Лабораторная работа № 4

Линейная алгебра

Беличева Дарья Михайловна

Содержание

Список иллюстраций

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, повторите примеры.
2. Выполните задания для самостоятельной работы.

3 Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [1]. 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 [2].

4 Выполнение лабораторной работы

Выполним примеры из раздела про поэлементные операции над многомерными массивами (рис. 4.1-4.2).

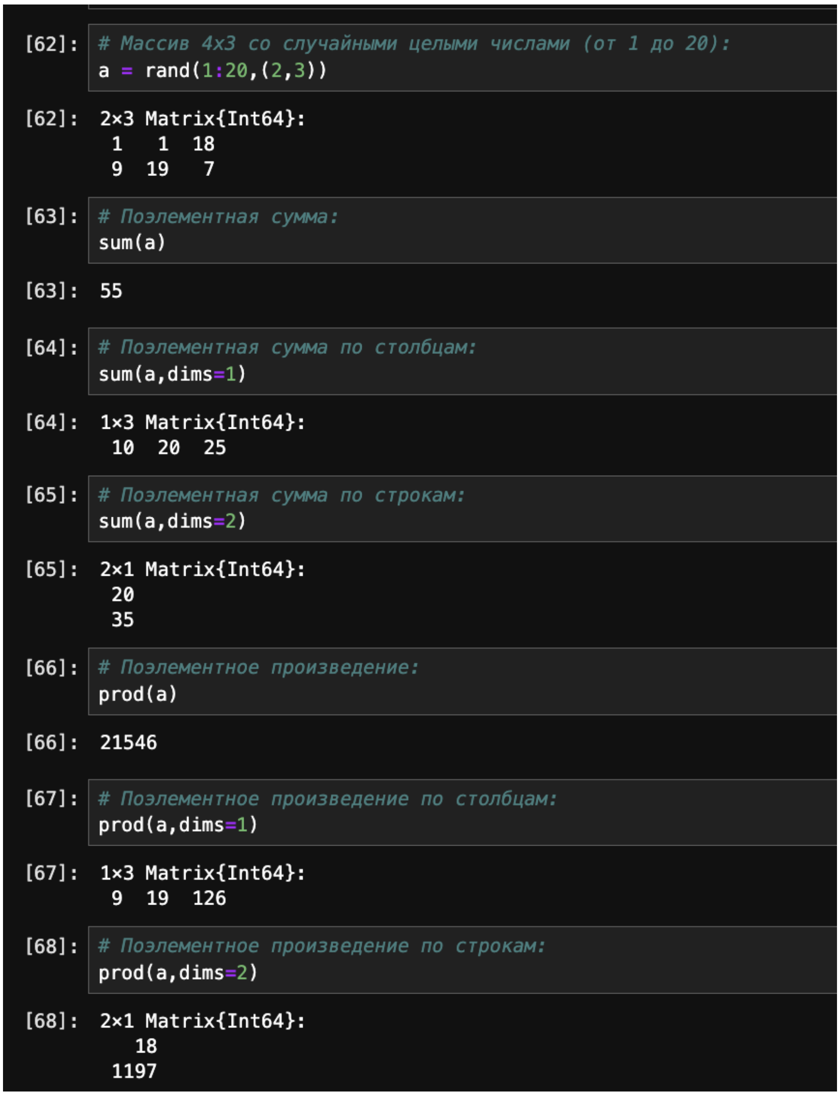


Рис. 4.1: Поэлементные операции над многомерными массивами

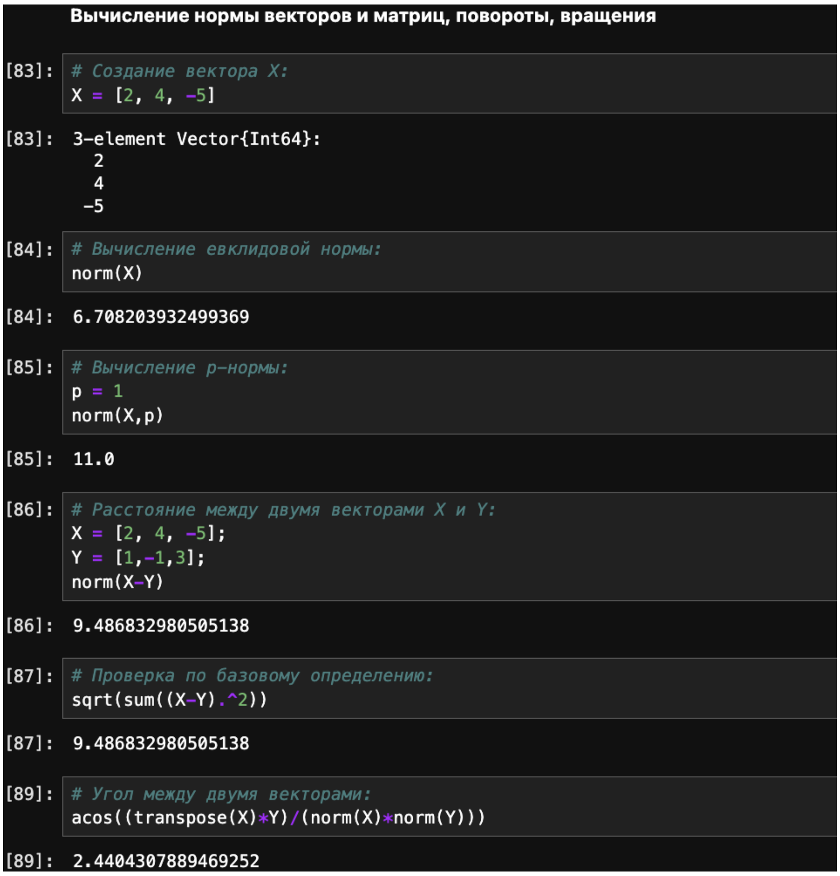
```
[70]: # Вычисление среднего значения массива:
      mean(a)

[70]: 9.166666666666666

[71]: # Среднее по столбцам:
      mean(a,dims=1)

[71]: 1×3 Matrix{Float64}:
       5.0  10.0  12.5

[72]: # Среднее по строкам:
      mean(a,dims=2)

[72]: 2×1 Matrix{Float64}:
        6.666666666666667
       11.666666666666666
```

Транспонирование, след, ранг, определитель и инверсия матрицы

```
 [4]: # Подключение пакета LinearAlgebra:
      import Pkg
      Pkg.add("LinearAlgebra`b")

        Resolving package versions...
       No Changes to `C:\Users\dasha\.julia\environments\v1.9\Project.toml`
       No Changes to `C:\Users\dasha\.julia\environments\v1.9\Manifest.toml`

 [7]: using LinearAlgebra

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

[73]: 4×4 Matrix{Int64}:
       11   7   1   5
       14   4   2   9
        7  16   9  10
       10  15  16   5

[74]: # Транспонирование:
      transpose(b)

[74]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
       11  14   7  10
        7   4  16  15
        1   2   9  16
        5   9  10   5
```

Рис. 4.2: Поэлементные операции над многомерными массивами

Выполним примеры из раздела про транспонирование,след,ранг,определители инверсия матрицы (рис. 4.3).



Вычисление нормы векторов и матриц, повороты, вращения

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

[83]: 3-element Vector{Int64}:
        2
        4
       -5

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

[84]: 6.708203932499369

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

[85]: 11.0

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

[86]: 9.486832980505138

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

[87]: 9.486832980505138

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

[89]: 2.4404307889469252
```

Рис. 4.3: Транспонирование,след,ранг,определительи инверсия матрицы

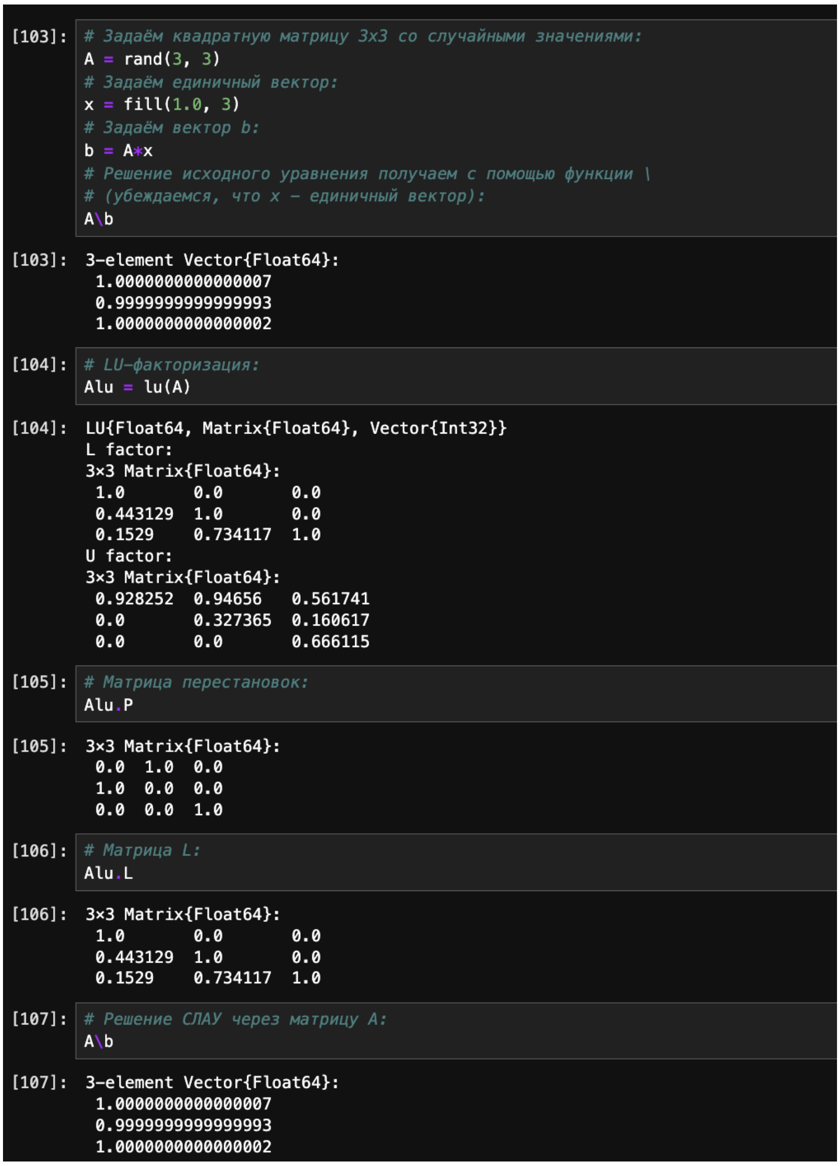
Выполним примеры из раздела про матричное умножение,единичная матрица,скалярное произведение (рис. 4.4).

Матричное умножение, единичная матрица, скалярное произведение

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

[97]: 2×3 Matrix{Int64}:
       5  7  9
       1  5  5

[98]: B = rand(2:4,(3,2))

[98]: 3×2 Matrix{Int64}:
       4  2
       4  2
       3  3

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

[99]: 2×2 Matrix{Int64}:
       75  51
       39  27

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

[100]: 3×3 Matrix{Int64}:
        1  0  0
        0  1  0
        0  0  1

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

[101]: -17

[102]: X'Y

[102]: -17
```

Рис. 4.4: Матричное умножение,единичная матрица,скалярное произведение

Выполним примеры из раздела про факторизацию и специальные матричные структуры (рис. 4.4-4.7).



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

[103]: 3-element Vector{Float64}:
        1.0000000000000007
        0.9999999999999993
        1.0000000000000002

[104]: # LU-факторизация:
       Alu = lu(A)

[104]: LU{Float64, Matrix{Float64}, Vector{Int32}}
       L factor:
       3×3 Matrix{Float64}:
        1.0       0.0       0.0
        0.443129  1.0       0.0
        0.1529    0.734117  1.0
       U factor:
       3×3 Matrix{Float64}:
        0.928252  0.94656   0.561741
        0.0       0.327365  0.160617
        0.0       0.0       0.666115

[105]: # Матрица перестановок:
       Alu.P

[105]: 3×3 Matrix{Float64}:
        0.0  1.0  0.0
        1.0  0.0  0.0
        0.0  0.0  1.0

[106]: # Матрица L:
       Alu.L

[106]: 3×3 Matrix{Float64}:
        1.0       0.0       0.0
        0.443129  1.0       0.0
        0.1529    0.734117  1.0

[107]: # Решение СЛАУ через матрицу A:
       A\b

[107]: 3-element Vector{Float64}:
        1.0000000000000007
        0.9999999999999993
        1.0000000000000002
```

Рис. 4.5: Факторизация.Специальные матричные структуры

```
[115]: # Симметризация матрицы A:
       Asym = A + A'

[115]: 3×3 Matrix{Float64}:
        0.822671  1.67506   0.55147
        1.67506   1.89312   0.946794
        0.55147   0.946794  1.73983

[116]: # Спектральное разложение симметризованной матрицы:
       AsymEig = eigen(Asym)

[116]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
       values:
       3-element Vector{Float64}:
        -0.4076568632552292
         1.1454565616037984
         3.7178240445947823
       vectors:
       3×3 Matrix{Float64}:
         0.794497   -0.336367  -0.5056
        -0.604064   -0.35234   -0.714817
         0.0622975   0.873335  -0.48312

[117]: #Собственные векторы:
       AsymEig.vectors

[117]: 3×3 Matrix{Float64}:
         0.794497   -0.336367  -0.5056
        -0.604064   -0.35234   -0.714817
         0.0622975   0.873335  -0.48312

 [51]: # Проверяем, что получится единичная матрица:
       inv(AsymEig)*Asym

 [51]: 3×3 Matrix{Float64}:
         1.0           1.9984e-15   2.44249e-15
         1.44329e-15   1.0          1.55431e-15
        -3.55271e-15  -2.22045e-15  1.0

[118]: # Матрица 1000 х 1000:
       n = 1000
       A = randn(n,n)

[118]: 1000×1000 Matrix{Float64}:
        -0.574533    0.0643334  -2.55939    …   1.56795    -1.76232    -0.824679
         1.11881     0.771541   -0.33128        0.433536   -1.40129     1.19907
        -0.278492   -0.417667   -0.0462546     -0.493284    0.783466   -0.372574
        -1.55815    -1.61922    -1.79691       -0.364171   -1.45956     0.574376
        -1.88479    -1.00454    -0.729409       0.228975    0.66527    -1.01256
         1.71465     0.560726   -0.147459   …  -0.0286493   0.571772    1.13623
```

Рис. 4.6: Факторизация.Специальные матричные структуры

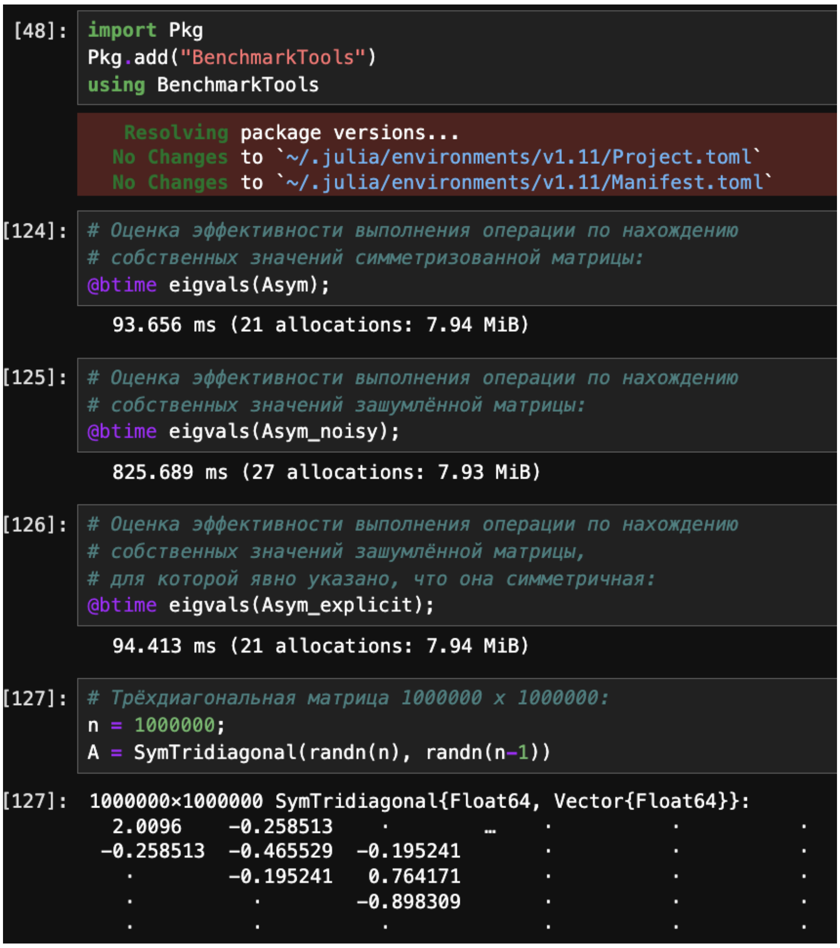


Рис. 4.7: Факторизация.Специальные матричные структуры

Выполним примеры из раздела про общую линейную алгебру (рис. 4.8).

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

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

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

[2]: 3-element Vector{Float64}:
      1.0
      1.0
      1.0

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

[3]: 3-element Vector{BigFloat}:
      0.8999999999999999999999999999999999999999999999999999999999999999999999999999965
      1.799999999999999999999999999999999999999999999999999999999999999999999999999993
      1.5

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

[4]: 3-element Vector{BigFloat}:
      0.9999999999999999999999999999999999999999999999999999999999999999999999999999914
      0.9999999999999999999999999999999999999999999999999999999999999999999999999999914
      1.0

[8]: # LU-факторизация:
     Alu = lu(Arational)

[8]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int32}}
     L factor:
     3×3 Matrix{Rational{BigInt}}:
       1      0     0
      1//8    1     0
      1//4  22//31  1
     U factor:
     3×3 Matrix{Rational{BigInt}}:
      4//5   1//10    9//10
       0    31//80   23//80
       0      0     177//310
```

Рис. 4.8: Общаялинейная алгебра

4.1 Заданиядля самостоятельного выполнения

Зададим вектор v. Умножим вектор v скалярно сам на себя и сохраним результат вdot_v (рис. 4.9).

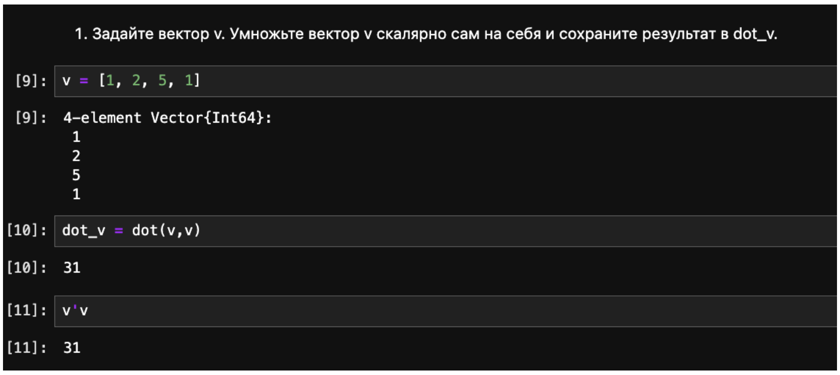


Рис. 4.9: Произведение векторов

Умножим v матрично на себя(внешнее произведение), присвоив результат переменной outer_v (рис. 4.10).

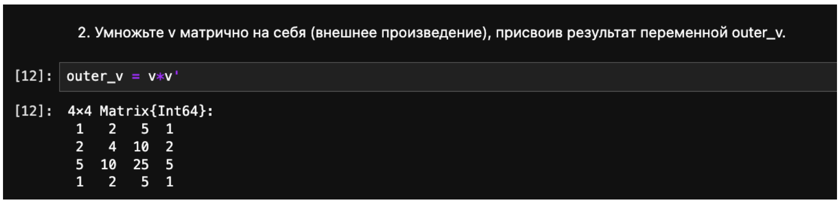
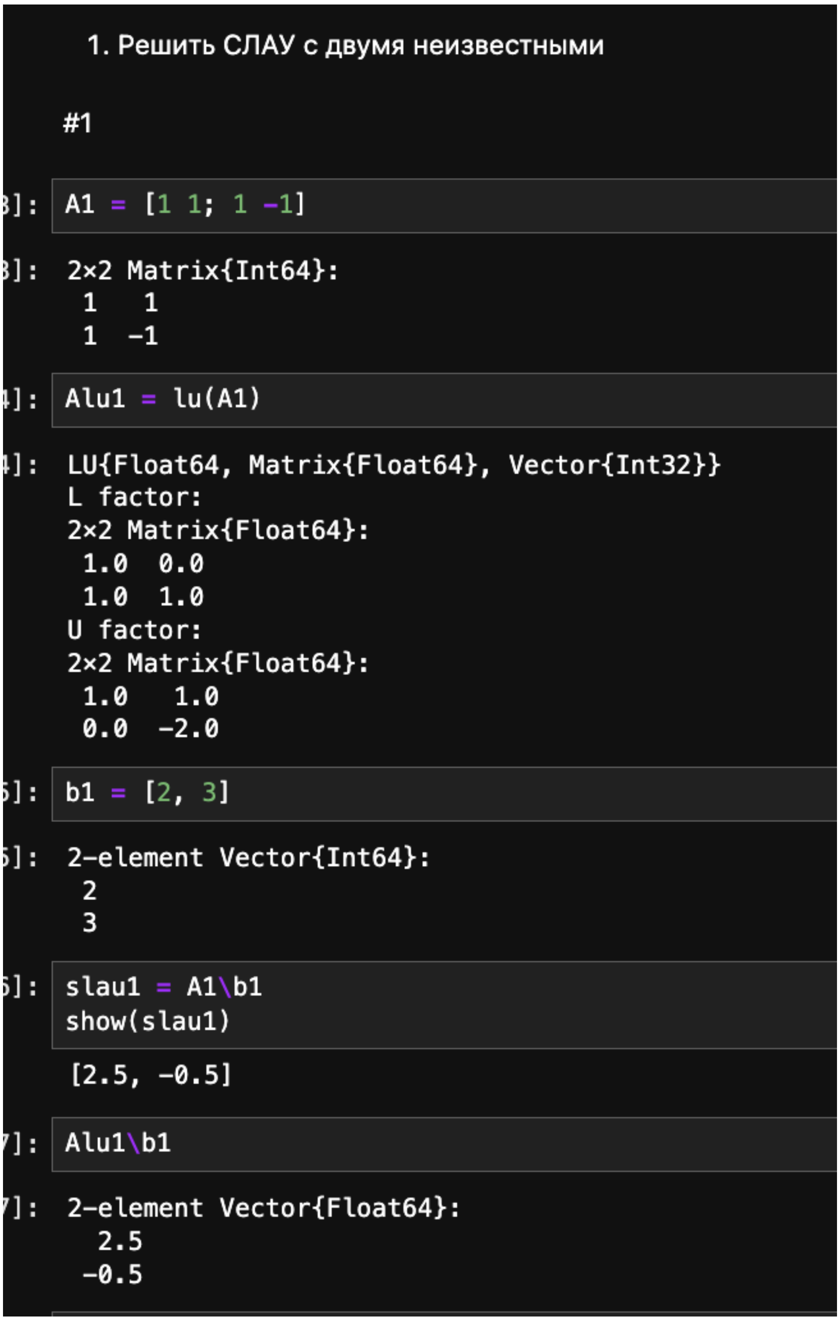


Рис. 4.10: Произведение векторов

Решим СЛАУ с двумя неизвестными (рис. 4.11-4.12).



1. Решить СЛАУ с двумя неизвестными

#1

```
]: A1 = [1 1; 1 -1]

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

]: Alu1 = lu(A1)

]: LU{Float64, Matrix{Float64}, Vector{Int32}}
   L factor:
   2×2 Matrix{Float64}:
    1.0  0.0
    1.0  1.0
   U factor:
   2×2 Matrix{Float64}:
    1.0   1.0
    0.0  -2.0

]: b1 = [2, 3]

]: 2-element Vector{Int64}:
    2
    3

]: slau1 = A1\b1
   show(slau1)

   [2.5, -0.5]

]: Alu1\b1

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

Рис. 4.11: Системы линейных уравнений

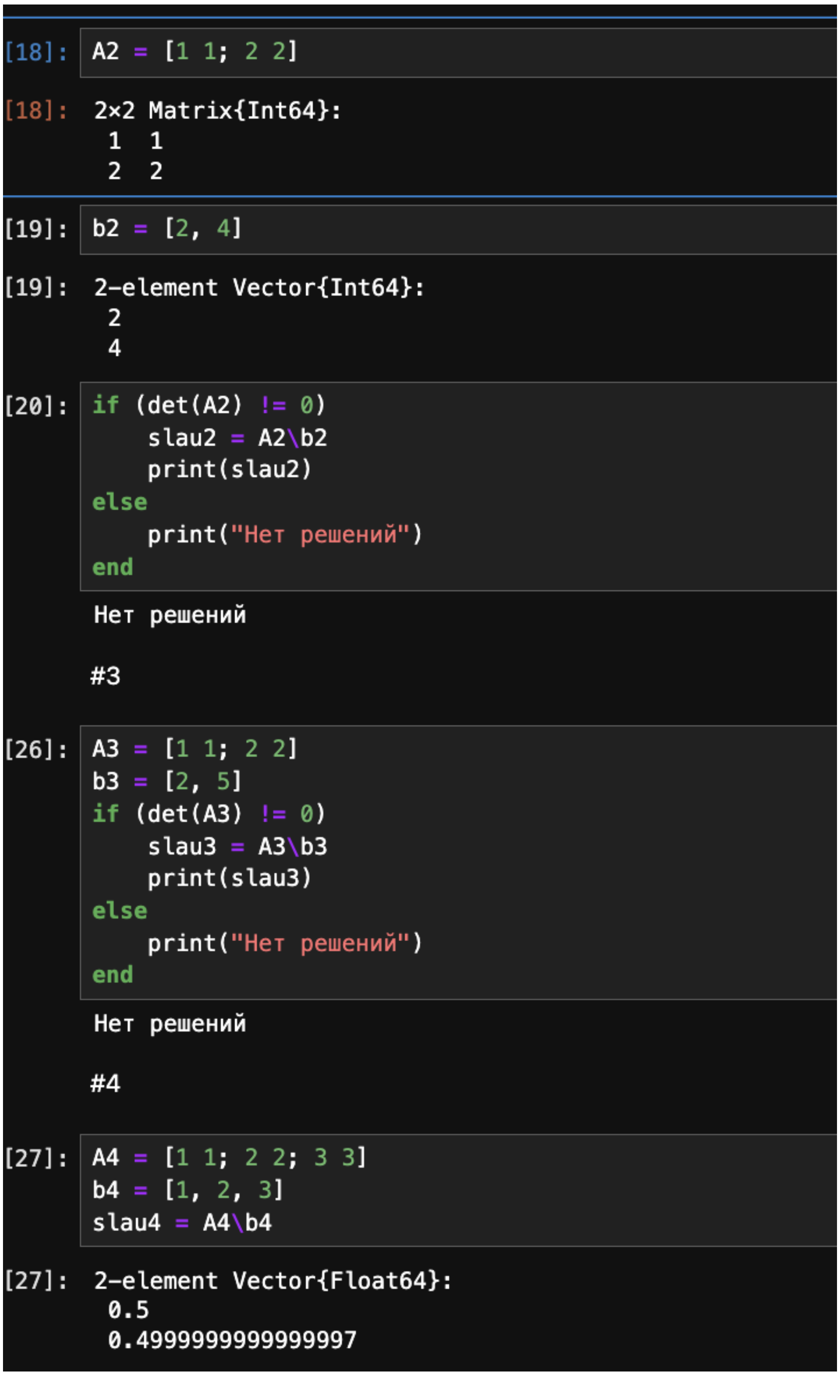
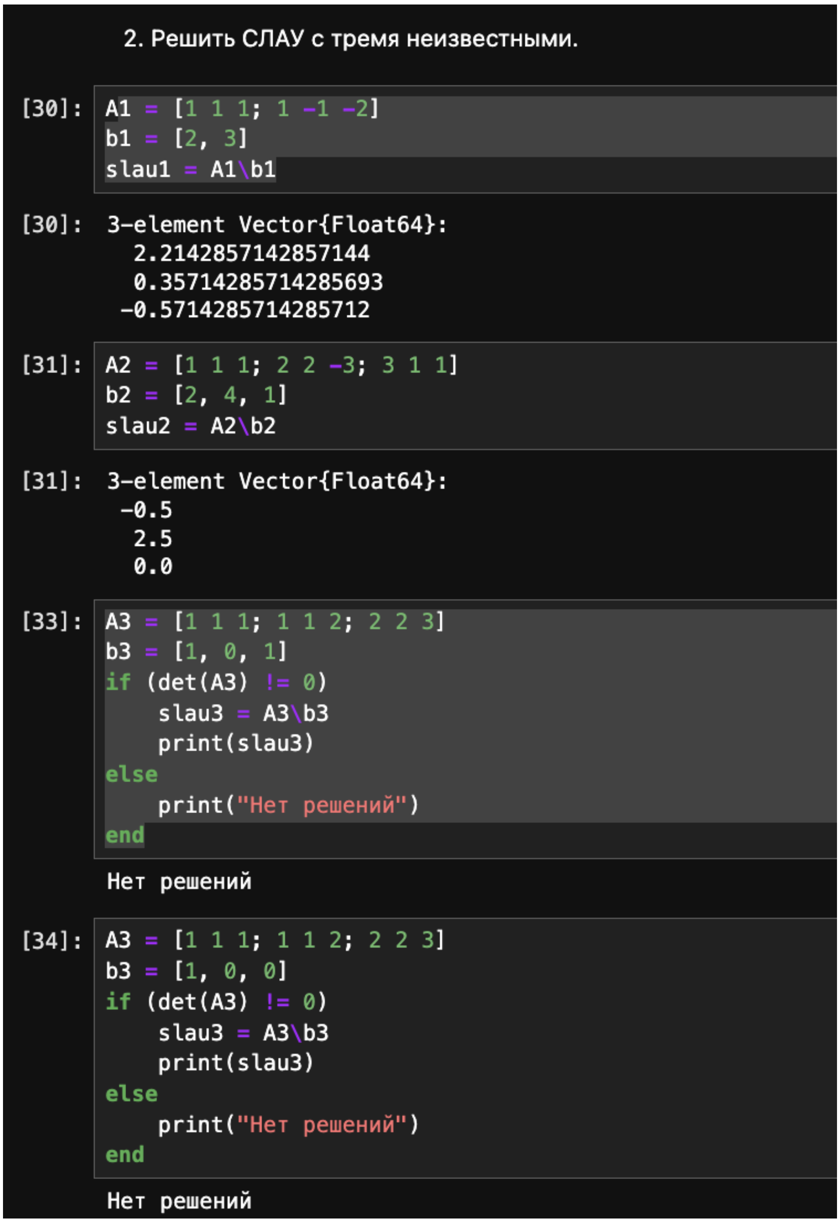


Рис. 4.12: Систем линейных уравнений

Решим СЛАУ с тремя неизвестными (рис. 4.13).



2. Решить СЛАУ с тремя неизвестными.

```
[30]: A1 = [1 1 1; 1 -1 -2]
      b1 = [2, 3]
      slau1 = A1\b1

[30]: 3-element Vector{Float64}:
        2.2142857142857144
        0.35714285714285693
       -0.5714285714285712

[31]: A2 = [1 1 1; 2 2 -3; 3 1 1]
      b2 = [2, 4, 1]
      slau2 = A2\b2

[31]: 3-element Vector{Float64}:
       -0.5
        2.5
        0.0

[33]: A3 = [1 1 1; 1 1 2; 2 2 3]
      b3 = [1, 0, 1]
      if (det(A3) != 0)
          slau3 = A3\b3
          print(slau3)
      else
          print("Нет решений")
      end

      Нет решений

[34]: A3 = [1 1 1; 1 1 2; 2 2 3]
      b3 = [1, 0, 0]
      if (det(A3) != 0)
          slau3 = A3\b3
          print(slau3)
      else
          print("Нет решений")
      end

      Нет решений
```

Рис. 4.13: Систем линейных уравнений

Приведем матрицы к диагональному виду (рис. 4.14).

```
[57]: A = [1 -2; -2 1]

[57]: 2×2 Matrix{Int64}:
        1  -2
       -2   1

[58]: AsymEig = eigen(A)

[58]: 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

[61]: display(diagm(AsymEig.values))

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

[35]: B = [1 -2; -2 3]
      Beig = eigen(B)
      display(diagm(Beig.values))

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

[36]: C = [1 -2 0; -2 1 2; 0 2 0]
      Ceig = eigen(C)
      display(diagm(Ceig.values))

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

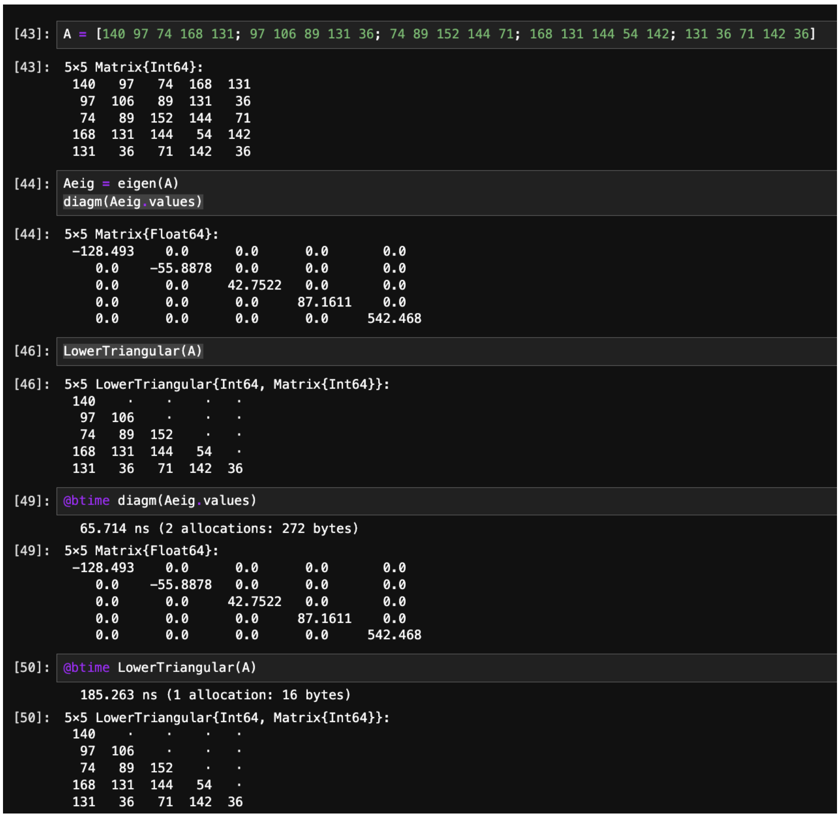
Рис. 4.14: Операции с матрицами

Вычислим (рис. 4.15).

```
[37]: A = [1 -2; -2 1]
      A^10

[37]: 2×2 Matrix{Int64}:
        29525  -29524
       -29524   29525

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

[38]: 2×2 Matrix{Float64}:
        2.1889   -0.45685
       -0.45685   2.1889

[31]: sqrt_B*sqrt_B

[31]: 2×2 Matrix{Float64}:
        5.0  -2.0
       -2.0   5.0

[40]: C = [1 -2; -2 1]
      C^(1/3)

[40]: 2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
        0.971125+0.433013im  -0.471125+0.433013im
       -0.471125+0.433013im   0.971125+0.433013im

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

[41]: 2×2 Matrix{ComplexF64}:
       0.568864+0.351578im  0.920442-0.217287im
       0.920442-0.217287im   1.48931+0.134291im
```

Рис. 4.15: 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. 4.16).



```
[43]: 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]

[43]: 5×5 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

[44]: Aeig = eigen(A)
      diagm(Aeig.values)

[44]: 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

[46]: LowerTriangular(A)

[46]: 5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36

[49]: @btime diagm(Aeig.values)

  65.714 ns (2 allocations: 272 bytes)

[49]: 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

[50]: @btime LowerTriangular(A)

  185.263 ns (1 allocation: 16 bytes)

[50]: 5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рис. 4.16: Операции с матрицами

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 матрицып родуктивными (рис. 4.17-4.18).

```
[71]: a = [1 2; 3 1]
      b = 1/2*a
      c = 1/10*a
      E = Matrix(I, 2, 2)

[71]: 2×2 Matrix{Bool}:
       1  0
       0  1

[72]: inv(E - a) # не продуктивная

[72]: 2×2 Matrix{Float64}:
       -0.0  -0.333333
       -0.5   0.0

[73]: inv(E - b) # не продуктивная

[73]: 2×2 Matrix{Float64}:
       -0.4  -0.8
       -1.2  -0.4

[74]: inv(E - c) # продуктивная

[74]: 2×2 Matrix{Float64}:
       1.2  0.266667
       0.4  1.2
```

Рис. 4.17: Линейные модели экономики

```
[75]: d = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]

[75]: 3×3 Matrix{Float64}:
       0.1  0.2  0.3
       0.0  0.1  0.2
       0.0  0.1  0.3

[76]: aeig = eigen(a)
      abs.(aeig.values).< 1

[76]: 2-element BitVector:
       0
       0

[77]: beig = eigen(b)
      abs.(beig.values).< 1

[77]: 2-element BitVector:
       1
       0

[78]: ceig = eigen(c)
      abs.(ceig.values).< 1

[78]: 2-element BitVector:
       1
       1

[80]: deig = eigen(d)
      abs.(deig.values).< 1

[80]: 3-element BitVector:
       1
       1
       1
```

Рис. 4.18: Линейные модели экономики

5 Выводы

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

Список литературы

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).