# Лабораторная работа №4

## Линейная алгебра

Легиньких Г.А.

Российский университет дружбы народов, Москва, Россия

# Информация

# Докладчик


- Легиньких Галина Андреевна
- НФИбд-02-21
- Российский университет дружбы народов
- 1032216447@pfur.ru
- https://github.com/galeginkikh

# Основная информация

# Цель работы

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

# Задание

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

# Выполнение

# Поэлементные операции над многомерными массивами

Примеры поэлементной суммы.

Так же повторила примеры поэлементного произведения.

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics.

```
[5]: # Массив 4x3 со случайными целыми числами (от 1 до 20):
     a = rand(1:20,(4,3))

[5]: 4x3 Matrix{Int64}:
      12  20  12
       9  16  14
      16   3   3
       7   8  12

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

[7]: 132

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

[9]: 1x3 Matrix{Int64}:
      44  47  41

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

[11]: 4x1 Matrix{Int64}:
       44
       39
       22
       27
```

Для выполнения таких операций над матрицами, как транспонирование, диагонализация, определение следа, ранга, определителя матрицы и т.п. можно воспользоваться библиотекой (пакетом) LinearAlgebra.

```
[39]: # Ранг матрицы:
      rank(b)

[39]: 4

[41]: # Инверсия матрицы (определение обратной матрицы):
      inv(b)

[41]: 4×4 Matrix{Float64}:
       -0.058881    -0.0152455   0.074322   -0.0185683
        0.00024432   0.0647935  -0.0658686   0.0789152
        0.0311507   -0.0388224   0.0017591   0.0616907
        0.0400684    0.0261422  -0.0024432  -0.0579037

[43]: # Определитель матрицы:
      det(b)

[43]: 40930.0

[45]: # Псевдобратная функция для прямоугольных матриц:
      pinv(a)

[45]: 3×4 Matrix{Float64}:
        0.000878478  -0.0121681    0.0705788  -0.00432708
        0.0847905     0.00185607  -0.0298596  -0.079491
       -0.0742372     0.0361774   -0.0170032   0.119614
```

**Рис. 2:** Ранг, инверсия, определитель, псевдобратная

Вычисление евклидовой нормы и p-нормы для векторов.

```
[47]: # Создание вектора X:
      X = [2, 4, -5]

[47]: 3-element Vector{Int64}:
        2
        4
       -5

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

[49]: 6.708203932499369

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

[51]: 11.0
```

**Рис. 3:** Нормы для векторов

Вычисление нормы для двумерной матрицы.

```
[59]: # Создание матрицы:
      d = [5 -4 2 ; -1 2 3; -2 1 0]

[59]: 3×3 Matrix{Int64}:
        5  -4  2
       -1   2  3
       -2   1  0

[61]: # Вычисление Евклидовой нормы:
      opnorm(d)

[61]: 7.147682841795258

[63]: # Вычисление p-нормы:
      p=1
      opnorm(d,p)

[63]: 8.0
```

**Рис. 4:** Нормы для матрицы

# Матричное умножение, единичная матрица, скалярное произведение

Произведение матриц A и B:

```
[71]: # Матрица 2х3 со случайными целыми значениями от 1 до 10:
      A = rand(1:10,(2,3))

[71]: 2×3 Matrix{Int64}:
       8  6  3
       3  4  3

[73]: # Матрица 3х4 со случайными целыми значениями от 1 до 10:
      B = rand(1:10,(3,4))

[73]: 3×4 Matrix{Int64}:
       9  6  3  10
       8  1  1   7
       7  6  8   3

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

[75]: 2×4 Matrix{Int64}:
       141  72  54  131
        80  40  37   67

[77]: # Единичная матрица 3х3:
      Matrix{Int}(I, 3, 3)

[77]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1
```

**Рис. 5:** Произведение

# Матричное умножение, единичная матрица, скалярное произведение

Скалярное произведение векторов X и Y:

```
[79]: # Скалярное произведение векторов X и Y:
      X = [2, 4, -5]
      Y = [1,-1,3]
      dot(X,Y)

[79]: -17

[81]: # тоже скалярное произведение:
      X'Y

[81]: -17
```

**Рис. 6:** Скалярное произведение

Решение систем линейный алгебраических уравнений ?? = ?:

```
[83]: # Задаём квадратную матрицу 3x3 со случайными значениями:
      A = rand(3, 3)

[83]: 3×3 Matrix{Float64}:
       0.564215   0.269493  0.101435
       0.195662   0.434349  0.294503
       0.0847998  0.704377  0.1274

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

[85]: 3-element Vector{Float64}:
       1.0
       1.0
       1.0

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

[87]: 3-element Vector{Float64}:
       0.935143164460376
       0.9245150405663181
       0.9165767302780387

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

[89]: 3-element Vector{Float64}:
       1.0
       1.0
       1.0
```

**Рис. 7:** $Ax = b$

Julia позволяет вычислять LU-факторизацию и определяет составной тип факторизации для его хранения.

Julia позволяет вычислять QR-факторизацию и определяет составной тип факторизации для его хранения.

Исходная система уравнений ?? = ? может быть решена или с использованием исходной матрицы, или с использованием объекта факторизации:

```
[101]: # Решение СЛАУ через матрицу A:
       A\b

[101]: 3-element Vector{Float64}:
        1.0
        1.0
        1.0

[103]: # Решение СЛАУ через объект факторизации:
       Alu\b

[103]: 3-element Vector{Float64}:
        1.0
        1.0
        1.0

[105]: # Детерминант матрицы A:
       det(A)

[105]: -0.07556391453190962

[107]: # Детерминант матрицы A через объект факторизации:
       det(Alu)

[107]: -0.07556391453190962
```

**Рис. 8:** СЛАУ

В следующем примере показано, как можно решить систему линейных уравнений с рациональными элементами без преобразования в типы элементов с плавающей запятой (для избежания проблемы с переполнением используем BigInt).

# Задание для самостоятельной работы

Произведение векторов. Задание 1.1 - 1.2

```
[171]: # Задание 1.1
       v = [1, 2, 3]
       dot_v = dot(v,v)

[171]: 14

[175]: # Задание 1.2
       outer_v = v * v'

[175]: 3×3 Matrix{Int64}:
        1  2  3
        2  4  6
        3  6  9
```

**Рис. 9:** Задание 1.1 - 1.2

Системы линейных уравнений. Допустим пример Задание 2.1.f

```
[211]: # Задание 2.1.f
       A = [1 1; 2 1; 3 2]
       b = [2; 1; 3]
       x_y = A\b

[211]: 2-element Vector{Float64}:
        -0.9999999999999994
         2.999999999999999
```

**Рис. 10:** Задание 2.1.f

## Операции с матрицами. Задание 3.1.a - 3.1.c

```
[255]: # Задание 3.1.a
       A = [1 -2; -2 1]
       eig_A = eigen(A)
       D =diagm(eig_A.values)

[255]: 2×2 Matrix{Float64}:
        -1.0  0.0
         0.0  3.0

[257]: # Задание 3.1.b
       A = [1 -2; -2 3]
       eig_A = eigen(A)
       D =diagm(eig_A.values)

[257]: 2×2 Matrix{Float64}:
        -0.236068  0.0
         0.0       4.23607

[259]: # Задание 3.1.c
       A = [1 -2 0; -2 1 2; 0 2 0]
       eig_A = eigen(A)
       D =diagm(eig_A.values)

[259]: 3×3 Matrix{Float64}:
        -2.14134  0.0       0.0
         0.0      0.515138  0.0
         0.0      0.0       3.6262
```

**Рис. 11:** Задание 3.1.a - 3.1.c

## Операции с матрицами. Задание 3.2.a - 3.2.d

```
[261]: # Задание 3.2.a
       A = [1 -2; -2 1]
       A_pow_10 = A^10

[261]: 2×2 Matrix{Int64}:
         29525  -29524
        -29524   29525

[265]: # Задание 3.2.b
       A = [5 -2; -2 5]
       A_sqrt = sqrt(A)

[265]: 2×2 Matrix{Float64}:
         2.1889   -0.45685
        -0.45685   2.1889

[267]: # Задание 3.2.c
       A = [1 -2; -2 1]
       A_pow = A^(1/3)

[267]: 2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
         0.971125+0.433013im  -0.471125+0.433013im
        -0.471125+0.433013im   0.971125+0.433013im

[269]: # Задание 3.2.d
       A = [1 2; 2 3]
       A_sqrt = sqrt(A)

[269]: 2×2 Matrix{ComplexF64}:
        0.568864+0.351578im  0.920442-0.217287im
        0.920442-0.217287im   1.48931+0.134291im
```

**Рис. 12:** Задание 3.2.a - 3.2.d

## Операции с матрицами. Задание 3.3

```
[271]: # Задание 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
       ]
        # Собственные значения
       eig_A = eigen(A)
       eigevalues = eig_A.values

[271]: 5-element Vector{Float64}:
        -128.49322764802136
         -55.88778455305702
          42.75216727931888
          87.16111477514494
         542.4677301466138

[273]: D = diagm(eigevalues)

[273]: 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

[275]: L = tril(A)

[275]: 5×5 Matrix{Int64}:
        140    0    0    0   0
         97  106    0    0   0
         74   89  152    0   0
        168  131  144   54   0
        131   36   71  142  36

[279]: @btime eigmax(A)

         2.267 μs (11 allocations: 2.61 KiB)

[279]: 542.4677301466143
```

Линейные модели экономики. Приведу один пример, остальные в отчете. Задание 4.2

```
[291]: # Задание 4.2
       function productive_2(A)
           E = I
           B = E - A
           if det(B) <= 0
               return false
           end
           inv_B = inv(B)
           return all(inv_B.>=0)
       end

       # a
       A = [1 2; 3 1]
       productive_2(A)

[291]: false

[293]: # b
       A = 0.5* [1 2; 3 1]
       productive_2(A)

[293]: false

[295]: # c
       A = 0.1 * [1 2; 3 1]
       productive_2(A)

[295]: true
```

**Рис. 14:** Задание 4.2

# Вывод

# Вывод

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