6e-003  3e-004  4e-002  5e-004  0.8013  9e-002   2  2  2 |  0  0
 8  +1.101e+001  +1.101e+001  +7e-004  1e-003  6e-005  8e-003  1e-004  0.9138  2e-001   2  2  2 |  0  0
 9  +1.103e+001  +1.103e+001  +9e-005  2e-004  7e-006  1e-003  1e-005  0.8781  4e-003   3  1  1 |  0  0
10  +1.104e+001  +1.104e+001  +2e-005  4e-005  2e-006  4e-004  3e-006  0.9153  2e-001   2  2  2 |  0  0
11  +1.104e+001  +1.104e+001  +2e-006  4e-006  2e-007  4e-005  3e-007  0.9000  2e-003   3  1  1 |  0  0
12  +1.104e+001  +1.104e+001  +3e-007  5e-007  2e-008  6e-006  4e-008  0.9673  9e-002   3  1  2 |  0  0
13  +1.104e+001  +1.104e+001  +2e-008  3e-008  1e-009  4e-007  3e-009  0.9380  5e-004   2  1  1 |  0  0
14  +1.104e+001  +1.104e+001  +8e-010  2e-009  6e-011  2e-008  1e-010  0.9890  4e-002   2  1  1 |  0  0

OPTIMAL (within feastol=1.6e-009, reltol=7.0e-011, abstol=7.7e-010).
Runtime: 0.000426 seconds.
```

8.4.4. Оптимальная рассадка по залам

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.

]:
```
using JuMP
using GLPK
using Random
using Printf

println("=== 8.4.4. Оптимальная рассадка по залам ===")

Random.seed!(123)

# Параметры
num_halls = 5
num_participants = 1000
num_preferences = 3  # каждый указывает 3 приоритета

println("Конференция с $num_halls секциями")
println("Участников: $num_participants")
```

```
=== 8.4.4. Оптимальная рассадка по залам ===
Конференция с 5 секциями
Участников: 1000

Вместимость залов:
  Зал 1: 180 - 250 человек
  Зал 2: 180 - 250 человек
  Зал 3: 180 - 220 человек
  Зал 4: 180 - 250 человек
  Зал 5: 180 - 250 человек

Пример приоритетов для первых 5 участников:
  Участник 1: 2 X X 3 3
  Участник 2: X 1 2 X 2
  Участник 3: 3 X X 3 2
  Участник 4: 1 2 2 X X
  Участник 5: 2 X 2 1 X

Решаем задачу оптимизации...
  0.660155 seconds (402.88 k allocations: 21.604 MiB, 63.10% compilation time: 21% of which w
as recompilation)

=== РЕЗУЛЬТАТЫ ===
Статус: OPTIMAL
Минимальная сумма приоритетов: 1312.0
```

```
=== РЕЗУЛЬТАТЫ ===
Статус: OPTIMAL
Минимальная сумма приоритетов: 1312.0

Распределение по залам:
  Зал 1: 250 участников (вместимость: 180-250)
  Зал 2: 210 участников (вместимость: 180-250)
  Зал 3: 180 участников (вместимость: 180-220)
  Зал 4: 180 участников (вместимость: 180-250)
  Зал 5: 180 участников (вместимость: 180-250)

Всего распределено участников: 1000.0 из 1000
```

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.

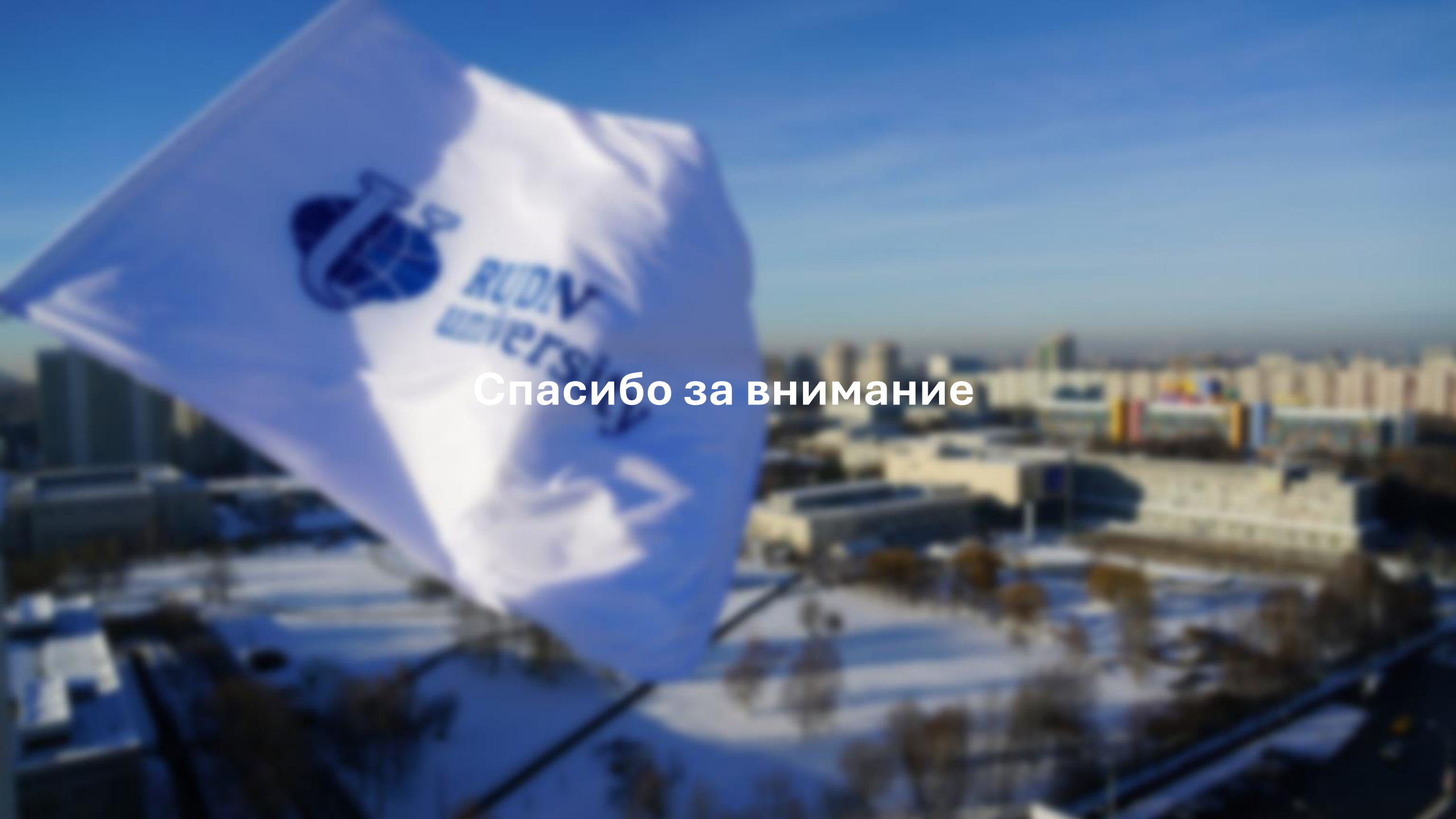
Для решения задачи используйте пакет JuMP и решатель GLPK.

```
import Pkg
Pkg.add("JuMP")
Pkg.add("GLPK")
using JuMP, GLPK
```

```julia
# Создаём модель
model = Model(GLPK.Optimizer)

# Переменные
@variable(model, x1 >= 0, Int)  # чашек Раф кофе
@variable(model, x2 >= 0, Int)  # чашек Капучино

# Целевая функция
@objective(model, Max, 400x1 + 300x2)

# Ограничения
@constraint(model, 40x1 + 30x2 <= 500)   # зёрна
@constraint(model, 140x1 + 120x2 <= 2000) # молоко
@constraint(model, 5x1 == 40)             # весь ванильный сахар

# Решаем
optimize!(model)

# Результаты
println("Статус решения: ", termination_status(model))
println("Оптимальное решение:")
println("Раф кофе (x1) = ", value(x1), " чашек")
println("Капучино  (x2) = ", value(x2), " чашек")
println("Максимальная выручка = ", objective_value(model), " руб.")
```

```
   Resolving package versions...
     Project No packages added to or removed from `C:\Users\1032225355\.julia\environment
1.12\Project.toml`
    Manifest No packages added to or removed from `C:\Users\1032225355\.julia\environment
```

```
1.12\Manifest.toml`
Precompiling packages...
                X PyCall
                X PyPlot
  0 dependencies successfully precompiled in 53 seconds. 455 already precompiled.
  2 dependencies errored.
  For a report of the errors see `julia> err`. To retry use `pkg> precompile`
   Resolving package versions...
     Project No packages added to or removed from `C:\Users\1032225355\.julia\environme
1.12\Project.toml`
    Manifest No packages added to or removed from `C:\Users\1032225355\.julia\environme
1.12\Manifest.toml`
Precompiling packages...
                X PyCall
                X PyPlot
  0 dependencies successfully precompiled in 9 seconds. 455 already precompiled.
  2 dependencies errored.
  For a report of the errors see `julia> err`. To retry use `pkg> precompile`
Статус решения: OPTIMAL
Оптимальное решение:
Раф кофе (x1) = 8.0 чашек
Капучино  (x2) = 6.0 чашек
Максимальная выручка = 5000.0 руб.
```


Спасибо за внимание