РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра теории вероятностей и кибербезопасности

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

дисциплина: Компьютерный практикум по статистическому анализу данных

Студент: Быстров Глеб

Группа: НПИбд-01-20

МОСКВА

2023 г.

Цель работы

В данной лабораторной работе мне будет необходимо освоить пакеты Julia для решения задач оптимизации.

Описание процесса выполнения работы

Линейное программирование

1. Линейное программирование рассматривает решения экстремальных задач на множествах n-мерного векторного пространства, задаваемых системами линейных уравнений и неравенств. Общей (стандартной) задачей линейного программирования называется задача нахождения минимума линейной целевой функции вида:

$$f(\vec{x}) = \sum_{j=1}^{n} c_j x_j,$$

где $\vec{c}$ — некоторые коэффициенты, $\vec{x} \in \mathbb{R}^n$.
Основной задачей линейного программирования называется задача, в которой есть ограничения в форме неравенств:

$$\sum_{j=1}^{n} a_{ij} x_j \geqslant b_i, \quad i = 1, 2, \ldots, m, \quad x_j \geqslant 0, \quad j = 1, 2, \ldots, n.$$

Задачи линейного программирования со смешанными ограничениями, такими как равенства и неравенства, с наличием переменных, свободных от ограничений, могут быть сведены к эквивалентным с тем же множеством решений путём замены переменных
и замены равенств на пару неравенств.

В Julia есть несколько средств, предназначенных для решения оптимизационных задач.

Одним из таких средств является JuMP (https://jump.dev/) — язык моделирования и вспомогательные пакеты для формулирования и решения задач математической оптимизации в Julia.

JuMP включает пакет Convex.jl (https://jump.dev/Convex.jl/stable/), позволяющий описать задачу оптимизации, используя естественный математический синтаксис, и решать её с помощью одного из решателей (COSMO, ECOS, SCS, GLPK, MathOptInterface).

Предположим, что требуется решить следующую задачу линейного

программирования:

$$12x + 20y \to \min$$

при заданных ограничениях:

$$6x + 8y \geqslant 100, \quad 7x + 12y \geqslant 120, \quad x \geqslant 0, \quad y \geqslant 0.$$

Воспользуемся JuMP и решателем линейного и смешанного целочисленного программирования GLPK (рис. 8.1):

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

```
    Updating registry at `C:\Users\GlebB\.julia\registries\General.toml`
   Resolving package versions...
   Installed CodecBzip2 ──────── v0.8.1
   Installed SnoopPrecompile ──── v1.0.3
   Installed MathOptInterface ─── v1.23.0
   Installed JuMP ─────────────── v1.17.0
    Updating `C:\Users\GlebB\.julia\environments\v1.9\Project.toml`
  [4076af6c] + JuMP v1.17.0
    Updating `C:\Users\GlebB\.julia\environments\v1.9\Manifest.toml`
```

Рис. 8.1. Подключение пакетов

2. Объект модели (контейнер для переменных, ограничений, параметров решателя и т. д.) в JuMP создаётся при помощи функции Model(), в которой в качестве аргумента указывается оптимизатор (решатель):

   # Определение объекта модели с именем model:

   model = Model(GLPK.Optimizer)

   Переменные задаются с помощью конструкции @variable (имя объекта модели, имя и привязка переменной, тип переменной). Здесь же задаются граничные условия на переменные (если тип переменной не определён, он считается действительным):

   # Определение переменных x, y и граничных условий для них:

   @variable(model, x >= 0)

   @variable(model, y >= 0)

   В качестве первого аргумента указан объект модели model, затем переменные x и y, связанные с этой моделью (причём указанные переменные не могут использоваться в другой модели).

Ограничения модели задаются с помощью конструкции @constraint (имя объекта

модели, ограничение):

Определение ограничений модели:

@constraint(model, 6x + 8y >= 100)

@constraint(model, 7x + 12y >= 120)

Далее следует задать собственно целевую функцию с помощью конструкции

@objective (имя объекта модели, Min или Max, функция для оптимизации):

Определение целевой функции:

@objective(model, Min, 12x + 20y)

Для решения задачи оптимизации необходимо вызывать функцию оптимизации:

Вызов функции оптимизации:

optimize!(model)

(рис. 8.2):

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

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

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

y

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

$7x + 12y \geq 120$

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

$12x + 20y$

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

Рис. 8.2. Определение объекта, переменных и функций

3. Следует проверять причину прекращения работы оптимизатора,

используя конструкцию termination_status(Объект модели):

Определение причины завершения работы оптимизатора:

termination_status(model)

Процесс решения мог быть прекращён по ряду причин. Во-первых, решатель мог найти оптимальное решение или доказать, что проблема невозможна. Однако он также мог столкнуться с численными трудностями или прерваться из-за таких настроек, как ограничение по времени. Если возвращено значение OPTIMAL, найдено оптимальное решение.

Наконец, можно посмотреть собственно результат решения оптимизационной задачи:

Демонстрация первичных результирующих значений переменных x и y:

@show value(x);

@show value(y);

Демонстрация результата оптимизации:

@show objective_value(model); (рис. 8.3):

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

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

```
# 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
```

Рис. 8.3. Определение причины и демонстрация

Векторизованные ограничения и целевая функция оптимизации

4. Часто бывает полезно создавать коллекции переменных JuMP внутри более сложных структур данных. Можно добавить ограничения и цель

в JuMP, используя векторизованную линейную алгебру.

Предположим, что требуется решить следующую задачу:

$$\vec{c}^T\vec{x} \to \min$$

при заданных ограничениях:

$$A\vec{x} = \vec{b}, \quad \vec{x} \succeq 0, \quad \vec{x} \in \mathbb{R},$$

где

$$A = \begin{pmatrix} 1 & 1 & 9 & 5 \\ 3 & 5 & 0 & 8 \\ 2 & 0 & 6 & 13 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} 7 \\ 3 \\ 5 \end{pmatrix}, \quad \vec{c} = \begin{pmatrix} 1 \\ 3 \\ 5 \\ 2 \end{pmatrix}.$$

Воспользуемся JuMP и решателем линейного и смешанного целочисленного программирования GLPK:

Подключение пакетов:

import Pkg

Pkg.add("JuMP")

Pkg.add("GLPK")

using JuMP

using GLPK

Определим объект модели:

Определение объекта модели с именем vector_model:

vector_model = Model(GLPK.Optimizer)

Зададим исходные значения матрицы A и векторов $\vec{b}$, :$\vec{c}$

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]

Далее зададим массив переменных для компонент вектора :$\vec{x}$

Определение вектора переменных:

@variable(vector_model, x[1:4] >= 0)

(рис. 8.4):

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

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

```
# 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]
```

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

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

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

Рис. 8.4. Определение объекта, данных и вектора

Оптимизация рациона питания

5. В некоторых задачах требуется использование массивов, в которых индексы не являются целыми диапазонами с отсчётом от единицы. Например, требуется использовать переменную, индексируемую по названию продукта или местоположению. Тогда необходимо в качестве индекса использовать произвольный вектор. В Julia это можно реализовать с помощью DenseAxisArrays.
6. Рассмотрим применение DenseAxisArrays на примере решения задачи оптимизации рациона питании в заведении быстрого питания при условии, что задано ограничение на количество потребляемых калорий (1800–2200), белков ($\geqslant$ 91), жиров (0–65) и соли (0–1779), а также перечень определённых продуктов питания с указанием их стоимости — гамбургер (2.49 ден.ед.), курица (2.89 ден.ед.), сосиска в тесте (1.50

ден.ед.), жареный картофель (1.89 ден.ед.), макароны (2.09 ден.ед.), пицца (1.99 ден.ед.), салат (2.49 ден.ед.), молочный коктейль (0.89 ден.ед.), мороженное (1.59 ден.ед.). Также известно содержание калорий, белков, жиров и соли в указанных продуктах. (рис. 8.5):

Таблица 8.1

Содержание калорий, белков, жиров и соли в продуктах питания

продукт	калории	белки	жиры	соль
гамбургер	10	24	26	730
курица	420	32	10	1190
сосиска в тесте	560	20	32	1800
жареный картофель	380	4	19	270
макароны	320	12	10	930
пицца	320	15	12	820
салат	320	31	12	1230
молочный коктейль	100	8	2.5	125
мороженное	330	8	10	180

Воспользуемся JuMP и решателем линейного и смешанного целочисленного программирования GLPK :

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

Создадим контейнер JuMP для хранения информации об ограничениях (минимальное,

максимальное) на количество потребляемых калорий, белков, жиров и соли:

```
# Контейнер для хранения данных об ограничениях на количество потребляемых калорий, белков, жиров и соли:
category_data = JuMP.Containers.DenseAxisArray(
```

```
# Контейнер для хранения данных об ограничениях на количество потребляемых калорий, белков, жиров и соли:
category_data = JuMP.Containers.DenseAxisArray(
[1800 2200;
91 Inf;
0 65;
0 1779],
["calories", "protein", "fat", "sodium"],
["min", "max"])
```

```
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, ["calories", "protein", "fat", "sodium"]
    Dimension 2, ["min", "max"]
And data, a 4x2 Matrix{Float64}:
 1800.0  2200.0
   91.0    Inf
    0.0    65.0
    0.0  1779.0
```

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

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

Рис. 8.5. Контейнер и массив

7. Введём данные о стоимости продуктов:

   # 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)

   Введём сведения о содержании калорий, белков, жиров и соли в продуктах питания:

   # Массив данных о содержании калорий, белков, жиров и соли в продуктах питания:

   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"])

(рис. 8.6).

```
# 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)
```

```
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
```

```
# Массив данных о содержании калорий, белков, жиров и соли в продуктах питания:
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"])
```

Рис. 8.6. Массивы данных

8. Определим объект модели:

   # Определение объекта модели с именем model:

   model = Model(GLPK.Optimizer)

   Определим массив:

   # Определим массив:

   categories = ["calories", "protein", "fat", "sodium"]

   (рис. 8.7)

```
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 9x4 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
```

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

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

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

```
4-element Vector{String}:
 "calories"
 "protein"
 "fat"
```

Рис. 8.7. Определение объекта и массива

9. Далее зададим переменные:

   # Определение переменных:

   @variables(model, begin

   category_data[c, "min"] <= nutrition[c = categories] <= category_data[c, "max"]

   # Сколько покупать продуктов (рис. 8.8):

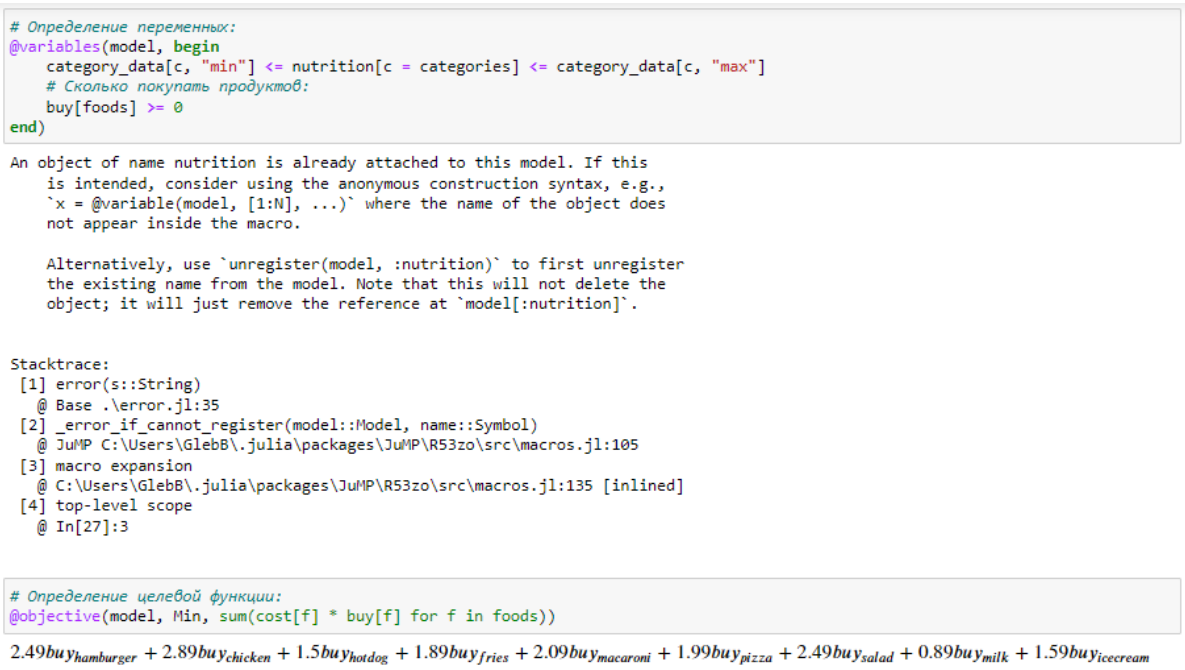
   buy[foods] >= 0

   end)

   Задаём целевую функцию минимизации цены:

   # Определение целевой функции:

   @objective(model, Min, sum(cost[f] * buy[f] for f in foods))



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

```
An object of name nutrition is already attached to this model. If this
    is intended, consider using the anonymous construction syntax, e.g.,
    `x = @variable(model, [1:N], ...)` where the name of the object does
    not appear inside the macro.

    Alternatively, use `unregister(model, :nutrition)` to first unregister
    the existing name from the model. Note that this will not delete the
    object; it will just remove the reference at `model[:nutrition]`.


Stacktrace:
 [1] error(s::String)
   @ Base .\error.jl:35
 [2] _error_if_cannot_register(model::Model, name::Symbol)
   @ JuMP C:\Users\GlebB\.julia\packages\JuMP\R53zo\src\macros.jl:105
 [3] macro expansion
   @ C:\Users\GlebB\.julia\packages\JuMP\R53zo\src\macros.jl:135 [inlined]
 [4] top-level scope
   @ In[27]:3
```

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

$$2.49buy_{hamburger} + 2.89buy_{chicken} + 1.5buy_{hotdog} + 1.89buy_{fries} + 2.09buy_{macaroni} + 1.99buy_{pizza} + 2.49buy_{salad} + 0.89buy_{milk} + 1.59buy_{icecream}$$

Рис. 8.8. Определение переменных и функции

10. # Определение ограничений модели:

@constraint(model, [c in categories],

sum(food_data[f, c] * buy[f] for f in foods) == nutrition[c]

Наконец, для решения задачи оптимизации вызовем функцию оптимизации:

Вызов функции оптимизации:

JuMP.optimize!(model)

term_status = JuMP.termination_status(model)

Для просмотра результата решения модно вывести значение переменной buy:

hcat(buy.data,JuMP.value.(buy.data))

В результате оптимальным по цене и пищевой ценности будет предложено купить

гамбургер, молочный коктейль и мороженное.

(рис. 8.9).

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

```
1-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float6
4}, MathOptInterface.EqualTo{Float64}}, ScalarShape},1,...} with index sets:
    Dimension 1, ["calories", "protein", "fat", "sodium"]
And data, a 4-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float6
4}, MathOptInterface.EqualTo{Float64}}, 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[milk] + 330 buy[ice cream] == 0
 -nutrition[protein] + 24 buy[hamburger] + 32 buy[chicken] + 20 buy[hot dog] + 4 buy[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] + 1
2 buy[salad] + 2.5 buy[milk] + 10 buy[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[milk] + 180 buy[ice cream] == 0
```

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

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

```
hcat(buy.data,JuMP.value.(buy.data))
```

```
9×2 Matrix{AffExpr}:
 buy[hamburger]  0.6045138888888888
 buy[chicken]    0
 buy[hot dog]    0
 buy[fries]      0
 buy[macaroni]   0
 buy[pizza]      0
```

Рис. 8.9 Определение модели и вызов функции

Путешествие по миру

11. Рассмотрим решение задачи определения оптимального числа паспортов, требующихся, чтобы объехать весь мир. При этом будем учитывать сведения о паспортах и ограничениях, указанных на ресурсе https://www.passportindex.org/. В табличной форме данные можно получить с ресурса https://github.com/ilyankou/ passport-index-dataset. Для решения задачи представляют интерес данные, указанные в файле passport-index-matrix.csv, в котором в первым столбце (Passport) указана страна отбытия (= от), в остальных столбцах указаны страны назначения (= до), на пересечении строк и столбцов указаны условия по наличию или отсутствию визы или другие ограничения (обозначения пояснены в табл. 8.2).

Таблица 8.2

Обозначения в сводной матрице по паспортам

Значение	Пояснение
7–360	Количество безвизовых дней (при наличии)
VF	viza free (безвизовый режим)
VOA	visa on arriva (виза по прибытии)
ETA	e-visa или electronic travel authority (электронная виза)
VR	visa required (требуется виза)
covid ban	запрет в связи с COVID=19
no admission	въезд запрещён
-1	паспорт из места назначения

Итак, попробуем решить задачу средствами Julia.

Сначала требуется скачать файл с данными. Например для ОС типа Linux можно воспользоваться стандартным вызовом команды git с соответствующими параметрами:

```
# Скачиваем данные с ресурса на git:
;git clone https://github.com/ilyankou/passport-index-dataset.git
```

Затем требуется подключить пакеты для обработки табличных файлов:

```
# Подключение пакетов:
import Pkg
Pkg.add("DelimitedFiles")
Pkg.add("CSV")
using DelimitedFiles
using CSV
```

Можем считать данные из имеющегося файла:

```
# Считывание данных:
passportdata = readdlm(joinpath("passport-index-dataset","passport- index-matrix.csv"),',')
```

```
# Подключение пакетов:
import Pkg
Pkg.add("DelimitedFiles")
Pkg.add("CSV")
using DelimitedFiles
using CSV
```

```
   Resolving package versions...
    Updating `C:\Users\GlebB\.julia\environments\v1.9\Project.toml`
  [8bb1440f] + DelimitedFiles v1.9.1
  No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Manifest.toml`
   Resolving package versions...
  No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Project.toml`
  No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Manifest.toml`
```

```
# Считывание данных:
passportdata = readdlm(joinpath("passport-index-matrix.csv"),',')
```

```
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"
```

Рис. 8.10. Подключение пакетов и считывание данных

12. Для решения задачи используем возможности пакета JuMP и решателя линейного и смешанного целочисленного программирования GLPK:

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

Далее просматриваем файл, задаём переменную для подсчёта числа паспортов, задаём переменную vf, в которую заносим сведения при отсутствии необходимости получать визу (если в поле указано число, «VF» или «VOA»):

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

Определяем объект модели:

```
# Определение объекта модели с именем model:
```

model = Model(GLPK.Optimizer)

Добавляем переменные, ограничения и целевую функцию:

Переменные, ограничения и целевая функция:

@variable(model, pass[1:length(cntr)], Bin)

@constraint(model, [j=1:length(cntr)], sum(vf[i,j]*pass[i] for i in 1:length(cntr)) >= 1)

@objective(model, Min, sum(pass))

Для решения задачи оптимизации вызываем функцию оптимизации:

Вызов функции оптимизации:

JuMP.optimize!(model)

termination_status(model)

(рис. 8.11).

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

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

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

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

$$pass_1 + pass_2 + pass_3 + pass_4 + pass_5 + pass_6 + pass_7 + pass_8 + pass_9 + pass_{10} + pass_{11} + pass_{12} + pass_{13} + pass_{14} + pass_{15} + pass_{16} + pass_{17} + pass_{18} + pass_{19} + pass_{20} + pass_{21} + pass_{22} + pass_{23} + pass_{24} + pass_{25} + pass_{26} + pass_{27} + pass_{28} + pass_{29} + pass_{30} + [[\dots \text{ 139 terms omitted } \dots]] + pass_{170} + pass_{171} + pass_{172} + pass_{173} + pass_{174} + pass_{175} + pass_{176} + pass_{177} + pass_{178} + pass_{179} + pass_{180} + pass_{181} + pass_{182} + pass_{183} + pass_{184} + pass_{185} + pass_{186} + pass_{187} + pass_{188} + pass_{189} + pass_{190} + pass_{191} + pass_{192} + pass_{193} + pass_{194} + pass_{195} + pass_{196} + pass_{197} + pass_{198} + pass_{199}$$

```
# Вызов функции оптимизации:
JuMP.optimize!(model)
termination_status(model)
```

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

Рис. 8.11. Переменные, определение объекта, вызов функции

13. Просматриваем результат:

Просмотр результата:

print(JuMP.objective_value(model)," passports:

",join(cntr[findall(JuMP.value.(pass) .== 1)],", "))

```
# Просмотр результата:
print(JuMP.objective_value(model)," passports: ",join(cntr[findall(JuMP.value.(pass) .== 1)],", "))
```

```
34.0 passports: Afghanistan, Australia, Bahrain, Cameroon, Canada, Comoros, Congo, Denmark, Djibouti, Eritrea, Guinea-Bissau, H
ong Kong, Iran, Kenya, Kuwait, Liberia, Libya, Madagascar, Maldives, Mauritania, Morocco, Nauru, Nepal, New Zealand, North Kore
a, Palestine, Papua New Guinea, Qatar, Saudi Arabia, Singapore, Somalia, Sri Lanka, Syria, Turkmenistan
```

Рис. 8.12. Просмотр результата

Портфельные инвестиции

14. Портфельные инвестиции — размещение капитала в ценные бумаги, формируемые в виде портфеля ценных бумаг, с целью получения прибыли.

    Инвестиционный портфель — процесс стратегического управления капиталом как оптимизированной, единой системой инвестиционных ценностей.

    Предположим требуется решить оптимизационную задачу в следующей формулировке. Имеется капитал в 1000 ден. ед., который планируется инвестировать в три компании — Microsoft, Facebook, Apple. При этом есть данные еженедельных значений цен на акции этих компаний за определённый период времени. Необходимо определить доходность акций каждой компании за рассматриваемый период времени, после чего инвестировать в эти три компании так, чтобы получить возврат в размере не менее 2% от вложенной суммы.

    Для решения оптимизационной задачи будем использовать пакет Convex.jl и оптимизатор (решатель) SCS. Кроме того, понадобится пакет Statistics.jl для получения матрицы рисков на основе расчёта ковариационных значений по доходности.

    Сначала требуется подгрузить имеющиеся данные о еженедельных значениях цен

    на акции рассматриваемых компаний, отобразить данные на графике. Предположим

    данные размещены в файле stock_prices.xlsx в каталоге data в проекте.

    Подключаем пакеты (рис. 8.13):

    # Подключение необходимых пакетов:

import Pkg

Pkg.add("DataFrames")

Pkg.add("XLSX")

Pkg.add("Plots")

Pkg.add("PyPlot")

Pkg.add("Convex")

Pkg.add("SCS")

Pkg.add("Statistics")

using DataFrames

using XLSX

using Plots

pyplot()

using Convex

using SCS

using Statistics

При установки PyCall происходит ошибка

Рис. 8.13. Информация об ошибке

Подгружаем данные еженедельных значений цен на акции компаний за определённый

период времени во фрейм:

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

Отображаем данные на графике (рис. 8.1):

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

Для дальнейших расчётов данные по ценам на акции переформируем из фрейма

в матрицу:

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

Доходность акций i-й компании за период времени t определяется формулой:

$R(i, t) = (\text{pr}(i, t) - \text{pr}(i, t-1))/\text{pr}(i, t-1)$, где $\text{pr}(i, t)$ — цена акций i-й компании за

период времени t:

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

Далее необходимо сформировать матрицу рисков — ковариационную матрицу рассчитанных цен доходности:

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

Доход от каждой из компаний получим из матрицы доходности как

вектор средних

значений:

Доход от каждой из компаний:

r = mean(R,dims=1)[:]

Далее нужно собственно сформулировать оптимизационную задачу. Пусть вектор $\vec{x} = (x1, x2, x3)$ — вектор инвестиций, вектор $\vec{r} = (r1, r2, r3)$ — вектор доходов от

компаний. Тогда задача оптимизации будет иметь вид:

$$\vec{x}^T \text{cov}(R)\vec{x} \to \min$$

при условии:

$$\sum_{i=1}^{3} x_i = 1, \quad \text{dot}(\vec{r}, \vec{x}) \geqslant 0,02, \quad x_i \geqslant 0,\ i = 1, 2, 3,$$

где dot($\vec{r}$, $\vec{x}$) — функция, определяющая возврат инвестиций.

Формируем вектор инвестиций:

Вектор инвестиций:

x = Variable(length(r))

Определяем объект модели в соответствии с формулировкой оптимизационной задачи (при этом делаем задачу совместимой с DCP в соответствии с требованием пакета

Convex.jl — см. http://cvxr.com/cvx/doc/dcp.html):

Объект модели:

problem = minimize(Convex.quadform(x,risk_matrix),[sum(x)==1;r'*x>=0.02;x.>=0])

Решаем поставленную задачу:

Находим решение:

solve!(problem, SCS.Optimizer)

Выводим значения компонент вектора инвестиций:

x

Проверяем выполнение условия $\sum_{i=1}^{3} x_i = 1$:

sum(x.value)

Проверяем выполнение условия на уровень доходности от 2%:

r'*x.value

Переводим процентные значения компонент вектора инвестиций в фактические денежные единицы:

x.value .* 1000

В результате, получаем: надо инвестировать 67.9 ден. ед. в Microsoft, 122.3 ден. ед. в Facebook, 809.7 ден. ед. в Apple.

Восстановление изображения

15. Предположим есть изображение, на котором были изменены некоторые пиксели. Требуется восстановить неизвестные пиксели путём решения задачи оптимизации.

    Будем использовать пакет Convex.jl и оптимизатор (решатель) SCS, пакет

    ImageMagick.jl для работы с изображениями:

    # Подключение необходимых пакетов:

    import Pkg

    Pkg.add("ImageMagick")

    Pkg.add("Convex")

    Pkg.add("SCS")

    using Images

    using Convex

    using SCS

```
# Подключение необходимых пакетов:
import Pkg
Pkg.add("ImageMagick")
Pkg.add("Convex")
Pkg.add("SCS")
using Images
using Convex
using SCS
```

```
  Resolving package versions...
 No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Project.toml`
 No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Project.toml`
 No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\GlebB\.julia\environments\v1.9\Project.toml`
```

Рис. 8.14. Подключение пакетов

16. Загрузим изображение для последующей обработки (рис. 8.15):

Считывание исходного изображения:

Kref = load("data/khiam-small.jpg")

Преобразуем изображение в оттенки серого и испортим некоторые пиксели (рис. ??):

K = copy(Kref)

p = prod(size(K))

missingids = rand(1:p,400)

K[missingids] .= RGBX{N0f8}(0.0,0.0,0.0)

K

Gray.(K)

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

```
[ Info: Precompiling Images [916415d5-f1e6-5110-898d-aaa5f9f070e0]
[ Info: Precompiling ForwardDiffStaticArraysExt [b74fd6d0-9da7-541f-a07d-1b6af30a262f]
[ Info: Precompiling ForwardDiffExt [4b8ca102-9d0f-5df8-84ed-253e0f005ba7]
```

```
K = copy(Kref)
p = prod(size(K))
missingids = rand(1:p,400)
K[missingids] .= RGBX{N0f8}(0.0,0.0,0.0)
K
Gray.(K)
```

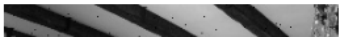


Рис. 8.15. Считывание и преобразование изображения

17. Формируем матрицу со значениями цветов (рис. 8.16):

Матрица цветов:

Y = Float64.(Gray.(K));

Далее необходимо сформировать новую матрицу X, в которой минимизируется норма ядра матрицы (т.е. сумма сингулярных чисел элементов матрицы) так, что элементы,ькоторые уже известны в матрице Y, остаются теми же самыми в матрице X:

correctids = findall(Y[:].!=0)

X = Convex.Variable(size(Y))

problem = minimize(nuclearnorm(X))

problem.constraints += X[correctids]==Y[correctids]

Решаем поставленную задачу:

Находим решение:

solve!(problem, SCS.Optimizer(eps=1e-3, alpha=1.5))

Выводим значение нормы и исправленное изображение:

@show norm(float.(Gray.(Kref))-X.value)

@show norm(-X.value)

colorview(Gray, X.value)

```
# Матрица цветов:
Y = Float64.(Gray.(K));
```

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

```
1-element Vector{Constraint}:
 == constraint (affine)
├─ index (affine; real)
│  └─ 283x283 real variable (id: 380…472)
└─ 79691-element Vector{Float64}
```

```
# Находим решение:
solve!(problem, SCS.Optimizer(eps=1e-3, alpha=1.5))
```

```
MethodError: no method matching SCS.Optimizer(; eps::Float64, alpha::Float64)

Closest candidates are:
  SCS.Optimizer() got unsupported keyword arguments "eps", "alpha"
   @ SCS C:\Users\GlebB\.julia\packages\SCS\mqg7w\src\MOI_wrapper\MOI_wrapper.jl:131


Stacktrace:
 [1] top-level scope
   @ In[67]:3
```

Рис. 8.16. Ошибка при работе с методом

Задания для самостоятельного выполнения

18. Линейное программирование

Решите задачу линейного программирования:

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

при заданных ограничениях:

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

(рис. 8.17-8.18):

```
#Задание №1
model = Model(GLPK.Optimizer)
```

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

```
@variable(model, 0 <= x1 <= 10)
@variable(model, x2 >= 0)
@variable(model, x3 >= 0)
```

$x3$

```
@constraint(model, -x1 + x2 + 3x3 <= -5)
@constraint(model, x1 + 3x2 - 7x3 <= 10)
```

$x1 + 3x2 - 7x3 \leq 10$

```
@objective(model, Max, x1 + 2x2 + 5x3)
```

$x1 + 2x2 + 5x3$

```
optimize!(model)
```

```
termination_status(model)
```

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

Рис. 8.17. Определение объекта, переменных, ограничений, функции и определение причины

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

@show objective_value(model);
```

```
value(x1) = 10.0
value(x2) = 2.1875
value(x3) = 0.9375
objective_value(model) = 19.0625
```

Рис. 8.18. Результаты

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

(рис. 8.19-8.20)

```
#Задание №2
vector_model_2 = Model(GLPK.Optimizer)
```

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

```
A= [-1 1 3;
    1 3 -7]
b = [-5; 10]
c = [1; 2; 5]
```

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

```
@variable(vector_model_2, x[1:3] >= 0)
set_upper_bound(x[1], 10)
```

```
@constraint(vector_model_2, A * x .== b)
```

```
2-element Vector{ConstraintRef{Model, MathOptInterf
nterface.EqualTo{Float64}}, ScalarShape}}:
 -x[1] + x[2] + 3 x[3] == -5
 x[1] + 3 x[2] - 7 x[3] == 10
```

Рис. 8.19. Определение объекта, данных, вектора и ограничений

```
@objective(vector_model_2, Max, c' * x)
```

$$x_1 + 2x_2 + 5x_3$$

```
optimize!(vector_model_2)
```

```
termination_status(vector_model_2)
```

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

```
@show value(x[1]);
@show value(x[2]);
@show value(x[3]);
@show objective_value(vector_model_2);
```

```
value(x[1]) = 10.0
value(x[2]) = 2.1875
value(x[3]) = 0.9375
objective_value(vector_model_2) = 19.0625
```

Рис. 8.20. Определение функции, вызов и результаты

20. 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. 8.21-8.22)

```
#Задание №3
using Convex
using SCS

m = 5
n = 4

A = rand(m, n)
b = rand(m)

display(A)
println()
display(b)

x = Variable(n)

display(x)

model_new = minimize(Convex.sumsquares(A*x - b),[x >= 0])

solve!(model_new, SCS.Optimizer)

model_new.status

model_new.optval
```

```
5x4 Matrix{Float64}:
 0.773851   0.731559  0.357961   0.927942
 0.314247   0.176215  0.0499934  0.0422559
 0.466721   0.716408  0.0630813  0.348065
 0.0164345  0.729675  0.735458   0.333719
```

Рис. 8.21. Задаём размерность, создаём вектор и объект, выводим решение

```
sign: real
vexity: affine
id: 144…234

------------------------------------------------------------------
               SCS v3.2.4 - Splitting Conic Solver
        (c) Brendan O'Donoghue, Stanford University, 2012
------------------------------------------------------------------
problem:  variables n: 7, constraints m: 15
cones:    z: primal zero / dual free vars: 1
          l: linear vars: 5
          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): 30, nnz(P): 0
------------------------------------------------------------------
 iter | pri res | dua res |   gap   |   obj   |  scale  | time (s)
------------------------------------------------------------------
     0|1.71e+001 1.00e+000 1.62e+001 -8.04e+000 1.00e-001 4.58e-003
   125|1.00e-005 5.56e-006 1.04e-005 4.84e-002 5.56e-001 4.71e-003
------------------------------------------------------------------
status:  solved
timings: total: 4.71e-003s = setup: 4.52e-003s + solve: 1.91e-004s
         lin-sys: 5.93e-005s, cones: 3.31e-005s, accel: 7.20e-006s
------------------------------------------------------------------
objective = 0.048427
------------------------------------------------------------------

0.048422158379626865
```

Рис. 8.22. Результаты

Вывод

В данной лабораторной работе мне успешно удалось освоить пакеты Julia для решения задач оптимизации.

Приложение

```
#Задание №1
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)
termination_status(model)
@show value(x1);
@show value(x2);
@show value(x3);
@show objective_value(model);

#Задание №2
vector_model_2 = Model(GLPK.Optimizer)
A= [-1 1 3;
   1 3 -7]
b = [-5; 10]
c = [1; 2; 5]
@variable(vector_model_2, x[1:3] >= 0)
set_upper_bound(x[1], 10)
@constraint(vector_model_2, A * x .== b)
@objective(vector_model_2, Max, c' * x)
optimize!(vector_model_2)
termination_status(vector_model_2)
@show value(x[1]);
@show value(x[2]);
@show value(x[3]);
@show objective_value(vector_model_2);

#Задание №3
using Convex
using SCS
m = 5
n = 4
A = rand(m, n)
b = rand(m)
display(A)
println()
display(b)
x = Variable(n)
display(x)
model_new = minimize(Convex.sumsquares(A*x - b),[x >= 0])
solve!(model_new, SCS.Optimizer)
model_new.status
model_new.optval
```