

Лабораторная работа № 4. Линейная алгебра

Компьютерный практикум по статистическому анализу данных

Демидова Е. А.

24 ноября 2024

Российский университет дружбы народов, Москва, Россия

Информация

- Демидова Екатерина Алексеевна
- студентка группы НКНбд-01-21
- Российский университет дружбы народов
- <https://github.com/eademidova>



Введение

Цель работы

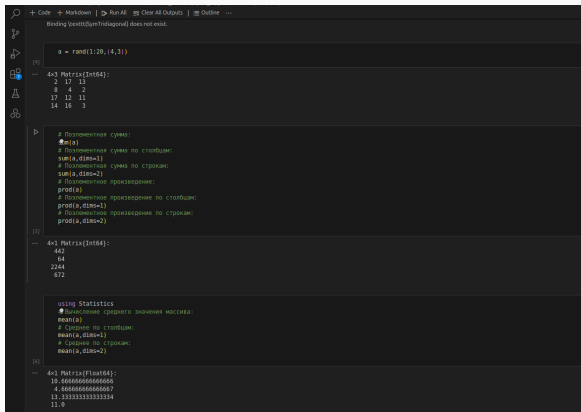
Основной целью работы является изучение возможностей специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Задачи

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

Выполнение лабораторной работы

Выполнение примеров



The screenshot shows a MATLAB script with the following content:

```
+ Code + Markdown | ▶ Run All | ⌵ Clear All Outputs | Outline ...
Binding Variable [Symbolic] does not exist.

a = rand(1:20, [4,3])

4x3 Matrix (int64):
2 17 13
8 4 2
17 12 11
14 16 3

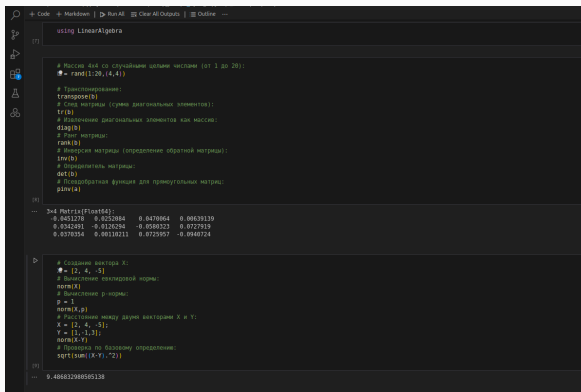
# Поэлементная сумма:
sum(a)
# Поэлементная сумма по столбцам:
sum(a, dim=1)
# Поэлементная сумма по строкам:
sum(a, dim=2)
# Поэлементное произведение:
prod(a)
# Поэлементное произведение по столбцам:
prod(a, dim=1)
# Поэлементное произведение по строкам:
prod(a, dim=2)

4x1 Matrix (int64):
442
64
2244
672

using Statistics
# Вычисление среднего значения массива:
mean(a)
# Среднее по столбцам:
mean(a, dim=1)
# Среднее по строкам:
mean(a, dim=2)

4x1 Matrix (float64):
10.466666666666666
4.466666666666667
13.333333333333334
11.0
```

Рис. 1: Поэлементные операции над многомерными массивами



```
using LinearAlgebra

[7]

# Массив 4x4 со случайными целыми числами (от 1 до 20):
a = 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)

...
3x4 Matrix{Float64}:
-0.6451278  0.6252084  0.6470064  0.00530119
 0.4242491 -0.9130294 -0.6500323  0.0727619
 0.6370354  0.40110211  0.0725957 -0.0940724

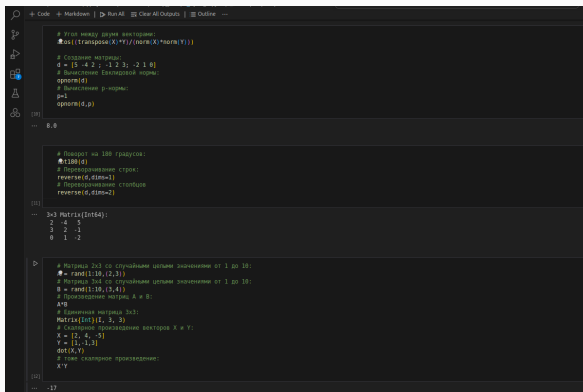
[8]

# Создание вектора X:
x = [2, 4, -5]
# Вычисление евклидовой нормы:
norm(x)
# Вычисление p-нормы:
p = 1
norm(x,p)
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1, -1, 3];
norm(x-y)
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))

[9]
...
0.406032980505138
```

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

Выполнение примеров



```
# Угол между двумя векторами:
cos((transpose(X)*Y)/(norm(X)*norm(Y)))

# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
# Вычисление Евклидовой нормы:
opnorm(d)
# Вычисление p-нормы:
p=1
opnorm(d,p)

... 6.0

# Поворот на 180 градусов:
rot180(d)
# Перекормирование строк:
reverse(d,dims=1)
# Перекормирование столбцов:
reverse(d,dims=2)

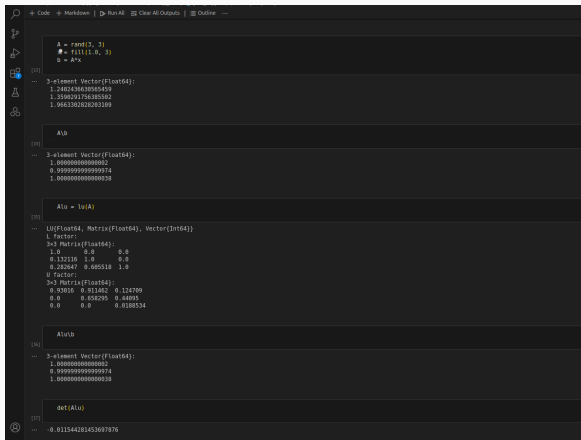
... 3x3 Matrix{Int64}:
 2 -4  5
 3  2 -1
 0  1 -2

# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand{Int}(1:10,(2,3))
# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand{Int}(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
```

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

Выполнение примеров



The screenshot shows a Jupyter Notebook interface with a dark theme. The code is written in C++ and demonstrates matrix operations. The output shows the results of these operations, including a 3x3 matrix A, its inverse A^-1, the LU decomposition of A, and the determinant of A.

```
A = rand(3, 3)
// = fill(1.0, 3)
B = A*x

...
3-element Vector{Float64}:
 1.240243648995459
 1.359020175638582
 1.966382828283109

A\b

...
3-element Vector{Float64}:
 1.000000000000002
 0.9999999999999974
 1.0000000000000028

A\b = lu(A)

...
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3x3 Matrix{Float64}:
 1.0      0.0      0.0
 0.132116 1.0      0.0
 0.282647 0.685518 1.0
U factor:
3x3 Matrix{Float64}:
 0.97016 0.911082 0.124789
 0.0     0.658295 0.44095
 0.0     0.0     0.0188534

A\b\b

...
3-element Vector{Float64}:
 1.000000000000002
 0.9999999999999974
 1.0000000000000028

det(A\b)

...
-0.011544281453697876
```

Рис. 4: Примеры. Матричное умножение, единичная матрица, скалярное произведение

Выполнение примеров

```
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...

In[ ]: In[AsymEig]*Asym

Out[ ]: 3x3 Matrix(Float64):
1.0 5.55112e-15 -6.38378e-15
-1.44329e-15 1.0 9.29612e-16
-1.27676e-15 -1.44329e-15 1.0

In[ ]: n = 1000
A = randn(n,n)
Asym = A + A'
issymmetric(Asym)

Out[ ]: true

In[ ]: Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
issymmetric(Asym_noisy)

Out[ ]: false

In[ ]: Asym_explicit = Symmetric(Asym_noisy)

Out[ ]: 1000x1000 Symmetric{Float64, Matrix{Float64}}:
-1.38705 2.36754 1.77734 ... -0.898384 -2.07477 0.0710199
2.36754 2.98918 -1.7326 -0.412254 0.927346 0.943464
1.77734 -1.7326 -1.92827 0.0117656 0.362245 -1.26035
0.0715804 0.362184 0.917652 -1.52021 -1.08014 0.868022
1.09047 -1.25572 0.326191 2.31923 0.051896 0.364824
-1.10098 -1.06752 -1.06744 -0.438759 -1.26461 0.490111
0.063176 1.13947 -0.143847 -1.01867 -0.159432 1.74265
0.0172167 -0.623325 0.8769264 1.15688 0.0152348 -2.93586
-0.0612332 0.119189 0.409365 1.71791 1.48875 2.19677
0.52526 -0.918453 -3.16274 -0.838246 -0.0907788 1.4786
-0.630662 1.11763 -2.21205 -1.00968 1.4799 -2.22543
1.48422 -0.310182 -1.34838 -0.00426986 0.981705 1.63343
-0.307627 -2.31148 -1.65298 0.52584 2.93294 -2.22985
...
-0.381884 0.518722 2.2419 1.6832 0.351898 -0.0945664
-0.524056 1.29909 -0.744196 -2.41559 1.05148 2.46367
0.593903 -1.07409 -0.453119 -0.715179 0.412781 -0.7908
0.304591 -1.10829 0.119616 1.06255 -0.742149 0.652269
2.413 2.40037 2.41776 0.000677 0.570359 -1.01182
-1.5141 -0.0180018 0.320218 1.97246 -0.310996 -1.685
-1.11468 -0.248895 1.29345 -2.2595 0.346618 0.456886
2.37688 -1.79427 -0.39733 -2.86656 0.855469 0.61363
-0.676839 0.0150166 -0.887244 1.04802 1.47958 -1.01649
```

Рис. 5: Примеры. Факторизация. Специальные матричные структуры

Выполнение примеров

[illegible]

Рис. 6: Примеры. Факторизация. Специальные матричные структуры

```
[0] 1//2
... 1//2

Arational = Matrix(Rational{BigInt})(rand(1:10, 3, 3))/10
# = fill{1, 3}
@ = Arational*x

[0] 3-element Vector{Rational{BigInt}}:
 17//10
 12//5
 4//5

Arational\b

[0] 3-element Vector{Rational{BigInt}}:
 1//1
 1//1
 1//1

lu(Arational)

[0] LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3x3 Matrix{Rational{BigInt}}:
 1//1  0//1  0//1
 2//2  1//1  0//1
 1//8 -13//21 1//1
U factor:
3x3 Matrix{Rational{BigInt}}:
 9//20 1//1 1//2
 0//1 -7//15 17//20
 0//1 0//1 25//42
```

Рис. 7: Примеры. Общая линейная алгебра

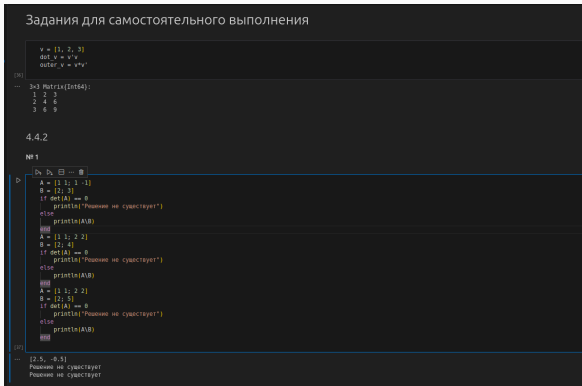


Рис. 8: Задание 1, 2

```

A = [1 1; 2 2; 3 3]
B = [1; 2;3]
println(A*B)

A = [1 1; 2 1; 1 -1]
B = [2; 1; 3]
println(A*B)

A = [1 1; 2 1; 3 2]
B = [2; 1; 3]
println(A*B)

```

```

[0.5, 0.5]
[1.5000000000000004, -0.0000000000000001]
[-0.0000000000000004, 2.0000000000000009]

```

№2

```

A = [1 1 1; 1 -1 -2]
B = [2; 3]
println(A*B)
A = [1 1 1; 2 2 -3; 3 1 1]
B = [2; 4;1]
println(A*B)
A = [1 1 1; 1 1 2; 2 2 3]
B = [1; 0;1]
if det(A) == 0
    println("Повторно не вычислять")
else
    println(A*B)
end
A = [1 1 1; 3 1 2; 2 2 3]
B = [1; 0;0]
if det(A) == 0
    println("Повторно не вычислять")
else
    println(A*B)
end
end

```

```

[2.14285714285715, 0.35714285714285715, -0.5714285714285711]
[-0.5, 2.5, 0.0]
Повторно не вычислять
Повторно не вычислять

```

Рис. 9: Задание 2

Выполнение заданий для самостоятельной работы

```
№ 1

A = [1 -2; -2 1]
Aeig = eig(A)
A_diag = inv(Aeig.vectors)*A*Aeig.vectors
A_diag = diag(Aeig.values)
display(A_diag)
A = [1 -2; -2 3]
Aeig = eig(A)
A_diag = diag(Aeig.values)
display(A_diag)
A = [1 -2 0; -2 1 2; 0 2 0]
Aeig = eig(A)
A_diag = diag(Aeig.values)
display(A_diag)

(4)
--- 2x2 Matrix(Float64):
 1.0  0.0
 0.0  3.0

--- 2x2 Matrix(Float64):
-0.236068  0.0
 0.0      4.23607

--- 3x3 Matrix(Float64):
-2.14134  0.0  0.0
 0.0     0.515138  0.0
 0.0     0.0  3.6262

№ 2

D =
A = [1 -2; -2 1]
display(A^10)
A = [5 -2; -2 5]
display(sqrt(A))
A = [1 -2; -2 1]
display(A^(1/3))
A = [1 2; 2 5]
display(sqrt(A))

(4)
--- 2x2 Matrix{Int64}:
29525  -29524
-29524  29525

--- 2x2 Matrix(Float64):
2.1889  -0.45685
-0.45685  2.1889

--- 2x2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
0.971125+0.433011im -0.471125+0.433011im
-0.471125+0.433011im 0.971125+0.433011im
```

Рис. 10: Задание 3

Выполнение заданий для самостоятельной работы

```
№ 3

A = [140 97 74 168 131; 97 106 89 131 36; 74 89 152 144 71; 168 131 144 54 142; 131 36 71 142 36]

In [42]:
5x5 Matrix{Int64}:
 140  97  74 168 131
  97 106  89 131  36
  74  89 152 144  71
168 131 144  54 142
131  36  71 142  36

Aeig = eigen(A)

In [43]:
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
5-element Vector{Float64}:
-128.49322764802147
-55.88778455105702
 42.752167279318826
 87.18111477514487
542.4877301486138
vectors:
5x5 Matrix{Float64}:
-0.347575  0.647178  0.818882  0.548993 -0.507907
-0.250795 -0.173868  0.834628  0.238894 -0.387253
-0.185537  0.239762 -0.422161 -0.731925 -0.449631
 0.819704 -0.247506 -0.8273194  0.8386467 -0.314526
-0.433883 -0.657619 -0.352577  0.322668 -0.364928

D = diagm(Aeig.values)

In [44]:
5x5 Matrix{Float64}:
-128.493  0.0  0.0  0.0  0.0
 0.0 -55.8878  0.0  0.0  0.0
 0.0  0.0  42.7522  0.0  0.0
 0.0  0.0  0.0  87.1811  0.0
 0.0  0.0  0.0  0.0 542.488

LowerTriangular(A)

In [45]:
5x5 LowerTriangular{Int64, Matrix{Int64}}:
 140  .  .  .  .
  97 106  .  .  .
  74  89 152  .  .
168 131 144  54  .
```

Рис. 11: Задания 3

Выполнение заданий для самостоятельной работы

```
@time diag(Aeig.values)
[14]
... 109.921 ns (1 allocation: 256 bytes)
... 5x5 Matrix{Float64}:
...  -128.492  0.0  0.0  0.0  0.0
...    0.0  -55.8878  0.0  0.0  0.0
...    0.0  0.0  42.7522  0.0  0.0
...    0.0  0.0  0.0  87.1611  0.0
...    0.0  0.0  0.0  0.0  542.468

@time LowerTriangular(A)
[10]
... 59.206 ns (1 allocation: 16 bytes)
... 5x5 LowerTriangular{Int64, Matrix{Int64}}:
...  140  '  '  '  '
...  97 106  '  '  '
...  74 89 152  '  '
... 100 131 144  54  '
... 131 36  71 142 36
```

Рис. 12: Задание 3

Выполнение заданий для самостоятельной работы

```
№ 1

$$x - Ax = y$$


A = [1 2; 3 4]
B = (1/2)*[1 2; 3 4]
C = (1/10)*[1 2; 3 4]

(18)
--- 2x2 Matrix(Float64):
0.1 0.2
0.3 0.4

№ 2

A = [1 2; 3 1]
B = (1/2)*[1 2; 3 1]
C = (1/10)*[1 2; 3 1]
E = Matrix{I,2,2}

(19)
--- 2x2 Matrix{Bool}:
1 0
0 1

Inv(E-A) для произведения
(20)
--- 2x2 Matrix(Float64):
0.5 -0.333333
-0.5 0.0

D= Inv(E-B) для произведения
(20)
--- 2x2 Matrix(Float64):
0.5 -0.5
-0.75 -0.25

Inv(E-C) для произведения
(20)
--- 2x2 Matrix(Float64):
1.25 0.416667
0.625 1.875
```

Рис. 13: Задания 4

Выполнение заданий для самостоятельной работы

```
№3

D = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]

[99]
--- 3x3 Matrix{Float64}:
 0.1 0.2 0.3
 0.0 0.1 0.2
 0.0 0.1 0.3

abs.(eigen(A).values).<1 #не продуктивная

[94]
--- 2-element BitVector:
 1
 0

abs.(eigen(B).values).<1 #не продуктивная

[95]
--- 2-element BitVector:
 1
 0

abs.(eigen(C).values).<1 #продуктивная

[96]
--- 2-element BitVector:
 1
 1

abs.(eigen(D).values).<1 #продуктивная

[97]
--- 3-element BitVector:
 1
 1
 1

D =
 0.1 0.2 0.3
 0.0 0.1 0.2
 0.0 0.1 0.3

[98]
--- search: KeyError eltype keytype supertype code_typed supertypes @code_typed
findkey::Base.Key{...}
```

Рис. 14: Задание 4

Выводы

В результате выполнения работы освоили применение циклов функций и сторонних для Julia пакетов для решения задач линейной алгебры и работы с матрицами.

1. JuliaLang [Электронный ресурс]. 2024 JuliaLang.org contributors. URL: <https://julialang.org/> (дата обращения: 11.10.2024).
2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.org contributors. URL: <https://docs.julialang.org/en/v1/> (дата обращения: 11.10.2024).