

# **Лабораторная работа 4**

**Линейная алгебра**

Ланцова Яна Игоревна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Задание 1 . . . . .	10
3.2	Задание 2 . . . . .	10
3.3	Задание 3 . . . . .	12
3.4	Задание 4 . . . . .	13
<b>4</b>	<b>Выводы</b>	<b>16</b>

# Список иллюстраций

3.1	Поэлементные операции над многомерными массивами . . . . .	7
3.2	Поэлементные операции над многомерными массивами . . . . .	7
3.3	Транспонирование, след, ранг, определители инверсия матрицы	8
3.4	Вычисление нормы векторов и матриц . . . . .	8
3.5	Факторизация. Специальные матричные структуры . . . . .	9
3.6	Факторизация. Специальные матричные структуры . . . . .	9
3.7	Общая линейная алгебра . . . . .	10
3.8	Произведение векторов . . . . .	10
3.9	Системы линейных уравнений . . . . .	11
3.10	Системы линейных уравнений . . . . .	11
3.11	Системы линейных уравнений . . . . .	11
3.12	Операции с матрицами . . . . .	12
3.13	Операции с матрицами . . . . .	12
3.14	Операции с матрицами . . . . .	13
3.15	Оценка эффективности . . . . .	13
3.16	Линейные модели экономики . . . . .	14
3.17	Линейные модели экономики . . . . .	14
3.18	Линейные модели экономики . . . . .	15

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Используя JupyterLab, повторим примеры.
2. Выполним задания для самостоятельной работы.

### 3 Выполнение лабораторной работы

Выполним примеры из раздела про поэлементные операции над многомерными массивами (рис. 3.1).

Поэлементные операции над многомерными массивами

```
[4]: a = rand(1:20, (4,3));  
sum(a)  
[4]: 134  
[5]: # Поэлементная сумма по строкам:  
sum(a,dims=2)  
[5]: 4×1 Matrix{Int64}:  
41  
32  
28  
33  
[6]: # Поэлементное произведение:  
prod(a)  
[6]: 181547827200  
[8]: prod(a,dims=1)  
[8]: 1×3 Matrix{Int64}:  
1216 7680 19440
```

Рис. 3.1: Поэлементные операции над многомерными массивами

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics (рис. 3.2).

```
[7]: import Pkg  
Pkg.add("Statistics")  
using Statistics  
  
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'  
[9]: # Вычисление среднего значения массива:  
mean(a)  
[9]: 11.166666666666666  
[10]: # Среднее по столбцам:  
mean(a,dims=1)  
[10]: 1×3 Matrix{Float64}:  
8.25 11.5 13.75
```

Рис. 3.2: Поэлементные операции над многомерными массивами

Выполним примеры из раздела про транспонирование,след,ранг,определитель и инверсия матрицы (рис. 3.3).

```

Транспонирование, след, ранг, определитель и инверсия матрицы

[11]: import Pkg
      Pkg.add("LinearAlgebra")
      using LinearAlgebra

      Resolving package versions...
      Updating `~/julia/environments/v1.11/Project.toml`
      [37e2e46d] + LinearAlgebra v1.11.0
      No changes to `~/julia/environments/v1.11/Manifest.toml`

[12]: b = rand(1:20, (4,4))

[12]: 4x4 Matrix{Int64}:
      11 16 20 7
      8 7 10 5
      16 5 20 1
      4 5 8 18

[13]: transpose(b)

[13]: 4x4 transpose{::Matrix{Int64}} with eltype Int64:
      11 8 16 4
      16 7 5 5
      20 10 20 8
      7 5 1 18

[15]: # След матрицы (сумма диагональных элементов):
      tr(b)

[15]: 56

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

[17]: -7649.999999999999

```

Рис. 3.3: Транспонирование, след, ранг, определитель и инверсия матрицы

Выполним примеры из раздела про вычисление нормы векторов и матриц, повороты, вращения (рис. 3.4).

```

Вычисление нормы векторов и матриц, повороты, вращения

[18]: # Создание вектора X:
      X = [2, 4, -5]
      # Вычисление евклидовой нормы:
      norm(X)

[18]: 6.708203932499369

[20]: # Вычисление p-нормы:
      p = 4
      norm(X,p)

[20]: 5.472655503843073

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

[21]: 9.486832980505138

[22]: # Угол между двумя векторами:
      acos((transpose(X)*Y)/(norm(X)*norm(Y)))

[22]: 2.4404307889469252

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

[23]: 7.147682841795258

[25]: opnorm(d,1)

[25]: 8.0

```

Рис. 3.4: Вычисление нормы векторов и матриц

Выполним примеры из раздела про матричное умножение, единичная матрица, скалярное произведение и примеры из раздела про факторизацию и специальные матричные структуры (рис. 3.5).



```

Матричное умножение, единичная матрица, скалярное произведение

[26]: A = rand(1:10, (2,3))
      B = rand(1:10, (3,4))
      # Произведение матриц A и B:
      A*B

[26]: 2×4 Matrix{Int64}:
      38  27  39  63
      64  47  69  101

[27]: X = [2, 4, -5]
      Y = [1, -1, 3]
      dot(X,Y)

[27]: -17

Факторизация. Специальные матричные структуры

[28]: A = rand(3, 3)
      x = fill(1.0, 3)
      b = A*x
      # Решение исходного уравнения получаем с помощью функции \
      # (убеждаемся, что x - единичный вектор):
      A\b

[28]: 3-element Vector{Float64}:
      0.9999999999999999
      1.0
      1.0000000000000002

[29]: # LU-факторизация:
      A\b

[29]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:

```

Рис. 3.5: Факторизация. Специальные матричные структуры

Для оценки эффективности выполнения операций над матрицами большой размерности и специальной структуры воспользуемся пакетом BenchmarkTools (рис. 3.6).

```

...
[34]: Asym = A + A'
      Asym_noisy = copy(Asym)
      Asym_noisy[1,2] += Seps();

[35]: 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);

      Resolving package versions...
      No Changes to `~/julia/environments/v1.11/Project.toml`
      No Changes to `~/julia/environments/v1.11/Manifest.toml`
      725.063 ns (15 allocations: 1.66 KiB)
      1.042 μs (17 allocations: 1.48 KiB)

[36]: # 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)

      370.908 ms (44 allocations: 183.24 MiB)

[36]: 6.229002583394929

```

Рис. 3.6: Факторизация. Специальные матричные структуры

Выполним примеры из раздела про общую линейную алгебру (рис. 3.7).

### Общая линейная алгебра

```
[37]: # Матрица с рациональными элементами:
Arational = Matrix(Rational{BigInt}]{rand(1:10, 3, 3)}/10
# Единичный вектор:
x = fill(1, 3)

[37]: 3-element Vector{Int64}:
      1
      1
      1

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

[38]: 3-element Vector{Rational{BigInt}}:
      8//5
     19//10
      6//5

[39]: # Решение исходного уравнения получаем с помощью функции (убеждаемся, что x – единичный вектор):
Arational\b
# LU-разложение:
lu(Arational)

[39]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
      L factor:
      3x3 Matrix{Rational{BigInt}}:
      1  0  0
     3//5  1  0
     3//5 -3//7  1
      U factor:
      3x3 Matrix{Rational{BigInt}}:
      1  4//5  1//10
      0 -7//25  17//50
      0  0  17//35
```

Рис. 3.7: Общая линейная алгебра

Теперь перейдем к выполнению заданий для самостоятельной работы.

## 3.1 Задание 1

Зададим вектор  $v$ . Умножим вектор  $v$  скалярно сам на себя и сохраним результат в `dot_v`. Также умножим  $v$  матрично на себя (внешнее произведение), присвоив результат переменной `outer_v` (рис. 3.8).

### Задания для самостоятельного выполнения

#### Задание 1. Произведение векторов

1. Задайте вектор  $v$ . Умножьте вектор  $v$  скалярно сам на себя и сохраните результат в `dot_v`.

```
[41]: v = [6, 2, 7, 4, 5]

[41]: 5-element Vector{Int64}:
      6
      2
      7
      4
      5

[42]: dot_v = dot(v, v)

[42]: 130
```

2. Умножьте  $v$  матрично на себя (внешнее произведение), присвоив результат переменной `outer_v`.

```
[44]: outer_v = v * transpose(v)

[44]: 5x5 Matrix{Int64}:
 36 12 42 24 30
 12  4 14  8 10
 42 14 49 28 35
 24  8 28 16 20
 30 10 35 20 25
```

Рис. 3.8: Произведение векторов

## 3.2 Задание 2

- Решим СЛАУ с двумя неизвестными (рис. 3.9 - 3.10).

```
Задание 2. Системы линейных уравнений
Решить СЛАУ с двумя неизвестными.

[45]: A_a = [1 1; 1 -1]
      b_a = [2, 3]
      x = A_a \ b_a

[45]: 2-element Vector{Float64}:
      2.5
      -0.5

[47]: A_b = [1 1; 2 2]
      b_b = [2, 4]
      if (det(A_b)!=0)
          print(A_b \ b_b)
      else
          print("No solution")
      end
      No solution

[51]: A_c = [1 1; 2 2]
      b_c = [2, 5]
      if (det(A_c)!=0)
          print(A_c \ b_c)
      else
          print("No solution")
      end
      No solution
```

Рис. 3.9: Системы линейных уравнений

```
[49]: A_d = [1 1; 2 2; 3 3]
      b_d = [1, 2, 3]
      print(A_d \ b_d)
      [0.5, 0.4999999999999997]

[50]: A_e = [1 1; 2 1; 1 -1]
      b_e = [2, 1, 3]
      print(A_e \ b_e)
      [1.5000000000000004, -0.9999999999999996]

[55]: A_f = [1 1; 2 1; 3 2]
      b_f = [2, 1, 3]
      print(A_f \ b_f)
      [-1.0000000000000009, 3.0000000000000018]
```

Рис. 3.10: Системы линейных уравнений

- Решим СЛАУ с тремя неизвестными (рис. 3.11).

```
Решить СЛАУ с тремя неизвестными.

[56]: A_a = [1 1 1; 1 -1 -2]
      b_a = [2, 3]
      # Решение через псевдообратную матрицу
      x_a = pinv(A_a) * b_a

[56]: 3-element Vector{Float64}:
      2.214285714285714
      0.35714285714285716
      -0.5714285714285715

[57]: A_b = [1 1 1; 2 2 -3; 3 1 1]
      b_b = [2, 4, 1]
      x_b = pinv(A_b) * b_b

[57]: 3-element Vector{Float64}:
      -0.4999999999999999
      2.5
      3.885780586180040e-16

[58]: A_c = [1 1 1; 1 1 2; 2 2 3]
      b_c = [1, 0, 1]
      x_c = pinv(A_c) * b_c

[58]: 3-element Vector{Float64}:
      0.9999999999999991
      1.0000000000000018
      -1.0000000000000002

[59]: A_d = [1 1 1; 1 1 2; 2 2 3]
      b_d = [1, 0, 0]
      x_d = pinv(A_d) * b_d

[59]: 3-element Vector{Float64}:
      0.8333333333333323
      0.8333333333333346
      -0.9999999999999999
```

Рис. 3.11: Системы линейных уравнений

### 3.3 Задание 3

Приведем матрицы к диагональному виду (рис. 3.12):

```
Задание 3. Операции с матрицами
Приведите приведённые ниже матрицы к диагональному виду

[61]: A1 = [1 -2; -2 1];
      Diagonal(eigen(A1).values)

[61]: 2x2 Diagonal(Float64, Vector{Float64}):
      -1.0  .
      .    3.0

[62]: A2 = [1 -2; -2 3]
      Diagonal(eigen(A2).values)

[62]: 2x2 Diagonal(Float64, Vector{Float64}):
      -0.236068  .
      .          4.23607

[63]: A3 = [1 -2 0; -2 1 2; 0 2 0]
      Diagonal(eigen(A3).values)

[63]: 3x3 Diagonal(Float64, Vector{Float64}):
      -2.14134  .  .
      .         0.515138  .
      .         .         3.6262
```

Рис. 3.12: Операции с матрицами

Вычислим следующие операции с матрицами (рис. 3.13):

```
Вычислите

[64]: A = [1 -2; -2 1]
      A^10

[64]: 2x2 Matrix{Int64}:
      29525  -29524
      -29524  29525

[65]: B = [5 -2; -2 5]
      sqrt(B)

[65]: 2x2 Matrix{Float64}:
      2.1889  -0.45685
      -0.45685  2.1889

[66]: A^(1/3)

[66]: 2x2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
      0.971125+0.433013im  -0.471125+0.433013im
      -0.471125+0.433013im  0.971125+0.433013im

[68]: D = [1 2; 2 3]
      sqrt(D)

[68]: 2x2 Matrix{ComplexF64}:
      0.568864+0.351578im  0.928442-0.217287im
      0.928442-0.217287im  1.48931+0.134291im
```

Рис. 3.13: Операции с матрицами

Найдем собственные значения матрицы A. Создадим диагональную матрицу из собственных значений матрицы A. Создадим нижнедиагональную матрицу из матрицы A. Оценим эффективность выполняемых операций (рис. 3.14 - 3.15):

Найдите собственные значения матрицы A, если

```
[69]: # Задаем матрицу A
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]]

[69]: 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

[70]: # собственные значения
@time F = eigen(A)

0.004099 seconds (19 allocations: 3.047 KiB)

[70]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
5-element Vector{Float64}:
 -128.4932276480214
 -55.88778455385699
 42.75216727931888
 87.161114775145
 542.4677301466138
vectors:
5x5 Matrix{Float64}:
 0.147575  0.647178  0.818882  0.548983 -0.587987
 0.256795 -0.173868  0.834628 -0.239864 -0.387253
 0.185537  0.239762 -0.422161 -0.731925 -0.440631
 -0.819784 -0.247586 -0.0273194 0.0366447 -0.514526
 0.453885 -0.657619 -0.352577  0.322668 -0.364928
```

Рис. 3.14: Операции с матрицами

```
[71]: @time D = Diagonal(F.values)

0.000134 seconds (1 allocation: 16 bytes)

[71]: 5x5 Diagonal{Float64, Vector{Float64}}:
 -128.493  .  .  .  .
 . -55.8878  .  .  .
 .  . 42.7522  .  .
 .  .  . 87.1611  .
 .  .  .  . 542.468

[72]: @time L = LowerTriangular(A)

0.010226 seconds (81 allocations: 3.028 KiB, 95.92% compilation time)

[72]: 5x5 LowerTriangular{Int64, Matrix{Int64}}:
 140  .  .  .  .
  97 106  .  .  .
  74  89 152  .  .
 168 131 144  54  .
 131  36  71 142  36
```

Рис. 3.15: Оценка эффективности

## 3.4 Задание 4

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. 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 (рис. 3.16).

**Задание 4. Линейные модели экономики**

Матрица  $A$  называется продуктивной, если решение  $x$  системы при любой неотрицательной правой части  $y$  имеет только неотрицательные элементы  $x_i$ . Используя это определение, проверьте, являются ли матрицы продуктивными

```

[75]: A1 = [1 2; 3 4]
      inv_I_A1 = inv(I - A1)

[75]: 2x2 Matrix{Float64}:
      0.5  -0.333333
      -0.5  0.0

[76]: all(>=(0), inv_I_A1) ? "Да" : "НЕТ"
[76]: "НЕТ"

[77]: A2 = 0.5 * [1 2; 3 4]
      inv_I_A2 = inv(I - A2)

[77]: 2x2 Matrix{Float64}:
      0.5  -0.5
      -0.75 -0.25

[78]: all(>=(0), inv_I_A2) ? "Да" : "НЕТ"
[78]: "НЕТ"

[79]: A3 = 0.1 * [1 2; 3 4]
      inv_I_A3 = inv(I - A3)

[79]: 2x2 Matrix{Float64}:
      1.25  0.416667
      0.625 1.875

[80]: all(>=(0), inv_I_A3) ? "Да" : "НЕТ"
[80]: "Да"

```

Рис. 3.16: Линейные модели экономики

Критерий продуктивности: матрица  $A$  является продуктивной тогда и только тогда, когда все элементы матрицы

$$(E - A)^{-1}$$

являются неотрицательными числами. Используя этот критерий, проверим, являются ли матрицы продуктивными (рис. 3.17).

Проверьте, являются ли матрицы продуктивными.

```

[84]: inv_EA1 = inv(I - A1)

[84]: 2x2 Matrix{Float64}:
      -0.0  -0.333333
      -0.5  0.0

[85]: all(>=(0), inv_EA1) ? "Да" : "НЕТ"
[85]: "НЕТ"

[86]: inv_EA2 = inv(I - A2)

[86]: 2x2 Matrix{Float64}:
      0.5  -0.5
      -0.75 -0.25

[87]: all(>=(0), inv_EA2) ? "Да" : "НЕТ"
[87]: "НЕТ"

[88]: inv_EA3 = inv(I - A3)

[88]: 2x2 Matrix{Float64}:
      1.25  0.416667
      0.625 1.875

[89]: all(>=(0), inv_EA3) ? "Да" : "НЕТ"
[89]: "Да"

```

Рис. 3.17: Линейные модели экономики

Спектральный критерий продуктивности: матрица  $A$  является продуктивной тогда и только тогда, когда все её собственные значения по модулю меньше 1. Используя этот критерий, проверим, являются ли матрицы продуктивными (рис. 3.18).

```

[89]: eigvalsA1 = eigvals(A1)
[89]: 2-element Vector{Float64}:
      -1.4494897427831779
       3.4494897427831783

[90]: all(x -> abs(x) < 1, eigvalsA1) ? "Да" : "НЕТ"
[90]: "НЕТ"

[91]: eigvalsA2 = eigvals(A2)
[91]: 2-element Vector{Float64}:
      -0.18614866163450715
       2.686148661634507

[92]: all(x -> abs(x) < 1, eigvalsA2) ? "Да" : "НЕТ"
[92]: "НЕТ"

[93]: eigvalsA3 = eigvals(A3)
[93]: 2-element Vector{Float64}:
      -0.037228132326901475
       0.5372281323269015

[94]: all(x -> abs(x) < 1, eigvalsA3) ? "Да" : "НЕТ"
[94]: "Да"

[95]: A4 = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
      eigvalsA4 = eigvals(A4)
[95]: 3-element Vector{Float64}:
       0.02679491924311228
       0.1
       0.37320580875688774

```

Рис. 3.18: Линейные модели экономики

## 4 Выводы

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