Отчет по лабораторной работе №4

Линейная алгебра

Легиньких Галина Андреевна

Содержание

Список иллюстраций

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

3 Выполнение лабораторной работы

1. Для начала рассмотрела тему “Поэлементные операции над многомерными массивами”.

Примеры поэлементной суммы. (рис. 3.1)

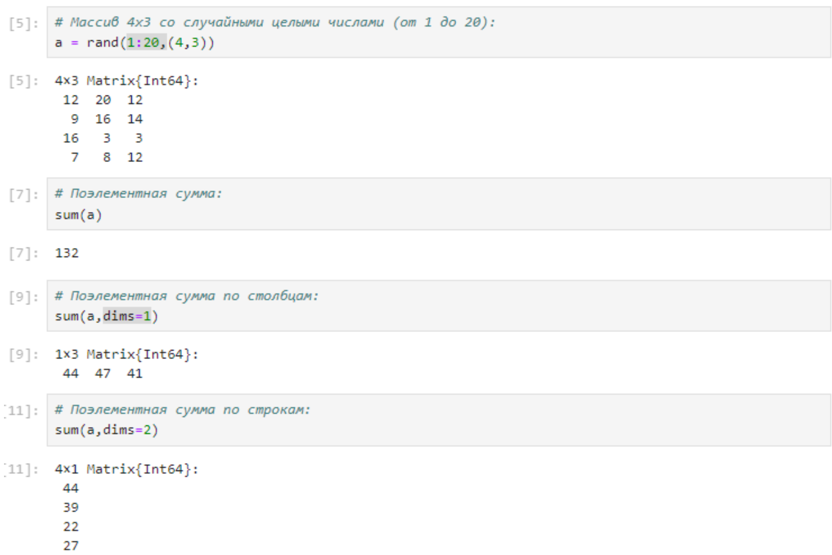

Рис. 3.1: Поэлементная сумма

Так же повторила примеры поэлементного произведения. (рис. 3.2)

```
[13]: # Поэлементное произведение:
      prod(a)

[13]: 561842749440

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

[15]: 1×3 Matrix{Int64}:
       12096  7680  6048

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

[17]: 4×1 Matrix{Int64}:
       2880
       2016
        144
        672
```

Рис. 3.2: Поэлементное произведение

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics: (рис. 3.3)

```
[19]: # Подключение пакета Statistics:
      import Pkg
      Pkg.add("Statistics")

          Updating registry at `C:\Users\galin\.julia\registries\General.toml`
         Resolving package versions...
          Updating `C:\Users\galin\.julia\environments\v1.10\Project.toml`
        [10745b16] + Statistics v1.10.0
        No Changes to `C:\Users\galin\.julia\environments\v1.10\Manifest.toml`

[21]: using Statistics

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

[23]: 11.0

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

[25]: 1×3 Matrix{Float64}:
       11.0  11.75  10.25

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

[27]: 4×1 Matrix{Float64}:
       14.666666666666666
       13.0
        7.333333333333333
        9.0
```

Рис. 3.3: mean

2. Далее перешла к рассмотрению примеров по теме “Транспонирование, след, ранг, определитель и инверсия матрицы”.

Для выполнения таких операций над матрицами, как транспонирование, диагонализация, определение следа, ранга, определителя матрицы и т.п. можно воспользоваться библиотекой (пакетом) LinearAlgebra: (рис. 3.4) (рис. 3.5)

```
[29]: # Подключение пакета LinearAlgebra:
      import Pkg
      Pkg.add("LinearAlgebra")
      using LinearAlgebra

        Resolving package versions...
         Updating `C:\Users\galin\.julia\environments\v1.10\Project.toml`
        [37e2e46d] + LinearAlgebra
        No Changes to `C:\Users\galin\.julia\environments\v1.10\Manifest.toml`

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

[31]: 4x4 Matrix{Int64}:
        2   2  14  17
       12  13   2  16
       19   6  14  17
        6   7  10   1

[33]: # Транспонирование:
      transpose(b)

[33]: 4x4 transpose(::Matrix{Int64}) with eltype Int64:
        2  12  19   6
        2  13   6   7
       14   2  14  10
       17  16  17   1

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

[35]: 30

[37]: # Извлечение диагональных элементов как массив:
      diag(b)

[37]: 4-element Vector{Int64}:
        2
       13
       14
        1
```

Рис. 3.4: Транспонирование, след матрицы, извлечение диагональных элементов как массив

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

[39]: 4

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

[41]: 4x4 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]: 3x4 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. 3.5: Ранг, инверсия, определитель, псевдобратная

3. Перешла к разделу “Вычисление нормы векторов и матриц, повороты, вращения”.

Вычисление евклидовой нормы и p-нормы. (рис. 3.6)

```
[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. 3.6: Нормы для векторов

Расстояние и угол между двумя векторами X и Y. (рис. 3.7)

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

[53]: 9.486832980505138

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

[55]: 9.486832980505138

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

[57]: 2.4404307889469252
```

Рис. 3.7: Два вектора

Вычисление нормы для двумерной матрицы: (рис. 3.8)

```
[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. 3.8: Нормы для матрицы

Повороты матриц. (рис. 3.9)

```
[65]: # Поворот на 180 градусов:
      rot180(d)

[65]: 3×3 Matrix{Int64}:
       0   1  -2
       3   2  -1
       2  -4   5

[67]: # Переворачивание строк:
      reverse(d,dims=1)

[67]: 3×3 Matrix{Int64}:
       -2   1  0
       -1   2  3
        5  -4  2

[69]: # Переворачивание столбцов
      reverse(d,dims=2)

[69]: 3×3 Matrix{Int64}:
       2  -4   5
       3   2  -1
       0   1  -2
```

Рис. 3.9: Поворот

4. Разобрала примеры раздела ” Матричное умножение, единичная матрица, скалярное произведение”.

Произведение матриц A и B: (рис. 3.10)

```
[71]: # Матрица 2x3 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 3x4 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 3x3:
      Matrix{Int}(I, 3, 3)

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

Рис. 3.10: Произведение

Скалярное произведение векторов X и Y: (рис. 3.11)

```
[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. 3.11: Скалярное произведение

5. Далее тема “Факторизация. Специальные матричные структуры”.

Решение систем линейный алгебраических уравнений ⊠⊠ = ⊠: (рис. 3.12)

```
[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. 3.12: Ax = b

Julia позволяет вычислять LU-факторизацию и определяет составной тип факторизации для его хранения: (рис. 3.13)

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

[91]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3x3 Matrix{Float64}:
       1.0       0.0       0.0
       0.150297  1.0       0.0
       0.346787  0.513491  1.0
      U factor:
      3x3 Matrix{Float64}:
       0.564215  0.269493  0.101435
       0.0       0.663873  0.112155
       0.0       0.0       0.201737

[93]: # Матрица перестановок:
      Alu.P

[93]: 3x3 Matrix{Float64}:
       1.0  0.0  0.0
       0.0  0.0  1.0
       0.0  1.0  0.0

[95]: # Вектор перестановок:
      Alu.p

[95]: 3-element Vector{Int64}:
       1
       3
       2

[97]: # Матрица L:
      Alu.L

[97]: 3x3 Matrix{Float64}:
       1.0       0.0       0.0
       0.150297  1.0       0.0
       0.346787  0.513491  1.0

[99]: # Матрица U:
      Alu.U

[99]: 3x3 Matrix{Float64}:
       0.564215  0.269493  0.101435
       0.0       0.663873  0.112155
       0.0       0.0       0.201737
```

Рис. 3.13: LU

Исходная система уравнений ⊠⊠ = ⊠ может быть решена или с использованием исходной матрицы, или с использованием объекта факторизации: (рис. 3.14)

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

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

[103]: # Решение СЛАУ через обьект 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 обьект факторизации:
       det(Alu)

[107]: -0.07556391453190962
```

Рис. 3.14: СЛАУ

Julia позволяет вычислять QR-факторизацию и определяет составной тип факторизации для его хранения: (рис. 3.15)

```
[111]: # QR-факторизация:
       Aqr = qr(A)

[111]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
       Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
       R factor:
       3×3 Matrix{Float64}:
        -0.60317  -0.492015  -0.208329
         0.0      -0.717881  -0.198487
         0.0       0.0       -0.174511

[113]: # Матрица Q:
       Aqr.Q

[113]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}

[115]: # Матрица R:
       Aqr.R

[115]: 3×3 Matrix{Float64}:
        -0.60317  -0.492015  -0.208329
         0.0      -0.717881  -0.198487
         0.0       0.0       -0.174511

[117]: # Проверка, что матрица Q - ортогональная:
       Aqr.Q'*Aqr.Q

[117]: 3×3 Matrix{Float64}:
         1.0          0.0  2.77556e-17
         0.0          1.0  1.11022e-16
        -2.77556e-17  0.0  1.0
```

Рис. 3.15: QR

Примеры собственной декомпозиции матрицы ⊠: (рис. 3.16)

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

[119]: 3×3 Matrix{Float64}:
        1.12843   0.465155  0.186235
        0.465155  0.868698  0.99888
        0.186235  0.99888   0.254801

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

[121]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
       values:
       3-element Vector{Float64}:
        -0.49384551866683535
         0.8357564386724663
         1.9100190147731346
       vectors:
       3×3 Matrix{Float64}:
         0.0835321   0.843768  -0.530168
        -0.607566   -0.37858   -0.698241
         0.789865   -0.380438  -0.481021

[123]: # Собственные значения:
       AsymEig.values

[123]: 3-element Vector{Float64}:
        -0.49384551866683535
         0.8357564386724663
         1.9100190147731346

[125]: #Собственные векторы:
       AsymEig.vectors

[125]: 3×3 Matrix{Float64}:
         0.0835321   0.843768  -0.530168
        -0.607566   -0.37858   -0.698241
         0.789865   -0.380438  -0.481021

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

[127]: 3×3 Matrix{Float64}:
         1.0          -8.32667e-16   1.40166e-15
         3.05311e-16   1.0          -4.71845e-15
        -3.05311e-16  -2.44249e-15   1.0
```

Рис. 3.16: Декомпозиция

Далее рассмотрим примеры работы с матрицами большой размерности и специальной структуры. (рис. 3.17)

```
[133]: # Матрица 1000 х 1000:
       n = 1000
       A = randn(n,n)
       # Симметризация матрицы:
       Asym = A + A'
```

```
[133]: 1000×1000 Matrix{Float64}:
 -0.417418  -0.594911  -2.1929      …  -0.274394   1.40063   -0.280964
 -0.594911  -1.22479   -0.144743        0.682625   1.6968     0.772137
 -2.1929    -0.144743  -2.44639        -1.22907   -1.16158   -0.348319
 -0.46608   -2.6828    -1.22889        -0.185428  -0.884075  -0.769141
 -0.126064   0.539408  -1.61926        -0.910877  -0.437402  -0.324212
  0.337017   0.523156   0.749698    …   0.526156   0.197285  -0.174337
  0.648176   0.145842   3.27901         2.59181    3.96276    1.94242
  0.703116   1.63807    1.5348         -1.57456    3.15766   -2.13444
  0.693767   0.803645  -3.63035         0.507219   0.0974712  0.471777
  0.313918  -2.89958    0.812097       -1.08361   -2.0067    -0.907463
  0.464385   0.127514  -1.6425      …   1.32788   -0.783623  -0.740144
 -0.783952  -0.229574   0.872381        1.76978    1.06422   -1.04115
 -2.20793   -0.509557  -1.26199         1.06351   -0.24933   -0.592422
  ⋮                                 ⋱
  1.71483   -1.24758   -0.00546236      1.02375   -0.632476   0.455259
 -3.63169    0.476929   2.92118        -0.550525  -0.537936  -0.0149842
  0.817758   0.779607   0.39907        -1.28412    1.00758   -1.06239
```

```
[135]: # Проверка, является ли матрица симметричной:
       issymmetric(Asym)

[135]: true
```

Рис. 3.17: Большие матрицы

Пример добавления шума в симметричную матрицу (матрица уже не будет симметричной): (рис. 3.18)

```
[137]: # Добавление шума:
       Asym_noisy = copy(Asym)
```

```
[137]: 1000×1000 Matrix{Float64}:
 -0.417418  -0.594911  -2.1929      …  -0.274394   1.40063   -0.280964
 -0.594911  -1.22479   -0.144743        0.682625   1.6968     0.772137
 -2.1929    -0.144743  -2.44639        -1.22907   -1.16158   -0.348319
 -0.46608   -2.6828    -1.22889        -0.185428  -0.884075  -0.769141
 -0.126064   0.539408  -1.61926        -0.910877  -0.437402  -0.324212
  0.337017   0.523156   0.749698    …   0.526156   0.197285  -0.174337
  0.648176   0.145842   3.27901         2.59181    3.96276    1.94242
  0.703116   1.63807    1.5348         -1.57456    3.15766   -2.13444
  0.693767   0.803645  -3.63035         0.507219   0.0974712  0.471777
  0.313918  -2.89958    0.812097       -1.08361   -2.0067    -0.907463
  0.464385   0.127514  -1.6425      …   1.32788   -0.783623  -0.740144
 -0.783952  -0.229574   0.872381        1.76978    1.06422   -1.04115
 -2.20793   -0.509557  -1.26199         1.06351   -0.24933   -0.592422
  ⋮                                 ⋱
  1.71483   -1.24758   -0.00546236      1.02375   -0.632476   0.455259
 -3.63169    0.476929   2.92118        -0.550525  -0.537936  -0.0149842
  0.817758   0.779607   0.39907        -1.28412    1.00758   -1.06239
```

```
[139]: Asym_noisy[1,2] += 5eps()

[139]: -0.5949111599476395
```

```
[141]: # Проверка, является ли матрица симметричной:
       issymmetric(Asym_noisy)

[141]: false
```

Рис. 3.18: Добавление шума

В Julia можно объявить структуру матрица явно, например, используя Diagonal, Triangular, Symmetric, Hermitian, Tridiagonal и SymTridiagonal: (рис. 3.19)

```
[143]: # Явно указываем, что матрица является симметричной:
       Asym_explicit = Symmetric(Asym_noisy)

[143]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
        -0.417418  -0.594911  -2.1929      …  -0.274394   1.40063    -0.280964
        -0.594911  -1.22479   -0.144743        0.682625   1.6968      0.772137
        -2.1929    -0.144743  -2.44639        -1.22907   -1.16158    -0.348319
        -0.46608   -2.6828    -1.22889        -0.185428  -0.884075   -0.769141
        -0.126064   0.539408  -1.61926        -0.910877  -0.437402   -0.324212
         0.337017   0.523156   0.749698    …   0.526156   0.197285   -0.174337
         0.648176   0.145842   3.27901         2.59181    3.96276     1.94242
         0.703116   1.63807    1.5348         -1.57456    3.15766    -2.13444
         0.693767   0.803645  -3.63035         0.507219   0.0974712   0.471777
         0.313918  -2.89958    0.812097       -1.08361   -2.0067     -0.907463
         0.464385   0.127514  -1.6425      …   1.32788   -0.783623   -0.740144
        -0.783952  -0.229574   0.872381        1.76978    1.06422    -1.04115
        -2.20793   -0.509557  -1.26199         1.06351   -0.24933    -0.592422
         ⋮                                 ⋱
         1.71483   -1.24758   -0.00546236      1.02375   -0.632476    0.455259
        -3.63169    0.476929   2.92118        -0.550525  -0.537936   -0.0149842
         0.817758   0.779607   0.39907        -1.28412    1.00758    -1.06239
```

Рис. 3.19: Структура матрицы

Далее для оценки эффективности выполнения операций над матрицами большой размерности и специальной структуры воспользуемся пакетом BenchmarkTools: (рис. 3.20)

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

          Resolving package versions...
          Installed BenchmarkTools ─ v1.5.0
           Updating `C:\Users\galin\.julia\environments\v1.10\Project.toml`
         [6e4b80f9] + BenchmarkTools v1.5.0
           Updating `C:\Users\galin\.julia\environments\v1.10\Manifest.toml`
         [6e4b80f9] + BenchmarkTools v1.5.0
         [9abbd945] + Profile
       Precompiling project...
         ✓ BenchmarkTools
         1 dependency successfully precompiled in 8 seconds. 27 already precompiled.

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

         80.088 ms (11 allocations: 7.99 MiB)

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

         564.229 ms (14 allocations: 7.93 MiB)

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

         79.853 ms (11 allocations: 7.99 MiB)
```

Рис. 3.20: Эффективность

Использование типов Tridiagonal и SymTridiagonal для хранения трёхдиагональных матриц позволяет работать с потенциально очень большими трёхдиагональными матрицами: (рис. 3.21)

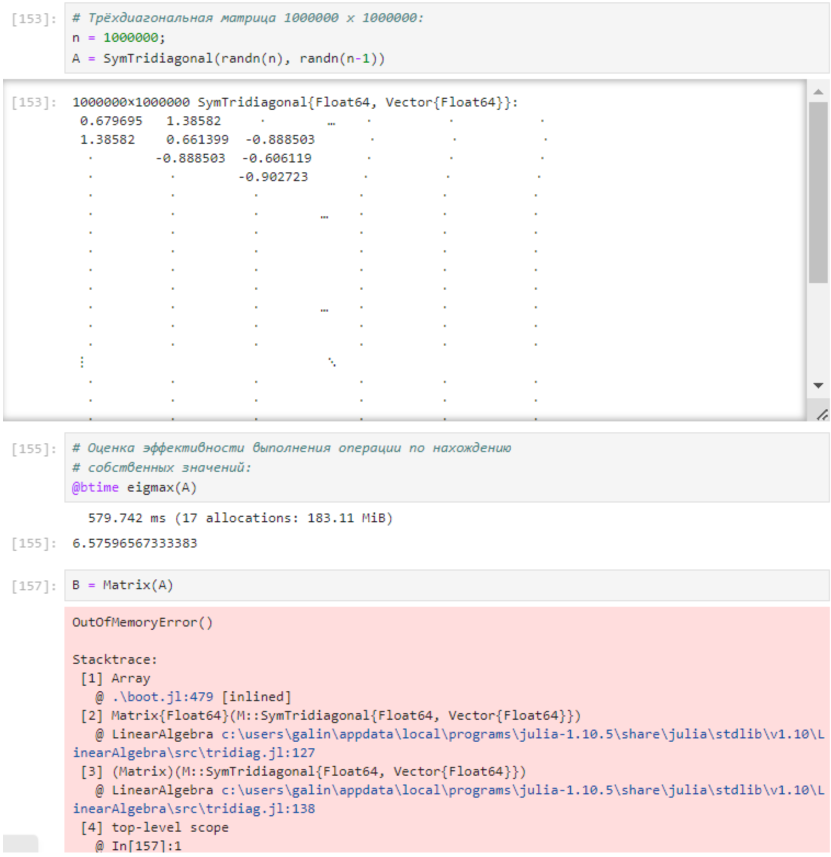

Рис. 3.21: Трёхдиагональные матрицы

6. Следующий раздел “Общая линейная алгебра”.

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): (рис. 3.22) (рис. 3.23)

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

[159]: 3×3 Matrix{Rational{BigInt}}:
        9//10  4//5  1//10
         1     1//2  4//5
        7//10  1//5  2//5

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

[161]: 3-element Vector{Int64}:
        1
        1
        1

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

[163]: 3-element Vector{Rational{BigInt}}:
          9//5
        23//10
        13//10

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

[165]: 3-element Vector{Rational{BigInt}}:
        1
        1
        1
```

Рис. 3.22: Рационыльные элементы 1

```
[167]: # LU-разложение:
       lu(Arational)

[167]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
       L factor:
       3×3 Matrix{Rational{BigInt}}:
         1       0    0
        9//10    1    0
        7//10  -3//7  1
       U factor:
       3×3 Matrix{Rational{BigInt}}:
        1  1//2      4//5
        0  7//20   -31//50
        0   0     -149//350
```

Рис. 3.23: Рационыльные элементы 2

7. Перешла к выполнению задания для самостоятельного выполнения. Переписывать задания не буду. Нумерация сохраняется.

- Задание 1.1 - 1.2 (рис. 3.24)

```
[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. 3.24: Задание 1.1 - 1.2

- Задание 2.1.a - 2.1.e (рис. 3.25)

```
[203]: # Задание 2.1.a
       A = [1 1; 1 -1]
       b = [2; 3]
       x_y = A\b

[203]: 2-element Vector{Float64}:
         2.5
        -0.5

[217]: # Задание 2.1.b
       A = [1 1; 2 2.01]
       b = [2; 4.01]
       x_y = A \ b

[217]: 2-element Vector{Float64}:
        1.0
        1.0

[219]: # Задание 2.1.c
       A = [1 1; 2 2.01]
       b = [2; 5.01]
       x_y = A\b

[219]: 2-element Vector{Float64}:
        -99.00000000000213
        101.00000000000213

[207]: # Задание 2.1.d
       A = [1 1; 2 2; 3 3]
       b = [1; 2; 3]
       x_y = A\b

[207]: 2-element Vector{Float64}:
        0.5
        0.5

[209]: # Задание 2.1.e
       A = [1 1; 2 1; 1 -1]
       b = [2; 1; 3]
       x_y = A\b

[209]: 2-element Vector{Float64}:
         1.5000000000000004
        -0.9999999999999997
```

Рис. 3.25: Задание 2.1.a - 2.1.e

- Задание 2.1.f (рис. 3.26)

```
[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. 3.26: Задание 2.1.f

- Задание 2.2.a - 2.2.d (рис. 3.27)

```
[223]: # Задание 2.2.a
       A = [1 1 1; 1 -1 -2]
       b = [2; 3]
       x_y_z = A\b

[223]: 3-element Vector{Float64}:
         2.214285714285715
         0.35714285714285715
        -0.5714285714285711

[227]: # Задание 2.2.b
       A = [1 1 1; 2 2 3; 3 1 1]
       b = [2; 4; 1]
       x_y_z = A\b

[227]: 3-element Vector{Float64}:
        -0.5
         2.5
        -0.0

[243]: # Задание 2.2.c
       A = [1.01 1 1; 1 1 2.01; 2 2 3.01]
       b = [1; 0; 1]
       x_y_z = A\b

[243]: 3-element Vector{Float64}:
         2.220446049250313e-16
         1.99009900990099
        -0.9900990099009903

[253]: # Задание 2.2.d
       A = [1 1.01 1; 1 1 2.01; 2 2 3.01]
       b = [1; 0; 0]
       x_y_z = A\b

[253]: 3-element Vector{Float64}:
        -99.99999999999991
         99.99999999999991
          0.0
```

Рис. 3.27: Задание 2.2.a - 2.2.d

- Задание 3.1.a - 3.1.c (рис. 3.28)

```
[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. 3.28: Задание 3.1.a - 3.1.c

- Задание 3.2.a - 3.2.d (рис. 3.29)

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

[261]: 2x2 Matrix{Int64}:
         29525  -29524
        -29524   29525

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

[265]: 2x2 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]: 2x2 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]: 2x2 Matrix{ComplexF64}:
        0.568864+0.351578im  0.920442-0.217287im
        0.920442-0.217287im   1.48931+0.134291im
```

Рис. 3.29: Задание 3.2.a - 3.2.d

- Задание 3.3 (рис. 3.30)

```
[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
```

Рис. 3.30: Задание 3.3

- Задание 4.1 (рис. 3.31)

```
[281]: # Задание 4.1
function productive(A, y)
    E = I
    B = E - A
    if det(B) <= 0
        return false
    end
    x = B\y
    return all(x.>=0)
end

# a
A = [1 2; 3 4]
y = [1; 1]
productive(A, y)

[281]: false

[285]: # b
A = 0.5 * [1 2; 3 4]
productive(A, y)

[285]: false

[287]: # c
A = 0.1 * [1 2; 3 4]
productive(A, y)

[287]: true
```

Рис. 3.31: Задание 4.1

- Задание 4.2 (рис. 3.32)

```
[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. 3.32: Задание 4.2

- Задание 4.3 (рис. 3.33)

```
[297]: # Задание 4.3
function productive_3(A)
    eigevalues = eigen(A).values
    return all(abs.(eigevalues) .< 1)
end

# a
A = [1 2; 3 1]
productive_3(A)

[297]: false

[299]: # b
A = 0.5 * [1 2; 3 1]
productive_3(A)

[299]: false

[301]: # c
A = 0.1 * [1 2; 3 1]
productive_3(A)

[301]: true

[303]: # d
A = [0.1 0.2 0.3;0 0.1 0.2; 0 0.1 0.3]
productive_3(A)

[303]: true
```

Рис. 3.33: Задание 4.3

4 Вывод

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