Отчет по лабораторной работе №4

Дисциплина: Компьютерный практикум по статистическому анализу данных

Оразгелдиев Язгелди

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. Повторила примеры с поэлементными операциями над многомерными массивами.

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

[4]: 4×3 Matrix{Int64}:
       7  9   9
       5  6  17
      13  2   9
       3  2   6

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

[5]: 88

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

[6]: 1×3 Matrix{Int64}:
      28  19  41

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

[7]: 4×1 Matrix{Int64}:
      25
      28
      24
      11
```

Рисунок 3.1: *Примеры с поэлементными операциями над многомерными массивами*

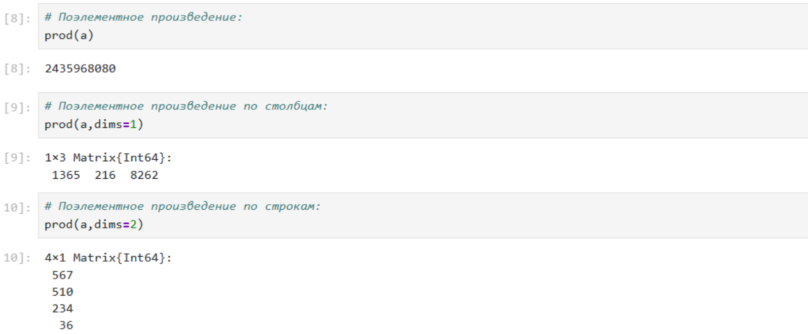


Рисунок 3.2: *Примеры с поэлементными операциями над многомерными массивами*

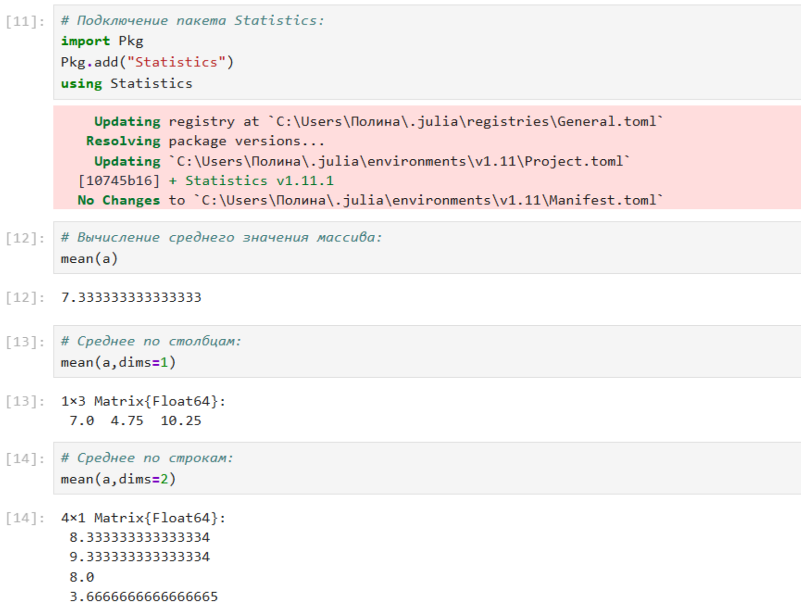


Рисунок 3.3: *Примеры с поэлементными операциями над многомерными массивами*

2. Повторил примеры с транспонированием, следом, рангом, определителем и инверсией матрицы.

```
[15]: #Транспонирование, след, ранг, определитель и инверсия матрицы
      # Подключение пакета LinearAlgebra:
      import Pkg
      Pkg.add("LinearAlgebra")
      using LinearAlgebra
```

```
   Resolving package versions...
    Updating `C:\Users\Полина\.julia\environments\v1.11\Project.toml`
  [37e2e46d] + LinearAlgebra v1.11.0
  No Changes to `C:\Users\Полина\.julia\environments\v1.11\Manifest.toml`
```

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

```
[16]: 4×4 Matrix{Int64}:
        2  14   2  20
       12   3  15   9
        5  11  12  12
       19   5  11  15
```

```
[17]: # Транспонирование:
      transpose(b)
```

```
[17]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
        2  12   5  19
       14   3  11   5
        2  15  12  11
       20   9  12  15
```

```
[18]: b'
```

```
[18]: 4×4 adjoint(::Matrix{Int64}) with eltype Int64:
        2  12   5  19
       14   3  11   5
        2  15  12  11
       20   9  12  15
```

Рисунок 3.4: *Примеры с транспонированием, следом, рангом, определителем и инверсией матрицы*

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

[19]: 32

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

[20]: 4-element Vector{Int64}:
        2
        3
       12
       15

[21]: # Ранг матрицы:
      rank(b)

[21]: 4

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

[22]: 4×4 Matrix{Float64}:
       -0.0869308   -0.16578    0.0910704    0.142519
       -0.129509    -0.3014     0.278533     0.130692
        0.00620934   0.130889  -0.00650503  -0.0816085
        0.148729     0.214469  -0.20343     -0.0975754

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

[23]: 10145.999999999998
```

Рисунок 3.5: *Примеры с транспонированием, следом, рангом, определителем и инверсией матрицы*

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

[24]: 3×4 Matrix{Float64}:
        0.0180046  -0.0551283   0.0896517   -0.00528759
        0.158343   -0.0429604  -0.0642611   -0.0194021
       -0.061237    0.0857328  -0.00551091   0.0238789
```

Рисунок 3.6: *Примеры с транспонированием, следом, рангом, определителем и инверсией матрицы*

3. Повторил примеры с вычислением нормы векторов и матриц, поворотами, вращением.

```
[25]: #Вычисление нормы векторов и матриц, повороты, вращения
      # Создание вектора X:
      X = [2, 4, -5]

[25]: 3-element Vector{Int64}:
        2
        4
       -5

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

[26]: 6.708203932499369

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

[27]: 11.0

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

[28]: 9.486832980505138

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

[29]: 9.486832980505138
```

Рисунок 3.7: *Примеры с вычислением нормы векторов и матриц, поворотами, вращением*

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

[31]: 2.4404307889469252

[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

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

[33]: 7.147682841795258

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

[34]: 8.0
```

Рисунок 3.8: *Примеры с вычислением нормы векторов и матриц, поворотами, вращением*

```
[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
```

Рисунок 3.9: *Примеры с вычислением нормы векторов и матриц, поворотами, вращением*

4. Повторил примеры с матричным умножением, единичной матрицей, скалярным произведением.

```
[38]: #Матричное умножение, единичная матрица, скалярное произведение
      # Матрица 2х3 со случайными целыми значениями от 1 до 10:
      A = rand(1:10,(2,3))

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

[39]: # Матрица 3х4 со случайными целыми значениями от 1 до 10:
      B = rand(1:10,(3,4))

[39]: 3×4 Matrix{Int64}:
       9  1  2  1
       5  1  3  7
       6  1  9  3

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

[40]: 2×4 Matrix{Int64}:
       91  14  93  44
       93  15  53  73

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

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

Рисунок 3.10: *Примеры с матричным умножением, единичной матрицей, скалярным произведением*

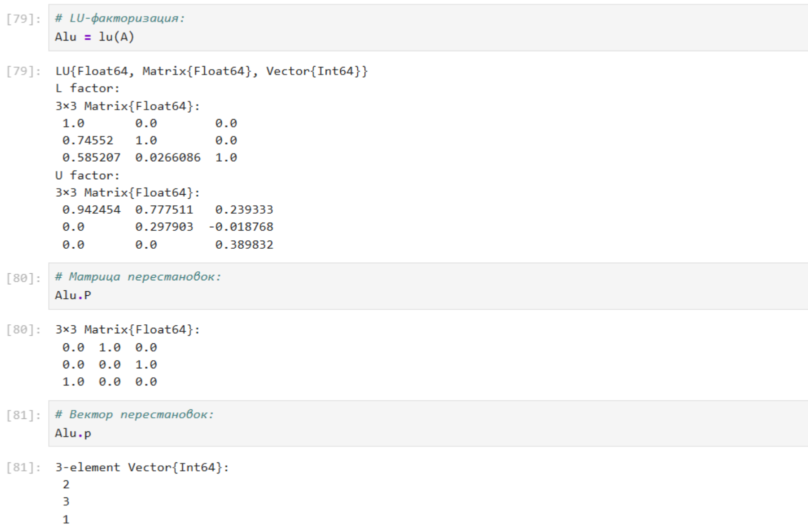
```
[42]: # Скалярное произведение векторов X и Y:
      X = [2, 4, -5]
      Y = [1,-1,3]
      dot(X,Y)

[42]: -17

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

[43]: -17
```

Рисунок 3.11: *Примеры с матричным умножением, единичной матрицей, скалярным произведением*

5. Повторил примеры с факторизацией.

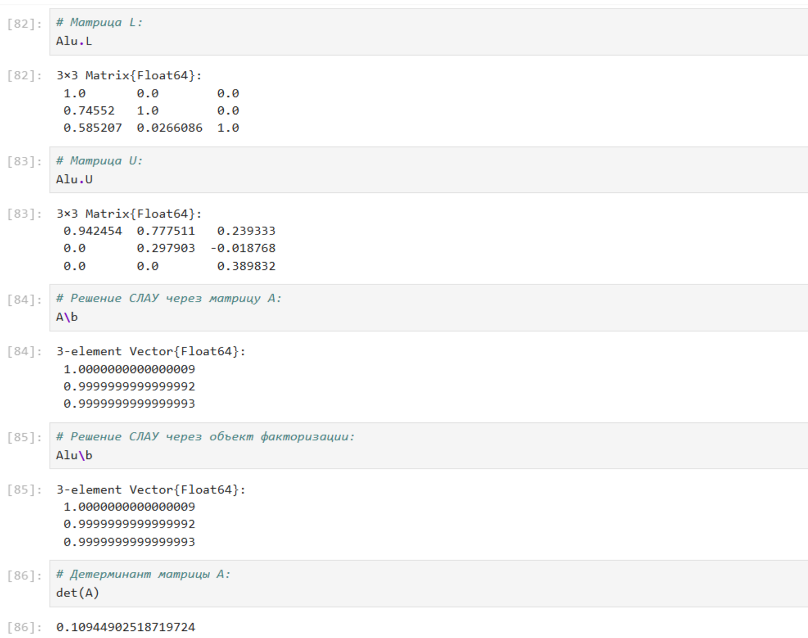
```
[75]: #Факторизация. Специальные матричные структуры
      # Задаём квадратную матрицу 3x3 со случайными значениями:
      A = rand(3, 3)

[75]: 3×3 Matrix{Float64}:
       0.551531  0.462932  0.529392
       0.942454  0.777511  0.239333
       0.702618  0.877552  0.15966

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

[76]: 3-element Vector{Float64}:
       1.0
       1.0
       1.0

[77]: # Задаём вектор b:
      b = A*x

[77]: 3-element Vector{Float64}:
       1.543855596934271
       1.9592989439076152
       1.739830453013854

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

[78]: 3-element Vector{Float64}:
       1.0000000000000009
       0.9999999999999992
       0.9999999999999993
```

Рисунок 3.12: *Примеры с факторизацией*



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

[79]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3×3 Matrix{Float64}:
       1.0       0.0        0.0
       0.74552   1.0        0.0
       0.585207  0.0266086  1.0
      U factor:
      3×3 Matrix{Float64}:
       0.942454  0.777511   0.239333
       0.0       0.297903  -0.018768
       0.0       0.0        0.389832

[80]: # Матрица перестановок:
      Alu.P

[80]: 3×3 Matrix{Float64}:
       0.0  1.0  0.0
       0.0  0.0  1.0
       1.0  0.0  0.0

[81]: # Вектор перестановок:
      Alu.p

[81]: 3-element Vector{Int64}:
       2
       3
       1
```

Рисунок 3.13: *Примеры с факторизацией*



```
[82]: # Матрица L:
      Alu.L

[82]: 3×3 Matrix{Float64}:
       1.0       0.0        0.0
       0.74552   1.0        0.0
       0.585207  0.0266086  1.0

[83]: # Матрица U:
      Alu.U

[83]: 3×3 Matrix{Float64}:
       0.942454  0.777511   0.239333
       0.0       0.297903  -0.018768
       0.0       0.0        0.389832

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

[84]: 3-element Vector{Float64}:
       1.0000000000000009
       0.9999999999999992
       0.9999999999999993

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

[85]: 3-element Vector{Float64}:
       1.0000000000000009
       0.9999999999999992
       0.9999999999999993

[86]: # Детерминант матрицы A:
      det(A)

[86]: 0.10944902518719724
```

Рисунок 3.14: *Примеры с факторизацией*

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

[87]: 0.10944902518719724

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

[88]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
      Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
      R factor:
      3×3 Matrix{Float64}:
       -1.29849  -1.2358    -0.484961
        0.0       0.248451  -0.112888
        0.0       0.0       -0.33926

[89]: # Матрица Q:
      Aqr.Q

[89]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}

[90]: # Матрица R:
      Aqr.R

[90]: 3×3 Matrix{Float64}:
       -1.29849  -1.2358    -0.484961
        0.0       0.248451  -0.112888
        0.0       0.0       -0.33926

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

[91]: 3×3 Matrix{Float64}:
        1.0          -8.32667e-17  -2.22045e-16
        0.0           1.0          -1.11022e-16
       -1.11022e-16   0.0           1.0
```

Рисунок 3.15: *Примеры с факторизацией*

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

[92]: 3×3 Matrix{Float64}:
       1.10306  1.40539  1.23201
       1.40539  1.55502  1.11689
       1.23201  1.11689  0.31932

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

[93]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
      values:
      3-element Vector{Float64}:
       -0.5873869922610678
       -0.02297178677478906
        3.5877632318320987
      vectors:
      3×3 Matrix{Float64}:
       -0.543112   0.589121  -0.598303
       -0.079495  -0.745429  -0.661828
        0.835889   0.311884  -0.451683

[94]: # Собственные значения:
      AsymEig.values

[94]: 3-element Vector{Float64}:
       -0.5873869922610678
       -0.02297178677478906
        3.5877632318320987

[95]: #Собственные векторы:
      AsymEig.vectors

[95]: 3×3 Matrix{Float64}:
       -0.543112   0.589121  -0.598303
       -0.079495  -0.745429  -0.661828
        0.835889   0.311884  -0.451683
```

Рисунок 3.16: *Примеры с факторизацией*

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

[96]: 3×3 Matrix{Float64}:
        1.0           -3.81917e-14  -5.77316e-15
        3.90799e-14    1.0           1.28786e-14
       -1.59872e-14  -2.13163e-14   1.0

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

[97]: 1000×1000 Matrix{Float64}:
       -2.21123     1.49552    -0.0800027  …   1.03644    0.287116   0.489334
       -1.61715    -1.19851    -0.520879      -0.340143   0.836955  -0.317155
        1.0039     -2.03757     0.0386619      0.684426   0.485467  -2.76429
       -1.11308    -0.503773    0.662727       1.69777   -1.30953   -0.778417
        0.422197    0.00782534  0.320535      -1.24173   -0.446     -0.0483074
        0.324208   -0.469507    1.69774    …   0.832636   0.894876   0.696623
        0.78462     0.877493   -1.77879       -2.06824    0.766819   1.01932
       -0.0452582   0.0214964  -1.6168        -0.894698  -0.284973   0.0288325
        1.68032     0.952413    0.134991      -1.2106     1.68641    0.955969
       -0.68542     0.730857   -1.02571       -0.258665  -1.09236   -1.57978
        1.88223    -0.028493   -1.30407    …  -0.70995    0.0860891 -1.39097
        1.37706     0.0634672   0.502746      -0.383901  -0.424257  -0.748692
       -0.463588   -2.28929     1.28139        0.0356126 -1.54675   -0.249698
        ⋮                                  ⋱
       -1.62627    -1.22571     0.837636       0.281486   0.501276  -1.1722
        0.150812   -1.18037    -1.15633       -1.49511    1.26137    0.576786
       -0.415434   -0.0905179  -1.49928    …  -0.643082   0.88309   -0.898478
       -0.499404   -2.28215     1.65194        0.520678  -0.375621  -0.251754
       -1.31017     0.704631    0.777236       0.0546234 -0.443469   1.48167
        1.52842    -0.859274    1.64819        0.986682   0.597711   0.564677
       -1.58124     0.315807    0.661945      -0.373102  -0.159406  -0.577015
       -0.725211   -0.335825    0.692725   …   1.68964   -0.235074   0.111309
        0.533782    0.942887   -0.562174      -0.673415  -0.763457  -0.263159
        0.5751      0.556507   -0.612258      -0.245984   0.495394   1.54353
        1.6237      1.30644     3.2001         0.263866   1.00224   -0.114302
```

Рисунок 3.17: *Примеры с факторизацией*

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

[98]: 1000×1000 Matrix{Float64}:
       -4.42247    -0.121627    0.923896   …   1.61154    1.91082    0.0643163
       -0.121627   -2.39702    -2.55845        0.216364   2.14339   -0.786335
        0.923896   -2.55845     0.0773238      0.0721682  3.68557   -4.07451
       -2.04304    -1.34501    -0.689001       1.92147   -1.32493   -1.4391
       -1.5396     -0.778088    0.554126      -1.42853    0.0744139 -0.540589
        0.205376   -0.447025    3.65558    …   1.48534   -0.414697   1.81156
        0.154791    1.34348    -1.29527       -2.13208   -0.66017    1.54498
        0.852045    3.13639    -0.781097      -0.697632  -0.630185   0.0521465
        3.3503     -0.237203   -0.70463       -1.93532    2.35026   -0.0314685
       -2.10326    -0.0592971  -1.50522        0.416827  -2.06459   -2.04976
        0.629218    1.10381    -1.85902    …  -1.31295   -0.510115   0.382771
        0.763118    1.66157    -0.288943       1.75658   -1.1151    -0.985488
       -1.74386    -0.738943    0.307054      -0.265365  -1.09749    0.0363151
        ⋮                                  ⋱
       -3.00941    -1.37024     2.39034        1.82314   -0.380292  -1.79772
       -0.0551953  -0.899751   -1.71652       -1.8575     1.74603    0.463956
       -1.24547    -0.849534   -2.47423    …   0.108653   0.940387  -0.146214
       -0.772616   -2.07813     3.09038        0.346363  -0.960159  -1.50093
       -2.64617    -2.04766     1.08653        0.822263   0.0911967  1.65553
        1.60719    -0.841101    0.157655       1.96522    2.13915    0.658992
       -0.55921    -1.16312     0.901499      -0.867904  -1.78357   -2.24253
       -0.496712   -0.0442799  -0.172507   …   2.9909     0.653923  -0.694089
        1.99153    -0.402386   -0.436316       1.62488   -2.21985    0.275544
        1.61154     0.216364    0.0721682     -0.491969   0.75926    4.27267
        1.91082     2.14339     3.68557        0.75926    2.00448   -0.342474
        0.0643163  -0.786335   -4.07451        4.27267   -0.342474  -1.15349

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

[99]: true
```

Рисунок 3.18: *Примеры с факторизацией*

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

[100]: -0.12162668870773974

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

[101]: false

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

[102]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
        -4.42247    -0.121627   0.923896   …   1.61154    1.91082    0.0643163
        -0.121627   -2.39702   -2.55845        0.216364   2.14339   -0.786335
         0.923896   -2.55845    0.0773238      0.0721682  3.68557   -4.07451
        -2.04304    -1.34501   -0.689001       1.92147   -1.32493   -1.4391
        -1.5396     -0.778088   0.554126      -1.42853    0.0744139 -0.540589
         0.205376   -0.447025   3.65558    …   1.48534   -0.414697   1.81156
         0.154791    1.34348   -1.29527       -2.13208   -0.66017    1.54498
         0.852045    3.13639   -0.781097      -0.697632  -0.630185   0.0521465
         3.3503     -0.237203  -0.70463       -1.93532    2.35026   -0.0314685
        -2.10326    -0.0592971 -1.50522        0.416827  -2.06459   -2.04976
         0.629218    1.10381   -1.85902    …  -1.31295   -0.510115   0.382771
         0.763118    1.66157   -0.288943       1.75658   -1.1151    -0.985488
        -1.74386    -0.738943   0.307054      -0.265365  -1.09749    0.0363151
         ⋮                                 ⋱
        -3.00941    -1.37024    2.39034        1.82314   -0.380292  -1.79772
        -0.0551953  -0.899751  -1.71652       -1.8575     1.74603    0.463956
        -1.24547    -0.849534  -2.47423    …   0.108653   0.940387  -0.146214
        -0.772616   -2.07813    3.09038        0.346363  -0.960159  -1.50093
        -2.64617    -2.04766    1.08653        0.822263   0.0911967  1.65553
         1.60719    -0.841101   0.157655       1.96522    2.13915    0.658992
        -0.55921    -1.16312    0.901499      -0.867904  -1.78357   -2.24253
        -0.496712   -0.0442799 -0.172507   …   2.9909     0.653923  -0.694089
```

Рисунок 3.19: *Примеры с факторизацией*

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

          Resolving package versions...
          Installed BenchmarkTools ─ v1.6.2
           Updating `C:\Users\Полина\.julia\environments\v1.11\Project.toml`
         [6e4b80f9] + BenchmarkTools v1.6.2
           Updating `C:\Users\Полина\.julia\environments\v1.11\Manifest.toml`
         [6e4b80f9] + BenchmarkTools v1.6.2
         [9abbd945] + Profile v1.11.0
       Precompiling project...
          3741.6 ms  ✓ BenchmarkTools
         1 dependency successfully precompiled in 6 seconds. 465 already precompiled.

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

         41.776 ms (21 allocations: 7.99 MiB)

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

         359.688 ms (27 allocations: 7.93 MiB)

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

         42.324 ms (21 allocations: 7.99 MiB)
```

Рисунок 3.20: *Примеры с факторизацией*

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

[107]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
        0.754391   1.12079     ⋅         …  ⋅         ⋅          ⋅
        1.12079    0.808854  -0.589236      ⋅         ⋅          ⋅
         ⋅        -0.589236   0.0751564     ⋅         ⋅          ⋅
         ⋅          ⋅         0.540651      ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅         …  ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅         …  ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
        ⋮                                ⋱
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅         …  ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅         …  ⋅         ⋅          ⋅
         ⋅          ⋅          ⋅             1.69466   ⋅          ⋅
         ⋅          ⋅          ⋅            -0.720398  1.75078    ⋅
         ⋅          ⋅          ⋅             1.75078   0.373654   0.454938
         ⋅          ⋅          ⋅            ⋅         0.454938  -0.0790711

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

         426.351 ms (44 allocations: 183.11 MiB)

[108]: 6.8310673689134
```

Рисунок 3.21: *Примеры с факторизацией*

6. Повторил примеры с общей линейной алгеброй.

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

[111]: 3×3 Matrix{Rational{BigInt}}:
        4//5   9//10  2//5
        3//5   1//10  7//10
        7//10  3//5   9//10

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

[112]: 3-element Vector{Int64}:
        1
        1
        1

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

[113]: 3-element Vector{Rational{BigInt}}:
        21//10
         7//5
        11//5

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

[114]: 3-element Vector{Rational{BigInt}}:
        1
        1
        1
```

Рисунок 3.22: *Примеры с общей линейной алгеброй*

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

[115]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
       L factor:
       3×3 Matrix{Rational{BigInt}}:
         1      0     0
        3//4    1     0
        7//8  15//46  1
       U factor:
       3×3 Matrix{Rational{BigInt}}:
        4//5    9//10    2//5
         0    -23//40    2//5
         0       0     193//460
```

Рисунок 3.23: *Примеры с общей линейной алгеброй*

7. Задал вектор v. Умножил вектор v скалярно сам на себя и сохранил результат в dot_v.

8. Умножил v матрично на себя (внешнее произведение), присвоив результат переменной outer_v.

```
•[119]:
# 1
v = [25, 86, 36]
dot_v = v'v

[119]:

9317

[120]:
outer_v = v*v'

[120]:

3×3 Matrix{Int64}:
  625  2150   900
 2150  7396  3096
  900  3096  1296
```

Рисунок 3.24: *Произведение векторов*

9. Решил СЛАУ с двумя неизвестными.

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

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

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

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

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

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

Рисунок 3.25: *Задание 2.1*

```
#2.1
A1 = [1 1; 1 -1]
b1 = [2, 3]
x1 = A1\b1
```

[190]:

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

[191]:

```
A2 = [1 1; 2 2]
b2 = [2, 4]
if det(A2) == 0
    print("Нет решений")
else
    x2 = A2\b2
    print(x2)
end
```

```
Нет решений
```

[192]:

```
A3 = [1 1; 2 2]
b3 = [2, 5]
if det(A3) == 0
    print("Нет решений")
else
    x3 = A3\b3
    print(x3)
end
```

```
Нет решений
```

Рисунок 3.26: *Системы линейных уравнений*

[193]:

```
A4 = [1 1; 2 2; 3 3]
b4 = [1, 2, 3]
x4 = A4\b4
```

[193]:

```
2-element Vector{Float64}:
 0.4999999999999999
 0.5
```

[194]:

```
A5 = [1 1; 2 1; 1 -1]
b5 = [2, 1, 3]
x5 = A5\b5
```

[194]:

```
2-element Vector{Float64}:
  1.5000000000000004
 -0.9999999999999997
```

[195]:

```
A6 = [1 1; 2 1; 3 2]
b6 = [2, 1, 3]
x6 = A6\b6
```

[195]:

```
2-element Vector{Float64}:
 -0.9999999999999989
  2.9999999999999982
```

Рисунок 3.27: *Системы линейных уравнений*

10. Решил СЛАУ с тремя неизвестными.

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

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

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

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

Рисунок 3.28: *Задание 2.2*

```
#2.2
A7 = [1 1 1; 1 -1 -2]
b7 = [2, 3]
x7 = A7\b7
```

[196]:

```
3-element Vector{Float64}:
  2.2142857142857144
  0.35714285714285704
 -0.5714285714285712
```

[197]:

```
A8 = [1 1 1; 2 2 -3; 3 1 1]
b8 = [2, 4, 1]
x8 = A8\b8
```

[197]:

```
3-element Vector{Float64}:
 -0.5
  2.5
  0.0
```

[198]:

```
A9 = [1 1 1; 1 1 2; 2 2 3]
b9 = [1, 0, 1]
if det(A9) == 0
    print("Нет решений")
else
    x9 = A9\b9
    print(x9)
end
```

```
Нет решений
```

Рисунок 3.29: *Системы линейных уравнений*

[199]:

```
A10 = [1 1 1; 1 1 2; 2 2 3]
b10 = [1, 0, 0]
if det(A10) == 0
    print("Нет решений")
else
    x10 = A10\b10
    print(x10)
end
```

```
Нет решений
```

Рисунок 3.30: *Системы линейных уравнений*

11. Привел приведённые ниже матрицы к диагональному виду.

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

Рисунок 3.31: *Задание 3.1*

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

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

Рисунок 3.32: *Задание 3.1*

```
#3.1
a = [1 -2; -2 1]
```

[200]:

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

[201]:

```
eigs = eigen(a)
```

[201]:

```
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
```

[202]:

```
a_diag = diagm(eigs.values)
```

[202]:

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

Рисунок 3.33: *Системы линейных уравнений*

```
b = [1 -2; -2 3]
eigs2 = eigen(b)
b_diag = diagm(eigs2.values)
```

[203]:

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

[204]:

```
c = [1 -2 0; -2 1 2; 0 2 0]
eigs3 = eigen(c)
c_diag = diagm(eigs3.values)
```

[204]:

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

Рисунок 3.34: *Системы линейных уравнений*

12. Вычислил

$$\text{a)} \quad \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}^{10}$$

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

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

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

Рисунок 3.35: *Задание 3.2*

```
#3.2
a = [1 -2; -2 1]
a^10
```

```
[205]:

2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
```

```
[206]:
b = [5 -2; -2 5]
sqrt(b)
```

```
[206]:

2×2 Matrix{Float64}:
  2.1889   -0.45685
 -0.45685   2.1889
```

```
[207]:
c = [1 -2; -2 1]
cbrt(c)
```

```
[207]:

2×2 Matrix{Float64}:
  0.221125  -1.22112
 -1.22112    0.221125
```

```
[208]:
d = [1 2; 2 3]
sqrt(d)
```

```
[208]:

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

Рисунок 3.36: *Операции с матрицами*

13. Нашел собственные значения матрицы A, если

$$A = \begin{pmatrix} 140 & 97 & 74 & 168 & 131 \\ 97 & 106 & 89 & 131 & 36 \\ 74 & 89 & 152 & 144 & 71 \\ 168 & 131 & 144 & 54 & 142 \\ 131 & 36 & 71 & 142 & 36 \end{pmatrix}.$$

Рисунок 3.37: *Задание 3.3*

Создал диагональную матрицу из собственных значений матрицы A. Создал нижнедиагональную матрицу из матрица A. Оценила эффективность выполняемых операций.

```
[209]:
#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]
eigs = eigen(A)
A_diag = diagm(eigs.values)

[209]:
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

[210]:
LowerTriangular(A)

[210]:
5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рисунок 3.38: *Операции с матрицами*

```
[211]:
@btime diagm(eigs.values)
  105.668 ns (2 allocations: 272 bytes)

[211]:
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

[212]:
@btime LowerTriangular(A)
  183.310 ns (1 allocation: 16 bytes)

[212]:
5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рисунок 3.39: *Операции с матрицами*

14. Линейная модель экономики может быть записана как СЛАУ x - Ax = y, где элементы матрицы A и столбца y — неотрицательные числа. По своему смыслу в экономике элементы матрицы A и столбцов x, y не могут быть отрицательными числами.

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

$$\text{a)}\quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

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

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

Рисунок 3.40: *Задание 4.1*

```
[213]:
#4.1
E = Matrix{Int}(I, 2, 2)
A1 = [1 2; 3 4]
y1 = [1, 1]
x1 = y1'*((E-A1)^-1)
#матрица не продуктивная
[213]:
1×2 adjoint(::Vector{Float64}) with eltype Float64:
 0.0  -0.333333
[214]:
A2 = 1/2*A1
x2 = y1'*((E-A2)^-1)
#матрица не продуктивная
[214]:
1×2 adjoint(::Vector{Float64}) with eltype Float64:
 -0.25  -0.75
[215]:
A3 = 1/10*A1
x3 = y1'*((E-A3)^(-1))
#матрица не продуктивная
[215]:
1×2 adjoint(::Vector{Float64}) with eltype Float64:
 1.875  2.29167
```

Рисунок 3.41: *Линейные модели экономики*

15. Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрица (E-A)^-1 являются неотрицательными числами. Используя этот критерий, проверил, являются ли матрицы продуктивными.

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

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

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

Рисунок 3.42: *Задание 4.2*

```
[216]:

#4.2
A4 = [1 2; 3 1]
x4 = (E-A4)^(-1)
#матрица не продуктивная

[216]:

2×2 Matrix{Float64}:
 -0.0  -0.333333
 -0.5   0.0

[217]:

A5 = 1/2*A4
x5 = (E-A5)^(-1)
#матрица не продуктивная

[217]:

2×2 Matrix{Float64}:
 -0.4  -0.8
 -1.2  -0.4

[218]:

A6 = 1/10*A4
x6 = (E-A6)^(-1)
#матрица продуктивная

[218]:

2×2 Matrix{Float64}:
 1.2  0.266667
 0.4  1.2
```

Рисунок 3.43: *Линейные модели экономики*

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

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

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

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

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

Рисунок 3.44: *Задание 4.3*

```
[219]:
#4.3
A7 = [1 2; 3 1]
eigs = eigen(A7)
abs.(eigs.values)
#матрица не продуктивная

[219]:
2-element Vector{Float64}:
 1.4494897427831779
 3.4494897427831783

[220]:
A8 = 1/2*A7
eigs = eigen(A8)
abs.(eigs.values)
#матрица не продуктивная

[220]:
2-element Vector{Float64}:
 0.7247448713915892
 1.724744871391589
```

Рисунок 3.45: *Линейные модели экономики*

[221]:

```
A9 = 1/10*A7
eigs = eigen(A9)
abs.(eigs.values)
#матрица продуктивная
```

[221]:

```
2-element Vector{Float64}:
 0.14494897427831785
 0.34494897427831783
```

[222]:

```
A10 = [0.1 0.2 0.2; 0 0.1 0.2; 0 0.1 0.3]
eigs = eigen(A10)
abs.(eigs.values)
#матрица продуктивная
```

[222]:

```
3-element Vector{Float64}:
 0.02679491924311228
 0.1
 0.37320508075688774
```

Рисунок 3.46: *Линейные модели экономики*

4 Выводы

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