Лабораторная работа №8

Дисциплина: Компьютерный практикум по статистическому моделированию

Манаева Варвара Евгеньевна

Содержание

Список иллюстраций

Список таблиц

1 Техническое оснащение:

- Персональный компьютер с операционной системой Windows 10;
- Планшет для записи видеосопровождения и голосовых комментариев;
- Microsoft Teams, использующийся для записи скринкаста лабораторной работы;
- Приложение Pycharm для редактирования файлов формата *md*;
- *pandoc* для конвертации файлов отчётов и презентаций.

2 Цели и задачи работы

2.1 Цель

Освоить пакеты Julia для решения задач оптимизации.

2.2 Задачи [1]

1. Повторить примеры из раздела 8.2
2. Выполнить задания для самостоятельной работы из раздела 8.4

3 Выполнение лабораторной работы

3.1 Повторение примеров

Повторение примеров (3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10 , 3.11, 3.12, 3.13)

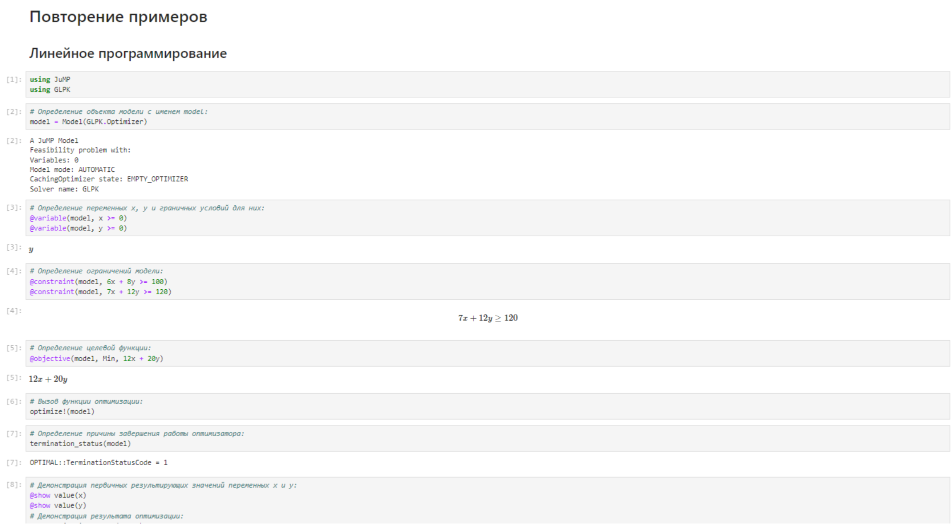


Рис. 3.1: Повторение примеров (1)

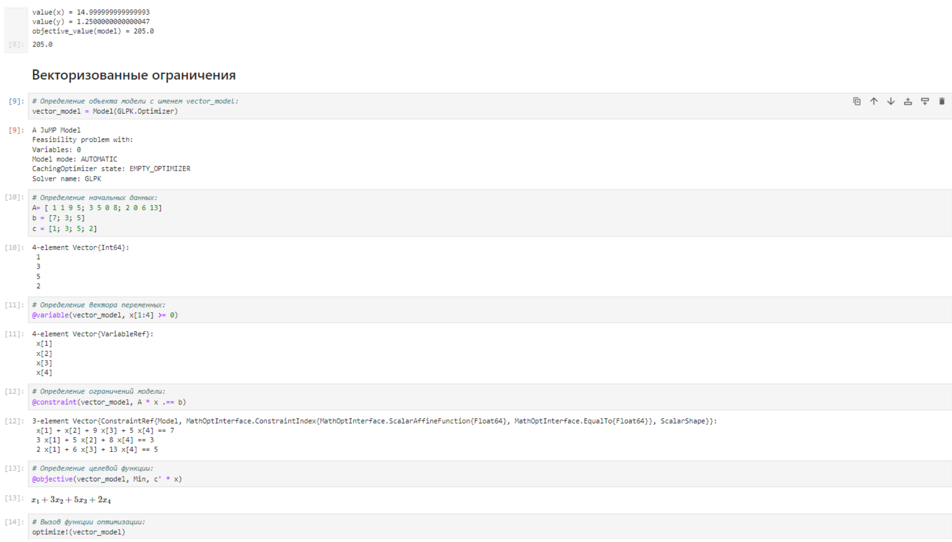


Рис. 3.2: Повторение примеров (2)

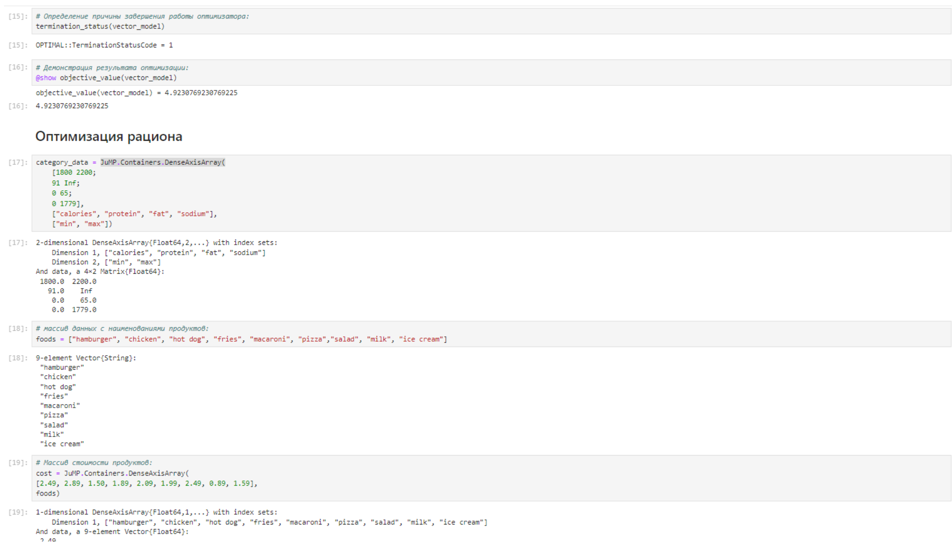


Рис. 3.3: Повторение примеров (3)

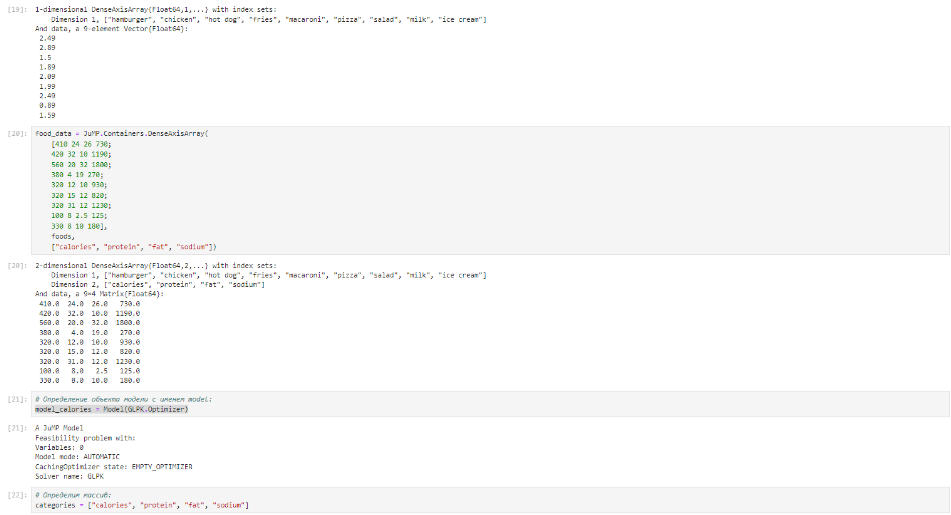


Рис. 3.4: Повторение примеров (4)

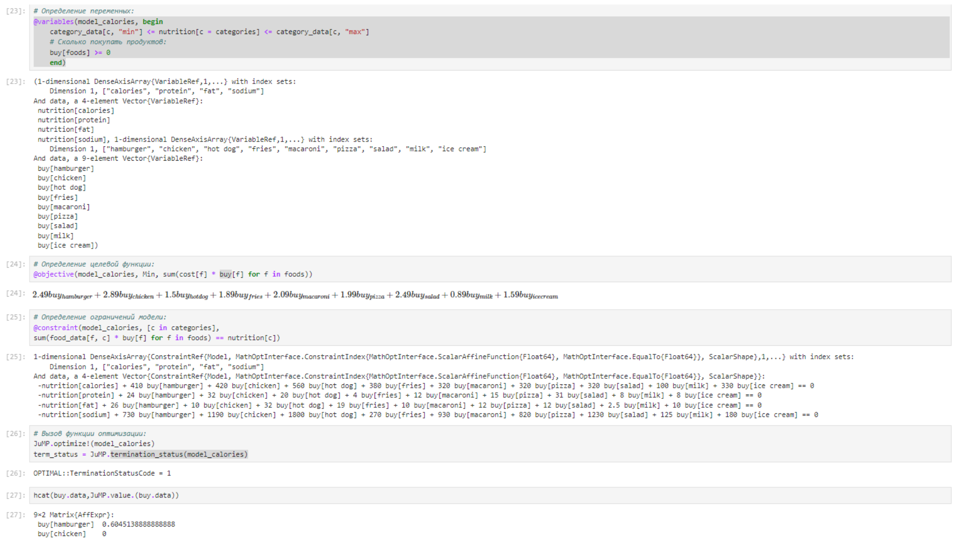


Рис. 3.5: Повторение примеров (5)

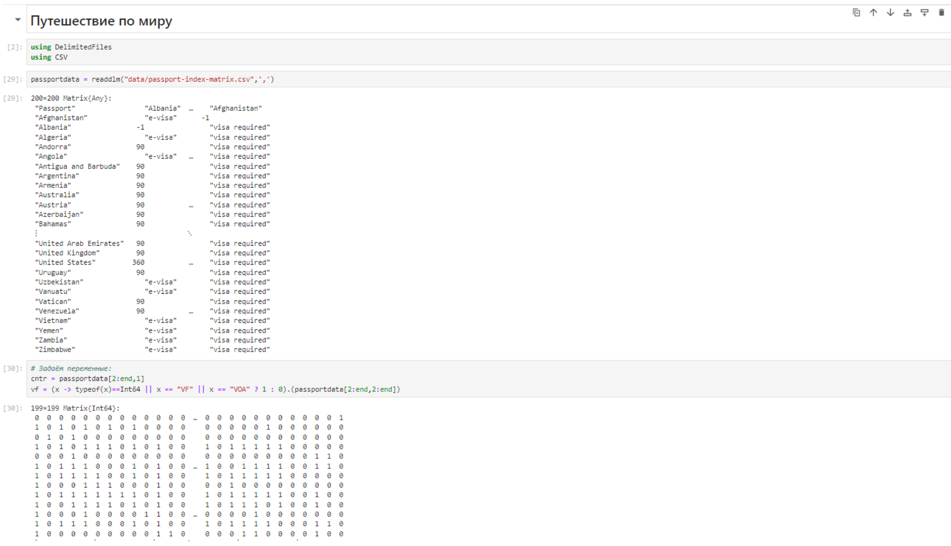


Рис. 3.6: Повторение примеров (6)

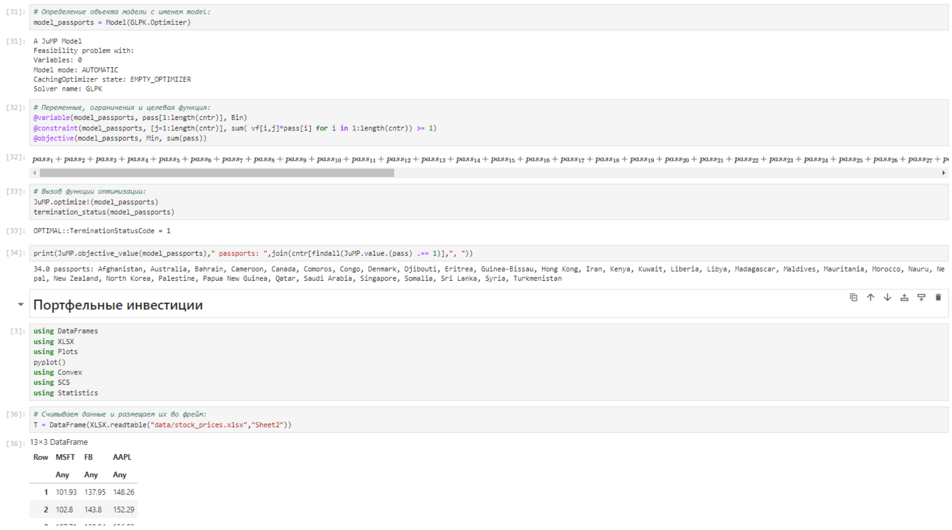


Рис. 3.7: Повторение примеров (7)

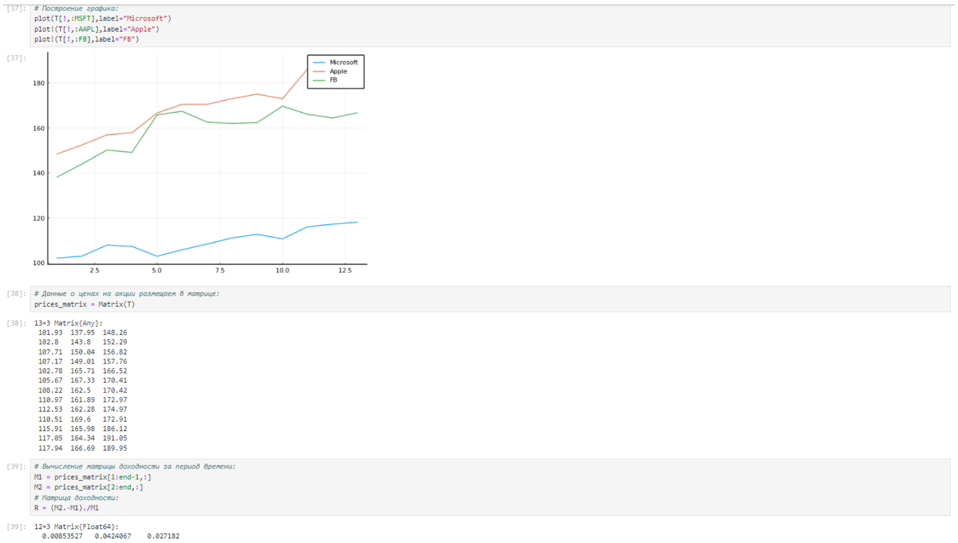


Рис. 3.8: Повторение примеров (8)

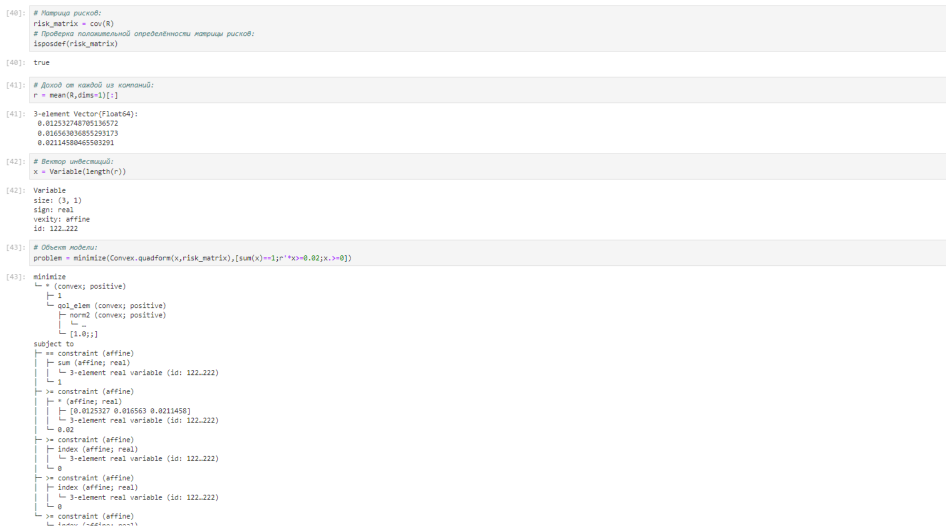


Рис. 3.9: Повторение примеров (9)

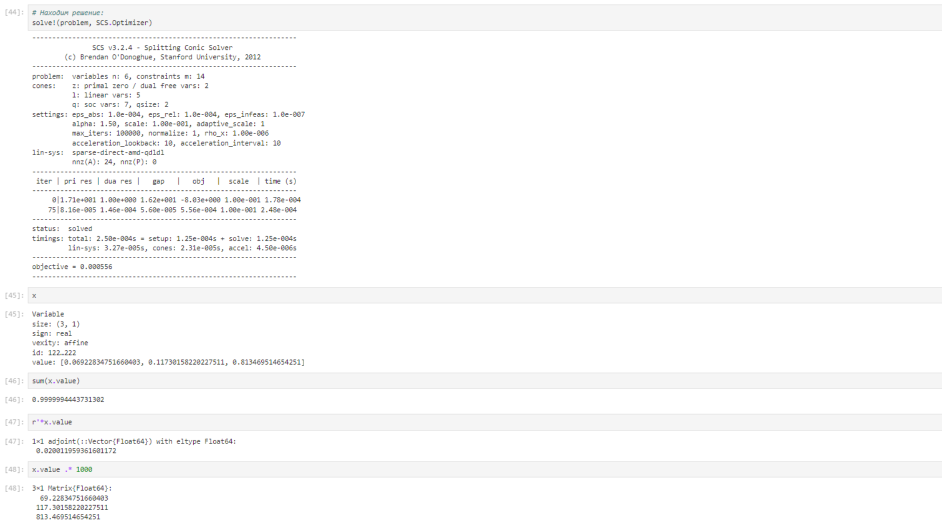


Рис. 3.10: Повторение примеров (10)

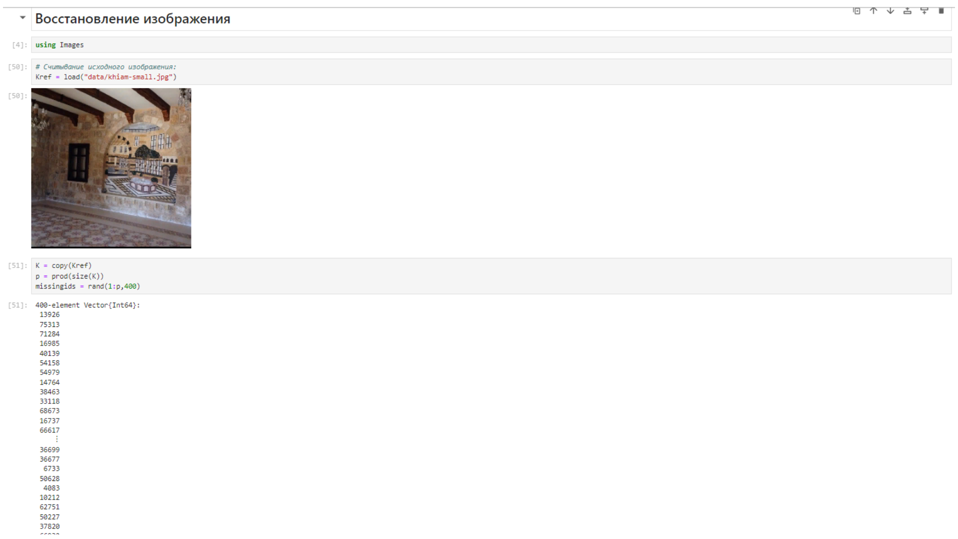


Рис. 3.11: Повторение примеров (11)

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

[52]:

```
[53]: # Матрица цветов:
      Y = Float64.(Gray.(K))
```

```
[53]: 283×283 Matrix{Float64}:
 0.101961   0.0627451  0.0784314  0.0941176  …  0.509804  0.552941  0.666667
 0.0666667  0.0980392  0.0745098  0.054902      0.505882  0.584314  0.501961
 0.0784314  0.0862745  0.0784314  0.0901961     0.6       0.701961  0.615686
 0.0862745  0.0666667  0.0745098  0.0941176     0.658824  0.705882  0.145098
 0.0784314  0.101961   0.0901961  0.0745098     0.713725  0.682353  0.231373
 0.0745098  0.0745098  0.0784314  0.0862745  …  0.729412  0.701961  0.168627
 0.12549    0.0980392  0.0862745  0.0862745     0.0       0.466667  0.192157
 0.439216   0.447059   0.305882   0.137255      0.231373  0.184314  0.137255
 0.458824   0.454902   0.45098    0.454902      0.196078  0.101961  0.117647
 0.45098    0.466667   0.458824   0.45098       0.584314  0.121569  0.137255
 0.458824   0.458824   0.458824   0.454902   …  0.521569  0.513725  0.12549
 0.466667   0.45098    0.458824   0.47451       0.576471  0.741176  0.117647
 0.45098    0.45098    0.462745   0.458824      0.560784  0.67451   0.117647
 ⋮                                           ⋱  ⋮
 0.494118   0.47451    0.47451    0.462745      0.427451  0.435294  0.443137
 0.47451    0.482353   0.470588   0.470588      0.439216  0.431373  0.431373
 0.494118   0.501961   0.470588   0.45098       0.447059  0.447059  0.45098
 0.470588   0.494118   0.490196   0.482353      0.431373  0.419608  0.419608
 0.458824   0.482353   0.47451    0.466667   …  0.454902  0.435294  0.423529
 0.443137   0.458824   0.45098    0.45098       0.309804  0.32549   0.34902
 0.47451    0.478431   0.462745   0.462745      0.341176  0.345098  0.360784
 0.482353   0.478431   0.458824   0.458824      0.423529  0.372549  0.321569
 0.552941   0.552941   0.541176   0.533333      0.447059  0.411765  0.372549
 0.552941   0.545098   0.576471   0.552941   …  0.435294  0.423529  0.407843
 0.564706   0.552941   0.54902    0.505882      0.439216  0.431373  0.419608
 0.568627   0.552941   0.517647   0.462745      0.439216  0.431373  0.427451
```

Рис. 3.12: Повторение примеров (12)

```
[54]: correctids = findall(Y[:].!=0)
      X = Convex.Variable(size(Y))
      problem = minimize(nuclearnorm(X))
      problem.constraints += X[correctids]==Y[correctids]
```

```
[54]: 1-element Vector{Constraint}:
 == constraint (affine)
├─ index (affine; real)
│  └─ 283×283 real variable (id: 141…996)
└─ 79690-element Vector{Float64}
```

```
[55]: # Находим решение:
      solve!(problem, SCS.Optimizer)
```

```
------------------------------------------------------------------
               SCS v3.2.4 - Splitting Conic Solver
        (c) Brendan O'Donoghue, Stanford University, 2012
------------------------------------------------------------------
problem:  variables n: 240268, constraints m: 400047
cones:    z: primal zero / dual free vars: 239586
          s: psd vars: 160461, ssize: 1
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
lin-sys:  sparse-direct-amd-qdldl
          nnz(A): 400330, nnz(P): 0
------------------------------------------------------------------
 iter | pri res | dua res |   gap    |   obj    |  scale   | time (s)
------------------------------------------------------------------
     0|1.50e+001 9.96e-001 8.34e+003 1.76e+002 1.00e-001 1.01e+000
   250|3.11e-004 2.58e-005 6.72e-006 4.46e+002 3.40e-001 9.48e+001
------------------------------------------------------------------
status:  solved
timings: total: 9.48e+001s = setup: 5.96e-001s + solve: 9.42e+001s
         lin-sys: 4.95e+000s, cones: 8.73e+001s, accel: 3.16e-001s
------------------------------------------------------------------
objective = 445.530879
------------------------------------------------------------------
```

```
[56]: @show norm(float.(Gray.(Kref))-X.value)
      @show norm(-X.value)
      colorview(Gray, X.value)
```

```
norm(float.(Gray.(Kref)) - X.value) = 1.2447077854577675
norm(-(X.value)) = 124.33581441728488
```

[56]:

Рис. 3.13: Повторение примеров (13)

3.2 Самостоятельная работа [2]

Самостоятельная работа (3.14, 3.15, 3.16, 3.17, 3.18, 3.19, 3.20, 3.21, 3.22)

Самостоятельная работа

Линейное программирование

```
[41]: model = Model(GLPK.Optimizer)

[41]: A JuMP Model
      Feasibility problem with:
      Variables: 0
      Model mode: AUTOMATIC
      CachingOptimizer state: EMPTY_OPTIMIZER
      Solver name: GLPK

[42]: @variable(model, x[1:3] >= 0)

[42]: 3-element Vector{VariableRef}:
       x[1]
       x[2]
       x[3]

[43]: @constraint(model, -x[1] + x[2] + 3x[3] <= -5)
      @constraint(model, x[1] + 3x[2] - 7*x[3] <= 10)
      @constraint(model, 0 <= x[1] <= 10)
```

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

```
[44]: @objective(model, Max, x[1] + 2x[2] + 5x[3])
```

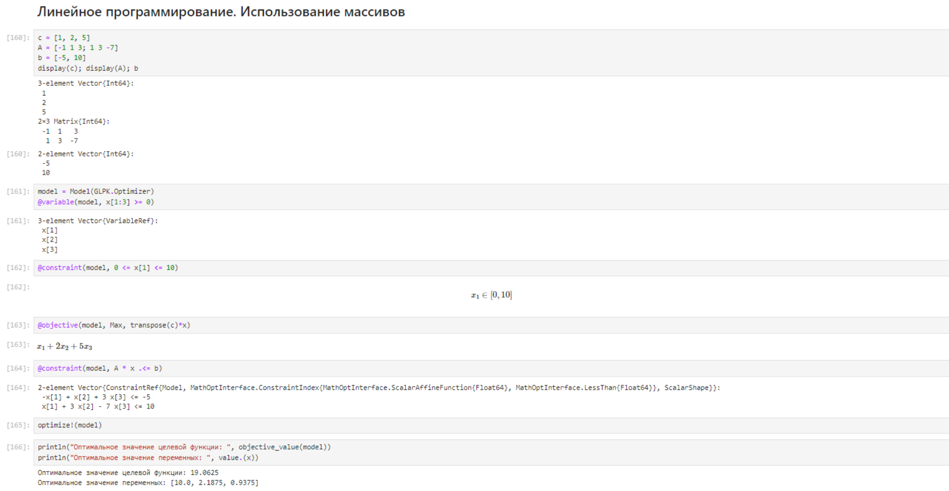
[44]: $x_1 + 2x_2 + 5x_3$

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

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

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

Рис. 3.14: Самостоятельная работа (1)



Линейное программирование. Использование массивов

```
[160]: 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
[160]: 2-element Vector{Int64}:
        -5
        10

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

[161]: 3-element Vector{VariableRef}:
        x[1]
        x[2]
        x[3]

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

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

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

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

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

[164]: 2-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.LessThan{Float64}}, ScalarShape}}:
        -x[1] + x[2] + 3 x[3] <= -5
        x[1] + 3 x[2] - 7 x[3] <= 10

[165]: optimize!(model)

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

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

Рис. 3.15: Самостоятельная работа (2)

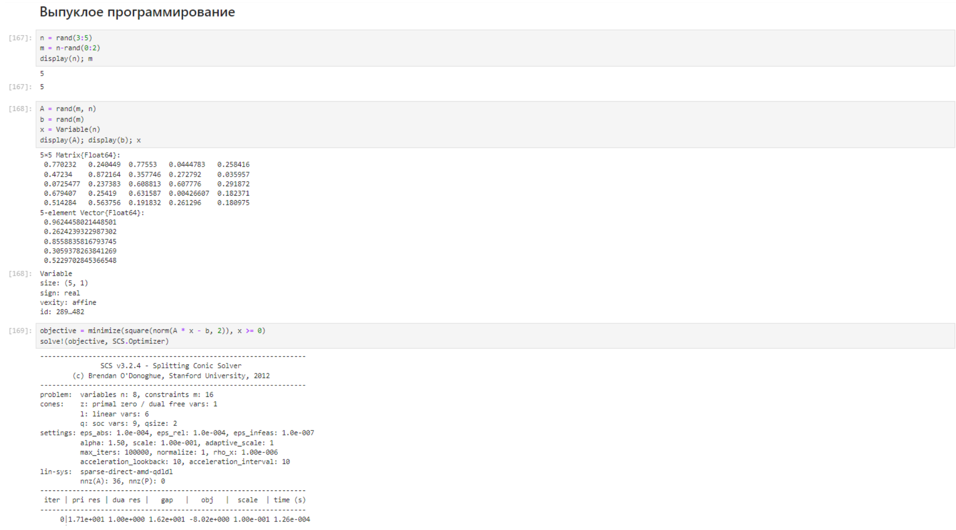


Рис. 3.16: Самостоятельная работа (3)

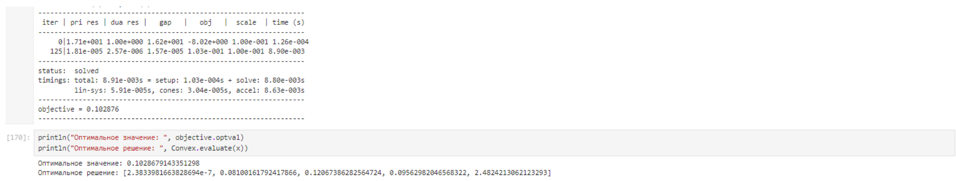


Рис. 3.17: Самостоятельная работа (4)

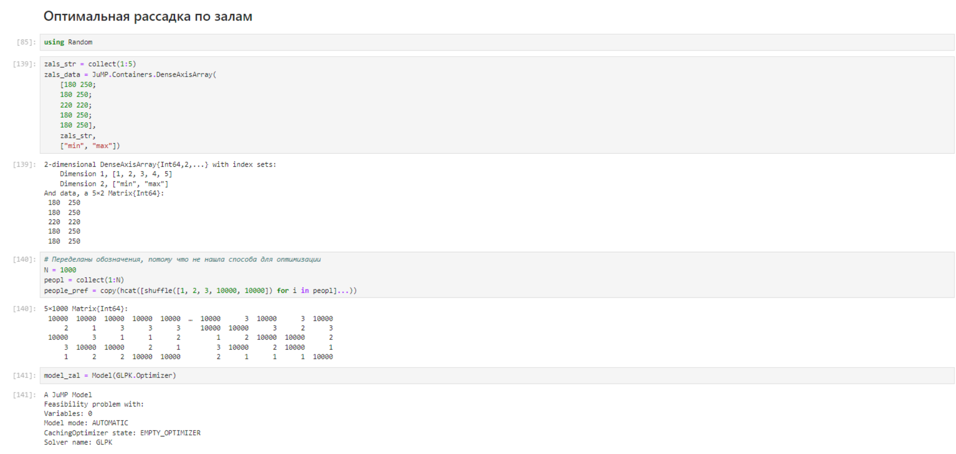


Рис. 3.18: Самостоятельная работа (5)

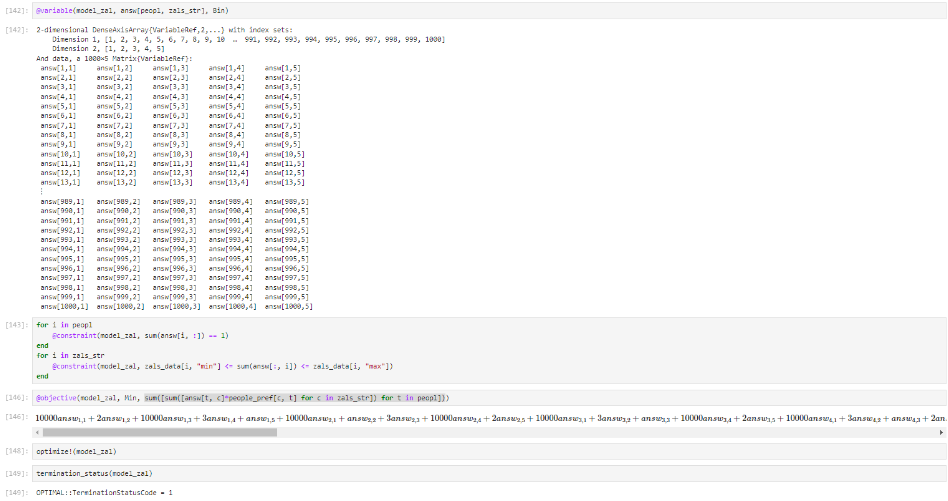


Рис. 3.19: Самостоятельная работа (6)

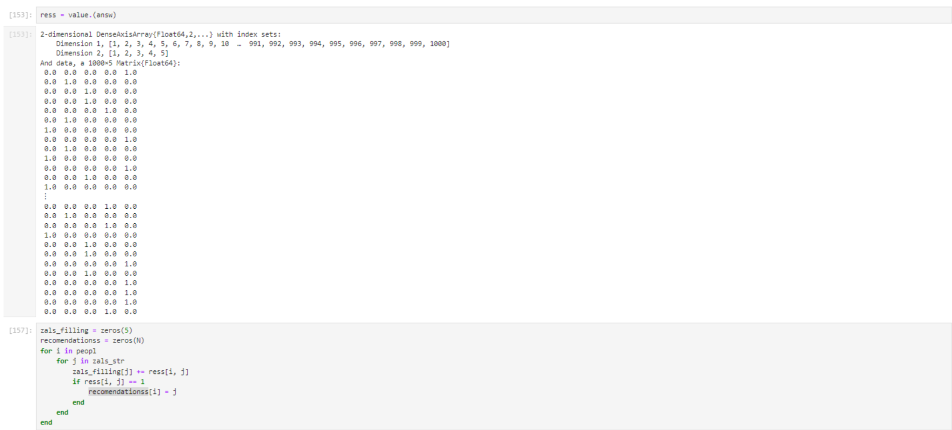


Рис. 3.20: Самостоятельная работа (7)

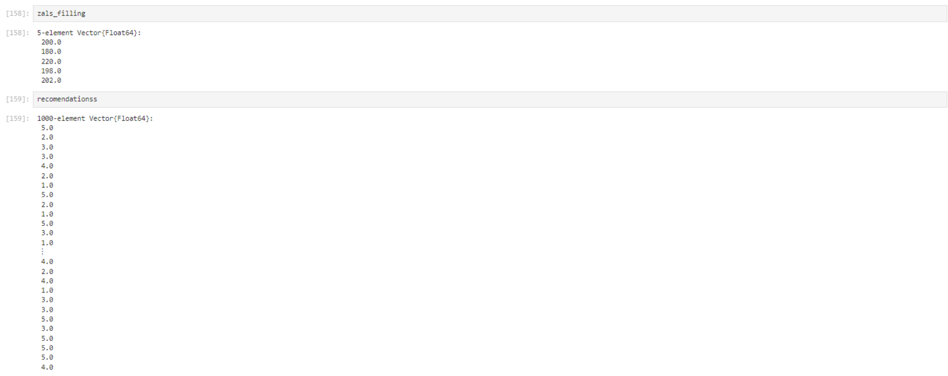


Рис. 3.21: Самостоятельная работа (8)

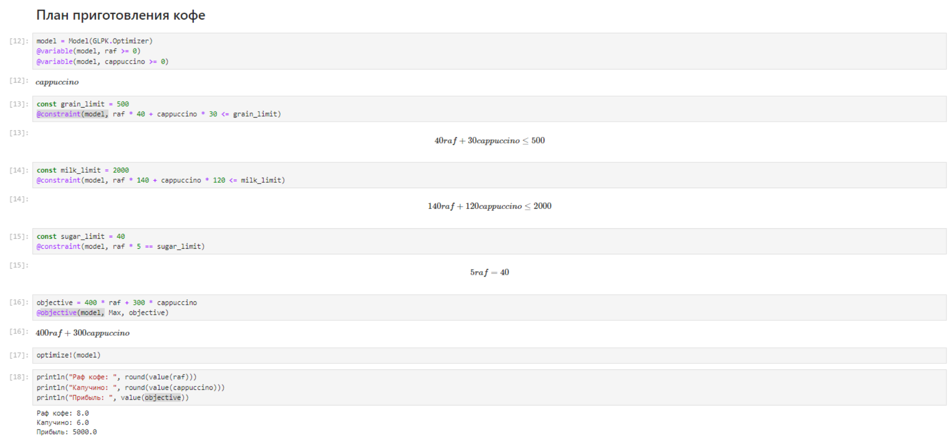


Рис. 3.22: Самостоятельная работа (9)

4 Выводы по проделанной работе

4.1 Вывод

В результате выполнения работы мы освоили пакеты Julia для решения задач оптимизации.

Были записаны скринкасты выполнения и защиты лабораторной работы.

Ссылки на скринкасты:

- Выполнение, Youtube
- Выполнение, Rutube
- Защита презентации, Youtube
- Защита презентации, Rutube

Список литературы

1. Лабораторная работа № 8 [Электронный ресурс]. Российский Университет Дружбы Народов имени Патрису Лумумбы, 2023. URL: https://esystem.rudn.ru/mod/resource/view.php?id=1069857.
2. Julia official documentation [Электронный ресурс]. 2023. URL: https://docs.julialang.org/en/v1/.

Повторение примеров

Линейное программирование

In [1]:
```
using JuMP
using GLPK
```

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

Out[2]:
```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
```

In [3]:
```
# Определение переменных x, y и граничных условий для них:
@variable(model, x >= 0)
@variable(model, y >= 0)
```

Out[3]: y

In [4]:
```
# Определение ограничений модели:
@constraint(model, 6x + 8y >= 100)
@constraint(model, 7x + 12y >= 120)
```

Out[4]:

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

In [5]:
```
# Определение целевой функции:
@objective(model, Min, 12x + 20y)
```

Out[5]: $12x + 20y$

In [6]:
```
# Вызов функции оптимизации:
optimize!(model)
```

In [7]:
```
# Определение причины завершения работы оптимизатора:
termination_status(model)
```

Out[7]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [8]:
```
# Демонстрация первичных результирующих значений переменных x и y:
@show value(x)
@show value(y)
# Демонстрация результата оптимизации:
@show objective_value(model)
```

```
value(x) = 14.999999999999993
value(y) = 1.2500000000000047
objective_value(model) = 205.0
```

Out[8]:
```
205.0
```

Векторизованные ограничения

In [9]:
```
# Определение объекта модели с именем vector_model:
vector_model = Model(GLPK.Optimizer)
```

Out[9]:
```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
```

In [10]:
```
# Определение начальных данных:
A= [ 1 1 9 5; 3 5 0 8; 2 0 6 13]
b = [7; 3; 5]
c = [1; 3; 5; 2]
```

Out[10]:
```
4-element Vector{Int64}:
 1
 3
 5
 2
```

In [11]:
```
# Определение вектора переменных:
@variable(vector_model, x[1:4] >= 0)
```

Out[11]:
```
4-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
 x[4]
```

In [12]:
```
# Определение ограничений модели:
@constraint(vector_model, A * x .== b)
```

Out[12]:
```
3-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInte
rface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float64}}, ScalarSha
pe}}:
 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
```

In [13]:
```
# Определение целевой функции:
@objective(vector_model, Min, c' * x)
```

Out[13]: $x_1 + 3x_2 + 5x_3 + 2x_4$

In [14]:
```
# Вызов функции оптимизации:
```

```
optimize!(vector_model)
```

In [15]:
```
# Определение причины завершения работы оптимизатора:
termination_status(vector_model)
```

Out[15]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [16]:
```
# Демонстрация результата оптимизации:
@show objective_value(vector_model)
```

```
objective_value(vector_model) = 4.9230769230769225
```

Out[16]:
```
4.9230769230769225
```

Оптимизация рациона

In [17]:
```
category_data = JuMP.Containers.DenseAxisArray(
    [1800 2200;
    91 Inf;
    0 65;
    0 1779],
    ["calories", "protein", "fat", "sodium"],
    ["min", "max"])
```

Out[17]:
```
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, ["calories", "protein", "fat", "sodium"]
    Dimension 2, ["min", "max"]
And data, a 4×2 Matrix{Float64}:
 1800.0  2200.0
   91.0     Inf
    0.0    65.0
    0.0  1779.0
```

In [18]:
```
# массив данных с наименованиями продуктов:
foods = ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza","salad", "
```

Out[18]:
```
9-element Vector{String}:
 "hamburger"
 "chicken"
 "hot dog"
 "fries"
 "macaroni"
 "pizza"
 "salad"
 "milk"
 "ice cream"
```

In [19]:
```
# Массив стоимости продуктов:
cost = JuMP.Containers.DenseAxisArray(
[2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59],
foods)
```

Out[19]:
```
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]
And data, a 9-element Vector{Float64}:
 2.49
 2.89
 1.5
 1.89
 2.09
 1.99
 2.49
 0.89
 1.59
```

In [20]:
```
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"])
```

Out[20]:
```
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"]
    Dimension 2, ["calories", "protein", "fat", "sodium"]
And data, a 9×4 Matrix{Float64}:
 410.0  24.0  26.0   730.0
 420.0  32.0  10.0  1190.0
 560.0  20.0  32.0  1800.0
 380.0   4.0  19.0   270.0
 320.0  12.0  10.0   930.0
 320.0  15.0  12.0   820.0
 320.0  31.0  12.0  1230.0
 100.0   8.0   2.5   125.0
 330.0   8.0  10.0   180.0
```

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

Out[21]:
```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
```

In [22]:
```
# Определим массив:
categories = ["calories", "protein", "fat", "sodium"]
```

Out[22]: 4-element Vector{String}:

```
 "calories"
 "protein"
 "fat"
 "sodium"
```

In [23]:

```
# Определение переменных:
@variables(model_calories, begin
    category_data[c, "min"] <= nutrition[c = categories] <= category_data[c, "max"]
    # Сколько покупать продуктов:
    buy[foods] >= 0
    end)
```

Out[23]:

```
(1-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
    Dimension 1, ["calories", "protein", "fat", "sodium"]
And data, a 4-element Vector{VariableRef}:
 nutrition[calories]
 nutrition[protein]
 nutrition[fat]
 nutrition[sodium], 1-dimensional DenseAxisArray{VariableRef,1,...} with index set
s:
    Dimension 1, ["hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza",
"salad", "milk", "ice cream"]
And data, a 9-element Vector{VariableRef}:
 buy[hamburger]
 buy[chicken]
 buy[hot dog]
 buy[fries]
 buy[macaroni]
 buy[pizza]
 buy[salad]
 buy[milk]
 buy[ice cream])
```

In [24]:

```
# Определение целевой функции:
@objective(model_calories, Min, sum(cost[f] * buy[f] for f in foods))
```

Out[24]: $2.49 buy_{hamburger} + 2.89 buy_{chicken} + 1.5 buy_{hotdog} + 1.89 buy_{fries} + 2.09 buy_{macaroni} + 1.99 buy_{pizza} + 2.4$

In [25]:

```
# Определение ограничений модели:
@constraint(model_calories, [c in categories],
sum(food_data[f, c] * buy[f] for f in foods) == nutrition[c])
```

Out[25]:
```
1-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex
{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float6
4}}, ScalarShape},1,...} with index sets:
    Dimension 1, ["calories", "protein", "fat", "sodium"]
And data, a 4-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex
{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float6
4}}, ScalarShape}}:
 -nutrition[calories] + 410 buy[hamburger] + 420 buy[chicken] + 560 buy[hot dog] +
380 buy[fries] + 320 buy[macaroni] + 320 buy[pizza] + 320 buy[salad] + 100 buy[mil
k] + 330 buy[ice cream] == 0
 -nutrition[protein] + 24 buy[hamburger] + 32 buy[chicken] + 20 buy[hot dog] + 4 b
uy[fries] + 12 buy[macaroni] + 15 buy[pizza] + 31 buy[salad] + 8 buy[milk] + 8 buy
[ice cream] == 0
 -nutrition[fat] + 26 buy[hamburger] + 10 buy[chicken] + 32 buy[hot dog] + 19 buy
[fries] + 10 buy[macaroni] + 12 buy[pizza] + 12 buy[salad] + 2.5 buy[milk] + 10 bu
y[ice cream] == 0
 -nutrition[sodium] + 730 buy[hamburger] + 1190 buy[chicken] + 1800 buy[hot dog] +
270 buy[fries] + 930 buy[macaroni] + 820 buy[pizza] + 1230 buy[salad] + 125 buy[mi
lk] + 180 buy[ice cream] == 0
```

In [26]:
```
# Вызов функции оптимизации:
JuMP.optimize!(model_calories)
term_status = JuMP.termination_status(model_calories)
```

Out[26]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [27]:
```
hcat(buy.data,JuMP.value.(buy.data))
```

Out[27]:
```
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]       6.970138888888935
 buy[ice cream]  2.591319444444441
```

Путешествие по миру

In [2]:
```
using DelimitedFiles
using CSV
```

In [29]:
```
passportdata = readdlm("data/passport-index-matrix.csv",',')
```

Out[29]:
```
200×200 Matrix{Any}:
 "Passport"                   "Albania"  …    "Afghanistan"
 "Afghanistan"                "e-visa"      -1
 "Albania"                  -1                "visa required"
 "Algeria"                    "e-visa"        "visa required"
 "Andorra"                  90                "visa required"
 "Angola"                     "e-visa"   …    "visa required"
 "Antigua and Barbuda"      90                "visa required"
 "Argentina"                90                "visa required"
 "Armenia"                  90                "visa required"
 "Australia"                90                "visa required"
 "Austria"                  90           …    "visa required"
 "Azerbaijan"               90                "visa required"
 "Bahamas"                  90                "visa required"
 ⋮                                       ⋱
 "United Arab Emirates"     90                "visa required"
 "United Kingdom"           90                "visa required"
 "United States"           360           …    "visa required"
 "Uruguay"                  90                "visa required"
 "Uzbekistan"                 "e-visa"        "visa required"
 "Vanuatu"                    "e-visa"        "visa required"
 "Vatican"                  90                "visa required"
 "Venezuela"                90           …    "visa required"
 "Vietnam"                    "e-visa"        "visa required"
 "Yemen"                      "e-visa"        "visa required"
 "Zambia"                     "e-visa"        "visa required"
 "Zimbabwe"                   "e-visa"        "visa required"
```

In [30]:
```julia
# Задаём переменные:
cntr = passportdata[2:end,1]
vf = (x -> typeof(x)==Int64 || x == "VF" || x == "VOA" ? 1 : 0).(passportdata[2:end
```

Out[30]:
```
199×199 Matrix{Int64}:
 0  0  0  0  0  0  0  0  0  0  0  0  0  …  0  0  0  0  0  0  0  0  0  0  0  1
 1  0  1  0  1  0  1  0  1  0  0  0  0     0  0  0  0  0  1  0  0  0  0  0  0
 0  1  0  1  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 1  0  1  0  1  1  1  0  1  0  1  0  0     1  0  1  1  1  1  1  0  0  0  0  0
 0  0  0  1  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  1  1  0
 1  0  1  1  1  0  0  0  1  0  1  0  0  …  1  0  0  1  1  1  1  0  0  1  1  0
 1  0  1  1  1  1  0  0  1  0  1  0  0     1  0  1  1  1  1  1  0  0  0  0  0
 1  0  0  0  1  1  1  0  0  0  1  0  0     0  0  1  0  0  0  0  0  0  0  0  0
 1  0  1  1  1  1  1  1  1  0  1  0  0     1  0  1  1  1  1  1  0  0  1  0  0
 1  0  0  1  1  1  1  0  1  0  1  0  0     1  0  1  1  1  0  1  0  0  1  0  0
 1  0  0  0  1  0  0  0  0  1  1  0  0  …  0  0  0  0  1  0  0  0  0  0  0  0
 1  0  1  1  1  0  0  0  1  0  1  0  0     1  0  1  1  1  1  0  0  0  1  1  0
 1  0  0  0  0  0  0  0  0  0  1  1  0     0  0  0  1  1  0  0  0  0  1  0  0
 ⋮              ⋮              ⋮        ⋱        ⋮              ⋮
 1  0  1  1  1  1  1  0  1  1  1  0  0     0  0  1  1  1  1  0  0  0  1  0  0
 1  0  1  1  1  1  1  0  1  0  1  0  0     1  0  1  1  1  1  1  1  0  1  0  0
 1  0  1  1  1  1  1  0  1  0  1  0  0     1  1  1  0  1  1  0  0  0  1  0  0
 1  0  1  1  0  1  1  0  1  0  1  0  0  …  1  0  1  0  1  1  1  0  0  0  0  0
 0  0  0  0  1  0  1  0  0  1  0  0  0     0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0  1  0  0     0  0  0  0  1  0  0  0  0  1  1  0
 1  0  1  1  1  1  1  0  1  0  1  0  0     1  0  1  1  1  1  0  0  0  0  0  0
 1  0  1  0  1  1  0  0  1  0  1  0  0     0  0  1  0  0  1  1  0  0  0  0  0
 0  0  0  0  0  0  0  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  0  0  0  0     0  0  0  0  0  0  0  0  1  0  0  0
 0  0  0  1  1  0  0  0  0  0  1  0  1     0  0  0  0  1  0  0  0  0  1  1  0
 0  0  0  1  0  0  0  0  0  0  1  0  0     0  0  0  0  1  0  0  0  0  1  1  0
```

In [31]:
```julia
# Определение объекта модели с именем model:
model_passports = Model(GLPK.Optimizer)
```

Out[31]:
```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
```

In [32]:
```julia
# Переменные, ограничения и целевая функция:
@variable(model_passports, pass[1:length(cntr)], Bin)
@constraint(model_passports, [j=1:length(cntr)], sum( vf[i,j]*pass[i] for i in 1:le
@objective(model_passports, Min, sum(pass))
```

Out[32]:

$$pass_1 + pass_2 + pass_3 + pass_4 + pass_5 + pass_6 + pass_7 + pass_8 + pass_9 + pass_{10} + pass_{11} + pas$$
$$+ pass_{22} + pass_{23} + pass_{24} + pass_{25} + pass_{26} + pass_{27} + pass_{28} + pass_{29} + pass_{30} + [[\ldots 139 \text{ term}$$
$$+ pass_{177} + pass_{178} + pass_{179} + pass_{180} + pass_{181} + pass_{182} + pass_{183} + pass_{184} + pass_{185} + pass_1$$
$$+ pass_{195} + pass_{196} + pass_{197} + pass_{198} + pass_{199}$$

In [33]:
```julia
# Вызов функции оптимизации:
JuMP.optimize!(model_passports)
termination_status(model_passports)
```

Out[33]:
```
OPTIMAL::TerminationStatusCode = 1
```

In [34]:
```julia
print(JuMP.objective_value(model_passports)," passports: ",join(cntr[findall(JuMP.v
```

```
34.0 passports: Afghanistan, Australia, Bahrain, Cameroon, Canada, Comoros, Congo, D
enmark, Djibouti, Eritrea, Guinea-Bissau, Hong Kong, Iran, Kenya, Kuwait, Liberia, L
ibya, Madagascar, Maldives, Mauritania, Morocco, Nauru, Nepal, New Zealand, North Ko
rea, Palestine, Papua New Guinea, Qatar, Saudi Arabia, Singapore, Somalia, Sri Lank
a, Syria, Turkmenistan
```

Портфельные инвестиции

In [3]:
```
using DataFrames
using XLSX
using Plots
pyplot()
using Convex
using SCS
using Statistics
```

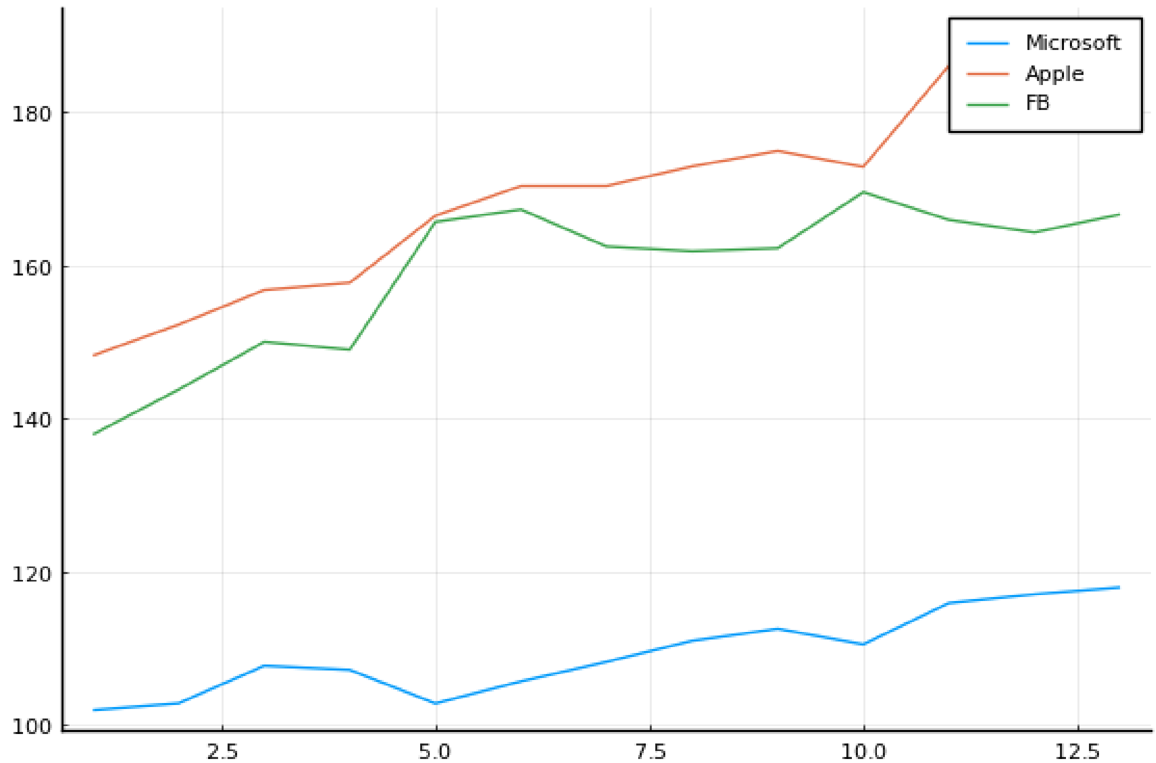
In [36]:
```
# Считываем данные и размещаем их во фрейм:
T = DataFrame(XLSX.readtable("data/stock_prices.xlsx","Sheet2"))
```

Out[36]: 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
4	107.17	149.01	157.76
5	102.78	165.71	166.52
6	105.67	167.33	170.41
7	108.22	162.5	170.42
8	110.97	161.89	172.97
9	112.53	162.28	174.97
10	110.51	169.6	172.91
11	115.91	165.98	186.12
12	117.05	164.34	191.05
13	117.94	166.69	189.95

In [37]:
```
# Построение графика:
plot(T[!,:MSFT],label="Microsoft")
plot!(T[!,:AAPL],label="Apple")
plot!(T[!,:FB],label="FB")
```

Out[37]:



In [38]:

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

Out[38]:

```
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
 112.53  162.28  174.97
 110.51  169.6   172.91
 115.91  165.98  186.12
 117.05  164.34  191.05
 117.94  166.69  189.95
```

In [39]:

```
# Вычисление матрицы доходности за период времени:
M1 = prices_matrix[1:end-1,:]
M2 = prices_matrix[2:end,:]
# Матрица доходности:
R = (M2.-M1)./M1
```

Out[39]:
```
12×3 Matrix{Float64}:
  0.00853527   0.0424067    0.027182
  0.0477626    0.0433936    0.0297459
 -0.00501346  -0.00686484   0.00599413
 -0.040963     0.112073     0.0555274
  0.0281183    0.00977611   0.0233606
  0.0241317   -0.0288651    5.8682e-5
  0.0254112   -0.00375385   0.014963
  0.0140579    0.00240904   0.0115627
 -0.0179508    0.0451072   -0.0117734
  0.0488644   -0.0213443    0.0763981
  0.00983522  -0.00988071   0.0264883
  0.00760359   0.0142996   -0.00575766
```

In [40]:
```
# Матрица рисков:
risk_matrix = cov(R)
# Проверка положительной определённости матрицы рисков:
isposdef(risk_matrix)
```

Out[40]:
```
true
```

In [41]:
```
# Доход от каждой из компаний:
r = mean(R,dims=1)[:]
```

Out[41]:
```
3-element Vector{Float64}:
 0.012532748705136572
 0.016563036855293173
 0.02114580465503291
```

In [42]:
```
# Вектор инвестиций:
x = Variable(length(r))
```

Out[42]:
```
Variable
size: (3, 1)
sign: real
vexity: affine
id: 122…222
```

In [43]:
```
# Объект модели:
problem = minimize(Convex.quadform(x,risk_matrix),[sum(x)==1;r'*x>=0.02;x.>=0])
```

Out[43]:
```
minimize
└─ * (convex; positive)
   ├─ 1
   └─ qol_elem (convex; positive)
      ├─ norm2 (convex; positive)
      │  └─ …
      └─ [1.0;;]
subject to
├─ == constraint (affine)
│  ├─ sum (affine; real)
│  │  └─ 3-element real variable (id: 122…222)
│  └─ 1
├─ >= constraint (affine)
│  ├─ * (affine; real)
│  │  ├─ [0.0125327 0.016563 0.0211458]
│  │  └─ 3-element real variable (id: 122…222)
│  └─ 0.02
├─ >= constraint (affine)
│  ├─ index (affine; real)
│  │  └─ 3-element real variable (id: 122…222)
│  └─ 0
├─ >= constraint (affine)
│  ├─ index (affine; real)
│  │  └─ 3-element real variable (id: 122…222)
│  └─ 0
└─ >= constraint (affine)
   ├─ index (affine; real)
   │  └─ 3-element real variable (id: 122…222)
   └─ 0

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

In [44]:
```julia
# Находим решение:
solve!(problem, SCS.Optimizer)
```

```
----------------------------------------------------------------
              SCS v3.2.4 - Splitting Conic Solver
        (c) Brendan O'Donoghue, Stanford University, 2012
----------------------------------------------------------------
problem:  variables n: 6, constraints m: 14
cones:    z: primal zero / dual free vars: 2
          l: linear vars: 5
          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
lin-sys:  sparse-direct-amd-qdldl
          nnz(A): 24, nnz(P): 0
----------------------------------------------------------------
 iter | pri res | dua res |   gap   |   obj   |  scale  | time (s)
----------------------------------------------------------------
     0|1.71e+001 1.00e+000 1.62e+001 -8.03e+000 1.00e-001 1.78e-004
    75|8.16e-005 1.46e-004 5.60e-005 5.56e-004 1.00e-001 2.48e-004
----------------------------------------------------------------
status:  solved
timings: total: 2.50e-004s = setup: 1.25e-004s + solve: 1.25e-004s
         lin-sys: 3.27e-005s, cones: 2.31e-005s, accel: 4.50e-006s
----------------------------------------------------------------
objective = 0.000556
----------------------------------------------------------------
```

In [45]:
```
x
```

Out[45]:
```
Variable
size: (3, 1)
sign: real
vexity: affine
id: 122…222
value: [0.06922834751660403, 0.11730158220227511, 0.813469514654251]
```

In [46]:
```
sum(x.value)
```

Out[46]:
```
0.9999994443731302
```

In [47]:
```
r'*x.value
```

Out[47]:
```
1×1 adjoint(::Vector{Float64}) with eltype Float64:
 0.020011959361601172
```

In [48]:
```
x.value .* 1000
```

Out[48]:
```
3×1 Matrix{Float64}:
  69.22834751660403
 117.30158220227511
 813.469514654251
```

Восстановление изображения

In [4]:
```julia
using Images
```

In [50]:
```julia
# Считывание исходного изображения:
Kref = load("data/khiam-small.jpg")
```

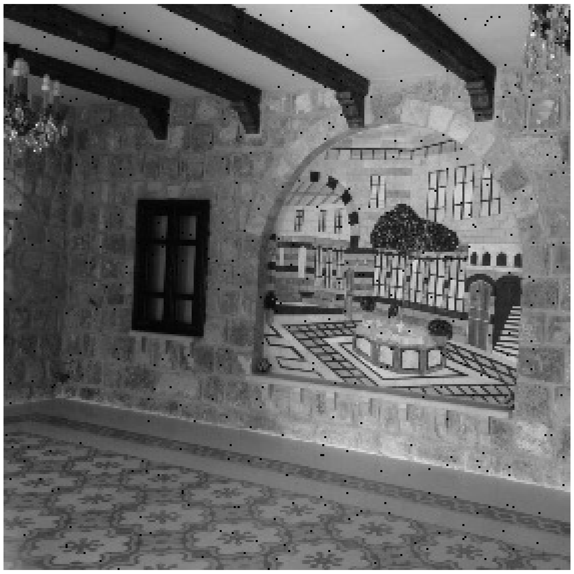
Out[50]:

In [51]:
```julia
K = copy(Kref)
p = prod(size(K))
missingids = rand(1:p,400)
```

Out[51]:
```
400-element Vector{Int64}:
 13926
 75313
 71284
 16985
 40139
 54158
 54979
 14764
 38463
 33118
 68673
 16737
 66617
     ⋮
 36699
 36677
  6733
 50628
  4083
 10212
 62751
 50227
 37820
 66932
 64624
 61869
```

In [52]:
```
K[missingids] .= RGBX{N0f8}(0.0,0.0,0.0)
K
Gray.(K)
```

Out[52]:



In [53]:
```
# Матрица цветов:
Y = Float64.(Gray.(K))
```

Out[53]:
```
283×283 Matrix{Float64}:
 0.101961   0.0627451  0.0784314  0.0941176  …  0.509804  0.552941  0.666667
 0.0666667  0.0980392  0.0745098  0.054902      0.505882  0.584314  0.501961
 0.0784314  0.0862745  0.0784314  0.0901961     0.6       0.701961  0.615686
 0.0862745  0.0666667  0.0745098  0.0941176     0.658824  0.705882  0.145098
 0.0784314  0.101961   0.0901961  0.0745098     0.713725  0.682353  0.231373
 0.0745098  0.0745098  0.0784314  0.0862745  …  0.729412  0.701961  0.168627
 0.12549    0.0980392  0.0862745  0.0862745     0.0       0.466667  0.192157
 0.439216   0.447059   0.305882   0.137255      0.231373  0.184314  0.137255
 0.458824   0.454902   0.45098    0.454902      0.196078  0.101961  0.117647
 0.45098    0.466667   0.458824   0.45098       0.584314  0.121569  0.137255
 0.458824   0.458824   0.458824   0.454902   …  0.521569  0.513725  0.12549
 0.466667   0.45098    0.458824   0.47451       0.576471  0.741176  0.117647
 0.45098    0.45098    0.462745   0.458824      0.560784  0.67451   0.117647
 ⋮                                           ⋱  ⋮
 0.494118   0.47451    0.47451    0.462745      0.427451  0.435294  0.443137
 0.47451    0.482353   0.470588   0.470588      0.439216  0.431373  0.431373
 0.494118   0.501961   0.470588   0.45098       0.447059  0.447059  0.45098
 0.470588   0.494118   0.490196   0.482353      0.431373  0.419608  0.419608
 0.458824   0.482353   0.47451    0.466667   …  0.454902  0.435294  0.423529
 0.443137   0.458824   0.45098    0.45098       0.309804  0.32549   0.34902
 0.47451    0.478431   0.462745   0.462745      0.341176  0.345098  0.360784
 0.482353   0.478431   0.458824   0.458824      0.423529  0.372549  0.321569
 0.552941   0.552941   0.541176   0.533333      0.447059  0.411765  0.372549
 0.552941   0.545098   0.576471   0.552941   …  0.435294  0.423529  0.407843
 0.564706   0.552941   0.54902    0.505882      0.439216  0.431373  0.419608
 0.568627   0.552941   0.517647   0.462745      0.439216  0.431373  0.427451
```

In [54]:
```
correctids = findall(Y[:].!=0)
X = Convex.Variable(size(Y))
```

```julia
problem = minimize(nuclearnorm(X))
problem.constraints += X[correctids]==Y[correctids]
```

Out[54]:

```
1-element Vector{Constraint}:
 == constraint (affine)
├─ index (affine; real)
│  └─ 283×283 real variable (id: 141…996)
└─ 79690-element Vector{Float64}
```

In [55]:

```julia
# Находим решение:
solve!(problem, SCS.Optimizer)
```

```
------------------------------------------------------------------
	       SCS v3.2.4 - Splitting Conic Solver
	(c) Brendan O'Donoghue, Stanford University, 2012
------------------------------------------------------------------
problem:  variables n: 240268, constraints m: 400047
cones: 	  z: primal zero / dual free vars: 239586
	  s: psd vars: 160461, ssize: 1
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
lin-sys:  sparse-direct-amd-qdldl
	  nnz(A): 400330, nnz(P): 0
------------------------------------------------------------------
 iter | pri res | dua res |   gap   |   obj   |  scale  | time (s)
------------------------------------------------------------------
     0|1.50e+001 9.96e-001 8.34e+003 1.76e+002 1.00e-001 1.01e+000
   250|3.11e-004 2.58e-005 6.72e-006 4.46e+002 3.40e-001 9.48e+001
------------------------------------------------------------------
status:  solved
timings: total: 9.48e+001s = setup: 5.96e-001s + solve: 9.42e+001s
	 lin-sys: 4.95e+000s, cones: 8.73e+001s, accel: 3.16e-001s
------------------------------------------------------------------
objective = 445.530879
------------------------------------------------------------------
```

In [56]:

```julia
@show norm(float.(Gray.(Kref))-X.value)
@show norm(-X.value)
colorview(Gray, X.value)
```

```
norm(float.(Gray.(Kref)) - X.value) = 1.2447077854577675
norm(-(X.value)) = 124.33581441728488
```

Out[56]:

Самостоятельная работа

Линейное программирование

In [41]:
```
model = Model(GLPK.Optimizer)
```

Out[41]:
```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
```

In [42]:
```
@variable(model, x[1:3] >= 0)
```

Out[42]:
```
3-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
```

In [43]:
```
@constraint(model, -x[1] + x[2] + 3x[3] <= -5)
@constraint(model, x[1] + 3x[2] - 7*x[3] <= 10)
@constraint(model, 0 <= x[1] <= 10)
```

Out[43]:

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

In [44]:
```
@objective(model, Max, x[1] + 2x[2] + 5x[3])
```

Out[44]: $x_1 + 2x_2 + 5x_3$

In [45]:
```
optimize!(model)
```

In [46]:
```
println("Оптимальное значение целевой функции: ", objective_value(model))
println("Оптимальное значение переменных: ", value.(x))
```

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

Линейное программирование. Использование массивов

In [160…
```
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
```

Out[160…
```
2-element Vector{Int64}:
 -5
 10
```

In [161…
```
model = Model(GLPK.Optimizer)
@variable(model, x[1:3] >= 0)
```

Out[161…
```
3-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
```

In [162…
```
@constraint(model, 0 <= x[1] <= 10)
```

Out[162…

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

In [163…
```
@objective(model, Max, transpose(c)*x)
```

Out[163…

$x_1 + 2x_2 + 5x_3$

In [164…
```
@constraint(model, A * x .<= b)
```

Out[164…
```
2-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.LessThan{Float64}}, ScalarShape}}:
 -x[1] + x[2] + 3 x[3] <= -5
 x[1] + 3 x[2] - 7 x[3] <= 10
```

In [165…
```
optimize!(model)
```

In [166…

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

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

Выпуклое программирование

In [167…

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

```
5
```

Out[167…

```
5
```

In [168…

```
A = rand(m, n)
b = rand(m)
x = Variable(n)
display(A); display(b); x
```

```
5×5 Matrix{Float64}:
 0.770232   0.240449  0.77553   0.0444783   0.258416
 0.47234    0.872164  0.357746  0.272792    0.035957
 0.0725477  0.237383  0.608813  0.607776    0.291872
 0.679407   0.25419   0.631587  0.00426607  0.182371
 0.514284   0.563756  0.191832  0.261296    0.180975
5-element Vector{Float64}:
 0.9624458021448501
 0.2624239322987302
 0.8558835816793745
 0.3059378263841269
 0.5229702845366548
```

Out[168…

```
Variable
size: (5, 1)
sign: real
vexity: affine
id: 289…482
```

In [169…

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

```
------------------------------------------------------------------
	       SCS v3.2.4 - Splitting Conic Solver
	(c) Brendan O'Donoghue, Stanford University, 2012
------------------------------------------------------------------
problem:  variables n: 8, constraints m: 16
cones: 	  z: primal zero / dual free vars: 1
	  l: linear vars: 6
	  q: soc vars: 9, 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
lin-sys:  sparse-direct-amd-qdldl
	  nnz(A): 36, nnz(P): 0
------------------------------------------------------------------
 iter | pri res | dua res |   gap   |   obj   |  scale  | time (s)
------------------------------------------------------------------
     0|1.71e+001 1.00e+000 1.62e+001 -8.02e+000 1.00e-001 1.26e-004
   125|1.81e-005 2.57e-006 1.57e-005 1.03e-001 1.00e-001 8.90e-003
------------------------------------------------------------------
status:  solved
timings: total: 8.91e-003s = setup: 1.03e-004s + solve: 8.80e-003s
	 lin-sys: 5.91e-005s, cones: 3.04e-005s, accel: 8.63e-003s
------------------------------------------------------------------
objective = 0.102876
------------------------------------------------------------------
```

In [170…

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

```
Оптимальное значение: 0.1028679143351298
Оптимальное решение: [2.3833981663828694e-7, 0.08100161792417866, 0.1206738628256472
4, 0.09562982046568322, 2.4824213062123293]
```

Оптимальная рассадка по залам

In [85]:

```julia
using Random
```

In [139…

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

Out[139…

```
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
```

In [140…

```
# Переделаны обозначения, потому что не нашла способа для оптимизации
N = 1000
peopl = collect(1:N)
people_pref = copy(hcat([shuffle([1, 2, 3, 10000, 10000]) for i in peopl]...))
```

Out[140…

```
5×1000 Matrix{Int64}:
 10000  10000  10000  10000  10000  …  10000      3  10000      3  10000
     2      1      3      3      3     10000  10000      3      2      3
 10000      3      1      1      2         1      2  10000  10000      2
     3  10000  10000      2      1         3  10000      2  10000      1
     1      2      2  10000  10000         2      1      1      1  10000
```

In [141…

```
model_zal = Model(GLPK.Optimizer)
```

Out[141…

```
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: GLPK
```

In [142…

```
@variable(model_zal, answ[peopl, zals_str], Bin)
```

Out[142...

```
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]
 answ[9,1]     answ[9,2]     answ[9,3]     answ[9,4]     answ[9,5]
 answ[10,1]    answ[10,2]    answ[10,3]    answ[10,4]    answ[10,5]
 answ[11,1]    answ[11,2]    answ[11,3]    answ[11,4]    answ[11,5]
 answ[12,1]    answ[12,2]    answ[12,3]    answ[12,4]    answ[12,5]
 answ[13,1]    answ[13,2]    answ[13,3]    answ[13,4]    answ[13,5]
 ⋮
 answ[989,1]   answ[989,2]   answ[989,3]   answ[989,4]   answ[989,5]
 answ[990,1]   answ[990,2]   answ[990,3]   answ[990,4]   answ[990,5]
 answ[991,1]   answ[991,2]   answ[991,3]   answ[991,4]   answ[991,5]
 answ[992,1]   answ[992,2]   answ[992,3]   answ[992,4]   answ[992,5]
 answ[993,1]   answ[993,2]   answ[993,3]   answ[993,4]   answ[993,5]
 answ[994,1]   answ[994,2]   answ[994,3]   answ[994,4]   answ[994,5]
 answ[995,1]   answ[995,2]   answ[995,3]   answ[995,4]   answ[995,5]
 answ[996,1]   answ[996,2]   answ[996,3]   answ[996,4]   answ[996,5]
 answ[997,1]   answ[997,2]   answ[997,3]   answ[997,4]   answ[997,5]
 answ[998,1]   answ[998,2]   answ[998,3]   answ[998,4]   answ[998,5]
 answ[999,1]   answ[999,2]   answ[999,3]   answ[999,4]   answ[999,5]
 answ[1000,1]  answ[1000,2]  answ[1000,3]  answ[1000,4]  answ[1000,5]
```

In [143...

```
for i in peopl
    @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, "
end
```

In [146...

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

Out[146...

$$10000answ_{1,1} + 2answ_{1,2} + 10000answ_{1,3} + 3answ_{1,4} + answ_{1,5} + 10000answ_{2,1} + answ_{2,2} + 3ans$$
$$+ 2answ_{3,5} + 10000answ_{4,1} + 3answ_{4,2} + answ_{4,3} + 2answ_{4,4} + 10000answ_{4,5} + 10000answ_{5,1} + 3$$
$$+ answ_{6,4} + 10000answ_{6,5} + [[\ldots 4940 \text{ terms omitted} \ldots]] + 3answ_{995,1} + 10000answ_{995,2} + 2answ$$
$$+ 3answ_{996,4} + 2answ_{996,5} + 3answ_{997,1} + 10000answ_{997,2} + 2answ_{997,3} + 10000answ_{997,4} + answ_{9}$$
$$+ 2answ_{999,2} + 10000answ_{999,3} + 10000answ_{999,4} + answ_{999,5} + 10000answ_{1000,1} + 3answ_{1000,2} + 2$$

In [148...

```
optimize!(model_zal)
```

In [149...

```
termination_status(model_zal)
```

Out[149...

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

In [153…

```
ress = value.(answ)
```

Out[153…

```
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  1.0  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  0.0  0.0  1.0  0.0
 0.0  1.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0
 0.0  1.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0
 0.0  0.0  1.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 ⋮
 0.0  0.0  0.0  1.0  0.0
 0.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  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  0.0  0.0  0.0  1.0
 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  0.0  1.0
 0.0  0.0  0.0  1.0  0.0
```

In [157…

```
zals_filling = zeros(5)
recomendationss = zeros(N)
for i in peopl
    for j in zals_str
        zals_filling[j] += ress[i, j]
        if ress[i, j] == 1
            recomendationss[i] = j
        end
    end
end
```

In [158…

```
zals_filling
```

Out[158…

```
5-element Vector{Float64}:
 200.0
 180.0
 220.0
 198.0
 202.0
```

In [159…

```
recomendationss
```

Out[159…

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

План приготовления кофе

In [12]:

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

Out[12]: $cappuccino$

In [13]:

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

Out[13]:

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

In [14]:

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

Out[14]:

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

In [15]:

```
const sugar_limit = 40
@constraint(model, raf * 5 == sugar_limit)
```

Out[15]:

$$5raf = 40$$

In [16]:

```
objective = 400 * raf + 300 * cappuccino
@objective(model, Max, objective)
```

Out[16]: $400raf + 300cappuccino$

In [17]:

```
optimize!(model)
```

In [18]:

```
println("Раф кофе: ", round(value(raf)))
println("Капучино: ", round(value(cappuccino)))
println("Прибыль: ", value(objective))
```

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