Лабораторная работы №4

Компьютерный практикум по статистическому анализу данных

Кузнецова С. В.

24 октября 2025

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Кузнецова София Вадимовна
- Российский университет дружбы народов

Цель

Цель

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

Выполнение

Поэлементные операции над многомерными массивами

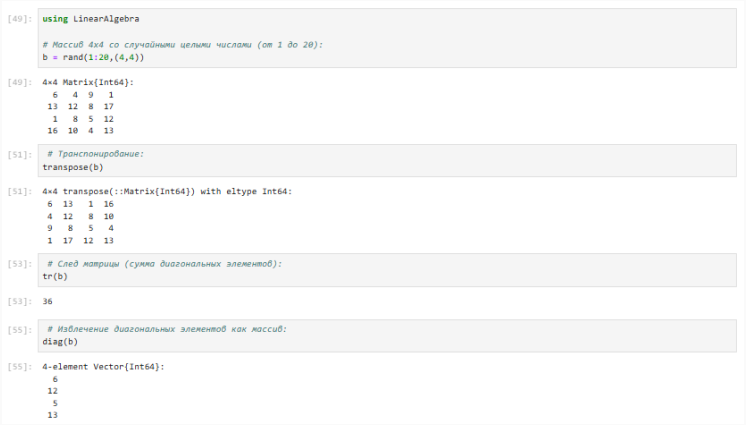
```
[1]: # Массив 4x3 со случайными целыми числами (от 1 до 20):
     a = rand(1:20,(4,3))

[1]: 4×3 Matrix{Int64}:
      20  14  20
       5  16   7
      17  19  12
      14   3   6

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

[3]: 153

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

[5]: 1×3 Matrix{Int64}:
      56  52  45

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

[7]: 4×1 Matrix{Int64}:
      54
      28
      48
      23

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

[9]: 3063094272000

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

11]: 1×3 Matrix{Int64}:
      23800  12768  10080

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

39]: 4×1 Matrix{Int64}:
      5600
       560
      3876
       252
```

Рис. 1: Поэлементные операции сложения и произведения элементов матрицы

Использование возможностей пакета Statistics

Рис. 2: Использование возможностей пакета Statistics для работы со средними значениями

Транспонирование, след, ранг, определитель и инверсия матрицы



```
[49]: using LinearAlgebra

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

[49]: 4×4 Matrix{Int64}:
        6   4  9   1
       13  12  8  17
        1   8  5  12
       16  10  4  13

[51]:  # Транспонирование:
      transpose(b)

[51]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
       6  13   1  16
       4  12   8  10
       9   8   5   4
       1  17  12  13

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

[53]: 36

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

[55]: 4-element Vector{Int64}:
        6
       12
        5
       13
```

Рис. 3: Использование библиотеки LinearAlgebra для выполнения определённых операций

Транспонирование, след, ранг, определитель и инверсия матрицы

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

[59]: 4

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

[61]: 4×4 Matrix{Float64}:
       -0.0160891   0.12005    -0.147277   -0.019802
        0.223391   -1.013       0.660272    0.69802
        0.0408416   0.310644   -0.164604   -0.257426
       -0.164604    0.535891   -0.27599    -0.356436

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

[63]: -1615.9999999999995

[65]: # Псевдообратная функция для прямоугольных матриц:
      pinv(a)

[65]: 3×4 Matrix{Float64}:
       -0.0418674   -0.0333706     0.0449694   0.0885515
       -0.0379454    0.0493012     0.0411936  -0.0134207
        0.118409    -0.000520226  -0.0743949  -0.0786333
```

Рис. 4: Использование библиотеки LinearAlgebra для выполнения определённых операций

Вычисление нормы векторов и матриц, повороты, вращения

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

[67]: 3-element Vector{Int64}:
       2
       4
      -5

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

[69]: 6.708203932499369

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

[71]: 11.0

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

[73]: 9.486832980505138

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

[75]: 9.486832980505138

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

[77]: 2.4404307889469252
```

Рис. 5: Использование LinearAlgebra.norm(x)

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

[79]: 3×3 Matrix{Int64}:
       5  -4  2
      -1   2  3
      -2   1  0

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

[81]: 7.147682841795258

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

[83]: 8.0

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

[85]: 3×3 Matrix{Int64}:
       0   1  -2
       3   2  -1
       2  -4   5

[87]: # Перебарачивание строк:
      reverse(d,dims=1)

[87]: 3×3 Matrix{Int64}:
      -2   1  0
      -1   2  3
       5  -4  2

[89]: # Перебарачивание столбцов:
      reverse(d,dims=2)

[89]: 3×3 Matrix{Int64}:
       2  -4   5
       3   2  -1
       0   1  -2
```

Рис. 6: Вычисление нормы для двумерной матрицы

Матричное умножение, единичная матрица, скалярное произведение

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

[93]: 2×3 Matrix{Int64}:
       2   4  10
      10  10   6

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

[95]: 3×4 Matrix{Int64}:
       4   8   6   2
       1   2   1  10
       9   5   8   3

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

[97]: 2×4 Matrix{Int64}:
       102   74   96   74
       104  130  118  138

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

[99]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1

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

[101]: -17

[103]: # Тоже скалярное произведение:
       X'Y

[103]: -17
```

Рис. 7: Примеры матричного умножения, единичной матрицы и скалярного произведения

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

[105]: 3×3 Matrix{Float64}:
        0.527302  0.681964   0.0360534
        0.574161  0.0967311  0.941675
        0.221858  0.603313   0.847529

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

[107]: 3-element Vector{Float64}:
        1.0
        1.0
        1.0

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

[109]: 3-element Vector{Float64}:
        1.2453191583252567
        1.6125669532785905
        1.6727008863162585

[111]: # Решение исходного уравнения получаем с помощью функции\
       # (убеждаемся,чтох-единичныйвектор):
       A\b

[111]: 3-element Vector{Float64}:
        1.0000000000000002
        0.9999999999999998
        0.9999999999999998
```

Рис. 8: Решение систем линейный алгебраических уравнений Ax = b

Факторизация. Специальные матричные структуры

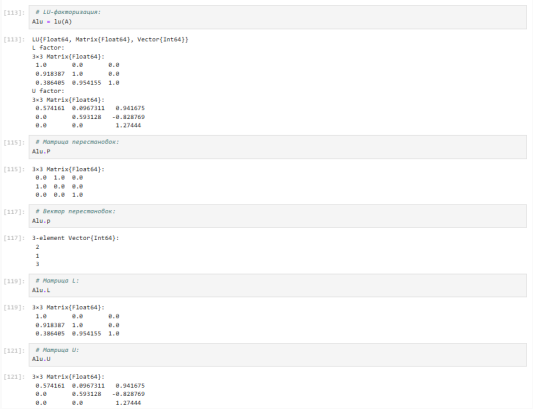
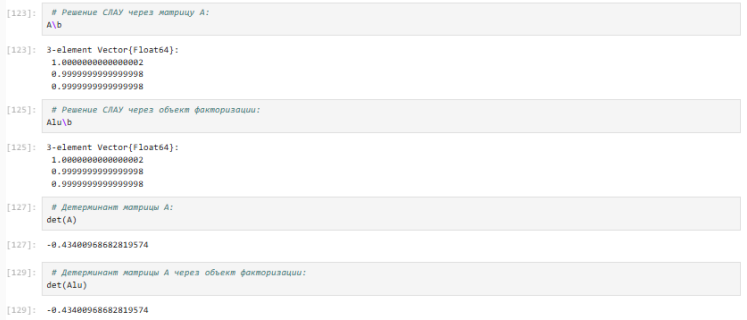


Рис. 9: Пример вычисления LU-факторизации и определение составного типа факторизации для его хранения



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

[123]: 3-element Vector{Float64}:
        1.0000000000000002
        0.9999999999999998
        0.9999999999999998

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

[125]: 3-element Vector{Float64}:
        1.0000000000000002
        0.9999999999999998
        0.9999999999999998

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

[127]: -0.4340096868281957 4

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

[129]: -0.4340096868281957 4
```

Рис. 10: Пример решения с использованием исходной матрицы и с использованием объекта факторизации

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

[131]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
       Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
       R factor:
       3×3 Matrix{Float64}:
        -0.810511  -0.677338  -0.922523
         0.0        0.616143   0.00347651
         0.0        0.0        0.869078

[133]: # Матрица Q:
       Aqr.Q

[133]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}

[135]: # Матрица R:
       Aqr.R

[135]: 3×3 Matrix{Float64}:
        -0.810511  -0.677338  -0.922523
         0.0        0.616143   0.00347651
         0.0        0.0        0.869078

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

[137]: 3×3 Matrix{Float64}:
         1.0          5.55112e-17  -2.22045e-16
         0.0          1.0          -1.11022e-16
        -2.22045e-16  0.0           1.0
```

Рис. 11: Пример вычисления QR-факторизации и определение составного типа факторизации для его хранения

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

[149]: 3×3 Matrix{Float64}:
        1.0546    1.25612   0.257912
        1.25612   0.193462  1.54499
        0.257912  1.54499   1.69506

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

[151]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
       values:
       3-element Vector{Float64}:
        -1.1943846277109258
         1.0441785138313682
         1.0933299302841823
       vectors:
       3×3 Matrix{Float64}:
         0.411223  -0.880671  0.435686
        -0.818594  -0.114113  0.562922
         0.400998   0.580137  0.702351

[153]: # Собственные значения:
       AsymEig.values

[153]: 3-element Vector{Float64}:
        -1.1943846277109258
         1.0441785138313682
         1.0933299302841823

[155]: # Собственные векторы:
       AsymEig.vectors

[155]: 3×3 Matrix{Float64}:
         0.411223  -0.880671  0.435686
        -0.818594  -0.114113  0.562922
         0.400998   0.580137  0.702351

[157]: # Проверим, что получится единичная матрица:
       inv(AsymEig)*Asym

[157]: 3×3 Matrix{Float64}:
         1.0          -3.38862e-15   2.66454e-15
         3.747e-16     1.0           -1.66533e-15
        -1.94289e-16   5.55112e-16    1.0
```

Рис. 12: Примеры собственной декомпозиции матрицы A

Факторизация. Специальные матричные структуры

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

```
[159]: 1000×1000 Matrix{Float64}:
        -1.17534     0.0684532   1.21081   …   0.332002   -0.300338    1.08963
         1.37781     0.468895    0.138705      0.715582   -0.127683   -0.979926
        -0.139675   -0.494688    1.09263       1.14976    -0.177964    1.2212
        -0.441907    0.731104   -0.369627      0.7362      0.739588    0.783094
         0.249947    0.739086    0.742876     -2.68047     0.455422    0.590739
         0.153848    0.686872   -0.984271  …   0.753231    0.948394    0.773453
        -0.16179    -0.43104     0.437299     -1.29949     0.260508    0.797469
        -1.05836     0.501389    0.657223     -0.220854    1.65408     0.478116
         1.79979    -1.12349     0.425852     -1.31802    -0.303627    0.591483
        -0.205502   -1.85456     1.64603      -0.753123   -0.470401    1.83961
         1.40937     1.1547      0.57341   …   0.785327    0.989018    0.345891
        -0.271228    0.672036   -0.690399      1.08188     0.122238   -0.0285673
         0.664249    0.5067      1.74936       0.191251   -0.0534473  -0.212219
                                                                ⋮
         0.137533   -1.05004    -0.811184      0.267977    2.10579     1.63683
         0.547193   -0.0340197   0.257343      1.55995    -1.00064    -0.548276
        -0.0394447  -0.550474   -2.20466   …  -0.311342   -0.0037587  -0.438719
         1.80858    -1.08709    -0.0530896    -0.635975   -2.21803    -1.45874
        -0.37211     0.166729   -0.993035     -1.81519    -0.0598573   1.67137
        -1.62599     1.53308    -1.52491      0.336111   -2.23549    -0.507685
         0.680747    2.5961     -0.339865     -1.42953     1.17323     0.107737
         1.15732     0.306048   -1.68074   …   0.300232    0.855597   -0.623833
         0.876841    0.958418   -0.166328     -0.957556    0.932118   -1.17832
         0.41736     0.847411    0.578494     -0.367679    1.10689     0.0660642
        -0.592137    2.21681     0.296898      1.52435    -0.527224   -0.13481
         1.10391     0.990593    0.362526      1.65636     0.686678   -1.63999
```

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

```
[161]: 1000×1000 Matrix{Float64}:
        -2.35869     1.44626     0.880111  …   0.758262   -0.892675    2.19355
         1.44626     0.93779    -0.363983      1.56293     2.00912     0.0106673
         0.880111   -0.365983    2.18526       1.72826     0.118134   -0.858678
        -1.92841     1.82863    -0.698332     -0.681973    1.22167    -0.488772
         0.517264    0.522002    1.18931      -2.21172     1.12622    -0.417693
        -0.185323    0.183076    0.327769  …   1.6548     -1.72754     0.221689
        -0.991409   -1.22231    -1.19436      -1.80664    -0.713351    2.32826
         0.891838    0.0818796   1.2878        0.0949028   2.21334     0.737791
         1.39934    -2.699      0.814663     -1.94438    -0.567002   -0.214042
        -0.560068   -1.69357    2.79061      -1.90138     1.28306     0.0004
         2.91243    -0.385417   -0.698671  …   1.55575     3.69028     1.37357
        -0.516356   -0.0116484  0.219428      0.911156    1.29962     1.55691
        -0.638639   -0.197656   2.58506       1.86344    -1.0776     -2.23549
                                                                ⋮
        -1.79222    -0.638007   -0.693538     2.41898     1.71785     3.85931
        -1.29327     0.0546353  2.62131       0.781614   -1.54348     0.478431
        -0.685307    0.480966   -2.85647   …  -1.09484     0.29178    -0.893044
         1.72707    -2.47819    0.321298     -0.148738   -1.15318    -2.97902
        -0.0858276  -0.243105   -1.66669     -0.838212   -1.19282     0.590454
        -2.29433     2.68759    -1.98583     -0.734156   -1.9004     -2.17635
        -0.0109699   2.34292    -0.595694    -0.6104     3.5229      -0.110978
         2.14381    -0.952126   -0.811001  …   0.1092     0.356424    -0.205193
         0.942865    1.77428    -0.437756     0.262112    0.585657    -3.32195
         0.750262    1.56223     1.72826     -0.735358    2.63044     1.7224
        -0.892675    2.00912     0.118134     2.63044    -1.05445     0.551869
         2.19355     0.0106673  -0.858678     1.7224      0.551869    -3.27997
```

Рис. 13: Примеры работы с матрицами большой размерности и специальной структуры

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

[163]: true

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

[167]: 1.4462602305066259

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

[169]: false
```

Рис. 14: Пример добавления шума в симметричную матрицу

Факторизация. Специальные матричные структуры

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

```
[171]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
 -2.35069     1.44626     0.880131  …   0.750262   -0.892475    2.19355
  1.44626     0.93779    -0.363903      1.56293     2.08912     0.0106673
  0.880131   -0.363903    2.18526       1.72826     0.118134   -0.858678
 -1.92841     1.82863    -0.698312     -0.681973    1.22167    -0.488772
  0.517264    0.522082    1.10931      -2.21172     1.12022    -0.417693
 -0.185323    0.183076    0.327769  …   1.6548     -1.72754     0.221689
 -0.991409   -1.22231    -1.19436      -1.80664    -0.713351    2.32826
 -0.891838    0.0818796   1.2878        0.0949028   2.21334     0.737791
  1.39934    -2.699       0.814663     -1.94438    -0.567002   -0.214042
 -0.560068   -1.69357     2.79061      -1.90338     1.28306     4.0004
  2.91243    -0.385417   -0.698671  …   1.55575     3.69028     1.37357
 -0.516356   -0.0316684   0.219428      0.911356    1.29962     1.55691
 -0.638639   -0.197656    2.58506       1.86344    -1.0776     -2.23549
  ⋮                                  ⋱
 -1.79222    -0.638007   -0.693538      2.41098     1.71785     3.85931
 -1.29327     0.0546353   2.62131       0.781614   -1.54348     0.478431
 -0.685307    0.480966   -2.85647   …  -1.99484     0.29378    -0.893044
  1.72707    -2.47019     0.321298      0.148738   -1.35318    -2.97902
 -0.0850276  -0.243105   -1.66669      -0.830212   -1.19282     0.506054
 -2.29433     2.68759    -1.80583      -0.714356   -1.3604     -2.17635
 -0.0199699   2.34292    -0.595694     -0.6394      3.5299     -0.110978
  2.14381    -0.952326   -0.811001  …   0.1992      0.356424   -0.205393
  0.942865    1.77428    -0.437756      0.262112    0.585657   -3.32395
  0.750262    1.56293     1.72826      -0.735358    2.63044     1.7224
 -0.892475    2.08912     0.118134      2.63044    -1.05445     0.551869
  2.19355     0.0106673  -0.858678      1.7224      0.551869   -3.27997
```

Рис. 15: Пример явного объявления структуры матрицы

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

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

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

```
  1.389 s (27 allocations: 7.93 MiB)
```

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

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

Рис. 16: Использование пакета BenchmarkTools

Факторизация. Специальные матричные структуры

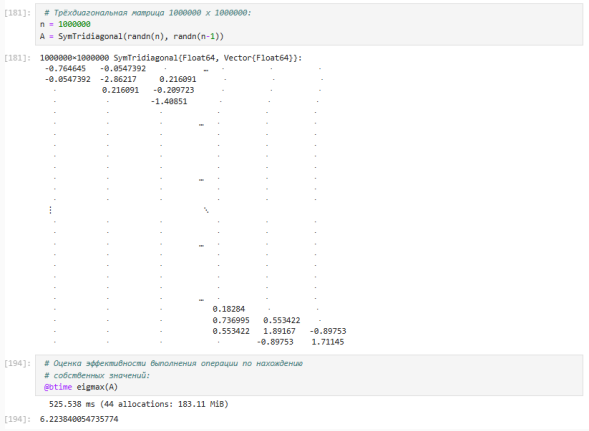


Рис. 17: Примеры работы с разряженными матрицами большой размерности

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

[196]: 3×3 Matrix{Rational{BigInt}}:
        2//5   1     9//10
        1//5  3//10  9//10
        1//5  9//10  2//5

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

[198]: 3-element Vector{Int64}:
        1
        1
        1

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

[200]: 3-element Vector{Rational{BigInt}}:
        23//10
         7//5
         3//2

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

[202]: 3-element Vector{Rational{BigInt}}:
        1
        1
        1

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

[204]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
       L factor:
       3×3 Matrix{Rational{BigInt}}:
        1      0    0
        1//2   1    0
        1//2  -1//2  1
       U factor:
       3×3 Matrix{Rational{BigInt}}:
        2//5   1     9//10
        0     2//5  -1//20
        0     0     17//40
```

Рис. 18: Решение системы линейных уравнений с рациональными элементами без преобразования в

Самостоятельная работа

Выполнение задания “Произведение векторов”:

```
[206]: using LinearAlgebra

       # Задаём вектор
       v = [1, 2, 3]

       # Скалярное произведение
       dot_v = dot(v, v)

[206]: 14

[208]: # Матричное (внешнее) произведение
       outer_v = v * v'

[208]: 3×3 Matrix{Int64}:
        1  2  3
        2  4  6
        3  6  9
```

Рис. 19: Решение задания “Произведение векторов”

Выполнение задания “Системы линейных уравнений”:

```
[211]: using LinearAlgebra

       # Система a
       A1 = [1 1; 1 -1]
       b1 = [2; 3]
       x1 = A1 \ b1
       println("Система a: x = $x1")

       # Система b
       A2 = [1 1; 2 2]
       b2 = [2; 4]
       try
           x2 = A2 \ b2
           println("Система b: x = $x2")
       catch e
           println("Система b: Нет решения (линейно зависимая система)")
       end

       # Система c
       A3 = [1 1; 2 2]
       b3 = [2; 5]
       try
           x3 = A3 \ b3
           println("Система c: x = $x3")
       catch e
           println("Система c: Нет решения (несовместная система)")
       end

       # Система d
       A4 = [1 1; 2 2; 3 3]
       b4 = [1; 2; 3]
       try
           x4 = A4 \ b4
           println("Система d: x = $x4")
       catch e
           println("Система d: Нет решения (линейно зависимая система)")
       end

       # Система e
       A5 = [1 1; 2 1; 1 -1]
       b5 = [2; 1; 3]
       try
           x5 = A5 \ b5
           println("Система e : x = $x5")
       catch e
           println("Система e: Нет решения (несовместная система)")
       end

       # Система f
       A6 = [1 1; 2 1; 3 2]
       b6 = [2; 1; 3]
       try
           x6 = A6 \ b6
           println("Система f : x = $x6")
       catch e
           println("Система f: Нет решения (сингулярная матрица или неопределённое решение)")
       end
```

```
Система a: x = [2.5, -0.5]
Система b: Нет решения (линейно зависимая система)
Система c: Нет решения (несовместная система)
Система d: x = [0.5, 0.5]
Система e : x = [1.5000000000000004, -0.9999999999999997]
Система f : x = [-0.9999999999999964, 2.9999999999999982]
```

Рис. 20: Решение задания “Системы линейных уравнений”

Выполнение задания “Системы линейных уравнений”:

[213]:
```
using LinearAlgebra

# Система a
A1 = [1 1 1; 1 -1 -2]
b1 = [2; 3]
try
    x1 = A1 \ b1
    println("Система a : x = $x1")
catch e
    println("Система a: Нет решения (недостаточно уравнений для определения решения)")
end

# Система b
A2 = [1 1 1; 2 2 -3; 3 1 1]
b2 = [2; 4; 1]
try
    x2 = A2 \ b2
    println("Система b : x = $x2")
catch e
    println("Система b: Нет решения")
end

# Система c
A3 = [1 1 1; 1 1 2; 2 2 3]
b3 = [1; 0; 1]
try
    x3 = A3 \ b3
    println("Система c : x = $x3")
catch e
    println("Система c: Нет решения (сингулярная матрица или неопределённое решение)")
end

# Система d
A4 = [1 1 1; 1 1 2; 2 2 3]
b4 = [1; 0; 0]
try
    x4 = A4 \ b4
    println("Система d : x = $x4")
catch e
    println("Система d: Нет решения (недостаточно уравнений для определения решения)")
end
```

```
Система a : x = [2.2142857142857115, 0.3571428571428571, -0.5714285714285711]
Система b : x = [-0.5, 2.5, 0.0]
Система c: Нет решения (сингулярная матрица или неопределённое решение)
Система d: Нет решения (недостаточно уравнений для определения решения)
```

Рис. 21: Решение задания “Системы линейных уравнений”

Выполнение задания “Операции с матрицами”:

```
[219]: # Матрица a

       A = [1 -2; -2 1]

       eigen_A = eigen(A) # Собственные значения и векторы
       diag_matrix = Diagonal(eigen_A.values) # Диагональная матрица
```

```
[219]: 2×2 Diagonal{Float64, Vector{Float64}}:
        -1.0    ⋅
         ⋅     3.0
```

```
[223]: # Матрица b

       B = [1 -2; -2 3]

       eigen_B = eigen(B) # Собственные значения и векторы
       diag_matrix = Diagonal(eigen_B.values) # Диагональная матрица
```

```
[223]: 2×2 Diagonal{Float64, Vector{Float64}}:
        -0.236068    ⋅
         ⋅          4.23607
```

```
[225]: # Матрица c

       C = [1 -2 0; -2 1 2; 0 2 0]

       eigen_C = eigen(C) # Собственные значения и векторы
       diag_matrix = Diagonal(eigen_C.values) # Диагональная матрица
```

```
[225]: 3×3 Diagonal{Float64, Vector{Float64}}:
        -2.14134    ⋅          ⋅
         ⋅         0.515138    ⋅
         ⋅          ⋅         3.6262
```

Рис. 22: Решение задания “Операции с матрицами”

Выполнение задания “Операции с матрицами”:

[229]:
```
# Исходная матрица (a)
A = [1 -2
     -2 1]

# Собственные значения и векторы
eigen_decomp = eigen(A)
P = eigen_decomp.vectors # Матрица собственных векторов
D = Diagonal(eigen_decomp.values) # Диагональная матрица собственных значений

# Возведение диагональной матрицы в 10-ю степень
D_10 = D.^10

# Вычисление A^10
A_10 = P * D_10 * inv(P)

println("Матрица A^10:")
println(A_10)
```

```
Матрица A^10:
[29525.0 -29524.0; -29524.0 29525.0]
```

Рис. 23: Решение задания “Операции с матрицами”

Выполнение задания “Операции с матрицами”:

[236]:
```
# Исходная матрица (b)
A = [5 -2;
     -2 5]

# Собственные значения и векторы
eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

# Проверяем, что собственные значения неотрицательные
if all(eigenvalues .>= 0)
    # Диагональная матрица с квадратными корнями собственных значений
    sqrt_D = Diagonal(sqrt.(eigenvalues))

    # Квадратный корень матрицы
    sqrt_A = eigenvectors * sqrt_D * inv(eigenvectors)

    println("Исходная матрица A:")
    println(A)

    println("\nКвадратный корень матрицы sqrt(A):")
    println(sqrt_A)

    # Проверка, что sqrt(A)^2 = A
    println("\nПроверка: sqrt(A)^2:")
    println(sqrt_A * sqrt_A)
else
    println("Матрица A имеет отрицательные собственные значения, квадратный корень не определён.")
end
```

```
Исходная матрица A:
[5 -2; -2 5]

Квадратный корень матрицы sqrt(A):
[2.188901059316734 -0.4568502517485676; -0.4568502517485676 2.188901059316734]

Проверка: sqrt(A)^2:
[5.000000000000002 -2.000000000000001; -2.000000000000001 5.000000000000002]
```

Рис. 24: Решение задания “Операции с матрицами”

Выполнение задания “Операции с матрицами”:

[241]:
```
# Исходная матрица (c)
A = [1 -2;
     -2 1]

# Собственные значения и векторы
eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

# Преобразуем собственные значения в комплексные для вычисления кубического корня
complex_eigenvalues = Complex.(eigenvalues)
cube_root_D = Diagonal(complex_eigenvalues .^ (1/3))

# Кубический корень матрицы
cube_root_A = eigenvectors * cube_root_D * inv(eigenvectors)

println("Исходная матрица A:")
println(A)

println("\nКубический корень матрицы ∛(A):")
println(cube_root_A)

# Проверка: (∛(A))^3 = A
println("\nПроверка: (∛(A))^3:")
println(cube_root_A * cube_root_A * cube_root_A)
```

```
Исходная матрица A:
[1 -2; -2 1]

Кубический корень матрицы ∛(A):
ComplexF64[0.971124785153704 + 0.43301270189221933im -0.47112478515370404 + 0.43301270189221933im; -0.47112478515370404
+ 0.43301270189221933im 0.971124785153704 + 0.43301270189221933im]

Проверка: (∛(A))^3:
ComplexF64[0.9999999999999991 + 0.0im -1.9999999999999991 + 5.551115123125783e-17im; -1.9999999999999991 + 5.55111512
3125783e-17im 0.9999999999999991 + 0.0im]
```

Рис. 25: Решение задания “Операции с матрицами”

Выполнение задания “Операции с матрицами”:

[243]:
```
# Исходная матрица (d)
A = [1 2;
     2 3]

# Собственные значения и векторы
eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

# Проверяем, что собственные значения неотрицательные
if all(eigenvalues .>= 0)
    # Диагональная матрица с квадратными корнями собственных значений
    sqrt_D = Diagonal(sqrt.(eigenvalues))

    # Квадратный корень матрицы
    sqrt_A = eigenvectors * sqrt_D * inv(eigenvectors)

    println("Исходная матрица A:")
    println(A)

    println("\nКвадратный корень матрицы sqrt(A):")
    println(sqrt_A)

    # Проверка, что sqrt(A)^2 = A
    println("\nПроверка: sqrt(A)^2:")
    println(sqrt_A * sqrt_A)
else
    println("Матрица A имеет отрицательные собственные значения, квадратный корень не определён.")
end
```

```
Матрица A имеет отрицательные собственные значения, квадратный корень не определён.
```

Рис. 26: Решение задания “Операции с матрицами”

Выполнение задания “Операции с матрицами”:

```
[246]: # Исходная матрица A
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
]

# Нахождение собственных значений и векторов
@btime eigen_decomp = eigen(A)
eigenvalues = eigen_decomp.values
eigenvectors = eigen_decomp.vectors

println("Собственные значения матрицы A:")
println(eigenvalues)

# Создание диагональной матрицы из собственных значений
# Прямое создание переменной и вывод без использования @btime
diag_matrix = Diagonal(eigenvalues)

println("\nДиагональная матрица из собственных значений:")
println(Matrix(diag_matrix))  # Преобразуем в стандартный массив для вывода

# Создание нижнедиагональной матрицы из A
lower_triangular = LowerTriangular(A)

println("\nНижнедиагональная матрица из A:")
println(Matrix(lower_triangular))

# Оценка эффективности
println("\nЭффективность выполнения операций:")
@btime eigen(A)
@btime Diagonal(eigenvalues)
@btime LowerTriangular(A)
```

```
  7.433 µs (19 allocations: 3.05 KiB)
Собственные значения матрицы A:
[-0.2368579774997897, 4.236857977499789]

Диагональная матрица из собственных значений:
[-0.2368579774997897 0.0; 0.0 4.236857977499789]

Нижнедиагональная матрица из A:
[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]

Эффективность выполнения операций:
  9.975 µs (19 allocations: 3.05 KiB)
  283.402 ns (1 allocation: 16 bytes)
  309.129 ns (1 allocation: 16 bytes)
```

```
[246]: 5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140   ⋅   ⋅   ⋅   ⋅
  97  106   ⋅   ⋅   ⋅
  74   89  152   ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рис. 27: Решение задания “Операции с матрицами”

Выполнение задания “Линейные модели экономики”:

[254]:
```
# Матрицы
A1 = [1 2; 3 4]
A2 = (1/2) * A1
A3 = (1/10) * A1
A4 = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]

# Функция проверки через (E - A)^(-1)
function check_productivity_via_inverse(A)
    E = I(size(A, 1))
    B = E - A
    try
        B_inv = inv(B)
        all(B_inv .>= 0)
    catch
        false
    end
end

# Функция проверки спектральная критерия
function check_productivity_via_spectrum(A)
    eigenvalues = eigvals(A)
    all(abs.(eigenvalues) .< 1)
end

# Проверка продуктивности для всех матриц
matrices = [A1, A2, A3, A4]
for (i, A) in enumerate(matrices)
    println("Matrix A$i:")
    println(A)
    println("Via (E - A)^(-1): ", check_productivity_via_inverse(A))
    println("Via spectrum: ", check_productivity_via_spectrum(A))
    println("-"^30)
end
```

```
Matrix A1:
[1.0 2.0; 3.0 4.0]
Via (E - A)^(-1): false
Via spectrum: false
------------------------------
Matrix A2:
[0.5 1.0; 1.5 2.0]
Via (E - A)^(-1): false
Via spectrum: false
------------------------------
Matrix A3:
[0.1 0.2; 0.30000000000000004 0.4]
Via (E - A)^(-1): true
Via spectrum: true
------------------------------
Matrix A4:
[0.1 0.2 0.3; 0.0 0.1 0.2; 0.0 0.1 0.3]
Via (E - A)^(-1): true
Via spectrum: true
------------------------------
```

Рис. 28: Решение задания “Линейные модели экономики”

Вывод

Вывод

- В ходе выполнения лабораторной работы были изучены возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Список литературы. Библиография

Список литературы. Библиография

[1] Julia Documentation: https://docs.julialang.org/en/v1/

Спасибо за внимание!