Отчёт по лабораторной работе №4

Линейная алгебра

Ким Реачна

Содержание

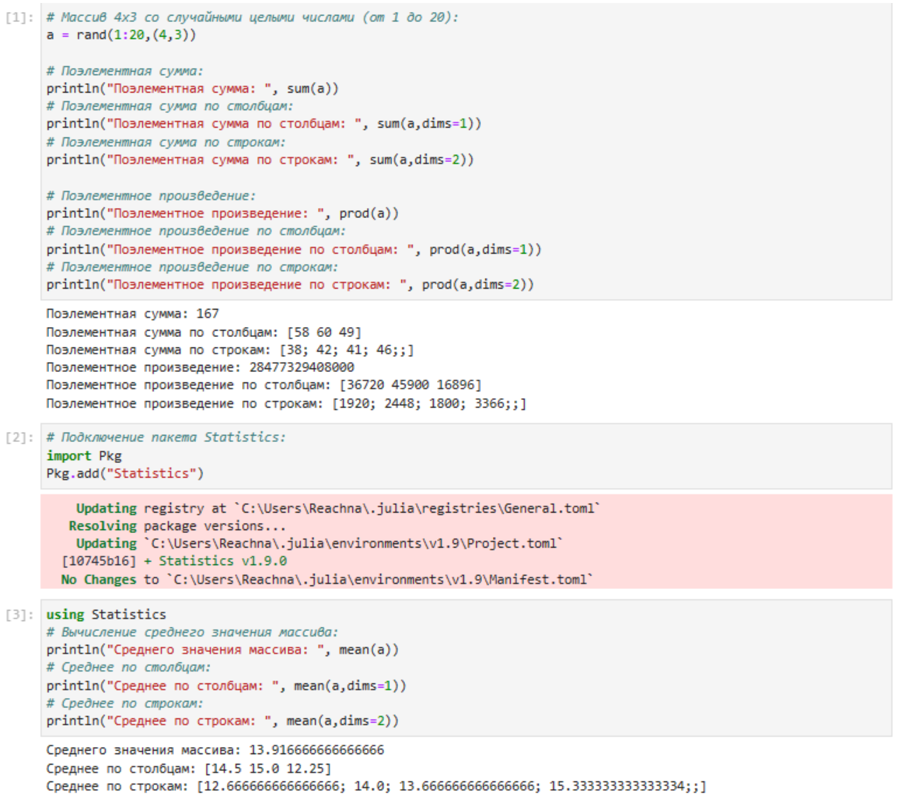
Список иллюстраций

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

2.1 Поэлементные операции над многомерными массивами



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

# Поэлементная сумма:
println("Поэлементная сумма: ", sum(a))
# Поэлементная сумма по столбцам:
println("Поэлементная сумма по столбцам: ", sum(a,dims=1))
# Поэлементная сумма по строкам:
println("Поэлементная сумма по строкам: ", sum(a,dims=2))

# Поэлементное произведение:
println("Поэлементное произведение: ", prod(a))
# Поэлементное произведение по столбцам:
println("Поэлементное произведение по столбцам: ", prod(a,dims=1))
# Поэлементное произведение по строкам:
println("Поэлементное произведение по строкам: ", prod(a,dims=2))

Поэлементная сумма: 167
Поэлементная сумма по столбцам: [58 60 49]
Поэлементная сумма по строкам: [38; 42; 41; 46;;]
Поэлементное произведение: 28477329408000
Поэлементное произведение по столбцам: [36720 45900 16896]
Поэлементное произведение по строкам: [1920; 2448; 1800; 3366;;]

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

    Updating registry at `C:\Users\Reachna\.julia\registries\General.toml`
   Resolving package versions...
    Updating `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
  [10745b16] + Statistics v1.9.0
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`

[3]: using Statistics
# Вычисление среднего значения массива:
println("Среднего значения массива: ", mean(a))
# Среднее по столбцам:
println("Среднее по столбцам: ", mean(a,dims=1))
# Среднее по строкам:
println("Среднее по строкам: ", mean(a,dims=2))

Среднего значения массива: 13.916666666666666
Среднее по столбцам: [14.5 15.0 12.25]
Среднее по строкам: [12.666666666666666; 14.0; 13.666666666666666; 15.333333333333334;;]
```

Рис. 2.1: Примеры с поэлементными операциями над многомерными массивами

2.2 Транспонирование, след, ранг, определитель и инверсия матрицы

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

       Resolving package versions...
        Updating `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
      [37e2e46d] + LinearAlgebra
      No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`

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

[5]: 4×4 Matrix{Int64}:
      17   8  1  16
       8  15  8  17
       2  15  5  20
       6  19  4   3

[6]: # Транспонирование:
     println("Транспонирование: ", transpose(b))
     # След матрицы (сумма диагональных элементов):
     println("След матрицы (сумма диагональных элементов): ", tr(b))
     # Извлечение диагональных элементов как массив:
     println("Извлечение диагональных элементов как массив: ", diag(b))
     # Ранг матрицы:
     println("Ранг матрицы: ", rank(b))

     Транспонирование: [17 8 2 6; 8 15 15 19; 1 8 5 4; 16 17 20 3]
     След матрицы (сумма диагональных элементов): 40
     Извлечение диагональных элементов как массив: [17, 15, 5, 3]
     Ранг матрицы: 4

[7]: # Инверсия матрицы (определение обратной матрицы):
     println("Инверсия матрицы: ")
     inv(b)

     Инверсия матрицы:

[7]: 4×4 Matrix{Float64}:
       0.0488384     0.032779   -0.0680218   0.00725977
      -0.000791975  -0.0590901   0.0407427   0.0674498
      -0.0815734     0.247052   -0.136836   -0.0526663
       0.0161035    -0.0207233   0.0604541  -0.0381468

[8]: # Определитель матрицы:
     println("Определитель матрицы: ", det(b))

     Определитель матрицы: 22728.000000000004

[9]: # Псевдобратная функция для прямоугольных матриц:
     println("Псевдобратная функция для прямоугольных матриц: ")
     pinv(a)

     Псевдобратная функция для прямоугольных матриц:

[9]: 3×4 Matrix{Float64}:
       0.0599931  -0.0750363   0.0543877  -0.00778499
      -0.113035    0.08467    -0.0099743   0.0466985
       0.0879067   0.00491167 -0.0325707  -0.0263336
```

Рис. 2.2: Примеры с транспонированием, трассировкой, ранжированием, определителем и матричной инверсией

2.3 Вычисление нормы векторов и матриц, повороты, вращения

```
[10]: # Создание вектора X:
      X = [2, 4, -5]
      # Вычисление евклидовой нормы:
      println("Евклидовой нормы: ", norm(X))
      # Вычисление p-нормы:
      p = 1
      println("p-нормы: ", norm(X,p))

      Евклидовой нормы: 6.708203932499369
      p-нормы: 11.0

[11]: # Расстояние между двумя векторами X и Y:
      X = [2, 4, -5];
      Y = [1,-1,3];
      println("Расстояние между двумя векторами X и Y: ", norm(X-Y))
      # Проверка по базовому определению:
      println("Расстояние по базовому определению: ", sqrt(sum((X-Y).^2)))

      Расстояние между двумя векторами X и Y: 9.486832980505138
      Расстояние по базовому определению: 9.486832980505138

[12]: # Угол между двумя векторами:
      println("Угол между двумя векторами: ", acos((transpose(X)*Y)/(norm(X)*norm(Y))))

      Угол между двумя векторами: 2.4404307889469252

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

[13]: 3×3 Matrix{Int64}:
        5  -4  2
       -1   2  3
       -2   1  0

[14]: # Вычисление Евклидовой нормы:
      println("Евклидовой нормы: ", opnorm(d))
      # Вычисление p-нормы:
      p=1
      println("p-нормы: ", opnorm(d,p))

      Евклидовой нормы: 7.147682841795258
      p-нормы: 8.0

[15]: # Поворот на 180 градусов:
      println("Поворот на 180 градусов", rot180(d))
      # Переворачивание строк:
      println("Переворачивание строк: ", reverse(d,dims=1))
      # Переворачивание столбцов
      println("Переворачивание столбцов: ", reverse(d,dims=2))

      Поворот на 180 градусов[0 1 -2; 3 2 -1; 2 -4 5]
      Переворачивание строк: [-2 1 0; -1 2 3; 5 -4 2]
      Переворачивание столбцов: [2 -4 5; 3 2 -1; 0 1 -2]
```

Рис. 2.3: Примеры с вычислением нормы векторов и матриц, повороты, вращения

2.4 Матричное умножение, единичная матрица, скалярное произведение

```
[17]: # Матрица 2х3 со случайными целыми значениями от 1 до 10:
      A = rand(1:10,(2,3))
      # Матрица 3х4 со случайными целыми значениями от 1 до 10:
      B = rand(1:10,(3,4))
      # Произведение матриц A и B:
      A*B

[17]: 2×4 Matrix{Int64}:
       73  37  18   66
       87  60  31  117

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

[18]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1

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

[19]: -17

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

[20]: -17
```

Рис. 2.4: Примеры с матричным умножением, единичной матрицей, скалярным произведением

2.5 Факторизация. Специальные матричные структуры

```
[21]: # Задаём квадратную матрицу 3x3 со случайными значениями:
      A = rand(3, 3)
      # Задаём единичный вектор:
      x = fill(1.0, 3)
      # Задаём вектор b:
      b = A*x
      # Решение исходного уравнения получаем с помощью функции \
      # (убеждаемся, что x - единичный вектор):
      A\b

[21]: 3-element Vector{Float64}:
       0.9999999999999998
       1.0
       1.0

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

[22]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3×3 Matrix{Float64}:
       1.0       0.0        0.0
       0.615141  1.0        0.0
       0.321768  0.0235118  1.0
      U factor:
      3×3 Matrix{Float64}:
       0.909254  0.606996  0.188041
       0.0       0.57954   0.63612
       0.0       0.0       0.861268

[23]: # Матрица перестановок:
      Alu.P

[23]: 3×3 Matrix{Float64}:
       0.0  1.0  0.0
       0.0  0.0  1.0
       1.0  0.0  0.0

[24]: # Вектор перестановок:
      Alu.p

[24]: 3-element Vector{Int64}:
       2
       3
       1

[25]: # Матрица L:
      Alu.L

[25]: 3×3 Matrix{Float64}:
       1.0       0.0        0.0
       0.615141  1.0        0.0
       0.321768  0.0235118  1.0
```

Рис. 2.5: Примеры с LU-факторизацией

```
[26]: # Матрица U:
      Alu.U

[26]: 3×3 Matrix{Float64}:
       0.909254  0.606996  0.188041
       0.0       0.57954   0.63612
       0.0       0.0       0.861268

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

[27]: 3-element Vector{Float64}:
       0.9999999999999998
       1.0
       1.0

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

[28]: 3-element Vector{Float64}:
       0.9999999999999998
       1.0
       1.0

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

[29]: 0.45384413687575187

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

[30]: 0.45384413687575187

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

[31]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
      Q factor:
      3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
       -0.264319  -0.12994   -0.955642
       -0.821459  -0.488837   0.293673
       -0.505313   0.862644   0.0224688
      R factor:
      3×3 Matrix{Float64}:
       -1.10688  -1.03538   -0.781953
        0.0       0.498166   0.434889
        0.0       0.0       -0.823064
```

Рис. 2.6: QR-факторизация

```
[32]: # Матрица Q:
      Aqr.Q

[32]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
       -0.264319  -0.12994   -0.955642
       -0.821459  -0.488837   0.293673
       -0.505313   0.862644   0.0224688

[33]: # Матрица R:
      Aqr.R

[33]: 3×3 Matrix{Float64}:
       -1.10688  -1.03538   -0.781953
        0.0       0.498166   0.434889
        0.0       0.0       -0.823064

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

[34]: 3×3 Matrix{Float64}:
       1.0  -1.66533e-16   0.0
       0.0   1.0          -4.44089e-16
       0.0  -3.33067e-16   1.0

[35]: # Симметризация матрицы A:
      Asym = A + A'

[35]: 3×3 Matrix{Float64}:
       0.585137  1.11819  1.49605
       1.11819   1.21399  1.14097
       1.49605   1.14097  1.50358

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

[36]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
      values:
      3-element Vector{Float64}:
       -0.5696462945958927
        0.2280379804830429
        3.6443212274452446
      vectors:
      3×3 Matrix{Float64}:
        0.844286   0.127328  -0.520546
       -0.215406  -0.808803  -0.54721
       -0.490694   0.574131  -0.655434

[37]: # Собственные значения:
      AsymEig.values

[37]: 3-element Vector{Float64}:
       -0.5696462945958927
        0.2280379804830429
        3.6443212274452446
```

Рис. 2.7: Симметризация матрицы, Спектральное разложение

[38]:
```
#Собственные векторы:
AsymEig.vectors
```

[38]:
```
3×3 Matrix{Float64}:
  0.844286   0.127328  -0.520546
 -0.215406  -0.808803  -0.54721
 -0.490694   0.574131  -0.655434
```

[39]:
```
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym
```

[39]:
```
3×3 Matrix{Float64}:
  1.0          6.66134e-16   1.11022e-15
 -1.33227e-15  1.0          -2.22045e-15
  2.66454e-15  6.66134e-16   1.0
```

[40]:
```
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
```

[40]:
```
1000×1000 Matrix{Float64}:
  0.563565      1.04686   -0.17552    …   0.732417   -0.352508    0.572473
  1.24608      -0.813521  -1.12819       -0.205106    0.486897    0.428047
  1.09677      -0.131553   0.0147621     -0.73089    -1.36957    -1.73431
  0.00612529   -0.061649   0.73033        1.33844     0.83394     1.00052
  0.327113     -0.490315   0.966332       0.681366   -0.57985     1.47336
 -0.447071      1.06384   -1.3739     …   1.06415     0.226066   -1.77447
  0.843396     -0.116513  -0.603763       0.642849    1.08515     0.313099
 -0.000157182   0.311643   0.522261       1.57937     1.20215     0.478557
 -0.150693      0.21262   -0.257269      -0.531689    1.04099    -1.76959
  0.568932     -0.656127  -0.618355       0.230223   -0.318063    0.666008
  0.589237      1.30122    0.988794   …  -0.747055    0.401636    0.805042
 -1.38645       0.170924  -0.911031      -0.122241   -2.61964    -0.655887
 -0.352182     -0.1377     0.731137      -0.481005    0.822418    0.611771
  ⋮                                   ⋱
  0.389215     -0.020974  -0.542638      -0.139909   -1.42999    -1.34067
 -1.13188       0.352305  -0.347941       1.06377    -0.277257   -0.532906
 -0.0315728    -0.250863  -0.229701   …  -0.0856085   0.098191    0.238011
 -1.3502        1.00534   -0.335383       0.94875     0.277358   -0.0610749
  0.0134547     0.881627   0.155217       1.82891    -0.0572179  -1.84264
 -0.678266     -0.994697  -0.223364       0.194877    0.0110706  -0.104862
 -0.721806      1.58744   -2.11496       -0.154554   -0.103842    0.117867
  0.838983     -0.310588   0.47783    …   1.52233     1.30001    -0.0295978
 -1.4324        1.18773   -0.099182       0.556455    0.833313   -0.926812
  0.142712     -1.35304    0.157123       0.269901   -1.89899     0.309658
  0.144455     -0.299778  -0.328117       0.478755    1.24456     0.120033
  0.454165     -0.09356    0.772868       0.296564   -0.445574   -0.409302
```

Рис. 2.8: Симметризация матрицы, Добавление шума

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

```
[41]: 1000×1000 Matrix{Float64}:
        1.12713     2.29294     0.92125     …   0.875129   -0.208053   1.02664
        2.29294    -1.62704    -1.25974        -1.55815     0.187119   0.334487
        0.92125    -1.25974     0.0295241      -0.573767   -1.69769   -0.961445
       -0.592356   -2.18621     1.105           1.15669     1.51627    0.611408
       -0.811693   -1.28434     0.554618        1.86542    -0.293424   2.83478
       -0.272431   -1.24724    -1.83987     …   0.413559   -1.19622   -2.10303
        1.38647    -0.473752   -2.18502         0.474592    1.17278    0.413386
        1.19554     0.494837    1.55956         2.86799     2.10379    1.09358
        0.326293   -0.536223    0.954788       -1.27036     1.89652   -0.920687
       -0.771797   -1.21269     0.858733        0.227956    0.306372   1.25796
       -0.244776    0.537374    2.49184     …  -0.969896    1.73563    1.08437
       -1.2631      2.12378    -2.0678          0.565227   -2.83178   -1.7204
       -0.26491    -0.0472591   1.18405         0.0641268  -0.8119     1.21369
        ⋮                                   ⋱
       -1.10117     0.489811   -0.702793        0.425125   -1.596     -1.86898
       -0.360239    1.5409      1.09111         1.49925     1.48906    2.00088
        0.835484   -0.0468494   0.00423505  …   0.427497   -0.42276    0.977794
       -2.43407     0.669173   -0.593056        0.912447    1.12643    1.24709
        0.313434    0.512936    1.44202         2.11698     1.86649   -3.70162
        0.0915174  -1.45202     0.0508664       0.521804   -0.923898   0.563592
       -2.01179     2.12807    -2.36717        -1.25802    -1.42504    0.262337
        0.317828    1.96872     2.35824     …   2.11963     2.15978   -2.04816
       -0.722444    2.03463    -1.32701         1.12356     0.620509  -0.660104
        0.875129   -1.55815    -0.573767        0.539803   -1.42023    0.606222
       -0.208053    0.187119   -1.69769        -1.42023     2.48911   -0.325541
        1.02664     0.334487   -0.961445        0.606222   -0.325541  -0.818604
```

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

```
[42]: true
```

```
[43]: # Добавление шума:
      Asym_noisy = copy(Asym)
      Asym_noisy[1,2] += 5eps()
      # Проверка, является ли матрица симметричной:
      issymmetric(Asym_noisy)
```

```
[43]: false
```

Рис. 2.9: Симметризация матрицы, Добавление шума

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

[44]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
        1.12713     2.29294     0.92125     …   0.875129   -0.208053   1.02664
        2.29294    -1.62704    -1.25974        -1.55815     0.187119   0.334487
        0.92125    -1.25974     0.0295241      -0.573767   -1.69769   -0.961445
       -0.592356   -2.18621     1.105           1.15669     1.51627    0.611408
       -0.811693   -1.28434     0.554618        1.86542    -0.293424   2.83478
       -0.272431   -1.24724    -1.83987     …   0.413559   -1.19622   -2.10303
        1.38647    -0.473752   -2.18502         0.474592    1.17278    0.413386
        1.19554     0.494837    1.55956         2.86799     2.10379    1.09358
        0.326293   -0.536223    0.954788       -1.27036     1.89652   -0.920687
       -0.771797   -1.21269     0.858733        0.227956    0.306372   1.25796
       -0.244776    0.537374    2.49184     …  -0.969896    1.73563    1.08437
       -1.2631      2.12378    -2.0678          0.565227   -2.83178   -1.7204
       -0.26491    -0.0472591   1.18405         0.0641268  -0.8119     1.21369
        ⋮                                   ⋱
       -1.10117     0.489811   -0.702793        0.425125   -1.596     -1.86898
       -0.360239    1.5409      1.09111         1.49925     1.48906    2.00088
        0.835484   -0.0468494   0.00423505  …   0.427497   -0.42276    0.977794
       -2.43407     0.669173   -0.593056        0.912447    1.12643    1.24709
        0.313434    0.512936    1.44202         2.11698     1.86649   -3.70162
        0.0915174  -1.45202     0.0508664       0.521804   -0.923898   0.563592
       -2.01179     2.12807    -2.36717        -1.25802    -1.42504    0.262337
        0.317828    1.96872     2.35824     …   2.11963     2.15978   -2.04816
       -0.722444    2.03463    -1.32701         1.12356     0.620509  -0.660104
        0.875129   -1.55815    -0.573767        0.539803   -1.42023    0.606222
       -0.208053    0.187119   -1.69769        -1.42023     2.48911   -0.325541
        1.02664     0.334487   -0.961445        0.606222   -0.325541  -0.818604

[45]: import Pkg
      Pkg.add("BenchmarkTools")

         Resolving package versions...
          Updating `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
        [6e4b80f9] + BenchmarkTools v1.3.2
          Updating `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
        [6e4b80f9] + BenchmarkTools v1.3.2
        [9abbd945] + Profile
      Precompiling project...
        ✓ BenchmarkTools
        1 dependency successfully precompiled in 2 seconds. 26 already precompiled.

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

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

Рис. 2.10: Явное указание симетричности матрицы

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

        732.106 ms (14 allocations: 7.93 MiB)

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

        174.454 ms (11 allocations: 7.99 MiB)

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

[49]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
        2.0704    -0.429507   ⋅          …   ⋅          ⋅           ⋅
       -0.429507   1.03656   -0.552678       ⋅          ⋅           ⋅
         ⋅        -0.552678  -0.84854        ⋅          ⋅           ⋅
         ⋅          ⋅        -1.84039        ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅         …   ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅         …   ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
        ⋮                                ⋱
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅         …   ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅             ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅         …   ⋅          ⋅           ⋅
         ⋅          ⋅          ⋅            -0.592926   ⋅           ⋅
         ⋅          ⋅          ⋅            -1.47854    0.869354    ⋅
         ⋅          ⋅          ⋅             0.869354  -0.0454058   1.52004
         ⋅          ⋅          ⋅             ⋅          1.52004     0.18073

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

        530.131 ms (17 allocations: 183.11 MiB)

[50]: 6.4179096859748075
```

Рис. 2.11: Расчет времени выполнения программы

2.6 Общая линейная алгебра


Общая линейная алгебра

```
[51]: 1//2

[51]: 1//2

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

[52]: 3×3 Matrix{Rational{BigInt}}:
       3//5   9//10  2//5
       2//5   4//5   1//5
       9//10  4//5   4//5

[53]: # Единичный вектор:
      x = fill(1, 3)
      # Задаём вектор b:
      b = Arational*x

[53]: 3-element Vector{Rational{BigInt}}:
       19//10
        7//5
        5//2

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

[54]: 3-element Vector{Rational{BigInt}}:
       1//1
       1//1
       1//1

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

[55]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
      L factor:
      3×3 Matrix{Rational{BigInt}}:
       1//1   0//1   0//1
       4//9   1//1   0//1
       2//3  33//40  1//1
      U factor:
      3×3 Matrix{Rational{BigInt}}:
       9//10  4//5   4//5
       0//1   4//9  -7//45
       0//1   0//1  -1//200
```

Рис. 2.12: Общая линейная алгебра

2.7 Задания для самостоятельного выполнения

2.7.1 Произведение векторов

1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результат в dot_v.
2. Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной outer_v.

Рис. 2.13: Произведение векторов

2.7.2 Системы линейных уравнений

1. Решить СЛАУ с двумя неизвестными.

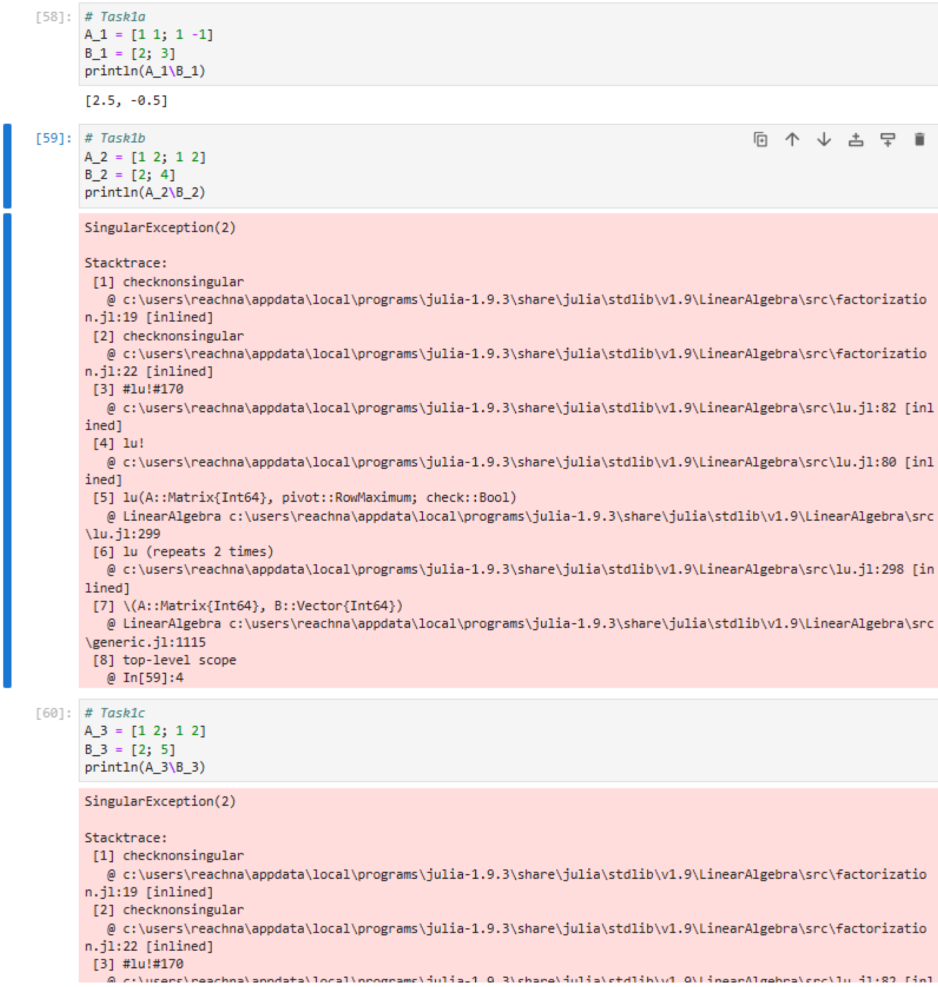


Рис. 2.14: Решение СЛАУ с двумя неизвестными

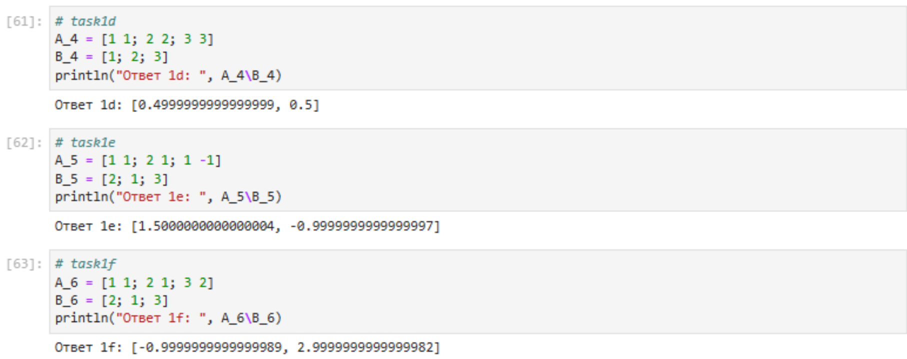
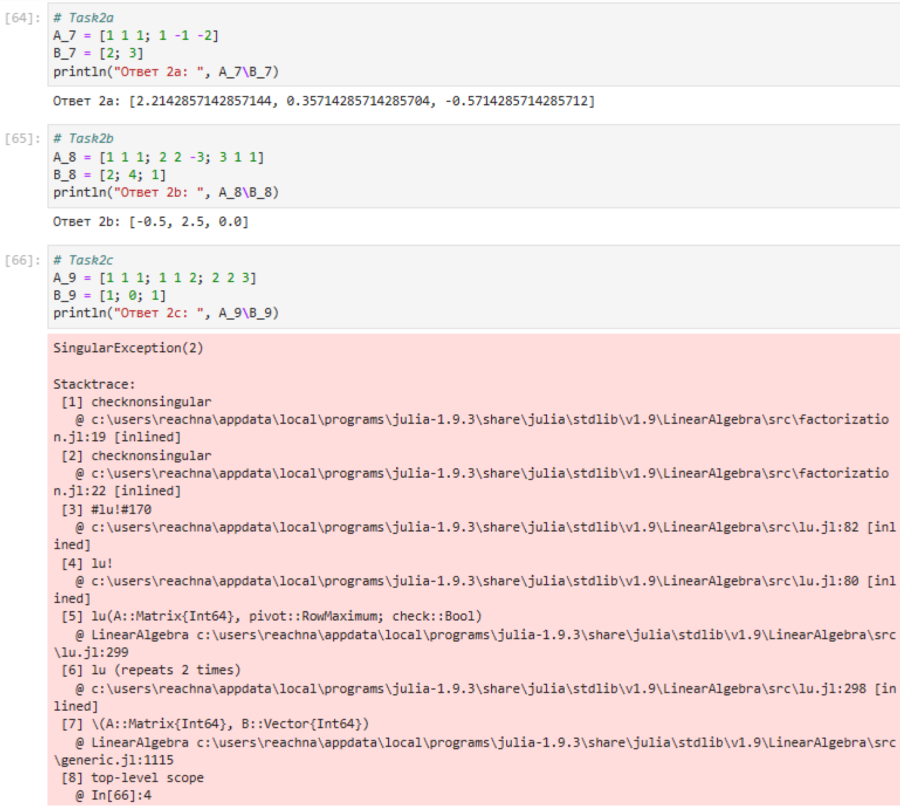


Рис. 2.15: Решение СЛАУ с двумя неизвестными

2. Решить СЛАУ с тремя неизвестными.



```
[64]: # Task2a
A_7 = [1 1 1; 1 -1 -2]
B_7 = [2; 3]
println("Ответ 2a: ", A_7\B_7)

Ответ 2a: [2.2142857142857144, 0.35714285714285704, -0.5714285714285712]

[65]: # Task2b
A_8 = [1 1 1; 2 2 -3; 3 1 1]
B_8 = [2; 4; 1]
println("Ответ 2b: ", A_8\B_8)

Ответ 2b: [-0.5, 2.5, 0.0]

[66]: # Task2c
A_9 = [1 1 1; 1 1 2; 2 2 3]
B_9 = [1; 0; 1]
println("Ответ 2c: ", A_9\B_9)

SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\factorizatio
n.jl:19 [inlined]
 [2] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\factorizatio
n.jl:22 [inlined]
 [3] #lu!#170
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:82 [inl
ined]
 [4] lu!
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:80 [inl
ined]
 [5] lu(A::Matrix{Int64}, pivot::RowMaximum; check::Bool)
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src
\lu.jl:299
 [6] lu (repeats 2 times)
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:298 [in
lined]
 [7] \(A::Matrix{Int64}, B::Vector{Int64})
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src
\generic.jl:1115
 [8] top-level scope
   @ In[66]:4
```

Рис. 2.16: Решение СЛАУ с тремя неизвестными

```
[67]: # Task2d
A_10 = [1 1 1; 1 1 2; 2 2 3]
B_10 = [1; 0; 0]
println("Ответ 2d: ", A_10\B_10)

SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\factorizatio
n.jl:19 [inlined]
 [2] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\factorizatio
n.jl:22 [inlined]
 [3] #lu!#170
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:82 [inl
ined]
 [4] lu!
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:80 [inl
ined]
 [5] lu(A::Matrix{Int64}, pivot::RowMaximum; check::Bool)
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src
\lu.jl:299
 [6] lu (repeats 2 times)
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:298 [in
lined]
 [7] \(A::Matrix{Int64}, B::Vector{Int64})
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src
\generic.jl:1115
 [8] top-level scope
   @ In[67]:4
```

Рис. 2.17: Решение СЛАУ с тремя неизвестными

2.7.3 Операции с матрицами

1. Приведите приведённые ниже матрицы к диагональному виду:

```
[68]: function diagonal_matrix(matrix)
          Asym = matrix + matrix'
          AsymEig = eigen(Asym)
          return inv(AsymEig.vectors) * matrix * AsymEig.vectors
      end

[68]: diagonal_matrix (generic function with 1 method)

[69]: # task1a
      a = [1 -2; -2 1]
      diagonal_matrix(a)

[69]: 2×2 Matrix{Float64}:
       -1.0  0.0
        0.0  3.0

[70]: # task1b
      b = [1 -2; -2 3]
      diagonal_matrix(b)

[70]: 2×2 Matrix{Float64}:
       -0.236068     3.46945e-16
        4.44089e-16  4.23607

[71]: # task1c
      c = [1 -2 0; -2 1 2; 0 2 0]
      diagonal_matrix(c)

[71]: 3×3 Matrix{Float64}:
       -2.14134       3.55271e-15  -1.9984e-15
        3.38618e-15   0.515138      1.11022e-16
       -6.66134e-16  -4.44089e-16   3.6262
```

Рис. 2.18: Диагонализация

2. Вычислите

```
[72]: # Task2a
      a = [1 -2; -2 1]
      a^10

[72]: 2×2 Matrix{Int64}:
        29525  -29524
       -29524   29525

[74]: # Task2b
      b = [5 -2; -2 5]
      sqrt(b)

[74]: 2×2 Matrix{Float64}:
        2.1889   -0.45685
       -0.45685   2.1889

[75]: # Task2c
      c = [1 -2; -2 1]
      c^(1/3)

[75]: 2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
        0.971125+0.433013im  -0.471125+0.433013im
       -0.471125+0.433013im   0.971125+0.433013im

[76]: # Task2d
      d = [1 2; 2 3]
      sqrt(d)

[76]: 2×2 Matrix{ComplexF64}:
       0.568864+0.351578im  0.920442-0.217287im
       0.920442-0.217287im   1.48931+0.134291im
```

Рис. 2.19: Диагонализация

3. Найдите собственные значения матрицы A:

```
[79]: # Task3
      using BenchmarkTools
      A = [140 97 74 168 131; 97 106 89 131 36; 74 89 152 144 7; 168 131 144 54 142; 131 36 71 142 36]

      eigen_value = eigvals(A)

[79]: 5-element Vector{Float64}:
       -133.26347908274784
        -46.31722914150222
         34.7455206405183
        100.90643294309879
        531.9287546406343

[80]: @btime eigvals(A);

        4.157 μs (12 allocations: 2.38 KiB)

[81]: B = zeros(5, 5)
      @btime for i in 1:5
          B[i, i] = eigen_value[i]
      end
      B

        227.292 ns (5 allocations: 80 bytes)

[81]: 5×5 Matrix{Float64}:
       -133.263    0.0       0.0       0.0      0.0
          0.0    -46.3172    0.0       0.0      0.0
          0.0      0.0      34.7455    0.0      0.0
          0.0      0.0       0.0     100.906    0.0
          0.0      0.0       0.0       0.0    531.929

[82]: Alu = lu(A)
      @btime Alu.L

        110.108 ns (1 allocation: 256 bytes)

[82]: 5×5 Matrix{Float64}:
       1.0        0.0        0.0        0.0       0.0
       0.779762   1.0        0.0        0.0       0.0
       0.440476  -0.47314    1.0        0.0       0.0
       0.833333   0.183929  -0.556312   1.0       0.0
       0.577381  -0.459012  -0.189658   0.897068  1.0
```

Рис. 2.20: Собственные значения матрицы

2.7.4 Линейные модели экономики

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

Создала функцию productive_matrix для определения продуктивности матрицы первым вариантом.

```
[85]: function productive_matrix(matrix, size)
          answer = ""
           # единичная матрица
          E = [1 0; 0 1]
          # зададим любые неотрицательные числа
          Y = rand(0:1000, size)
          # По формуле вычислим x - A*x = y
          S = E - matrix
          # найдем значения x
          X = S\Y
          # теперь проверим есть ли среди x отрицательное число
          for i in 1:1:size
              if X[i] < 0
                  answer = "Матрица непродуктивная"
                  break
              else
                  answer = "Матрица продуктивная"
              end
          end
          return answer
      end

[85]: productive_matrix (generic function with 1 method)

[86]: # task4.1.a
      a = [1 2; 3 4]
      productive_matrix(a, 2)

[86]: "Матрица непродуктивная"

[88]: # task 4.1.b
      b = [1 2; 3 4] * (1/2)
      productive_matrix(b, 2)

[88]: "Матрица непродуктивная"

[90]: # Task 4.1.c
      c = [1 2; 3 4] * (1/10)
      productive_matrix(c, 2)

[90]: "Матрица продуктивная"
```

Рис. 2.21: Линейные модели экономики

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

Также создала функцию productive_matrix_2 в которой проверяю матрицы на продуктивность.

```
[91]: # Task2
      function productive_matrix_2(matrix, size)
          # единичная матрица
          ans = ""
          E = [1 0; 0 1]
          matrix_new = E - matrix
          inv_matrix_new = inv(matrix_new)
          for i in 1:1:size
              for j in 1:1:size
                  if inv_matrix_new[i, j] < 0
                      ans = "Матрица непродуктивная"
                      break
                  else
                      ans = "Матрица продуктивная"
                  end
              end
          end
          return ans
      end

[91]: productive_matrix_2 (generic function with 1 method)

[92]: # task4.2.a
      a = [1 2; 3 1]
      productive_matrix_2(a, 2)

[92]: "Матрица непродуктивная"

[93]: # task 4.2.b
      b = [1 2; 3 1] * (1/2)
      productive_matrix_2(b, 2)

[93]: "Матрица непродуктивная"

[94]: # task 4.2.c
      c = [1 2; 3 1] * (1/10)
      productive_matrix_2(c, 2)

[94]: "Матрица продуктивная"
```

Рис. 2.22: Критерий продуктивности

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.

```
[95]: # Task3
function productive_matrix_3(matrix, size)
    ans=""
    # найдем собственные значения переданной матрицы
    eigenvalues = eigvals(matrix)
    for i in 1:1:size
        if abs(eigenvalues[i]) > 1
            ans = "Матрица непродуктивная"
            break
        else
            ans = "Матрица продуктивная"
        end
    end
    return ans
end

[95]: productive_matrix_3 (generic function with 1 method)

[96]: # task4.3.a
a = [1 2; 3 1]
productive_matrix_3(a, 2)

[96]: "Матрица непродуктивная"

[97]: # task4.3.b
b = [1 2; 3 1] * (1/2)
productive_matrix_3(b, 2)

[97]: "Матрица непродуктивная"

[98]: # task4.3.c
c = [1 2; 3 1] * (1/10)
productive_matrix_3(c, 2)

[98]: "Матрица продуктивная"

[99]: # task 4.3.d
d = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
productive_matrix_3(d, 3)

[99]: "Матрица продуктивная"
```

Рис. 2.23: Спектральный критерий продуктивности

3 Листинги программы

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

# Поэлементная сумма:
println("Поэлементная сумма: ", sum(a))
# Поэлементная сумма по столбцам:
println("Поэлементная сумма по столбцам: ", sum(a,dims=1))
# Поэлементная сумма по строкам:
println("Поэлементная сумма по строкам: ", sum(a,dims=2))

# Поэлементное произведение:
println("Поэлементное произведение: ", prod(a))
# Поэлементное произведение по столбцам:
println("Поэлементное произведение по столбцам: ", prod(a,dims=1))
# Поэлементное произведение по строкам:
println("Поэлементное произведение по строкам: ", prod(a,dims=2))
Поэлементная сумма: 167
Поэлементная сумма по столбцам: [58 60 49]
Поэлементная сумма по строкам: [38; 42; 41; 46;;]
Поэлементное произведение: 28477329408000
Поэлементное произведение по столбцам: [36720 45900 16896]
Поэлементное произведение по строкам: [1920; 2448; 1800; 3366;;]
```

```
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics
# Вычисление среднего значения массива:
println("Среднего значения массива: ", mean(a))
# Среднее по столбцам:
println("Среднее по столбцам: ", mean(a,dims=1))
# Среднее по строкам:
println("Среднее по строкам: ", mean(a,dims=2))
Среднего значения массива: 13.916666666666666
Среднее по столбцам: [14.5 15.0 12.25]
Среднее по строкам: [12.666666666666666; 14.0; 13.666666666666666;
15.333333333333334;;]
```

Транспонирование, след, ранг, определитель и инверсия матрицы

```
# Подключение пакета LinearAlgebra:
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
# Массив 4x4 со случайными целыми числами (от 1 до 20):
b = rand(1:20,(4,4))
4×4 Matrix{Int64}:
 17   8  1  16
  8  15  8  17
  2  15  5  20
  6  19  4   3
# Транспонирование:
println("Транспонирование: ", transpose(b))
# След матрицы (сумма диагональных элементов):
```

```
println("След матрицы (сумма диагональных элементов): ", tr(b))
# Извлечение диагональных элементов как массив:
println("Извлечение диагональных элементов как массив: ", diag(b))
# Ранг матрицы:
println("Ранг матрицы: ", rank(b))
Транспонирование: [17 8 2 6; 8 15 15 19; 1 8 5 4; 16 17 20 3]
След матрицы (сумма диагональных элементов): 40
Извлечение диагональных элементов как массив: [17, 15, 5, 3]
Ранг матрицы: 4
# Инверсия матрицы (определение обратной матрицы):
println("Инверсия матрицы: ")
inv(b)
Инверсия матрицы:
4×4 Matrix{Float64}:
  0.0488384     0.032779   -0.0680218   0.00725977
 -0.000791975  -0.0590901   0.0407427   0.0674498
 -0.0815734     0.247052   -0.136836   -0.0526663
  0.0161035    -0.0207233   0.0604541  -0.0381468
# Определитель матрицы:
println("Определитель матрицы: ", det(b))
Определитель матрицы: 22728.000000000004
# Псевдобратная функция для прямоугольных матриц:
println("Псевдобратная функция для прямоугольных матриц: ")
pinv(a)
Псевдобратная функция для прямоугольных матриц:
3×4 Matrix{Float64}:
  0.0599931  -0.0750363    0.0543877  -0.00778499
 -0.113035    0.08467     -0.0099743   0.0466985
  0.0879067   0.00491167  -0.0325707  -0.0263336
```

Вычисление нормы векторов и матриц, повороты, вращения

```
# Создание вектора X:
X = [2, 4, -5]
# Вычисление евклидовой нормы:
println("Евклидовой нормы: ", norm(X))
# Вычисление p-нормы:
p = 1
println("p-нормы: ", norm(X,p))
Евклидовой нормы: 6.708203932499369
p-нормы: 11.0
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
println("Расстояние между двумя векторами X и Y: ", norm(X-Y))
# Проверка по базовому определению:
println("Расстояние по базовому определению: ", sqrt(sum((X-Y).^2)))
Расстояние между двумя векторами X и Y: 9.486832980505138
Расстояние по базовому определению: 9.486832980505138
# Угол между двумя векторами:
println("Угол между двумя векторами: ", acos((transpose(X)*Y)/(norm(X)*norm(Y))))
Угол между двумя векторами: 2.4404307889469252
# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
3×3 Matrix{Int64}:
  5  -4  2
 -1   2  3
 -2   1  0
# Вычисление Евклидовой нормы:
println("Евклидовой нормы: ", opnorm(d))
```

```
# Вычисление p-нормы:
p=1
println("p-нормы: ", opnorm(d,p))
Евклидовой нормы: 7.147682841795258
p-нормы: 8.0
# Поворот на 180 градусов:
println("Поворот на 180 градусов", rot180(d))
# Переворачивание строк:
println("Переворачивание строк: ", reverse(d,dims=1))
# Переворачивание столбцов
println("Переворачивание столбцов: ", reverse(d,dims=2))
Поворот на 180 градусов[0 1 -2; 3 2 -1; 2 -4 5]
Переворачивание строк: [-2 1 0; -1 2 3; 5 -4 2]
Переворачивание столбцов: [2 -4 5; 3 2 -1; 0 1 -2]
```

Матричное умножение, единичная матрица, скалярное произведение

```
# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand(1:10,(2,3))
# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand(1:10,(3,4))
# Произведение матриц A и B:
A*B
2×4 Matrix{Int64}:
 73  37  18   66
 87  60  31  117
# Единичная матрица 3x3:
Matrix{Int}(I, 3, 3)
3×3 Matrix{Int64}:
 1  0  0
 0  1  0
```

```
 0  0  1
# Скалярное произведение векторов X и Y:
X = [2, 4, -5]
Y = [1,-1,3]
dot(X,Y)
-17
# тоже скалярное произведение:
X'Y
-17
```

Факторизация. Специальные матричные структуры

```
# Задаём квадратную матрицу 3x3 со случайными значениями:
A = rand(3, 3)
# Задаём единичный вектор:
x = fill(1.0, 3)
# Задаём вектор b:
b = A*x
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
A\b
3-element Vector{Float64}:
 0.9999999999999998
 1.0
 1.0
# LU-факторизация:
Alu = lu(A)
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
 1.0       0.0        0.0
```

```
 0.615141  1.0        0.0
 0.321768  0.0235118  1.0
U factor:
3×3 Matrix{Float64}:
 0.909254  0.606996  0.188041
 0.0       0.57954   0.63612
 0.0       0.0       0.861268
# Матрица перестановок:
Alu.P
3×3 Matrix{Float64}:
 0.0  1.0  0.0
 0.0  0.0  1.0
 1.0  0.0  0.0
# Вектор перестановок:
Alu.p
3-element Vector{Int64}:
 2
 3
 1
# Матрица L:
Alu.L
3×3 Matrix{Float64}:
 1.0       0.0        0.0
 0.615141  1.0        0.0
 0.321768  0.0235118  1.0
# Матрица U:
Alu.U
3×3 Matrix{Float64}:
 0.909254  0.606996  0.188041
```

```
 0.0       0.57954   0.63612
 0.0       0.0       0.861268
# Решение СЛАУ через матрицу A:
A\b
3-element Vector{Float64}:
 0.9999999999999998
 1.0
 1.0
# Решение СЛАУ через объект факторизации:
Alu\b
3-element Vector{Float64}:
 0.9999999999999998
 1.0
 1.0
# Детерминант матрицы A:
det(A)
0.45384413687575187
# Детерминант матрицы A через объект факторизации:
det(Alu)
0.45384413687575187
# QR-факторизация:
Aqr = qr(A)
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor:
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.264319  -0.12994   -0.955642
 -0.821459  -0.488837   0.293673
 -0.505313   0.862644   0.0224688
R factor:
```

```
3×3 Matrix{Float64}:
 -1.10688  -1.03538   -0.781953
  0.0       0.498166   0.434889
  0.0       0.0       -0.823064
# Матрица Q:
Aqr.Q
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.264319  -0.12994   -0.955642
 -0.821459  -0.488837   0.293673
 -0.505313   0.862644   0.0224688
# Матрица R:
Aqr.R
3×3 Matrix{Float64}:
 -1.10688  -1.03538   -0.781953
  0.0       0.498166   0.434889
  0.0       0.0       -0.823064
# Проверка, что матрица Q - ортогональная:
Aqr.Q'*Aqr.Q
3×3 Matrix{Float64}:
 1.0  -1.66533e-16   0.0
 0.0   1.0          -4.44089e-16
 0.0  -3.33067e-16   1.0
# Симметризация матрицы A:
Asym = A + A'
3×3 Matrix{Float64}:
 0.585137  1.11819  1.49605
 1.11819   1.21399  1.14097
 1.49605   1.14097  1.50358
# Спектральное разложение симметризованной матрицы:
```

```
AsymEig = eigen(Asym)
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 -0.5696462945958927
  0.2280379804830429
  3.6443212274452446
vectors:
3×3 Matrix{Float64}:
  0.844286    0.127328   -0.520546
 -0.215406   -0.808803   -0.54721
 -0.490694    0.574131   -0.655434
# Собственные значения:
AsymEig.values
3-element Vector{Float64}:
 -0.5696462945958927
  0.2280379804830429
  3.6443212274452446
#Собственные векторы:
AsymEig.vectors
3×3 Matrix{Float64}:
  0.844286    0.127328   -0.520546
 -0.215406   -0.808803   -0.54721
 -0.490694    0.574131   -0.655434
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym
3×3 Matrix{Float64}:
  1.0           6.66134e-16   1.11022e-15
 -1.33227e-15  1.0          -2.22045e-15
```

```
  2.66454e-15  6.66134e-16   1.0
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
1000×1000 Matrix{Float64}:
  0.563565      1.04686   -0.17552    ...   0.732417   -0.352508    0.572473
  1.24608      -0.813521  -1.12819        -0.205106    0.486897    0.428047
  1.09677      -0.131553   0.0147621      -0.73089    -1.36957    -1.73431
  0.00612529   -0.061649   0.73033         1.33844     0.83394     1.00052
  0.327113     -0.490315   0.966332        0.681366   -0.57985     1.47336
 -0.447071      1.06384   -1.3739     ...   1.06415     0.226066   -1.77447
  0.843396     -0.116513  -0.603763        0.642849    1.08515     0.313099
 -0.000157182   0.311643   0.522261        1.57937     1.20215     0.478557
 -0.150693      0.21262   -0.257269       -0.531689    1.04099    -1.76959
  0.568932     -0.656127  -0.618355        0.230223   -0.318063    0.666008
  0.589237      1.30122    0.988794   ...  -0.747055    0.401636    0.805042
 -1.38645       0.170924  -0.911031       -0.122241   -2.61964    -0.655887
 -0.352182     -0.1377     0.731137       -0.481005    0.822418    0.611771
  ⋮                                   ⋱
  0.389215     -0.020974  -0.542638       -0.139909   -1.42999    -1.34067
 -1.13188       0.352305  -0.347941        1.06377    -0.277257   -0.532906
 -0.0315728    -0.250863  -0.229701   ...  -0.0856085   0.098191    0.238011
 -1.3502        1.00534   -0.335383        0.94875     0.277358   -0.0610749
  0.0134547     0.881627   0.155217        1.82891    -0.0572179  -1.84264
 -0.678266     -0.994697  -0.223364        0.194877    0.0110706  -0.104862
 -0.721806      1.58744   -2.11496        -0.154554   -0.103842    0.117867
  0.838983     -0.310588   0.47783    ...   1.52233     1.30001    -0.0295978
 -1.4324        1.18773   -0.099182        0.556455    0.833313   -0.926812
  0.142712     -1.35304    0.157123        0.269901   -1.89899     0.309658
```

```
  0.144455     -0.299778  -0.328117      0.478755    1.24456     0.120033
  0.454165     -0.09356    0.772868      0.296564   -0.445574   -0.409302
# Симметризация матрицы:
Asym = A + A'
1000×1000 Matrix{Float64}:
  1.12713      2.29294     0.92125     .  0.875129   -0.208053    1.02664
  2.29294     -1.62704    -1.25974       -1.55815     0.187119    0.334487
  0.92125     -1.25974     0.0295241     -0.573767   -1.69769    -0.961445
 -0.592356    -2.18621     1.105          1.15669     1.51627     0.611408
 -0.811693    -1.28434     0.554618       1.86542    -0.293424    2.83478
 -0.272431    -1.24724    -1.83987     .  0.413559   -1.19622    -2.10303
  1.38647     -0.473752   -2.18502        0.474592    1.17278     0.413386
  1.19554      0.494837    1.55956        2.86799     2.10379     1.09358
  0.326293    -0.536223    0.954788      -1.27036     1.89652    -0.920687
 -0.771797    -1.21269     0.858733       0.227956    0.306372    1.25796
 -0.244776     0.537374    2.49184     . -0.969896    1.73563     1.08437
 -1.2631       2.12378    -2.0678         0.565227   -2.83178    -1.7204
 -0.26491     -0.0472591   1.18405        0.0641268  -0.8119      1.21369
  ⋮                                    ⋱
 -1.10117      0.489811   -0.702793       0.425125   -1.596      -1.86898
 -0.360239     1.5409      1.09111        1.49925     1.48906     2.00088
  0.835484    -0.0468494   0.00423505  .  0.427497   -0.42276     0.977794
 -2.43407      0.669173   -0.593056       0.912447    1.12643     1.24709
  0.313434     0.512936    1.44202        2.11698     1.86649    -3.70162
  0.0915174   -1.45202     0.0508664      0.521804   -0.923898    0.563592
 -2.01179      2.12807    -2.36717       -1.25802    -1.42504     0.262337
  0.317828     1.96872     2.35824     .  2.11963     2.15978    -2.04816
 -0.722444     2.03463    -1.32701        1.12356     0.620509   -0.660104
  0.875129    -1.55815    -0.573767       0.539803   -1.42023     0.606222
```

```
 -0.208053    0.187119   -1.69769        -1.42023     2.48911   -0.325541
  1.02664     0.334487   -0.961445        0.606222   -0.325541  -0.818604
# Проверка, является ли матрица симметричной:
issymmetric(Asym)
true
# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)
false
# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
1000×1000 Symmetric{Float64, Matrix{Float64}}:
  1.12713     2.29294     0.92125     .   0.875129   -0.208053    1.02664
  2.29294    -1.62704    -1.25974        -1.55815     0.187119    0.334487
  0.92125    -1.25974     0.0295241      -0.573767   -1.69769    -0.961445
 -0.592356   -2.18621     1.105           1.15669     1.51627     0.611408
 -0.811693   -1.28434     0.554618        1.86542    -0.293424    2.83478
 -0.272431   -1.24724    -1.83987     .   0.413559   -1.19622    -2.10303
  1.38647    -0.473752   -2.18502         0.474592    1.17278     0.413386
  1.19554     0.494837    1.55956         2.86799     2.10379     1.09358
  0.326293   -0.536223    0.954788       -1.27036     1.89652    -0.920687
 -0.771797   -1.21269     0.858733        0.227956    0.306372    1.25796
 -0.244776    0.537374    2.49184     .  -0.969896    1.73563     1.08437
 -1.2631      2.12378    -2.0678          0.565227   -2.83178    -1.7204
 -0.26491    -0.0472591   1.18405         0.0641268  -0.8119      1.21369
  .                                  .
 -1.10117     0.489811   -0.702793        0.425125   -1.596      -1.86898
```

```
 -0.360239    1.5409      1.09111          1.49925     1.48906     2.00088
  0.835484   -0.0468494   0.00423505  .    0.427497   -0.42276     0.977794
 -2.43407     0.669173   -0.593056         0.912447    1.12643     1.24709
  0.313434    0.512936    1.44202          2.11698     1.86649    -3.70162
  0.0915174  -1.45202     0.0508664        0.521804   -0.923898    0.563592
 -2.01179     2.12807    -2.36717         -1.25802    -1.42504     0.262337
  0.317828    1.96872     2.35824     .    2.11963     2.15978    -2.04816
 -0.722444    2.03463    -1.32701          1.12356     0.620509   -0.660104
  0.875129   -1.55815    -0.573767         0.539803   -1.42023     0.606222
 -0.208053    0.187119   -1.69769         -1.42023     2.48911    -0.325541
  1.02664     0.334487   -0.961445         0.606222   -0.325541   -0.818604
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);
  170.188 ms (11 allocations: 7.99 MiB)
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы:
@btime eigvals(Asym_noisy);
  732.106 ms (14 allocations: 7.93 MiB)
# Оценка эффективности выполнения операции по нахождению
# собственных значений зашумлённой матрицы,
# для которой явно указано, что она симметричная:
@btime eigvals(Asym_explicit);
  174.454 ms (11 allocations: 7.99 MiB)
# Трёхдиагональная матрица 1000000 x 1000000:
n = 1000000;
```

```
A = SymTridiagonal(randn(n), randn(n-1))
1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
  2.0704    -0.429507    .        .    .           .           .
 -0.429507   1.03656   -0.552678       .           .           .
   .        -0.552678  -0.84854        .           .           .
   .          .        -1.84039        .           .           .
   .          .          .             .           .           .
   .          .          .        .    .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .        .    .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
  .                               .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .        .    .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .             .           .           .
   .          .          .        .    .           .           .
   .          .          .           -0.592926     .           .
   .          .          .           -1.47854     0.869354     .
   .          .          .            0.869354  -0.0454058   1.52004
   .          .          .             .          1.52004     0.18073
# Оценка эффективности выполнения операции по нахождению
```

```
# собственных значений:
@btime eigmax(A)
  530.131 ms (17 allocations: 183.11 MiB)
6.4179096859748075
```

Общая линейная алгебра

```
1//2
1//2
# Матрица с рациональными элементами:
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
3×3 Matrix{Rational{BigInt}}:
 3//5   9//10  2//5
 2//5   4//5   1//5
 9//10  4//5   4//5
# Единичный вектор:
x = fill(1, 3)
# Задаём вектор b:
b = Arational*x
3-element Vector{Rational{BigInt}}:
 19//10
  7//5
  5//2
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
Arational\b
3-element Vector{Rational{BigInt}}:
 1//1
 1//1
 1//1
# LU-разложение:
```

```
lu(Arational)
LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3×3 Matrix{Rational{BigInt}}:
 1//1   0//1   0//1
 4//9   1//1   0//1
 2//3  33//40  1//1
U factor:
3×3 Matrix{Rational{BigInt}}:
 9//10  4//5   4//5
 0//1   4//9  -7//45
 0//1   0//1  -1//200
```

Задания для самостоятельного выполнения

Произведение векторов

```
# task1
v = [4, 2, -10, 3]
dot_v = dot(v, v)
129
outer_v = v * v'
4×4 Matrix{Int64}:
  16    8  -40   12
   8    4  -20    6
 -40  -20  100  -30
  12    6  -30    9
```

Системы линейных уравнений

```
# Task1a
A_1 = [1 1; 1 -1]
B_1 = [2; 3]
println(A_1\B_1)
```

```
[2.5, -0.5]
# Task1b
A_2 = [1 2; 1 2]
B_2 = [2; 4]
println(A_2\B_2)
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.

   9\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.

   9\LinearAlgebra\src\factorization.jl:22 [inlined]
 [3] #lu!#170
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.

   9\LinearAlgebra\src\lu.jl:82 [inlined]
 [4] lu!
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.

   9\LinearAlgebra\src\lu.jl:80 [inlined]
 [5] lu(A::Matrix{Int64}, pivot::RowMaximum; check::Bool)
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.

   3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:299
 [6] lu (repeats 2 times)
```

```
  @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
  9\LinearAlgebra\src\lu.jl:298 [inlined]
 [7] \(A::Matrix{Int64}, B::Vector{Int64})
  @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.
  3\share\julia\stdlib\v1.9\LinearAlgebra\src\generic.jl:1115
 [8] top-level scope
  @ In[59]:4
# Task1c
A_3 = [1 2; 1 2]
B_3 = [2; 5]
println(A_3\B_3)
SingularException(2)

Stacktrace:
 [1] checknonsingular
  @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
  9\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular
  @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
  9\LinearAlgebra\src\factorization.jl:22 [inlined]
 [3] #lu!#170
  @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
  9\LinearAlgebra\src\lu.jl:82 [inlined]
 [4] lu!
  @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
  9\LinearAlgebra\src\lu.jl:80 [inlined]
 [5] lu(A::Matrix{Int64}, pivot::RowMaximum; check::Bool)
```

```
  @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.
  3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:299
A_4 = [1 1; 2 2; 3 3]
B_4 = [1; 2; 3]
println("Ответ 1d: ", A_4\B_4)
Ответ 1d: [0.4999999999999999, 0.5]
# task1e
A_5 = [1 1; 2 1; 1 -1]
B_5 = [2; 1; 3]
println("Ответ 1e: ", A_5\B_5)
Ответ 1e: [1.5000000000000004, -0.9999999999999997]
# task1f
A_6 = [1 1; 2 1; 3 2]
B_6 = [2; 1; 3]
println("Ответ 1f: ", A_6\B_6)
Ответ 1f: [-0.9999999999999989, 2.9999999999999982]
# Task2a
A_7 = [1 1 1; 1 -1 -2]
B_7 = [2; 3]
println("Ответ 2a: ", A_7\B_7)
Ответ 2a: [2.2142857142857144, 0.35714285714285704, -0.5714285714285712]
# Task2b
A_8 = [1 1 1; 2 2 -3; 3 1 1]
B_8 = [2; 4; 1]
println("Ответ 2b: ", A_8\B_8)
Ответ 2b: [-0.5, 2.5, 0.0]
# Task2c
A_9 = [1 1 1; 1 1 2; 2 2 3]
B_9 = [1; 0; 1]
```

```
println("Ответ 2c: ", A_9\B_9)
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\factorization.jl:22 [inlined]
 [3] #lu!#170
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\lu.jl:82 [inlined]
 [4] lu!
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\lu.jl:80 [inlined]
# Task2d
A_10 = [1 1 1; 1 1 2; 2 2 3]
B_10 = [1; 0; 0]
println("Ответ 2d: ", A_10\B_10)
SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\factorization.jl:19 [inlined]
 [2] checknonsingular
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\factorization.jl:22 [inlined]
```

```
 [3] #lu!#170
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\lu.jl:82 [inlined]
 [4] lu!
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.9\
 [5] lu(A::Matrix{Int64}, pivot::RowMaximum; check::Bool)
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.
   3\share\julia\stdlib\v1.9\LinearAlgebra\src\lu.jl:299
 [6] lu (repeats 2 times)
   @ c:\users\reachna\appdata\local\programs\julia-1.9.3\share\julia\stdlib\v1.
   9\LinearAlgebra\src\lu.jl:298 [inlined]
 [7] \(A::Matrix{Int64}, B::Vector{Int64})
   @ LinearAlgebra c:\users\reachna\appdata\local\programs\julia-1.9.
   3\share\julia\stdlib\v1.9\LinearAlgebra\src\generic.jl:1115
 [8] top-level scope
   @ In[67]:4
```

Операции с матрицами

```
function diagonal_matrix(matrix)
    Asym = matrix + matrix'
    AsymEig = eigen(Asym)
    return inv(AsymEig.vectors) * matrix * AsymEig.vectors
end
```

```
diagonal_matrix (generic function with 1 method)
```

```
# task1a
a = [1 -2; -2 1]
diagonal_matrix(a)
```

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

```
# task1b
b = [1 -2; -2 3]
diagonal_matrix(b)
2×2 Matrix{Float64}:
 -0.236068     3.46945e-16
  4.44089e-16  4.23607
# task1c
c = [1 -2 0; -2 1 2; 0 2 0]
diagonal_matrix(c)
3×3 Matrix{Float64}:
 -2.14134       3.55271e-15  -1.9984e-15
  3.38618e-15   0.515138      1.11022e-16
 -6.66134e-16  -4.44089e-16   3.6262
# Task2a
a = [1 -2; -2 1]
a^10
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
# Task2b
b = [5 -2; -2 5]
sqrt(b)
2×2 Matrix{Float64}:
  2.1889   -0.45685
 -0.45685   2.1889
# Task2c
c = [1 -2; -2 1]
c^(1/3)
2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
```

```
  0.971125+0.433013im  -0.471125+0.433013im
 -0.471125+0.433013im   0.971125+0.433013im
# Task2d
d = [1 2; 2 3]
sqrt(d)
2×2 Matrix{ComplexF64}:
 0.568864+0.351578im  0.920442-0.217287im
 0.920442-0.217287im   1.48931+0.134291im
# Task3
using BenchmarkTools
A = [140 97 74 168 131; 97 106 89 131 36; 74 89 152 144 7; 168 131 144 54 142;
131 36 71 142 36]

eigen_value = eigvals(A)
5-element Vector{Float64}:
 -133.26347908274784
  -46.31722914150222
   34.7455206405183
  100.90643294309879
  531.9287546406343
@btime eigvals(A);
  4.157 μs (12 allocations: 2.38 KiB)
B = zeros(5, 5)
@btime for i in 1:5
    B[i, i] = eigen_value[i]
end
B
  227.292 ns (5 allocations: 80 bytes)
5×5 Matrix{Float64}:
```

```
 -133.263    0.0      0.0       0.0      0.0
    0.0    -46.3172   0.0       0.0      0.0
    0.0      0.0     34.7455    0.0      0.0
    0.0      0.0      0.0     100.906    0.0
    0.0      0.0      0.0       0.0    531.929
Alu = lu(A)
@btime Alu.L
  110.108 ns (1 allocation: 256 bytes)
5×5 Matrix{Float64}:
 1.0        0.0        0.0        0.0        0.0
 0.779762   1.0        0.0        0.0        0.0
 0.440476  -0.47314    1.0        0.0        0.0
 0.833333   0.183929  -0.556312   1.0        0.0
 0.577381  -0.459012  -0.189658   0.897068   1.0
```

Линейные модели экономики

```
function productive_matrix(matrix, size)
    answer = ""
     # единичная матрица
    E = [1 0; 0 1]
    # зададим любые неотрицательные числа
    Y = rand(0:1000, size)
    # По формуле вычислим x - A*x = y
    S = E - matrix
    # найдем значения x
    X = S\Y
    # теперь проверим есть ли среди x отрциательное число
    for i in 1:1:size
        if X[i] < 0
            answer = "Матрица непродуктивная"
```

```
            break
        else
            answer = "Матрица продуктивная"
        end
    end
    return answer
end
productive_matrix (generic function with 1 method)
# task4.1.a
a = [1 2; 3 4]
productive_matrix(a, 2)
"Матрица непродуктивная"
# task 4.1.b
b = [1 2; 3 4] * (1/2)
productive_matrix(b, 2)
"Матрица непродуктивная"
# Task 4.1.c
c = [1 2; 3 4] * (1/10)
productive_matrix(c, 2)
"Матрица продуктивная"
# Task2
function productive_matrix_2(matrix, size)
    # единичная матрица
    ans = ""
    E = [1 0; 0 1]
    matrix_new = E - matrix
    inv_matrix_new = inv(matrix_new)
    for i in 1:1:size
        for j in 1:1:size
```

```
            if inv_matrix_new[i, j] < 0
                ans = "Матрица непродуктивная"
                break
            else
                ans = "Матрица продуктивная"
            end
        end
    end
    return ans
end
productive_matrix_2 (generic function with 1 method)
# task4.2.a
a = [1 2; 3 1]
productive_matrix_2(a, 2)
"Матрица непродуктивная"
# task 4.2.b
b = [1 2; 3 1] * (1/2)
productive_matrix_2(b, 2)
"Матрица непродуктивная"
# task 4.2.c
c = [1 2; 3 1] * (1/10)
productive_matrix_2(c, 2)
"Матрица продуктивная"
# Task3
function productive_matrix_3(matrix, size)
    ans=""
    # найдем собственные значения переданной матрицы
    eigenvalues = eigvals(matrix)
    for i in 1:1:size
```

```
        if abs(eigenvalues[i]) > 1
            ans = "Матрица непродуктивная"
            break
        else
            ans = "Матрица продуктивная"
        end
    end
    return ans
end
productive_matrix_3 (generic function with 1 method)
# task4.3.a
a = [1 2; 3 1]
productive_matrix_3(a, 2)
"Матрица непродуктивная"
# task4.3.b
b = [1 2; 3 1] * (1/2)
productive_matrix_3(b, 2)
"Матрица непродуктивная"
# task4.3.c
c = [1 2; 3 1] * (1/10)
productive_matrix_3(c, 2)
"Матрица продуктивная"
# task 4.3.d
d = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
productive_matrix_3(d, 3)
"Матрица продуктивная"
```

4 Вывод

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