Презентация по лабораторной работе №4

Дисциплина: Компьютерный практикум по статистическому анализу данных

Лобанова П.И.

20 октября 2025

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Лобанова Полина Иннокентьевна
- Учащаяся на направлении “Фундаментальная информатика и информационные технологии”
- Студентка группы НФИбд-02-22
- polla-2004@mail.ru

Цель

Цель

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

Задание

Задание

1. Используя Jupyter Lab, повторите примеры из раздела 4.2.
2. Выполните задания для самостоятельной работы (раздел 4.4)

Выполнение

Выполнение

```
[4]: #Поэлементные операции над многомерными массивами
     # Массив 4x3 со случайными целыми числами (от 1 до 20):
     a = rand(1:20, (4,3))

[4]: 4×3 Matrix{Int64}:
       7  9   9
       5  6  17
      13  2   9
       3  2   6

[5]: # Поэлементная сумма:
     sum(a)

[5]: 88

[6]: # Поэлементная сумма по столбцам:
     sum(a,dims=1)

[6]: 1×3 Matrix{Int64}:
      28  19  41

[7]: # Поэлементная сумма по строкам:
     sum(a,dims=2)

[7]: 4×1 Matrix{Int64}:
      25
      28
      24
      11
```

Рис. 1: *Примеры с поэлементными операциями над многомерными массивами*

```
[15]: #Транспонирование, след, ранг, определитель и инверсия матрицы
      # Подключение пакета LinearAlgebra:
      import Pkg
      Pkg.add("LinearAlgebra")
      using LinearAlgebra

   Resolving package versions...
    Updating `C:\Users\Полина\.julia\environments\v1.11\Project.toml`
  [37e2e46d] + LinearAlgebra v1.11.0
  No Changes to `C:\Users\Полина\.julia\environments\v1.11\Manifest.toml`

[16]: # Массив 4x4 со случайными целыми числами (от 1 до 20):
      b = rand(1:20,(4,4))

[16]: 4×4 Matrix{Int64}:
        2  14   2  20
       12   3  15   9
        5  11  12  12
       19   5  11  15

[17]: # Транспонирование:
      transpose(b)

[17]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
        2  12   5  19
       14   3  11   5
        2  15  12  11
       20   9  12  15

[18]: b'

[18]: 4×4 adjoint(::Matrix{Int64}) with eltype Int64:
        2  12   5  19
       14   3  11   5
        2  15  12  11
       20   9  12  15
```

Рис. 2: *Примеры с транспонированием, следом, рангом, определителем и инверсией матрицы*

```
[25]: #Вычисление нормы векторов и матриц, повороты, вращения
      # Создание вектора X:
      X = [2, 4, -5]

[25]: 3-element Vector{Int64}:
        2
        4
       -5

[26]: # Вычисление евклидовой нормы:
      norm(X)

[26]: 6.708203932499369

[27]: # Вычисление p-нормы:
      p = 1
      norm(X,p)

[27]: 11.0

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

[28]: 9.486832980505138

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

[29]: 9.486832980505138
```

Рис. 3: *Примеры с вычислением нормы векторов и матриц, поворотами, вращением*

```
[38]: #Матричное умножение, единичная матрица, скалярное произведение
      # Матрица 2x3 со случайными целыми значениями от 1 до 10:
      A = rand(1:10,(2,3))

[38]: 2×3 Matrix{Int64}:
       3  2  9
       4  9  2

[39]: # Матрица 3x4 со случайными целыми значениями от 1 до 10:
      B = rand(1:10,(3,4))

[39]: 3×4 Matrix{Int64}:
       9  1  2  1
       5  1  3  7
       6  1  9  3

[40]: # Произведение матриц A и B:
      A*B

[40]: 2×4 Matrix{Int64}:
       91  14  93  44
       93  15  53  73

[41]: # Единичная матрица 3x3:
      Matrix{Int}(I, 3, 3)

[41]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1
```

Рис. 4: *Примеры с матричным умножением, единичной матрицей, скалярным произведением*

```
[75]: #Факторизация. Специальные матричные структуры
      # Задаём квадратную матрицу 3x3 со случайными значениями:
      A = rand(3, 3)

[75]: 3×3 Matrix{Float64}:
       0.551531  0.462932  0.529392
       0.942454  0.777511  0.239333
       0.702618  0.877552  0.15966

[76]: # Задаём единичный вектор:
      x = fill(1.0, 3)

[76]: 3-element Vector{Float64}:
       1.0
       1.0
       1.0

[77]: # Задаём вектор b:
      b = A*x

[77]: 3-element Vector{Float64}:
       1.543855596934271
       1.9592989439076152
       1.739830453013854

[78]: # Решение исходного уравнения получаем с помощью функции \
      # (убеждаемся, что x - единичный вектор):
      A\b

[78]: 3-element Vector{Float64}:
       1.0000000000000009
       0.9999999999999992
       0.9999999999999993
```

Рис. 5: *Примеры с факторизацией*

```
[111]: #Общая линейная алгебра
       # Матрица с рациональными элементами:
       Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10

[111]: 3×3 Matrix{Rational{BigInt}}:
        4//5   9//10  2//5
        3//5   1//10  7//10
        7//10  3//5   9//10

[112]: # Единичный вектор:
       x = fill(1, 3)

[112]: 3-element Vector{Int64}:
        1
        1
        1

[113]: # Задаём вектор b:
       b = Arational*x

[113]: 3-element Vector{Rational{BigInt}}:
        21//10
         7//5
        11//5

[114]: # Решение исходного уравнения получаем с помощью функции \
       # (убеждаемся, что x - единичный вектор):
       Arational\b

[114]: 3-element Vector{Rational{BigInt}}:
        1
        1
        1
```

Рис. 6: *Примеры с общей линейной алгеброй*

•[119]:

```
# 1
v = [25, 86, 36]
dot_v = v'v
```

[119]:

```
9317
```

[120]:

```
outer_v = v*v'
```

[120]:

```
3×3 Matrix{Int64}:
  625  2150   900
 2150  7396  3096
  900  3096  1296
```

Рис. 7: *Произведение векторов*

```
#2.1
A1 = [1 1; 1 -1]
b1 = [2, 3]
x1 = A1\b1
```

[190]:

```
2-element Vector{Float64}:
  2.5
 -0.5
```

[191]:

```
A2 = [1 1; 2 2]
b2 = [2, 4]
if det(A2) == 0
    print("Нет решений")
else
    x2 = A2\b2
    print(x2)
end
```

```
Нет решений
```

[192]:

```
A3 = [1 1; 2 2]
b3 = [2, 5]
if det(A3) == 0
    print("Нет решений")
else
    x3 = A3\b3
    print(x3)
end
```

```
Нет решений
```

Рис. 8: *Системы линейных уравнений*

[193]:

```
A4 = [1 1; 2 2; 3 3]
b4 = [1, 2, 3]
x4 = A4\b4
```

[193]:

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

[194]:

```
A5 = [1 1; 2 1; 1 -1]
b5 = [2, 1, 3]
x5 = A5\b5
```

[194]:

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

[195]:

```
A6 = [1 1; 2 1; 3 2]
b6 = [2, 1, 3]
x6 = A6\b6
```

[195]:

```
2-element Vector{Float64}:
 -0.9999999999999989
  2.9999999999999982
```

Рис. 9: *Системы линейных уравнений*

```
#2.2
A7 = [1 1 1; 1 -1 -2]
b7 = [2, 3]
x7 = A7\b7
```

[196]:

```
3-element Vector{Float64}:
  2.2142857142857144
  0.35714285714285704
 -0.5714285714285712
```

[197]:

```
A8 = [1 1 1; 2 2 -3; 3 1 1]
b8 = [2, 4, 1]
x8 = A8\b8
```

[197]:

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

[198]:

```
A9 = [1 1 1; 1 1 2; 2 2 3]
b9 = [1, 0, 1]
if det(A9) == 0
    print("Нет решений")
else
    x9 = A9\b9
    print(x9)
end
```

```
Нет решений
```

Рис. 10: *Системы линейных уравнений*

[199]:

```
A10 = [1 1 1; 1 1 2; 2 2 3]
b10 = [1, 0, 0]
if det(A10) == 0
    print("Нет решений")
else
    x10 = A10\b10
    print(x10)
end
```

```
Нет решений
```

Рис. 11: *Системы линейных уравнений*

```
#3.1
a = [1 -2; -2 1]
```

[200]:

```
2×2 Matrix{Int64}:
  1  -2
 -2   1
```

[201]:

```
eigs = eigen(a)
```

[201]:

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
2-element Vector{Float64}:
 -1.0
  3.0
vectors:
2×2 Matrix{Float64}:
 -0.707107  -0.707107
 -0.707107   0.707107
```

[202]:

```
a_diag = diagm(eigs.values)
```

[202]:

```
2×2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
```

Рис. 12: *Системы линейных уравнений*

```
b = [1 -2; -2 3]
eigs2 = eigen(b)
b_diag = diagm(eigs2.values)
```

[203]:

```
2×2 Matrix{Float64}:
 -0.236068  0.0
  0.0       4.23607
```

[204]:

```
c = [1 -2 0; -2 1 2; 0 2 0]
eigs3 = eigen(c)
c_diag = diagm(eigs3.values)
```

[204]:

```
3×3 Matrix{Float64}:
 -2.14134  0.0       0.0
  0.0      0.515138  0.0
  0.0      0.0       3.6262
```

Рис. 13: *Системы линейных уравнений*

```
#3.2
a = [1 -2; -2 1]
a^10
```

[205]:

```
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
```

[206]:

```
b = [5 -2; -2 5]
sqrt(b)
```

[206]:

```
2×2 Matrix{Float64}:
  2.1889    -0.45685
 -0.45685    2.1889
```

[207]:

```
c = [1 -2; -2 1]
cbrt(c)
```

[207]:

```
2×2 Matrix{Float64}:
  0.221125  -1.22112
 -1.22112    0.221125
```

[208]:

```
d = [1 2; 2 3]
sqrt(d)
```

[208]:

```
2×2 Matrix{ComplexF64}:
 0.568864+0.351578im  0.920442-0.217287im
 0.920442-0.217287im   1.48931+0.134291im
```

Рис. 14: *Операции с матрицами*

[209]:

```
#3.3
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]
eigs = eigen(A)
A_diag = diagm(eigs.values)
```

[209]:

```
5×5 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      87.1611    0.0
    0.0      0.0       0.0       0.0     542.468
```

[210]:

```
LowerTriangular(A)
```

[210]:

```
5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рис. 15: *Операции с матрицами*

[211]:

```
@btime diagm(eigs.values)
```

```
105.668 ns (2 allocations: 272 bytes)
```

[211]:

```
5×5 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     87.1611    0.0
    0.0      0.0       0.0      0.0     542.468
```

[212]:

```
@btime LowerTriangular(A)
```

```
183.310 ns (1 allocation: 16 bytes)
```

[212]:

```
5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рис. 16: *Операции с матрицами*

[213]:

```
#4.1
E = Matrix{Int}(I, 2, 2)
A1 = [1 2; 3 4]
y1 = [1, 1]
x1 = y1'*((E-A1)^-1)
#матрица не продуктивная
```

[213]:

```
1×2 adjoint(::Vector{Float64}) with eltype Float64:
 0.0  -0.333333
```

[214]:

```
A2 = 1/2*A1
x2 = y1'*((E-A2)^-1)
#матрица не продуктивная
```

[214]:

```
1×2 adjoint(::Vector{Float64}) with eltype Float64:
 -0.25  -0.75
```

[215]:

```
A3 = 1/10*A1
x3 = y1'*((E-A3)^(-1))
#матрица не продуктивная
```

[215]:

```
1×2 adjoint(::Vector{Float64}) with eltype Float64:
 1.875  2.29167
```

Рис. 17: *Линейные модели экономики*

[216]:

```
#4.2
A4 = [1 2; 3 1]
x4 = (E-A4)^(-1)
#матрица не продуктивная
```

[216]:

```
2×2 Matrix{Float64}:
 -0.0  -0.333333
 -0.5   0.0
```

[217]:

```
A5 = 1/2*A4
x5 = (E-A5)^(-1)
#матрица не продуктивная
```

[217]:

```
2×2 Matrix{Float64}:
 -0.4  -0.8
 -1.2  -0.4
```

[218]:

```
A6 = 1/10*A4
x6 = (E-A6)^(-1)
#матрица продуктивная
```

[218]:

```
2×2 Matrix{Float64}:
 1.2  0.266667
 0.4  1.2
```

Рис. 18: *Линейные модели экономики*

[219]:

```
#4.3
A7 = [1 2; 3 1]
eigs = eigen(A7)
abs.(eigs.values)
#матрица не продуктивная
```

[219]:

```
2-element Vector{Float64}:
 1.4494897427831779
 3.4494897427831783
```

[220]:

```
A8 = 1/2*A7
eigs = eigen(A8)
abs.(eigs.values)
#матрица не продуктивная
```

[220]:

```
2-element Vector{Float64}:
 0.7247448713915892
 1.724744871391589
```

Рис. 19: *Линейные модели экономики*

[221]:

```
A9 = 1/10*A7
eigs = eigen(A9)
abs.(eigs.values)
#матрица продуктивная
```

[221]:

```
2-element Vector{Float64}:
 0.14494897427831785
 0.34494897427831783
```

[222]:

```
A10 = [0.1 0.2 0.2; 0 0.1 0.2; 0 0.1 0.3]
eigs = eigen(A10)
abs.(eigs.values)
#матрица продуктивная
```

[222]:

```
3-element Vector{Float64}:
 0.02679491924311228
 0.1
 0.37320508075688774
```

Рис. 20: *Линейные модели экономики*

Вывод

Вывод

Я изучила возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.