Лабораторная работа №4

Компьютерный практикум по статистическому анализу данных

Николаев Дмитрий Иванович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

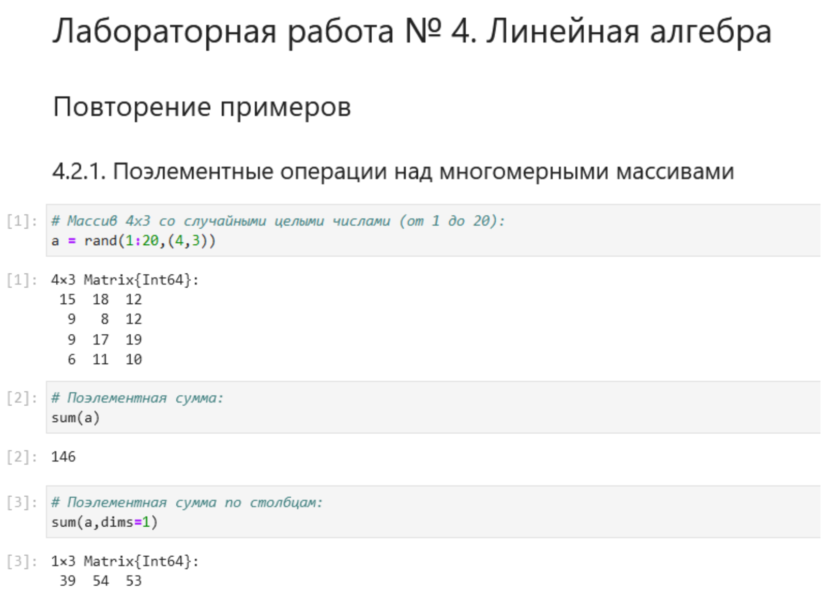
Основной целью работы является изучение возможностей специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

2 Выполнение лабораторной работы

Выполняем задания согласно указаниям [1].

2.1 Повторение примеров

Повторим примеры, представленные в лабораторной работе. Поэлементные операции над многомерными массивами ([2.1-2.3]); транспонирование, след, ранг, определитель и инверсия матрицы ([2.4,2.5]); вычисление нормы векторов и матриц, повороты, вращения ([2.6-2.8]); матричное умножение, единичная матрица, скалярное произведение и массивами ([2.9,2.10]); факторизация, специальные матричные структуры ([2.11-2.24]) и общая линейная алгебра ([2.25-2.26]).



Лабораторная работа № 4. Линейная алгебра

Повторение примеров

4.2.1. Поэлементные операции над многомерными массивами

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

[1]: 4×3 Matrix{Int64}:
      15  18  12
       9   8  12
       9  17  19
       6  11  10

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

[2]: 146

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

[3]: 1×3 Matrix{Int64}:
      39  54  53
```

Рис. 2.1: Поэлементные операции над многомерными массивами (1)

[4]:
```julia
# Поэлементная сумма по строкам:
sum(a,dims=2)
```

[4]:
```
4×1 Matrix{Int64}:
 45
 29
 45
 27
```

[5]:
```julia
# Поэлементное произведение:
prod(a)
```

[5]:
```
5370908083200
```

[6]:
```julia
# Поэлементное произведение по столбцам:
prod(a,dims=1)
```

[6]:
```
1×3 Matrix{Int64}:
 7290  26928  27360
```

[7]:
```julia
# Поэлементное произведение по строкам:
prod(a,dims=2)
```

[7]:
```
4×1 Matrix{Int64}:
 3240
  864
 2907
  660
```

Рис. 2.2: Поэлементные операции над многомерными массивами (2)

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics:

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

    Updating registry at `C:\Users\User\.julia\registries\General.toml`
   Resolving package versions...
    Updating `C:\Users\User\.julia\environments\v1.8\Project.toml`
  [10745b16] + Statistics
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`

[9]: using Statistics

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

[10]: 12.166666666666666

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

[11]: 1×3 Matrix{Float64}:
       9.75  13.5  13.25

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

[12]: 4×1 Matrix{Float64}:
       15.0
        9.666666666666666
       15.0
        9.0
```

Рис. 2.3: Поэлементные операции над многомерными массивами (3)

4.2.2. Транспонирование, след, ранг, определитель и инверсия матрицы

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

   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`

[15]: using LinearAlgebra

[16]: # Массив 4х4 со случайными целыми числами (от 1 до 20):
      b = rand(1:20,(4,4))

[16]: 4×4 Matrix{Int64}:
       9   1  13   3
       6  18   8  17
       9  12  10  19
       1  19  20   9

[17]: # Транспонирование:
      transpose(b)

[17]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
        9   6   9   1
        1  18  12  19
       13   8  10  20
        3  17  19   9

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

[18]: 46
```

Рис. 2.4: Транспонирование, след, ранг, определитель и инверсия матрицы (1)

[19]:
```
# Извлечение диагональных элементов как массив:
diag(b)
```

[19]:
```
4-element Vector{Int64}:
  9
 18
 10
  9
```

[20]:
```
# Ранг матрицы:
rank(b)
```

[20]:
```
4
```

[21]:
```
# Инверсия матрицы (определение обратной матрицы):
inv(b)
```

[21]:
```
4×4 Matrix{Float64}:
  0.142988     0.19193   -0.149266   -0.0950814
  0.039861     0.17712   -0.155848   -0.0188334
 -0.00420552  -0.113488   0.0730176   0.06162
 -0.090693    -0.143049   0.183336    0.0245017
```

[22]:
```
# Определитель матрицы:
det(b)
```

[22]:
```
-16406.999999999996
```

[23]:
```
# Псевдобратная функция для прямоугольных матриц:
pinv(a)
```

[23]:
```
3×4 Matrix{Float64}:
  0.0681024   0.133993  -0.0948392  -0.0623202
  0.052915   -0.174364   0.041201    0.0674566
 -0.0838008   0.096607   0.0501025  -0.0105621
```

Рис. 2.5: Транспонирование, след, ранг, определитель и инверсия матрицы (2)

4.2.3. Вычисление нормы векторов и матриц, повороты, вращения

Для вычисления нормы используется LinearAlgebra.norm(x).

Евклидова норма:

$$||\overline{X}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2};$$

p-норма:

$$||\overline{A}||_p = \left(\sum_{i=1}^{n} |a_i|^p \right)^{\frac{1}{p}}$$

[24]:
```
# Создание вектора X:
X = [2, 4, -5]
```

[24]:
```
3-element Vector{Int64}:
  2
  4
 -5
```

[25]:
```
# Вычисление евклидовой нормы:
norm(X)
```

[25]:
```
6.708203932499369
```

[26]:
```
# Вычисление p-нормы:
p = 1
norm(X,p)
```

[26]:
```
11.0
```

Рис. 2.6: Вычисление нормы векторов и матриц, повороты, вращения (1)

Евклидово расстояние между двумя векторами $\overline{X}$ и $\overline{Y}$ определяется как $||\overline{X} - \overline{Y}||^2$.

[27]:
```
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
norm(X-Y)
```

[27]:
```
9.486832980505138
```

[28]:
```
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))
```

[28]:
```
9.486832980505138
```

Угол между двумя векторами $\overline{X}$ и $\overline{Y}$ определяется как $\cos^{-1} \frac{\overline{X}^T \overline{Y}}{||\overline{X}||^2 ||\overline{Y}||^2}$.

[31]:
```
# Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))
```

[31]:
```
2.4404307889469252
```

Вычисление нормы для двумерной матрицы:

[32]:
```
# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
```

[32]:
```
3×3 Matrix{Int64}:
  5  -4  2
 -1   2  3
 -2   1  0
```

Рис. 2.7: Вычисление нормы векторов и матриц, повороты, вращения (2)

```
[33]: # Вычисление Евклидовой нормы:
      opnorm(d)

[33]: 7.147682841795258

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

[34]: 8.0

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

[35]: 3×3 Matrix{Int64}:
       0   1  -2
       3   2  -1
       2  -4   5

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

[36]: 3×3 Matrix{Int64}:
       -2   1  0
       -1   2  3
        5  -4  2

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

[37]: 3×3 Matrix{Int64}:
       2  -4   5
       3   2  -1
       0   1  -2
```

Рис. 2.8: Вычисление нормы векторов и матриц, повороты, вращения (3)

4.2.4. Матричное умножение, единичная матрица, скалярное произведение

```
[38]: # Матрица 2x3 со случайными целыми значениями от 1 до 10:
      A = rand(1:10,(2,3))

[38]: 2×3 Matrix{Int64}:
       2  9  2
       8  2  4

[39]: # Матрица 3x4 со случайными целыми значениями от 1 до 10:
      B = rand(1:10,(3,4))

[39]: 3×4 Matrix{Int64}:
       10   9  4   3
        5  10  9   7
        1  10  7  10

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

[40]: 2×4 Matrix{Int64}:
       67  128  103  89
       94  132   78  78

[41]: # Единичная матрица 3x3:
      Matrix{Int}(I, 3, 3)

[41]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1
```

Рис. 2.9: Матричное умножение, единичная матрица, скалярное произведение и массивами (1)

```
[43]: # Скалярное произведение векторов X и Y:
      X = [2, 4, -5]

[43]: 3-element Vector{Int64}:
        2
        4
       -5

[44]: Y = [1,-1,3]

[44]: 3-element Vector{Int64}:
        1
       -1
        3

[45]: dot(X,Y)

[45]: -17

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

[46]: -17
```

Рис. 2.10: Матричное умножение, единичная матрица, скалярное произведение и массивами (2)

4.2.5. Факторизация. Специальные матричные структуры

Решение систем линейный алгебраических уравнений $Ax = b$:

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

```
[47]: 3×3 Matrix{Float64}:
       0.0244786  0.295394  0.719576
       0.470582   0.665976  0.354524
       0.494353   0.586467  0.937499
```

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

```
[48]: 3-element Vector{Float64}:
       1.0
       1.0
       1.0
```

```
[49]: # Задаём вектор b:
      b = A*x
```

```
[49]: 3-element Vector{Float64}:
       1.039448668198395
       1.4910819593511615
       2.018319207610515
```

Рис. 2.11: Факторизация, специальные матричные структуры (1)

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

```
[50]: 3-element Vector{Float64}:
       0.9999999999999998
       0.9999999999999998
       1.0000000000000002
```

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

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

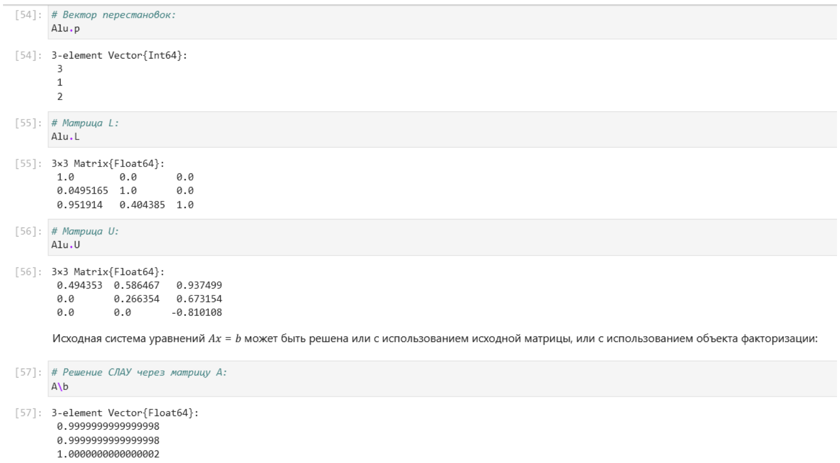
```
[51]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3×3 Matrix{Float64}:
       1.0        0.0       0.0
       0.0495165  1.0       0.0
       0.951914   0.404385  1.0
      U factor:
      3×3 Matrix{Float64}:
       0.494353  0.586467   0.937499
       0.0       0.266354   0.673154
       0.0       0.0       -0.810108
```

Различные части факторизации могут быть извлечены путём доступа к их специальным свойствам:

```
[53]: # Матрица перестановок:
      Alu.P
```

```
[53]: 3×3 Matrix{Float64}:
       0.0  0.0  1.0
       1.0  0.0  0.0
       0.0  1.0  0.0
```

Рис. 2.12: Факторизация, специальные матричные структуры (2)

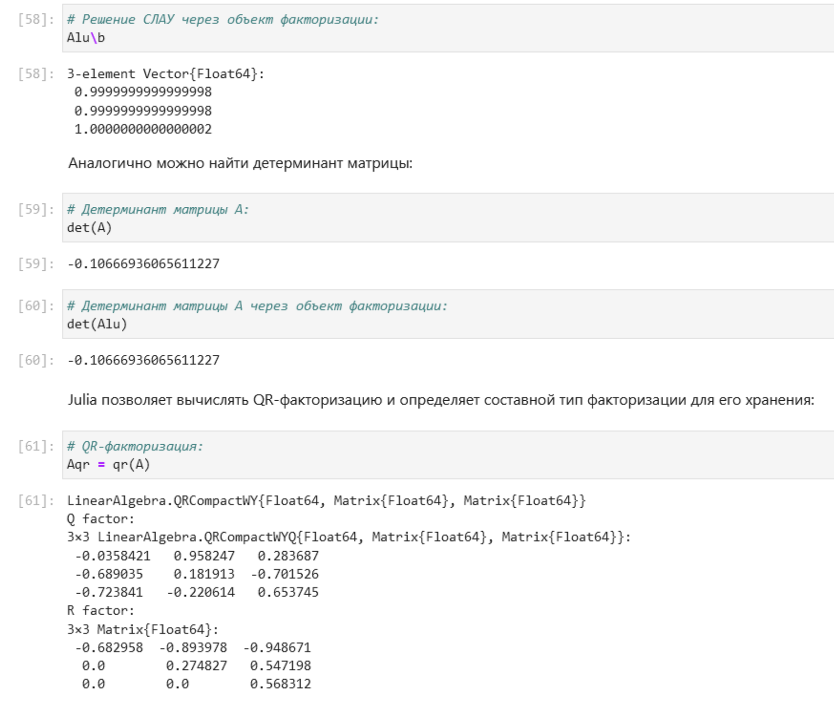


```
[54]: # Вектор перестановок:
      Alu.p

[54]: 3-element Vector{Int64}:
       3
       1
       2

[55]: # Матрица L:
      Alu.L

[55]: 3×3 Matrix{Float64}:
       1.0        0.0       0.0
       0.0495165  1.0       0.0
       0.951914   0.404385  1.0

[56]: # Матрица U:
      Alu.U

[56]: 3×3 Matrix{Float64}:
       0.494353  0.586467   0.937499
       0.0       0.266354   0.673154
       0.0       0.0       -0.810108
```

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

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

[57]: 3-element Vector{Float64}:
       0.9999999999999998
       0.9999999999999998
       1.0000000000000002
```

Рис. 2.13: Факторизация, специальные матричные структуры (3)



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

[58]: 3-element Vector{Float64}:
       0.9999999999999998
       0.9999999999999998
       1.0000000000000002
```

Аналогично можно найти детерминант матрицы:

```
[59]: # Детерминант матрицы A:
      det(A)

[59]: -0.10666936065611227

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

[60]: -0.10666936065611227
```

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

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

[61]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
      Q factor:
      3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
       -0.0358421   0.958247   0.283687
       -0.689035    0.181913  -0.701526
       -0.723841   -0.220614   0.653745
      R factor:
      3×3 Matrix{Float64}:
       -0.682958  -0.893978  -0.948671
        0.0        0.274827   0.547198
        0.0        0.0        0.568312
```

Рис. 2.14: Факторизация, специальные матричные структуры (4)

По аналогии с LU-факторизацией различные части QR-факторизации могут быть извлечены путём доступа к их специальным свойствам:

```
[62]: # Матрица Q:
      Aqr.Q

[62]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
       -0.0358421   0.958247   0.283687
       -0.689035    0.181913  -0.701526
       -0.723841   -0.220614   0.653745

[63]: # Матрица R:
      Aqr.R

[63]: 3×3 Matrix{Float64}:
       -0.682958  -0.893978  -0.948671
        0.0        0.274827   0.547198
        0.0        0.0        0.568312

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

[64]: 3×3 Matrix{Float64}:
       1.0  -2.22045e-16  -1.66533e-16
       0.0   1.0          -2.22045e-16
       0.0  -1.11022e-16   1.0
```

Примеры собственной декомпозиции матрицы A:

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

[65]: 3×3 Matrix{Float64}:
       0.0489572  0.765976  1.21393
       0.765976   1.33195   0.940991
       1.21393    0.940991  1.875
```

Рис. 2.15: Факторизация, специальные матричные структуры (5)

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

[66]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
      values:
      3-element Vector{Float64}:
       -0.6043968330748379
        0.6325671697101596
        3.227737441859591
      vectors:
      3×3 Matrix{Float64}:
        0.908856   0.071752  0.410891
       -0.17568   -0.827605  0.53311
       -0.378308   0.556705  0.739569

[67]: # Собственные значения:
      AsymEig.values

[67]: 3-element Vector{Float64}:
       -0.6043968330748379
        0.6325671697101596
        3.227737441859591

[68]: #Собственные векторы:
      AsymEig.vectors

[68]: 3×3 Matrix{Float64}:
        0.908856   0.071752  0.410891
       -0.17568   -0.827605  0.53311
       -0.378308   0.556705  0.739569
```

Рис. 2.16: Факторизация, специальные матричные структуры (6)

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

[69]: 3×3 Matrix{Float64}:
        1.0          2.66454e-15  -4.21885e-15
       -3.10862e-15  1.0           4.21885e-15
        4.44089e-16  1.4988e-15    1.0
```

Далее рассмотрим примеры работы с матрицами большой размерности и специальной структуры.

Рис. 2.17: Факторизация, специальные матричные структуры (7)

```
[70]: # Матрица 1000 x 1000:
      n = 1000
      A = randn(n,n)

[70]: 1000×1000 Matrix{Float64}:
        0.715121   -1.09378   -0.0644743  …  -0.608474   0.135291    0.477227
       -0.0257378  -0.583642  -0.428233       1.34519    2.10222    -0.69783
       -1.11208    -0.648552  -0.0955454     -1.78696    0.302478    0.156592
        0.0214497  -0.157924  -0.699911      -1.03782   -0.125016   -0.0160884
       -0.457676   -0.218426   1.26587        1.21362   -0.00822816 -0.709229
       -0.857719   -1.41396   -0.939895   …  -1.0237     0.223336    0.289345
        1.93189     0.618262   1.15489       -0.331677   0.297265   -1.37486
       -0.279122   -0.554029  -1.62564        0.592909  -0.444105    0.578417
       -1.01754    -0.46107    0.890266      -1.26405   -1.19266     0.519305
       -0.331716    0.357467  -1.74207       -1.12207   -2.17101    -1.61775
       -0.223045    0.981586   2.33064    …   0.117963   0.845698   -1.74915
        2.37672     0.062367   1.95454       -0.628635   0.950427    0.813157
       -1.28262    -0.859824  -0.708803       2.01096    0.781505    0.976032
        ⋮                                 ⋱
        0.777343    0.389775  -0.0231732     -1.17584   -1.23587     0.134063
       -0.212206    0.765117   0.0413582      0.420782  -1.00319     0.272628
        1.0237      0.612465  -1.8645     …   1.01826   -1.27172    -0.176564
        2.56778    -0.520432  -0.629332      -0.707429  -0.791852    1.57703
       -1.82739     1.66386    0.85783        0.270949   1.29468    -0.282906
        0.0971543   0.24198   -0.803953      -1.32409   -1.06221    -0.362605
       -0.0567814  -0.612306  -0.725842       1.09588   -0.0717177  -1.79129
        1.53218    -0.637768  -1.22399    …   0.559103   1.71369     1.13567
       -1.91554    -2.96016    2.09998        0.467376  -0.314868    0.0106346
        1.45246     1.19208    1.57033        0.980217   0.516328   -0.569184
       -1.03089    -1.23329    0.918618      -0.441951   1.12149     1.04079
        0.406171   -1.23483    2.16231       -1.92173    0.224729    0.376351
```

Рис. 2.18: Факторизация, специальные матричные структуры (8)

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

```
[71]: 1000×1000 Matrix{Float64}:
  1.43024    -1.11952   -1.17655   …   0.84399    -0.895594    0.883398
 -1.11952    -1.16728   -1.07678       2.53726     0.868932   -1.93266
 -1.17655    -1.07678   -0.191091     -0.216623    1.2211      2.3189
  0.0541985   0.823744  -1.53706      -0.785901   -0.559282    1.52263
 -0.387462   -1.30423   -0.445992      1.63118    -1.2959     -2.62596
 -2.03282    -3.47907   -0.784218  …  -1.09252    -0.636919    0.252326
  2.20544     0.96182   -0.174497     -1.87636    -0.492122   -1.81876
 -2.1409     -0.49753   -1.29366      -0.363027    0.294809    0.599053
 -0.651438   -0.107928   2.30529      -1.54557    -1.61155     1.44834
 -2.37908     1.60309   -1.75341       0.826037   -1.76863    -1.13807
  0.328566    0.259349   4.93238   …   0.621055    1.79743    -0.306787
  2.65319     0.454741   0.707302     -1.98627     1.92313     2.08757
 -2.43506    -0.825956  -0.916995      3.03427     2.96398    -0.00659231
  ⋮                                ⋱
 -0.449135    1.44031    0.589127     -0.751309    1.03124     0.131837
  0.913089   -1.07386    2.24422       1.29229    -1.73184     0.930101
  2.61202     1.16403   -1.33515   …  -1.22623    -1.13502    -1.41021
  1.63117    -0.676748  -1.2645       -1.19492     0.662504    2.72126
 -3.38065     1.47896    0.369715     -0.0987206   2.83172    -0.741428
  0.408206    0.317323  -0.470182     -0.587179   -1.35645    -0.550765
 -1.02969    -1.51806   -0.412233      2.81717     1.01752    -2.44839
  0.752629   -1.2514    -2.27441   …  -0.114144    1.97173    -0.49272
 -1.26177    -3.72216    3.59293       1.1674      1.53069    -0.684214
  0.84399     2.53726   -0.216623      1.96043     0.0743765  -2.49091
 -0.895594    0.868932   1.2211        0.0743765   2.24298     1.26552
  0.883398   -1.93266    2.3189       -2.49091     1.26552     0.752703
```

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

```
[72]: true
```

Рис. 2.19: Факторизация, специальные матричные структуры (9)

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

```
[73]: # Добавление шума:
      Asym_noisy = copy(Asym)
      Asym_noisy[1,2] += 5eps()
```

```
[73]: -1.1195158971718326
```

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

```
[74]: false
```

В Julia можно объявить структуру матрица явно, например, используя Diagonal, Triangular, Symmetric, Hermitian, Tridiagonal и SymTridiagonal:

Рис. 2.20: Факторизация, специальные матричные структуры (10)

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

[75]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
        1.43024    -1.11952    -1.17655   …   0.84399    -0.895594    0.883398
       -1.11952    -1.16728    -1.07678       2.53726     0.868932   -1.93266
       -1.17655    -1.07678    -0.191091     -0.216623    1.2211      2.3189
        0.0541985   0.823744   -1.53706      -0.785901   -0.559282    1.52263
       -0.387462   -1.30423    -0.445992      1.63118    -1.2959     -2.62596
       -2.03282    -3.47907    -0.784218  …  -1.09252    -0.636919    0.252326
        2.20544     0.96182    -0.174497     -1.87636    -0.492122   -1.81876
       -2.1409     -0.49753    -1.29366      -0.363027    0.294809    0.599053
       -0.651438   -0.107928    2.30529      -1.54557    -1.61155     1.44834
       -2.37908     1.60309    -1.75341       0.826037   -1.76863    -1.13807
        0.328566    0.259349    4.93238   …   0.621055    1.79743    -0.306787
        2.65319     0.454741    0.707302     -1.98627     1.92313     2.08757
       -2.43506    -0.825956   -0.916995      3.03427     2.96398    -0.00659231
        ⋮                                 ⋱
       -0.449135    1.44031     0.589127     -0.751309    1.03124     0.131837
        0.913089   -1.07386     2.24422       1.29229    -1.73184     0.930101
        2.61202     1.16403    -1.33515   …  -1.22623    -1.13502    -1.41021
        1.63117    -0.676748   -1.2645       -1.19492     0.662504    2.72126
       -3.38065     1.47896     0.369715     -0.0987206   2.83172    -0.741428
        0.408206    0.317323   -0.470182     -0.587179   -1.35645    -0.550765
       -1.02969    -1.51806    -0.412233      2.81717     1.01752    -2.44839
        0.752629   -1.2514     -2.27441   …  -0.114144    1.97173    -0.49272
       -1.26177    -3.72216     3.59293       1.1674      1.53069    -0.684214
        0.84399     2.53726    -0.216623      1.96043     0.0743765  -2.49091
       -0.895594    0.868932    1.2211        0.0743765   2.24298     1.26552
        0.883398   -1.93266     2.3189       -2.49091     1.26552     0.752703
```

Рис. 2.21: Факторизация, специальные матричные структуры (11)

Далее для оценки эффективности выполнения операций над матрицами большой размерности и специальной структуры воспользуемся пакетом BenchmarkTools:

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

        Resolving package versions...
       No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
       No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`

[77]: using BenchmarkTools

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

       75.297 ms (11 allocations: 7.99 MiB)

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

       454.170 ms (13 allocations: 7.92 MiB)

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

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

Рис. 2.22: Факторизация, специальные матричные структуры (12)

Далее рассмотрим примеры работы с разряженными матрицами большой размерности.

Использование типов Tridiagonal и SymTridiagonal для хранения трёхдиагональных матриц позволяет работать с потенциально очень большими трёхдиагональными матрицами:

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

```
[83]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
 -1.00882    -0.0972623    ⋅        …   ⋅          ⋅          ⋅
 -0.0972623   2.33981    -2.0367        ⋅          ⋅          ⋅
   ⋅         -2.0367      1.86911       ⋅          ⋅          ⋅
   ⋅           ⋅          1.14463       ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅        …   ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅        …   ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
  ⋮                                 ⋱
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅        …   ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅            ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅        …   ⋅          ⋅          ⋅
   ⋅           ⋅           ⋅           1.07083     ⋅          ⋅
   ⋅           ⋅           ⋅           1.3813    -0.385808    ⋅
   ⋅           ⋅           ⋅          -0.385808   0.0896784   0.258891
   ⋅           ⋅           ⋅            ⋅          0.258891  -0.561995
```

Рис. 2.23: Факторизация, специальные матричные структуры (13)

```
[84]: # Оценка эффективности выполнения операции по нахождению
      # собственных значений:
      @btime eigmax(A)

        426.209 ms (17 allocations: 183.11 MiB)

[84]: 6.749672457003237
```

При попытке задать подобную матрицу обычным способом и посчитать её собственные значения, вы скорее всего получите ошибку переполнения памяти:

```
[85]: B = Matrix(A)

OutOfMemoryError()

Stacktrace:
 [1] Array
   @ .\boot.jl:461 [inlined]
 [2] Array
   @ .\boot.jl:469 [inlined]
 [3] zeros
   @ .\array.jl:588 [inlined]
 [4] zeros
   @ .\array.jl:584 [inlined]
 [5] Matrix{Float64}(M::SymTridiagonal{Float64, Vector{Float64}})
   @ LinearAlgebra C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\tridiag.jl:127
 [6] (Matrix)(M::SymTridiagonal{Float64, Vector{Float64}})
   @ LinearAlgebra C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\tridiag.jl:137
 [7] top-level scope
   @ In[85]:1
```

Рис. 2.24: Факторизация, специальные матричные структуры (14)

4.2.6. Общая линейная алгебра

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

[86]: 3×3 Matrix{Rational{BigInt}}:
       1//10  1//1   1//5
       1//10  3//10  1//1
       3//5   9//10  3//5

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

[87]: 3-element Vector{Int64}:
       1
       1
       1

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

[88]: 3-element Vector{Rational{BigInt}}:
       13//10
        7//5
       21//10
```

Рис. 2.25: Общая линейная алгебра (1)

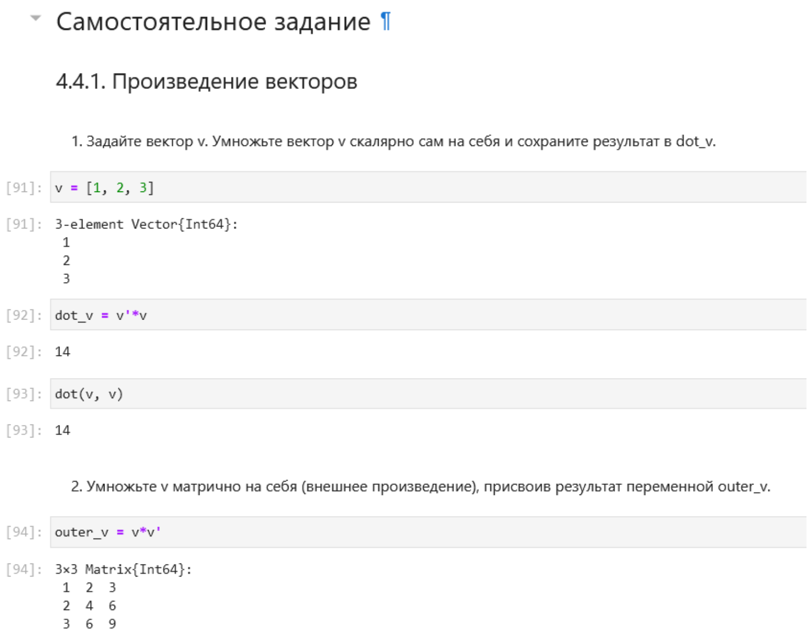
```
[89]: # Решение исходного уравнения получаем с помощью функции \
      # (убеждаемся, что x - единичный вектор):
      Arational\b

[89]: 3-element Vector{Rational{BigInt}}:
       1//1
       1//1
       1//1

[90]: # LU-разложение:
      lu(Arational)

[90]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
      L factor:
      3×3 Matrix{Rational{BigInt}}:
       1//1  0//1   0//1
       1//6  1//1   0//1
       1//6  3//17  1//1
      U factor:
      3×3 Matrix{Rational{BigInt}}:
       3//5   9//10   3//5
       0//1  17//20   1//10
       0//1   0//1   15//17
```

Рис. 2.26: Общая линейная алгебра (2)

2.2 Самостоятельная работа

2.2.1 Произведение векторов

Найдем скалярное и внешнее произведение векторов ([2.27]).


Самостоятельное задание ¶

4.4.1. Произведение векторов

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

```
[91]: v = [1, 2, 3]

[91]: 3-element Vector{Int64}:
       1
       2
       3

[92]: dot_v = v'*v

[92]: 14

[93]: dot(v, v)

[93]: 14
```

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

```
[94]: outer_v = v*v'

[94]: 3×3 Matrix{Int64}:
       1  2  3
       2  4  6
       3  6  9
```

Рис. 2.27: Задание 4.4.1. Произведение векторов

2.2.2 Системы линейных уравнений

Напишем функцию, решающую СЛАУ ([2.28]).

4.4.2. Системы линейных уравнений

1. Решить СЛАУ с двумя неизвестными.

[116]:

```
# Функция для решения СЛАУ с 2 неизвестными
function SLAE(A, b)
    if size(A)[1] == size(A)[2]
        return inv(A)*b
    else
        return pinv(A)*b
    end
end
```

Рис. 2.28: Задание 4.4.2. Функция решения СЛАУ

1. Решаем СЛАУ с двумя неизвестными:

   1) Пункт a ([2.29])

а) $\begin{cases} x + y = 2, \\ x - y = 3. \end{cases}$

```
[117]: A_42_1a = [1 1; 1 -1]

[117]: 2×2 Matrix{Int64}:
        1   1
        1  -1

[118]: b_42_1a = [2, 3]

[118]: 2-element Vector{Int64}:
        2
        3

[119]: x_42_1a = SLAE(A_42_1a, b_42_1a)

[119]: 2-element Vector{Float64}:
         2.5
        -0.5
```

Рис. 2.29: Задание 4.4.2. Номер 1. Пункт а

2) Пункты b, c и d ([2.30])

b) $\begin{cases} x + y = 2, \\ 2x + 2y = 4. \end{cases}$

Данная система линейно зависима, остается уравнение $x + y = 2$, то есть имеем решение $\begin{pmatrix} x \\ 2 - x \end{pmatrix}$

```
[120]: x_42_1b = ["x", "2 - x"]
```

```
[120]: 2-element Vector{String}:
 "x"
 "2 - x"
```

c) $\begin{cases} x + y = 2, \\ 2x + 2y = 5. \end{cases}$

Данная система несовместна, так как $\begin{cases} x + y = 2, \\ x + y = 2.5. \end{cases}$, то есть решений нет

d) $\begin{cases} x + y = 1, \\ 2x + 2y = 2, \\ 3x + 3y = 3. \end{cases}$

Данная система линейно зависима, остается уравнение $x + y = 1$, то есть имеем решение $\begin{pmatrix} x \\ 1 - x \end{pmatrix}$, где $x \in \mathbb{R}$(или $\mathbb{C}$)

Рис. 2.30: Задание 4.4.2. Номер 1. Пункты b, c и d

3) Пункты e и f ([2.31])

e) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ x - y = 3. \end{cases}$

Система несовместна, так как из 1 и 3-го уравнений, имеем $x = 2.5$, $y = -0.5$, и, подставив во 2-е уравнение, получим неверное равенство $5 - 0.5 \neq 1$

```
[121]: A_42_1e = [1 1; 2 1; 1 -1]
       b_42_1e = [2, 1, 3]
       x_42_1e = SLAE(A_42_1e, b_42_1e)
```

```
[121]: 2-element Vector{Float64}:
  1.5000000000000004
 -0.9999999999999998
```

f) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ 3x + 2y = 3. \end{cases}$

```
[122]: A_42_1f = [1 1; 2 1; 3 2]
       b_42_1f = [2, 1, 3]
       x_42_1f = SLAE(A_42_1f, b_42_1f)
```

```
[122]: 2-element Vector{Float64}:
 -0.9999999999999987
  2.9999999999999982
```

Рис. 2.31: Задание 4.4.2. Номер 1. Пункты e и f

2. Решаем СЛАУ с тремя неизвестными:

   1) Пункт a и b ([2.32])

2. Решить СЛАУ с тремя неизвестными.

a) $\begin{cases} x + y + z = 2, \\ x - y - 2z = 3. \end{cases}$

```
[123]: A_42_2a = [1 1 1; 1 -1 -2]
       b_42_2a = [2, 3]
       x_42_2a = SLAE(A_42_2a, b_42_2a)
```

```
[123]: 3-element Vector{Float64}:
         2.2142857142857144
         0.35714285714285676
        -0.5714285714285716
```

b) $\begin{cases} x + y + z = 2, \\ 2x + 2y - 3z = 4, \\ 3x + y + z = 1. \end{cases}$

```
[124]: A_42_2b = [1 1 1; 2 2 -3; 3 1 1]
       b_42_2b = [2, 4, 1]
       x_42_2b = SLAE(A_42_2b, b_42_2b)
```

```
[124]: 3-element Vector{Float64}:
        -0.4999999999999999
         2.4999999999999996
         0.0
```

Рис. 2.32: Задание 4.4.2. Номер 2. Пункты a и b

2) Пункт c ([2.33])

c) $\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 1. \end{cases}$

Система линейно зависима, так как 3-е уравнение получим из сумма первых двух. Тогда остается два уравнения $\begin{cases} x + y + z = 1, \\ x + y + 2z = 0. \end{cases}$, откуда получим $z = -1$, тогда $x + y = 2$ и итоговое решение $\begin{pmatrix} x \\ 2 - x \\ -1 \end{pmatrix}$, где $x \in \mathbb{R}(\text{или } \mathbb{C})$

```
[125]: A_42_2c = [1 1 1; 1 1 2; 2 2 3]
       b_42_2c = [1, 0, 1]
       x_42_2c = SLAE(A_42_2c, b_42_2c)
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\factorization.jl:21 [inlined]
 [3] #lu!#170
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\lu.jl:82 [inlined]
 [4] #lu#177
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\lu.jl:279 [inlined]
 [5] lu (repeats 2 times)
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\lu.jl:278 [inlined]
 [6] inv(A::Matrix{Int64})
   @ LinearAlgebra C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\dense.jl:893
 [7] SLAE(A::Matrix{Int64}, b::Vector{Int64})
   @ Main .\In[116]:4
 [8] top-level scope
   @ In[125]:3
```

Рис. 2.33: Задание 4.4.2. Номер 2. Пункт c

3) Пункт d ([2.34])

d) $\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 0. \end{cases}$

Система несовместна, так как из 1 и 2-го уравнений, имеем $z = -1$, $x + y = 2$, и, подставив в 3-е уравнение, получим неверное равенство $4 - 3 \neq 0$

```
126]: A_42_2d = [1 1 1; 1 1 2; 2 2 3]
      b_42_2d = [1, 0, 0]
      x_42_2d = SLAE(A_42_2d, b_42_2d)
```

```
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\factorization.jl:21 [inlined]
 [3] #lu!#170
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\lu.jl:82 [inlined]
 [4] #lu#177
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\lu.jl:279 [inlined]
 [5] lu (repeats 2 times)
   @ C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\lu.jl:278 [inlined]
 [6] inv(A::Matrix{Int64})
   @ LinearAlgebra C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\dense.jl:893
 [7] SLAE(A::Matrix{Int64}, b::Vector{Int64})
   @ Main .\In[116]:4
 [8] top-level scope
   @ In[126]:3
```

Рис. 2.34: Задание 4.4.2. Номер 2. Пункт d

2.2.3 Операции с матрицами

1. Приведем матрицы к диагональному виду:

   1) Пункт a ([2.35,2.36])

4.4.3. Операции с матрицами

1. Приведите приведённые ниже матрицы к диагональному виду

a) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$

```
•[132]: A_43_1a = [1 -2; -2 1]

 [132]: 2×2 Matrix{Float64}:
         -0.707107  -0.707107
         -0.707107   0.707107

 [133]: #A_43_1a_Eig = eigen(A_43_1a)
        A_43_1a_EigVect = eigvecs(A_43_1a)

 [133]: 2×2 Matrix{Float64}:
         -0.707107  -0.707107
         -0.707107   0.707107

•[134]: A_43_1a_Diag = A_43_1a_EigVect' * A_43_1a * A_43_1a_EigVect

 [134]: 2×2 Matrix{Float64}:
         -1.0  0.0
          0.0  3.0
```

Рис. 2.35: Задание 4.4.3. Номер 1. Пункт a (1)

```
[140]: A_43_1a_EigVals = eigvals(A_43_1a)

[140]: 2-element Vector{Float64}:
        -1.0
         3.0

[142]: A_43_1a_EigVal_Matrix = zeros(size(A_43_1a)[1], size(A_43_1a)[2])
       for i ∈ 1:size(A_43_1a)[1]
           A_43_1a_EigVal_Matrix[i, i] = A_43_1a_EigVals[i]
       end
       A_43_1a_EigVal_Matrix

[142]: 2×2 Matrix{Float64}:
        -1.0  0.0
         0.0  3.0
```

Рис. 2.36: Задание 4.4.3. Номер 1. Пункт a (2)

2) Пункт b ([2.37])

b) $\begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}$

```
[144]: A_43_1b = [1 -2; -2 3]

[144]: 2×2 Matrix{Int64}:
         1  -2
        -2   3

[146]: A_43_1b_EigVals = eigvals(A_43_1b)

[146]: 2-element Vector{Float64}:
        -0.2360679774997897
         4.23606797749979

[147]: A_43_1b_EigVal_Matrix = zeros(size(A_43_1b)[1], size(A_43_1b)[2])
       for i ∈ 1:size(A_43_1b)[1]
           A_43_1b_EigVal_Matrix[i, i] = A_43_1b_EigVals[i]
       end
       A_43_1b_EigVal_Matrix

[147]: 2×2 Matrix{Float64}:
        -0.236068  0.0
         0.0       4.23607

[148]: A_43_1b_EigVect = eigvecs(A_43_1b)

[148]: 2×2 Matrix{Float64}:
        -0.850651  -0.525731
        -0.525731   0.850651

[149]: A_43_1b_Diag = A_43_1b_EigVect' * A_43_1b * A_43_1b_EigVect

[149]: 2×2 Matrix{Float64}:
        -0.236068     3.46945e-16
         2.22045e-16  4.23607
```

Рис. 2.37: Задание 4.4.3. Номер 1. Пункт b

3) Пункт c ([2.38,2.39])

c) $\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix}$

```
[150]: A_43_1c = [1 -2 0; -2 1 2; 0 2 0]

[150]: 3×3 Matrix{Int64}:
         1  -2  0
        -2   1  2
         0   2  0

[151]: A_43_1c_EigVals = eigvals(A_43_1c)

[151]: 3-element Vector{Float64}:
        -2.1413361156553643
         0.51513804712807
         3.6261980685272945

[152]: A_43_1c_EigVal_Matrix = zeros(size(A_43_1c)[1], size(A_43_1c)[2])
       for i ∈ 1:size(A_43_1c)[1]
           A_43_1c_EigVal_Matrix[i, i] = A_43_1c_EigVals[i]
       end
       A_43_1c_EigVal_Matrix

[152]: 3×3 Matrix{Float64}:
        -2.14134  0.0       0.0
         0.0      0.515138  0.0
         0.0      0.0       3.6262
```

Рис. 2.38: Задание 4.4.3. Номер 1. Пункт c (1)

```
[153]: A_43_1c_EigVect = eigvecs(A_43_1c)

[153]: 3×3 Matrix{Float64}:
         0.421859  0.717093   0.554808
         0.6626    0.173846  -0.728518
        -0.618866  0.674948  -0.401808

[154]: A_43_1c_Diag = A_43_1c_EigVect' * A_43_1c * A_43_1c_EigVect

[154]: 3×3 Matrix{Float64}:
        -2.14134       3.55271e-15  -6.66134e-16
         3.52496e-15   0.515138     -3.88578e-16
        -6.66134e-16  -3.33067e-16   3.6262
```

Рис. 2.39: Задание 4.4.3. Номер 1. Пункт c (2)

2. Вычислим некоторые выражения:

    1) Пункт a ([2.40])

2. Вычислите

а) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}^{10}$

```
[155]: A_43_2a = [1 -2; -2 1]

[155]: 2×2 Matrix{Int64}:
         1  -2
        -2   1

[156]: A_43_2a^10

[156]: 2×2 Matrix{Int64}:
         29525  -29524
        -29524   29525
```

Рис. 2.40: Задание 4.4.3. Номер 2. Пункт a

2) Пункт b ([2.41,2.42])

b) $\sqrt{\begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}}$

```
[172]: A_43_2b = [5 -2; -2 5]

[172]: 2×2 Matrix{Int64}:
         5  -2
        -2   5

[173]: A_43_2b_Eig = eigen(A_43_2b)

[173]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
       values:
       2-element Vector{Float64}:
        3.0
        7.0
       vectors:
       2×2 Matrix{Float64}:
        -0.707107  -0.707107
        -0.707107   0.707107

[174]: A_43_2b_EigVect = eigvecs(A_43_2b)

[174]: 2×2 Matrix{Float64}:
        -0.707107  -0.707107
        -0.707107   0.707107
```

Рис. 2.41: Задание 4.4.3. Номер 2. Пункт b (1)

```
[175]: A_43_2b_Diag = A_43_2b_EigVect' * A_43_2b * A_43_2b_EigVect

[175]: 2×2 Matrix{Float64}:
        3.0  0.0
        0.0  7.0

[176]: Res_43_2b = sqrt.(A_43_2b_Diag)

[176]: 2×2 Matrix{Float64}:
        1.73205  0.0
        0.0      2.64575

[177]: # Проверка
       Res_43_2b*Res_43_2b

[177]: 2×2 Matrix{Float64}:
        3.0  0.0
        0.0  7.0
```

Рис. 2.42: Задание 4.4.3. Номер 2. Пункт b (2)

3) Пункт c ([2.43,2.44])

c) $\sqrt[3]{\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}}$

```
[180]: A_43_2c = [1 -2; -2 1]

[180]: 2×2 Matrix{Int64}:
   1  -2
  -2   1

[181]: A_43_2c_Eig = eigen(A_43_2c)

[181]: 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

[184]: A_43_2c_EigVect = eigvecs(A_43_2c)

[184]: 2×2 Matrix{Float64}:
        -0.707107  -0.707107
        -0.707107   0.707107
```

Рис. 2.43: Задание 4.4.3. Номер 2. Пункт с (1)

```
[185]: A_43_2c_Diag = A_43_2c_EigVect' * A_43_2c * A_43_2c_EigVect

[185]: 2×2 Matrix{Float64}:
        -1.0  0.0
         0.0  3.0

[186]: Res_43_2c = cbrt.(A_43_2c_Diag)

[186]: 2×2 Matrix{Float64}:
        -1.0  0.0
         0.0  1.44225

[187]: # Проверка
       Res_43_2c*Res_43_2c*Res_43_2c

[187]: 2×2 Matrix{Float64}:
        -1.0  0.0
         0.0  3.0
```

Рис. 2.44: Задание 4.4.3. Номер 2. Пункт с (2)

4) Пункт d ([2.45,2.46])

d) $\sqrt{\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}}$

```
[188]: A_43_2d = [1 2; 2 3]

[188]: 2×2 Matrix{Int64}:
        1  2
        2  3

[189]: A_43_2d_Eig = eigen(A_43_2d)

[189]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
       values:
       2-element Vector{Float64}:
        -0.2360679774997897
         4.23606797749979
       vectors:
       2×2 Matrix{Float64}:
        -0.850651  0.525731
         0.525731  0.850651

[190]: A_43_2d_EigVect = eigvecs(A_43_2d)

[190]: 2×2 Matrix{Float64}:
        -0.850651  0.525731
         0.525731  0.850651

[191]: A_43_2d_Diag = A_43_2d_EigVect' * A_43_2d * A_43_2d_EigVect

[191]: 2×2 Matrix{Float64}:
        -0.236068     -3.46945e-16
        -2.22045e-16   4.23607
```

Рис. 2.45: Задание 4.4.3. Номер 2. Пункт d (1)

```
[195]: # Одно из собственных значений отрицательно, поэтому извлекаем корень из комплексного числа
       Res_43_2d = sqrt.(Complex.(A_43_2d_Diag))

[195]: 2×2 Matrix{ComplexF64}:
        0.0+0.485868im       0.0+1.86265e-8im
        0.0+1.49012e-8im  2.05817+0.0im

[196]: # Проверка
       Res_43_2d*Res_43_2d

[196]: 2×2 Matrix{ComplexF64}:
        -0.236068+0.0im         -9.05e-9+3.83364e-8im
         -7.24e-9+3.06691e-8im   4.23607+0.0im
```

Рис. 2.46: Задание 4.4.3. Номер 2. Пункт d (2)

3. Найдем собственные значения матрицы A ([2.47]), создадим нижнедиаго-

нальную матрицу и диагональную матрицу из собственных значений матрицы A ([2.48]), оценим эффективность этих операций, рассчитав время на выполнение операций ([2.49]).

3. Найдите собственные значения матрицы A, если $A = \begin{pmatrix} 140 & 97 & 74 & 168 & 131 \\ 97 & 106 & 89 & 131 & 36 \\ 74 & 89 & 152 & 144 & 71 \\ 168 & 131 & 144 & 54 & 142 \\ 131 & 36 & 71 & 142 & 36 \end{pmatrix}$. Создайте диагональную матрицу из собственных значений матрицы A. Создайте нижнедиагональную матрицу из матрица A. Оцените эффективность выполняемых операций.

```
[199]: A_43_3 = [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]

[199]: 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

[200]: print(issymmetric(A_43_3))

       true

[201]: A_43_3_EigVals = eigvals(A_43_3)

[201]: 5-element Vector{Float64}:
        -128.49322764802145
         -55.887784553056875
          42.7521672793189
          87.16111477514521
         542.4677301466143
```

Рис. 2.47: Задание 4.4.3. Номер 3 (1)

```
[202]: A_43_3_EigVal_Matrix = zeros(size(A_43_3)[1], size(A_43_3)[2])
       for i ∈ 1:size(A_43_3)[1]
           A_43_3_EigVal_Matrix[i, i] = A_43_3_EigVals[i]
       end
       A_43_3_EigVal_Matrix

[202]: 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

[204]: LowerTriangular(A_43_3)

[204]: 5×5 LowerTriangular{Int64, Matrix{Int64}}:
        140    ⋅    ⋅    ⋅   ⋅
         97  106    ⋅    ⋅   ⋅
         74   89  152    ⋅   ⋅
        168  131  144   54   ⋅
        131   36   71  142  36
```

Рис. 2.48: Задание 4.4.3. Номер 3 (2)

```
[207]: @btime begin
           A_43_3_EigVals = eigvals(A_43_3)
           A_43_3_EigVal_Matrix = zeros(size(A_43_3)[1], size(A_43_3)[2])
           for i ∈ 1:size(A_43_3)[1]
               A_43_3_EigVal_Matrix[i, i] = A_43_3_EigVals[i]
           end
       end

         3.350 μs (25 allocations: 3.20 KiB)

[206]: @btime LowerTriangular(A_43_3)

         68.275 ns (1 allocation: 16 bytes)

[206]: 5×5 LowerTriangular{Int64, Matrix{Int64}}:
        140    ⋅    ⋅    ⋅   ⋅
         97  106    ⋅    ⋅   ⋅
         74   89  152    ⋅   ⋅
        168  131  144   54   ⋅
        131   36   71  142  36
```

Рис. 2.49: Задание 4.4.3. Номер 3 (3)

2.2.4 Линейные модели экономики

Напишем функции, которые будут необходимы для дальнейших заданий: создание диагональной матрицы из собственных значений некоторой матрицы, нахождение обратной матрицы к разности единичной и исходной матриц и решение СЛАУ линейной модели экономики ([2.50]).

4.4.4. Линейные модели экономики

Линейная модель экономики может быть записана как СЛАУ

$$x - Ax = y, \qquad (x(E - A) = y)$$

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.

```
[231]: function Diagonalize(A)
           A_EigVals = eigvals(A)
           A_EigVal_Matrix = zeros(size(A)[1], size(A)[2])
           for i ∈ 1:size(A)[1]
               A_EigVal_Matrix[i, i] = A_EigVals[i]
           end
           return A_EigVal_Matrix
       end

[231]: Diagonalize (generic function with 1 method)

[236]: function inverse_E_minus_A_Diag(A)
           A_Diag = Diagonalize(A)
           E_minus_A = Matrix{Int}(I, size(A)[1], size(A)[2]) - A_Diag
           return inv(E_minus_A)
       end

[236]: inverse_E_minus_A_Diag (generic function with 1 method)

[265]: function SLAE_44(A, y)
           Inv_E_minus_A_Diag = inverse_E_minus_A_Diag(A)
           x = zeros(length(y))
           # Умножаем элементы диагональной матрицы на соответствующий элемент из правой части
           for i ∈ 1:length(y)
               x[i] = y[i]*Inv_E_minus_A_Diag[i, i]
           end
           return x
       end

[265]: SLAE_44 (generic function with 1 method)
```

Рис. 2.50: Задание 4.4.4. Линейные модели экономики

1. Проверим является ли матрица продуктивной (определение на первом

скриншоте):

1) Пункт a ([2.51,2.52])

1. 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.

а) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```
[260]: A_44_1a = [1 2; 3 4]

[260]: 2×2 Matrix{Int64}:
        1  2
        3  4

[261]: b_44_1a = [2, 2]

[261]: 2-element Vector{Int64}:
        2
        2
```

Рис. 2.51: Задание 4.4.4. Номер 1. Пункт a (1)

```
[262]: Diagonalize(A_44_1a)

[262]: 2×2 Matrix{Float64}:
        -0.372281  0.0
         0.0       5.37228

[263]: inverse_E_minus_A_Diag(A_44_1a)

[263]: 2×2 Matrix{Float64}:
        0.728714   0.0
        0.0       -0.228714
```

Так как один из элементов на диагонали отрицательный, то матрицы не продуктивная

```
[271]: # Проверка
       SLAE_44(A_44_1a, b_44_1a)

[271]: 2-element Vector{Float64}:
         1.457427107756338
        -0.4574271077563381
```

Рис. 2.52: Задание 4.4.4. Номер 1. Пункт a (2)

2) Пункт b ([2.53])

b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```
[266]: A_44_1b = 1/2*[1 2; 3 4]
```

```
[266]: 2×2 Matrix{Float64}:
        0.5  1.0
        1.5  2.0
```

```
[267]: b_44_1b = b_44_1a
```

```
[267]: 2-element Vector{Int64}:
        2
        2
```

```
[268]: Diagonalize(A_44_1b)
```

```
[268]: 2×2 Matrix{Float64}:
        -0.186141  0.0
         0.0       2.68614
```

```
[269]: inverse_E_minus_A_Diag(A_44_1b)
```

```
[269]: 2×2 Matrix{Float64}:
        0.84307   0.0
        0.0      -0.59307
```

Так как один из элементов на диагонали отрицательный, то матрицы не продуктивная

```
[270]: # Проверка
       SLAE_44(A_44_1b, b_44_1b)
```

```
[270]: 2-element Vector{Float64}:
         1.6861406616345072
        -1.1861406616345072
```

Рис. 2.53: Задание 4.4.4. Номер 1. Пункт b

3) Пункт c ([2.54])

c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```
[272]: A_44_1c = 1/10*[1 2; 3 4]

[272]: 2×2 Matrix{Float64}:
        0.1  0.2
        0.3  0.4

[274]: b_44_1c = b_44_1a

[274]: 2-element Vector{Int64}:
        2
        2

[275]: Diagonalize(A_44_1c)

[275]: 2×2 Matrix{Float64}:
        -0.0372281  0.0
         0.0        0.537228

[276]: inverse_E_minus_A_Diag(A_44_1c)

[276]: 2×2 Matrix{Float64}:
        0.964108  0.0
        0.0       2.16089
```

Так как все элементы на диагонали неотрицательны, то матрица продуктивная

```
[277]: # Проверка
       SLAE_44(A_44_1c, b_44_1c)

[277]: 2-element Vector{Float64}:
        1.9282161153045774
        4.321783884695423
```

Рис. 2.54: Задание 4.4.4. Номер 1. Пункт c

2. Проверим матрицы на продуктивность с помощью критерия продуктивности. Напишем функцию для критерия продуктивности ([2.55]).

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

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

являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными.

```
[299]: Inverse_E_minus_A(A) = inv(Matrix{Int}(I, size(A)[1], size(A)[2]) - A)

       function Productivity_Criteria(A)
           inverse_E_minus_A = Inverse_E_minus_A(A)
           n = 0
           for i ∈ 1:length(A)
               if inverse_E_minus_A[i] >= 0
                   n += 1
               end
           end
           if n == length(A)
               return true
           else
               return false
           end
       end

[299]: Productivity_Criteria (generic function with 1 method)
```

Рис. 2.55: Задание 4.4.4. Номер 2

1) Пункт a ([2.56])

a) $\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

```
[300]: A_44_2a = [1 2; 3 1]

[300]: 2×2 Matrix{Int64}:
        1  2
        3  1

[303]: Inverse_E_minus_A(A_44_2a)

[303]: 2×2 Matrix{Float64}:
        -0.0  -0.333333
        -0.5   0.0

[304]: Productivity_Criteria(A_44_2a)

[304]: false
```

Матрица не продуктивна

Рис. 2.56: Задание 4.4.4. Номер 2. Пункт a

2) Пункт b ([2.57])

b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

```
[305]: A_44_2b = 1/2*[1 2; 3 1]

[305]: 2×2 Matrix{Float64}:
        0.5  1.0
        1.5  0.5

[306]: Inverse_E_minus_A(A_44_2b)

[306]: 2×2 Matrix{Float64}:
        -0.4  -0.8
        -1.2  -0.4

[307]: Productivity_Criteria(A_44_2b)

[307]: false
```

Матрица не продуктивна

Рис. 2.57: Задание 4.4.4. Номер 2. Пункт b

3) Пункт c ([2.58])

c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

```
[308]: A_44_2c = 1/10*[1 2; 3 1]

[308]: 2×2 Matrix{Float64}:
        0.1  0.2
        0.3  0.1

[309]: Inverse_E_minus_A(A_44_2c)

[309]: 2×2 Matrix{Float64}:
        1.2  0.266667
        0.4  1.2

[310]: Productivity_Criteria(A_44_2c)

[310]: true
```

Матрица продуктивна

Рис. 2.58: Задание 4.4.4. Номер 2. Пункт c

3. Проверим матрицы на продуктивность с помощью спектрального критерия продуктивности. Напишем функцию для спектрального критерия продуктивности ([2.59]).

3. Спектральный критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все её собственные значения по модулю меньше 1. Используя этот критерий, проверьте, являются ли матрицы продуктивными.

```
[324]: function Spectral_Productivity_Criteria(A)
           Eig_val = abs.(eigvals(A))
           n = 0
           for i ∈ 1:size(A)[1]
               if Eig_val[i] < 1
                   n += 1
               end
           end
           if n == size(A)[1]
               return true
           else
               return false
           end
       end

[324]: Spectral_Productivity_Criteria (generic function with 1 method)
```

Рис. 2.59: Задание 4.4.4. Номер 3

1) Пункт a ([2.60])

a) $\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

```
[325]: A_44_3a = [1 2; 3 1]

[325]: 2×2 Matrix{Int64}:
        1  2
        3  1

[326]: A_44_3a_Eigvals = eigvals(A_44_3a)

[326]: 2-element Vector{Float64}:
        -1.4494897427831779
         3.4494897427831783

[327]: Spectral_Productivity_Criteria(A_44_3a)

[327]: false
```

Матрица не продуктивна

Рис. 2.60: Задание 4.4.4. Номер 3. Пункт a

2) Пункт b ([2.61])

b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

```
[328]: A_44_3b = 1/2*[1 2; 3 1]
```

```
[328]: 2×2 Matrix{Float64}:
        0.5  1.0
        1.5  0.5
```

```
[329]: A_44_3b_Eigvals = eigvals(A_44_3b)
```

```
[329]: 2-element Vector{Float64}:
        -0.7247448713915892
         1.724744871391589
```

```
[330]: Spectral_Productivity_Criteria(A_44_3b)
```

```
[330]: false
```

Матрица не продуктивна

Рис. 2.61: Задание 4.4.4. Номер 3. Пункт b

3) Пункт c ([2.62])

c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$

[331]:
```
A_44_3c = 1/10*[1 2; 3 1]
```

[331]:
```
2×2 Matrix{Float64}:
 0.1  0.2
 0.3  0.1
```

[332]:
```
A_44_3c_Eigvals = eigvals(A_44_3c)
```

[332]:
```
2-element Vector{Float64}:
 -0.14494897427831785
  0.34494897427831783
```

[333]:
```
Spectral_Productivity_Criteria(A_44_3c)
```

[333]:
```
true
```

Матрица продуктивна

Рис. 2.62: Задание 4.4.4. Номер 3. Пункт c

4) Пункт d ([2.63])

d) $\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0 & 0.1 & 0.2 \\ 0 & 0.1 & 0.3 \end{pmatrix}$

```
[334]: A_44_3d = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]

[334]: 3×3 Matrix{Float64}:
        0.1  0.2  0.3
        0.0  0.1  0.2
        0.0  0.1  0.3

[335]: A_44_3d_Eigvals = eigvals(A_44_3d)

[335]: 3-element Vector{Float64}:
        0.02679491924311228
        0.1
        0.37320508075688774

[336]: Spectral_Productivity_Criteria(A_44_3d)

[336]: true
```

Рис. 2.63: Задание 4.4.4. Номер 3. Пункт d

3 Выводы

В ходе выполнения лабораторной работы я изучил специализированные пакеты Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Список литературы

1. Королькова А. В., Кулябов Д. С. Лабораторная работа № 4. Линейная алгебра [Электронный ресурс]. RUDN, 2023. URL: https://esystem.rudn.ru/pluginfile.php/2231349/mod_resource/content/3/004-lab_linear-algebra.pdf.