

Лабораторная работа №4. Компьютерный практикум по статистическому анализу данных

Леснухин Даниил Дмитриевич
Российский университет дружбы народов
Москва

Цель работы

Основной целью работы является изучение возможностей специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Задание

1. Используя JupyterLab, повторите примеры.
2. Выполните задания для самостоятельной работы.

Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [[@julialang](#)]. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia [[@juliadoc](#)].

Выполнение лабораторной работы

Выполнение примеров

Выполним примеры из раздела про поэлементные операции над многомерными массивами .

Выполним примеры из раздела про транспонирование, след, ранг, определитель, инверсия матрицы

Выполним примеры из раздела про вычисление нормы векторов и матриц, повороты, вращения

Выполним примеры из раздела про матричное умножение, единичную матрицу, скалярное произведение

Выполним примеры из раздела про факторизацию и специальные матричные структуры

```

  ▾ Поэлементные операции над многомерными массивами

[1]
✓ 0 сек.
# Массив 4x3 со случайными целыми числами (от 1 до 20):
a = rand(1:20, (4,3))

4x3 Matrix{Int64}:
 20  20   8
 15   2   8
 11  12  15
 18   6  19

[2]
✓ 0 сек.
▶ # Поэлементная сумма:
sum(a)
# Поэлементная сумма по столбцам:
sum(a,dims=1)
# Поэлементная сумма по строкам:
sum(a,dims=2)
# Поэлементное произведение:
prod(a)
# Поэлементное произведение по столбцам:
prod(a,dims=1)
# Поэлементное произведение по строкам:
prod(a,dims=2)

... 4x1 Matrix{Int64}:
3200
 240
1980
2052

[3]
✓ 30 сек.
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics
# Вычисление среднего значения массива:
mean(a)
# Среднее по столбцам:
mean(a,dims=1)
# Среднее по строкам:
mean(a,dims=2)

Updating registry at `~/.julia/registries/General.toml`
Resolving package versions...
Updating `~/.julia/environments/v1.11/Project.toml`
[10745b16] + Statistics v1.11.1
No Changes to `~/.julia/environments/v1.11/Manifest.toml`
4x1 Matrix{Float64}:
16.0
```

Рис. 1: Поэлементные операции над многомерными массивами

Транспонирование, след, ранг, определитель и инверсия матрицы

```
[4] 9 сек.
# Подключение пакета LinearAlgebra:
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
# Массив 4x4 со случайными целыми числами (от 1 до 20):
b = rand(1:20, (4,4))
# Транспонирование:
transpose(b)
# След матрицы (сумма диагональных элементов):
tr(b)
# Извлечение диагональных элементов как массив:
diag(b)
# Ранг матрицы:
rank(b)
# Инверсия матрицы (определение обратной матрицы):
inv(b)
# Определитель матрицы:
det(b)
# Псевдообратная функция для прямоугольных матриц:
pinv(a)

... Resolving package versions...
Updating `~/julia/environments/v1.11/Project.toml`
[37e2e46d] + LinearAlgebra v1.11.0
No Changes to `~/julia/environments/v1.11/Manifest.toml`
3x4 Matrix{Float64}:
 0.0255752  0.0674405 -0.0566417  0.00555265
 0.0395018 -0.0492086  0.0390821 -0.0267672
-0.0429881 -0.030739  0.0577149  0.0381102
```

Рис. 2: Транспонирование, след, ранг, определитель и инверсия матрицы

```
▼ Вычисление нормы векторов и матриц, повороты, вращения

[5] 0 сек. # Создание вектора X:
X = [2, 4, -5]
# Вычисление евклидовой нормы:
norm(X)
# Вычисление p-нормы:
p = 1
norm(X,p)

... 11.0

[6] 0 сек. # Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1, -1, 3];
norm(X-Y)

9.486832980505138

[7] 0 сек. # Проверка по базовому определению:
sqrt(sum((X-Y).^2))

9.486832980505138

[9] 0 сек. # Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))

# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
# Вычисление Евклидовой нормы:
norm(d)
# Вычисление p-нормы:
p=1
norm(d,p)
# Поворот на 180 градусов:
rot180(d)
# Переворачивание строк:
reverse(d,dims=1)
# Переворачивание столбцов
reverse(d,dims=2)
#

3x3 Matrix{Int64}:
 2 -4  5
 3  2 -1
 0  1 -2
```

Рис. 3: Вычисление нормы векторов и матриц, повороты, вращения

```
[10]
✓ 0 сек.
# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand(1:10, (2,3))
# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand(1:10, (3,4))
# Произведение матриц A и B:
A*B
# Единичная матрица 3x3:
Matrix{Int}(I, 3, 3)
# Скалярное произведение векторов X и Y:
X = [2, 4, -5]
Y = [1, -1, 3]
dot(X,Y)
# тоже скалярное произведение:
X'Y
```

-17

```
[11] # Задаём квадратную матрицу 3x3 со случайными значениями:
2 A = rand(3, 3)
сек. # Задаём единичный вектор:
x = fill(1.0, 3)
# Задаём вектор b:
b = A*x
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
A\b
```

```
[13] # LU-факторизация:
✓ 0 Alu = lu(A)
сек.

LU[Float64, Matrix{Float64}, Vector{Int64}]
L factor:
3×3 Matrix{Float64}:
 1.0      0.0      0.0
 0.630634 1.0      0.0
 0.802078 -0.663093 1.0

U factor:
3×3 Matrix{Float64}:
 0.572395 0.565877 0.232299
 0.0      -0.170278 0.550369
 0.0      0.0      0.208105
```

0.020283246786260464

```

# Матрица Q:
Aqr.Q
# Матрица R:
Aqr.R
# Проверка, что матрица Q - ортогональная:
Aqr.Q'*Aqr.Q

```

```

... 3x3 Matrix{Float64}:
 1.0  0.0 -1.11022e-16
 0.0  1.0 -1.11022e-16
 0.0 -2.77556e-17  1.0

```

```

# Симметризация матрицы A:
Asym = A + A'
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
# Собственные значения:
AsymEig.values
# Собственные векторы:
AsymEig.vectors
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym

```

```

... 3x3 Matrix{Float64}:
 1.0 -5.77316e-15  1.02141e-14
 2.22045e-15  1.0 -5.9952e-15
 1.77636e-15  1.9984e-15  1.0

```

```

# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:
Asym = A + A'
# Проверка, является ли матрица симметричной:
issymmetric(Asym)

```

```
true
```

```

# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)

```

```
false
```

```

import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы:
@btime eigvals(Asym_noisy);
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы,
# для которой явно указано, что она симметричная:
@btime eigvals(Asym_explicit);

...
Resolving package versions...
Installed BenchmarkTools - v1.6.3
Updating `~/.julia/environments/v1.11/Project.toml`
[6e4b80f9] + BenchmarkTools v1.6.3
Updating `~/.julia/environments/v1.11/Manifest.toml`
[6e4b80f9] + BenchmarkTools v1.6.3
[9abbd945] + Profile v1.11.0
Precompiling project...
2710.9 ms ✓ BenchmarkTools
1 dependency successfully precompiled in 4 seconds. 469 already precompiled.
123.690 ms (21 allocations: 7.99 MiB)
703.937 ms (27 allocations: 7.93 MiB)
123.717 ms (21 allocations: 7.99 MiB)

```

```

# Трёхдиагональная матрица 1000000 x 1000000:
n = 1000000;
A = SymTridiagonal(randn(n), randn(n-1))
# Оценка эффективности выполнения операции по нахождению
# собственных значений:
@btime eigmax(A)

```

```

709.663 ms (44 allocations: 183.11 MiB)
6.934806147575959

```

Общая линейная алгебра

[24]
0 сек.

```
# Матрица с рациональными элементами:  
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10  
# Единичный вектор:  
x = fill(1, 3)  
# Задаём вектор b:  
b = Arational*x  
# Решение исходного уравнения получаем с помощью функции \  
# (убеждаемся, что x - единичный вектор):  
Arational\b  
# LU-разложение:  
lu(Arational)
```

▼

...
UndefVarError: `ля` not defined in `main`
Suggestion: check for spelling errors or missing imports.

Далее: [Объяснить ошибку](#)

Задания для самостоятельного выполнения

Произведение векторов

Теперь перейдем к выполнению заданий для самостоятельной работы.

1. Зададим вектор v . Умножим вектор v скалярно сам на себя и сохраним результат в `dot_v`.
2. Умножим v матрично на себя (внешнее произведение), присвоив результат переменной `outer_v`

Системы линейных уравнений

1. Решим СЛАУ с двумя неизвестными

1

[25]

0

CEK.

v = [1, 2, 5, 1]

4-element Vector{Int64}:

1

2

5

1

[25]

0

CEK.

dot_v = dot(v, v)

31

[28]

0

CEK.

v'v

...

31

Рис. 5: Произведение векторов

2

29]

0

CEK.

A1 = [1 1; 1 -1]

...

2x2 Matrix{Int64}:

1 1

1 -1

30]

0

CEK.

Alu1 = lu(A1)

...

LU{Float64, Matrix{Float64}, Vector{Int64}}

L factor:

2x2 Matrix{Float64}:

1.0 0.0

1.0 1.0

U factor:

2x2 Matrix{Float64}:

1.0 1.0

0.0 -2.0

31]

0

CEK.

b1 = [2, 3]

slau1 = A1\b1

...

2-element Vector{Float64}:

2.5

-0.5

A2 = [1 1; 2 2]

b2 = [2 4]

...

1x2 Matrix{Int64}:

2 4

if (det(A2) != 0)

slau2 = A2\b2

print(slau2)

else

print("no way")

end

|

...

no way

9

```

A3 = [1 1; 2 2]
b3 = [2 5]

if (det(A3) != 0)
    slau2 = A3\b3
    print(slau3)
else
    print("no way")
end

... no way

```

```

A4 = [1 1; 2 2; 3 3]
b4 = [1, 2, 3]
slau4 = A4\b4

... 2-element Vector{Float64}:
 0.4999999999999999
 0.5

```

```

A5 = [1 1; 2 1; 1 -1]
b5 = [2, 1, 3]
slau5 = A5\b5

... 2-element Vector{Float64}:
 1.5000000000000004
-0.9999999999999997

```

Решим СЛАУ с тремя неизвестными

).

Операции с матрицами

1. Приведем матрицы к диагональному виду .
2. Вычислим .
3. Найдем собственные значения матрицы A . Создадим диагональную матрицу из собственных значений матрицы A . Создадим нижнедиагональную матрицу из матрицы A . Оценим эффективность выполняемых операций

Линейные модели экономики

Линейная модель может быть записана как СЛАУ

$$x - Ax = y,$$

где элементы матрицы A и столбца y – неотрицательные числа. По своему смыслу в экономике элементы матрицы A и столбцов x , y не могут быть отрицательными числами.

1. Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части y имеет только неотрицательные элементы x_i . Используя это определение, проверим, являются ли матрицы продуктивными
2. Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрицы $(E - A)^{-1}$ являются неотрицательными числами. Используя этот

```
[42] 0 сек.
A1 = [1 1 1; 1 -1 2]
b1 = [2, 3]
slau1 = A1\b1

3-element Vector{Float64}:
 0.7857142857142853
 0.07142857142857145
 1.1428571428571426

[45] 0 сек.
A2 = [1 1 1; 2 2 -3; 3 1 1]
b2 = [2, 4, 1]
slau2 = A2\b2

... 3-element Vector{Float64}:
 -0.5
  2.5
  0.0

[46] 0 сек.
A3 = [1 1 1; 1 1 2; 2 2 3]
b3 = [1, 0, 1]
if (det(A2) != 0)
    slau2 = A2\b2
    print(slau2)
else
    print("no way")
end

[-0.5, 2.5, 0.0]
```

Рис. 6: Системы линейных уравнений

```
47] A = [1 -2; 2 -1]
0
сек.

2x2 Matrix{Int64}:
 1 -2
 2 -1

51] AsymEig = eigen(A)
0
сек.

Eigen{ComplexF64, ComplexF64, Matrix{ComplexF64}, Vector{ComplexF64}}
values:
2-element Vector{ComplexF64}:
-1.1102230246251565e-16 - 1.7320508075688772im
-1.1102230246251565e-16 + 1.7320508075688772im
vectors:
2x2 Matrix{ComplexF64}:
-0.707107-0.0im -0.707107+0.0im
-0.353553-0.612372im -0.353553+0.612372im

53] display(diagm(AsymEig.values))
0
сек.

... 2x2 Matrix{ComplexF64}:
-1.11022e-16-1.73205im 0.0+0.0im
0.0+0.0im -1.11022e-16+1.73205im

54] B = [1 -2; -2 3]
0
сек.

Beig = eigen(B)
display(diagm(AsymEig.values))

... 2x2 Matrix{ComplexF64}:
-1.11022e-16-1.73205im 0.0+0.0im
0.0+0.0im -1.11022e-16+1.73205im
```

Рис. 7: Операции с матрицами

Вычисление

[57]
0 сек.

$A = \begin{bmatrix} 1 & -2 \\ -2 & 1 \end{bmatrix}$
 A^{10}

2x2 Matrix{Int64}:
29525 -29524
-29524 29525

[58]
22 сек.

$B = \begin{bmatrix} 5 & -2 \\ -2 & 5 \end{bmatrix}$
 $\text{sqrt}(B)$

2x2 Matrix{Float64}:
2.1889 -0.45685
-0.45685 2.1889

[60]
9 сек.

$C = \begin{bmatrix} 1 & -2 \\ -2 & 1 \end{bmatrix}$
 $C^{(1/3)}$

2x2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
0.971125+0.433013im -0.471125+0.433013im
-0.471125+0.433013im 0.971125+0.433013im

[61]
0 сек.

$D = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$
 $\text{sqrt}(D)$

... 2x2 Matrix{ComplexF64}:
0.568864+0.351578im 0.920442-0.217287im
0.920442-0.217287im 1.48931+0.134291im

Рис. 8: Операции с матрицами

```

[62] 0 A = [140 97 74 168 13; 97 106 89 131 36; 74 89 152 144 71; 168 131 144 54 142; 131 36 71 142 36]
    0
    Cbk.
    ... 5x5 Matrix{Int64}:
        140  97  74 168  13
         97 106  89 131  36
         74  89 152 144  71
        168 131 144  54 142
        131  36  71 142  36

[63] 0 Aeig = eigen(A)
    0
    Cbk.
    diagm(Aeig.values)

    5x5 Matrix{ComplexF64}:
    -133.689+0.0im  0.0+0.0im  ...  0.0+0.0im  0.0+0.0im
      0.0+0.0im 10.4119+0.0im  0.0+0.0im  0.0+0.0im
      0.0+0.0im  0.0+0.0im  0.0+0.0im  0.0+0.0im
      0.0+0.0im  0.0+0.0im 45.6515+7.34279im 0.0+0.0im
      0.0+0.0im  0.0+0.0im  0.0+0.0im 519.974+0.0im

[64] 0 LowerTriangular(A)
    0
    Cbk.
    ... 5x5 LowerTriangular{Int64, Matrix{Int64}}:
        140  .  .  .  .
         97 106  .  .  .
         74  89 152  .  .
        168 131 144  54  .
        131  36  71 142 36

[66] 4 @btime diagm(Aeig.values)
    4
    Cbk.
    ... 155.156 ns (2 allocations: 496 bytes)
    5x5 Matrix{ComplexF64}:
    -133.689+0.0im  0.0+0.0im  ...  0.0+0.0im  0.0+0.0im
      0.0+0.0im 10.4119+0.0im  0.0+0.0im  0.0+0.0im
      0.0+0.0im  0.0+0.0im  0.0+0.0im  0.0+0.0im
      0.0+0.0im  0.0+0.0im 45.6515+7.34279im 0.0+0.0im

```

Рис. 9: Операции с матрицами

критерий, проверим, являются ли матрицы продуктивными

Выводы

В процессе выполнения данной лабораторной работы я изучил возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.