Лабораторная работа № 4

Линейная алгебра

Беличева Д. М.

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Беличева Дарья Михайловна
- студентка
- Российский университет дружбы народов
- 1032216453@pfur.ru
- https://dmbelicheva.github.io/ru/

Цель работы

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

Задание

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

Выполнение лабораторной работы

```
[62]: # Массив 4x3 со случайными целыми числами (от 1 до 20):
      a = rand(1:20,(2,3))

[62]: 2×3 Matrix{Int64}:
       1   1  18
       9  19   7

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

[63]: 55

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

[64]: 1×3 Matrix{Int64}:
       10  20  25

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

[65]: 2×1 Matrix{Int64}:
       20
       35

[66]: # Поэлементное произведение:
      prod(a)

[66]: 21546

[67]: # Поэлементное произведение по столбцам:
      prod(a,dims=1)

[67]: 1×3 Matrix{Int64}:
       9  19  126

[68]: # Поэлементное произведение по строкам:
      prod(a,dims=2)

[68]: 2×1 Matrix{Int64}:
         18
       1197
```

Рис. 1: Поэлементные операции над многомерными массивами

Выполнение лабораторной работы

```
[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")

        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. 2: Поэлементные операции над многомерными массивами

Выполнение лабораторной работы

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. 3: Транспонирование,след,ранг,определительи инверсия матрицы

Выполнение лабораторной работы

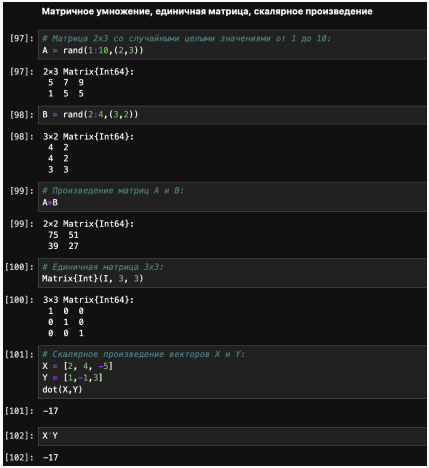
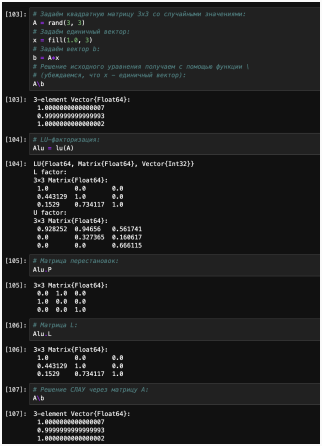


Рис. 4: Матричное умножение,единичная матрица,скалярное произведение

Выполнение лабораторной работы



```
[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       0.327305  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. 5: Факторизация.Специальные матричные структуры

Выполнение лабораторной работы

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

[115]: 3×3 Matrix{Float64}:
        0.822671  1.67586   0.55147
        1.67586   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
         1.1454565610037084
         3.7178248445947823
       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 x 1000:
       n = 1000
       A = randn(n,n)

[118]: 1000×1000 Matrix{Float64}:
        -0.574533    0.0643334  -2.55939    …   1.56795    -1.76232    -0.824679
         1.13083     0.771541   -0.33128        0.433536   -1.48129     1.19007
        -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. 6: Факторизация.Специальные матричные структуры

Выполнение лабораторной работы

```
[48]: import Pkg
      Pkg.add("BenchmarkTools")
      using BenchmarkTools

        Resolving package versions...
        No Changes to `~/.julia/environments/v1.11/Project.toml`
        No Changes to `~/.julia/environments/v1.11/Manifest.toml`

[124]: # Оценка эффективности выполнения операции по нахождению
       # собственных значений симметризованной матрицы:
       @btime eigvals(Asym);

         93.656 ms (21 allocations: 7.94 MiB)

[125]: # Оценка эффективности выполнения операции по нахождению
       # собственных значений зашумлённой матрицы:
       @btime eigvals(Asym_noisy);

         825.689 ms (27 allocations: 7.93 MiB)

[126]: # Оценка эффективности выполнения операции по нахождению
       # собственных значений зашумлённой матрицы,
       # для которой явно указано, что она симметричная:
       @btime eigvals(Asym_explicit);

         94.413 ms (21 allocations: 7.94 MiB)

[127]: # Трёхдиагональная матрица 1000000 x 1000000:
       n = 1000000;
       A = SymTridiagonal(randn(n), randn(n-1))

[127]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
        2.0096     -0.258513      …   ⋅          ⋅          ⋅
       -0.258513   -0.465529      -0.195241      ⋅          ⋅          ⋅
         ⋅         -0.195241       0.764171      ⋅          ⋅          ⋅
         ⋅          ⋅             -0.898309      ⋅          ⋅          ⋅
         ⋅          ⋅              ⋅             ⋅          ⋅          ⋅
```

Рис. 7: Факторизация.Специальные матричные структуры

Выполнение лабораторной работы

```
[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]: # Зададим единичный вектор:
     x = fill(1.0, 3)

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

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

[3]: 3-element Vector{BigFloat}:
      0.899999999999999999999999999999999999999999999999999999999999999999999999999965
      1.799999999999999999999999999999999999999999999999999999999999999999999999999993
      1.5

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

[4]: 3-element Vector{BigFloat}:
      0.999999999999999999999999999999999999999999999999999999999999999999999999999914
      0.999999999999999999999999999999999999999999999999999999999999999999999999999914
      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. 8: Общая линейная алгебра

Задания для самостоятельного выполнения

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

```
[9]: v = [1, 2, 5, 1]

[9]: 4-element Vector{Int64}:
      1
      2
      5
      1

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

[10]: 31

[11]: v'v

[11]: 31
```

Рис. 9: Произведение векторов

Задания для самостоятельного выполнения

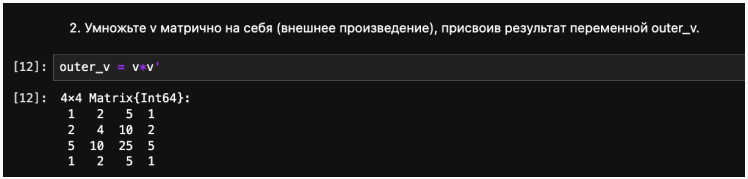
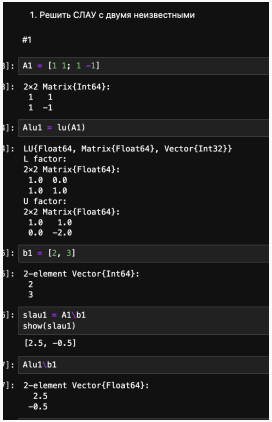


Рис. 10: Произведение векторов

Задания для самостоятельного выполнения



1. Решить СЛАУ с двумя неизвестными

#1

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

```
[5]: 2×2 Matrix{Int64}:
      1   1
      1  -1
```

```
[6]: Alu1 = lu(A1)
```

```
[6]: 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
```

```
[8]: b1 = [2, 3]
```

```
[8]: 2-element Vector{Int64}:
      2
      3
```

```
[9]: slau1 = A1\b1
     show(slau1)
```

```
     [2.5, -0.5]
```

```
[7]: Alu1\b1
```

```
[7]: 2-element Vector{Float64}:
       2.5
      -0.5
```

Рис. 11: Системы линейных уравнений

Задания для самостоятельного выполнения

```
[18]: A2 = [1 1; 2 2]

[18]: 2×2 Matrix{Int64}:
       1  1
       2  2

[19]: b2 = [2, 4]

[19]: 2-element Vector{Int64}:
       2
       4

[20]: if (det(A2) != 0)
          slau2 = A2\b2
          print(slau2)
      else
          print("Нет решений")
      end

      Нет решений

      #3

[26]: A3 = [1 1; 2 2]
      b3 = [2, 5]
      if (det(A3) != 0)
          slau3 = A3\b3
          print(slau3)
      else
          print("Нет решений")
      end

      Нет решений

      #4

[27]: A4 = [1 1; 2 2; 3 3]
      b4 = [1, 2, 3]
      slau4 = A4\b4

[27]: 2-element Vector{Float64}:
       0.5
       0.4999999999999997
```

Рис. 12: Систем линейных уравнений

Задания для самостоятельного выполнения

Рис. 13: Систем линейных уравнений

Задания для самостоятельного выполнения

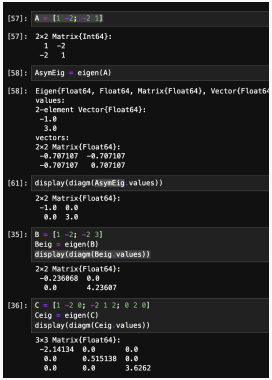


Рис. 14: Операции с матрицами

Задания для самостоятельного выполнения

```
[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. 15: Операции с матрицами

Задания для самостоятельного выполнения

```
[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. 16: Операции с матрицами

Задания для самостоятельного выполнения

```
[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. 17: Линейные модели экономики

Задания для самостоятельного выполнения

```
[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. 18: Линейные модели экономики

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[Электронныйресурс].2024JuliaLang.orgcontributors.URL:https://julialang.org/(дата-обращения:11.10.2024).
2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.orgcontributors. URL: https://docs.julialang.org/en/v1/ (дата обращения:11.10.2024)