8.4. Задания для самостоятельного выполнения

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 ⩽ −5,

$x_1 + 3x_2 - 7x_3 \leqslant 10$,

$0 \leqslant x_1 \leqslant 10$,

$x_2 \geqslant 0$,

$x_3 \geqslant 0$.

In [1]:
```
# Подключение пакетов:
using JuMP
using GLPK
```

In [2]:
```
# Определение объекта модели с именем model:
task_1 = Model(GLPK.Optimizer)
```

Out[2]:

$$\begin{aligned} \text{feasibility}\\ \text{Subject to} \end{aligned}$$

In [3]:
```
# Определение переменных x1, x2, x3 и граничных условий для них:
@variable(task_1, 0 <= x1 <= 10)
@variable(task_1, x2 >= 0)
@variable(task_1, x3 >= 0)
# Определение ограничений модели:
@constraint(task_1, -x1 + x2 +3x3 <= -5)
@constraint(task_1, x1 + 3x2 - 7x3 <= 10)
# Определение целевой функции:
@objective(task_1, Max, x1 + 2x2 +5x3)
```

Out[3]:

$$x1 + 2x2 + 5x3$$

In [4]:
```
# Вызов функции оптимизации:
optimize!(task_1)
# Определение причины завершения работы оптимизатора:
termination_status(task_1)
```

Out[4]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [5]:
```
# Демонстрация первичных результирующих значений переменных x1, x2 и x3:
```

```
@show value(x1);
@show value(x2);
@show value(x3);
```

```
value(x1) = 10.0
value(x2) = 2.1875
value(x3) = 0.9375
```

In [6]:

```
# Демонстрация результата оптимизации:
@show objective_value(task_1);
```

```
objective_value(task_1) = 19.0625
```

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

In [7]:

```
# Определение объекта модели с именем vector_model:
task_2 = Model(GLPK.Optimizer)
```

Out[7]:

$$\text{feasibility}$$
$$\text{Subject to}$$

In [8]:

```
# Определение начальных данных:
A= [ -1 1 3;
     1 3 -7 ]
b = [-5; 10]
c = [1; 2; 5];
```

In [9]:

```
# Определение вектора переменных и граничных условий для них:
@variable(task_2, x[1:3] >= 0)
set_upper_bound(x[1], 10) # 0 ⩽ x1 ⩽ 10
```

In [10]:

```
# Определение ограничений модели:
@constraint(task_2, A * x .== b)
```

Out[10]:

```
2-element Array{ConstraintRef{Model,MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},MathOptInterface.EqualTo{Float64}},ScalarShape},1}:
 -x[1] + x[2] + 3 x[3] = -5.0
 x[1] + 3 x[2] - 7 x[3] = 10.0
```

In [11]:

```
# Определение целевой функции:
@objective(task_2, Max, c' * x)
```

Out[11]:

$$x_1 + 2x_2 + 5x_3$$

In [12]:

```
# Вызов функции оптимизации:
optimize!(task_2)
```

In [13]:

```
# Определение причины завершения работы оптимизатора:
termination_status(task_2)
```

Out[13]:

```
OPTIMAL::TerminationStatusCode = 1
```

In [14]:

```
# Демонстрация результата оптимизации:
@show objective_value(task_2);
```

```
objective_value(task_2) = 19.0625
```

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 .

In [15]:

```
using Convex, SCS
```

In [16]:

```
# Задаем переменные для определения размерности матрицы A
m = 5
n = 3
# Задаем матрицу A и вектор b
A = randn(m, n)
b = randn(m, 1)
```

Out[16]:

```
5×1 Array{Float64,2}:
  0.37974429867172604
 -1.5525315174702539
  0.4883412588264075
 -0.4069536239842097
  0.3731970925049839
```

In [17]:

```
# Создаем вектор-столбец размера n x 1
x = Variable(n)
```

Out[17]:

```
Variable
size: (3, 1)
sign: real
vexity: affine
id: 318…347
```

In [18]:

```
# Объект модели:
task_3 = minimize(sumsquares(A * x - b), [x >= 0])
```

Out[18]:
```
minimize
└─ qol_elem (convex; positive)
   ├─ norm2 (convex; positive)
   │  └─ + (affine; real)
   │     ├─ …
   │     └─ …
   └─ [1.0]
subject to
└─ >= constraint (affine)
   ├─ 3-element real variable (id: 318…347)
   └─ 0

status: `solve!` not called yet
```

In [19]:
```
# Находим решение:
solve!(task_3, SCS.Optimizer)
```

```
----------------------------------------------------------------------------
	SCS v2.1.2 - Splitting Conic Solver
	(c) Brendan O'Donoghue, Stanford University, 2012
----------------------------------------------------------------------------
Lin-sys: sparse-indirect, nnz in A = 24, CG tol ~ 1/iter^(2.00)
eps = 1.00e-05, alpha = 1.50, max_iters = 5000, normalize = 1, scale = 1.00
acceleration_lookback = 10, rho_x = 1.00e-03
Variables n = 6, constraints m = 14
Cones:	primal zero / dual free vars: 1
	linear vars: 4
	soc vars: 9, soc blks: 2
Setup time: 1.08e-02s
----------------------------------------------------------------------------
 Iter | pri res | dua res | rel gap | pri obj | dua obj | kap/tau | time (s)
----------------------------------------------------------------------------
     0| 1.60e+19  1.34e+19  1.00e+00 -3.92e+19  2.50e+19  5.11e+19  2.10e-05 
    37| 5.15e-11  1.37e-10  1.98e-11  2.41e+00  2.41e+00  1.11e-16  8.74e-03 
----------------------------------------------------------------------------
Status: Solved
Timing: Solve time: 8.75e-03s
	Lin-sys: avg # CG iterations: 2.00, avg solve time: 5.74e-07s
	Cones: avg projection time: 9.62e-08s
	Acceleration: avg step time: 2.28e-04s
----------------------------------------------------------------------------
Error metrics:
dist(s, K) = 2.2204e-16, dist(y, K*) = 0.0000e+00, s'y/|s||y| = 0.0000e+00
primal res: |Ax + s - b|_2 / (1 + |b|_2) = 5.1514e-11
dual res:   |A'y + c|_2 / (1 + |c|_2) = 1.3742e-10
rel gap:    |c'x + b'y| / (1 + |c'x| + |b'y|) = 1.9793e-11
----------------------------------------------------------------------------
c'x = 2.4056, -b'y = 2.4056
============================================================================
```

In [20]:
```
# Проверяем статус (найдено ли оптимальное решение)
task_3.status
```

Out[20]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [21]:
```
# Получаем оптимальное значение
task_3.optval
```

Out[21]:
```
2.405571018460159
```

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 начение 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.

In [ ]:

```

```

In [ ]:

```

```

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.

In [39]:

```
# Контейнер для хранения данных о запасах на складе:
coffee_data = JuMP.Containers.DenseAxisArray(
    [0 500;
    0 2000;
    0 40;],
    ["coffee bean", "milk", "vanilla sugar"],
    ["min", "max"])
```

Out[39]:

```
2-dimensional DenseAxisArray{Int64,2,...} with index sets:
    Dimension 1, ["coffee bean", "milk", "vanilla sugar"]
    Dimension 2, ["min", "max"]
And data, a 3×2 Array{Int64,2}:
 0   500
 0  2000
 0    40
```

In [41]:

```
# массив данных с наименованиями кофе:
coffee = ["raf coffee", "cappuccino"]
```

Out[41]:
```
2-element Array{String,1}:
 "raf coffee"
 "cappuccino"
```

In [42]:
```
# Массив стоимости кофе:
cost = JuMP.Containers.DenseAxisArray(
    [400, 300],
    coffee)
```

Out[42]:
```
1-dimensional DenseAxisArray{Int64,1,...} with index sets:
    Dimension 1, ["raf coffee", "cappuccino"]
And data, a 2-element Array{Int64,1}:
 400
 300
```

In [43]:
```
# Расход продуктов для приготовления кофе:
coffee_data = JuMP.Containers.DenseAxisArray(
    [40 140 5;
     30 120 0],
    coffee,
    ["coffee bean", "milk", "vanilla sugar"])
```

Out[43]:
```
2-dimensional DenseAxisArray{Int64,2,...} with index sets:
    Dimension 1, ["raf coffee", "cappuccino"]
    Dimension 2, ["coffee bean", "milk", "vanilla sugar"]
And data, a 2×3 Array{Int64,2}:
 40  140  5
 30  120  0
```

In [44]:
```
# Определение объекта модели с именем model:
model = Model(GLPK.Optimizer)
```

Out[44]:

$$\text{feasibility}$$
$$\text{Subject to}$$

In [45]:
```
# Определим массив:
products = ["coffee bean", "milk", "vanilla sugar"]
```

Out[45]:
```
3-element Array{String,1}:
 "coffee bean"
 "milk"
 "vanilla sugar"
```

In [46]:
```
# Определение переменных:
@variables(model, begin
coffee_data[c, "min"] <= nutrition[c = products] <= coffee_data[c, "max"]
# Сколько покупать продуктов:
buy[coffee] >= 0
end)
```

In [47]:
```
# Определение целевой функции:
@objective(model, Max, sum(cost[f] * buy[f] for f in coffee))
```

Out[47]:

$$400buy_{rafcoffee} + 300buy_{cappuccino}$$

In [48]:
```
# Определение ограничений модели:
@constraint(model, [c in products],
sum(coffee_data[f, c] * buy[f] for f in coffee) == nutrition[c])
```

Out[48]:
```
1-dimensional DenseAxisArray{ConstraintRef{Model,MathOptInterface.ConstraintIndex
{MathOptInterface.ScalarAffineFunction{Float64},MathOptInterface.EqualTo{Float6
4}},ScalarShape},1,...} with index sets:
    Dimension 1, ["coffee bean", "milk", "vanilla sugar"]
And data, a 3-element Array{ConstraintRef{Model,MathOptInterface.ConstraintIndex{M
athOptInterface.ScalarAffineFunction{Float64},MathOptInterface.EqualTo{Float64}},S
calarShape},1}:
 -nutrition[coffee bean] + 40 buy[raf coffee] + 30 buy[cappuccino] = 0.0
 -nutrition[milk] + 140 buy[raf coffee] + 120 buy[cappuccino] = 0.0
 -nutrition[vanilla sugar] + 5 buy[raf coffee] = 0.0
```

In [49]:
```
# Вызов функции оптимизации:
JuMP.optimize!(model)
term_status = JuMP.termination_status(model)
```

Out[49]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [50]:
```
#Для просмотра результата решения модно вывести значение переменной buy:
hcat(buy.data,JuMP.value.(buy.data))
```

Out[50]:
```
2×2 Array{GenericAffExpr{Float64,VariableRef},2}:
 buy[raf coffee]  8
 buy[cappuccino]  6
```

In [ ]:
```

```

In [ ]:
```

```