Компьютерный практикум по статистическому анализу данных
презентация к лабораторной работе №4

Линейная алгебра

Ким И. В. НФИбд-01-21

Российский университет дружбы народов, Москва, Россия

Цель работы

Цель работы

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

Задание

Задание

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

Выполнение работы. Повтор примеров из раздела 4.2

4.2.1. Поэлементные операции над многомерными массивами

Для матрицы 4 × 3 рассмотрим поэлементные операции сложения и произведения её элементов:

[1]:
```
# Массив 4х3 со случайными целыми числами (от 1 до 20):
a= rand(1:20,(4,3))
```

[1]:
```
4×3 Matrix{Int64}:
 11   2  18
  1   9  11
 19  20  11
 13  16  13
```

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

[2]:
```
144
```

4.2.1. Поэлементные операции над многомерными массивами

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

[3]: 1×3 Matrix{Int64}:
      44  47  53

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

[4]: 4×1 Matrix{Int64}:
      31
      21
      50
      42
```

4.2.1. Поэлементные операции над многомерными массивами

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

[5]: 443111834880

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

[6]: 1×3 Matrix{Int64}:
      2717  5760  28314

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

[7]: 4×1 Matrix{Int64}:
       396
        99
      4180
      2704
```

4.2.1. Поэлементные операции над многомерными массивами

```
[8]: # Подключение пакета Statistics:
     import Pkg

[9]: Pkg.add("Statistics")
     using Statistics

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

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

[10]: 12.0
```

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

[11]: 1×3 Matrix{Float64}:
       11.0  11.75  13.25

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

[12]: 4×1 Matrix{Float64}:
       10.333333333333334
        7.0
       16.666666666666668
       14.0
```

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

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

13]:
```
# Подключение пакета LinearAlgebra:
import Pkg
```

14]:
```
Pkg.add("LinearAlgebra")
using LinearAlgebra
```

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

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

[15]: 4×4 Matrix{Int64}:
        7  14  11  11
        2  14  11  12
       14  12  14  20
        2  16  15   4

[17]: # Транспонирование:
      transpose(b)

[17]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
        7   2  14   2
       14  14  12  16
       11  11  14  15
       11  12  20   4
```

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

[18]: 39

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

[19]: 4-element Vector{Int64}:
        7
       14
       14
        4

[20]: # Ранг матрицы:
      rank(b)

[20]: 4
```

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

[21]: 4×4 Matrix{Float64}:
        0.183218   -0.17475     0.00654349  -0.0123172
        0.292533   -0.0227098  -0.13241     -0.0742879
       -0.314088    0.0138568   0.13164      0.163972
       -0.0839107   0.126251    0.0327175   -0.0615858

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

[22]: -5195.999999999999

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

[23]: 3×4 Matrix{Float64}:
        0.0338864  -0.0894004   0.0429001  -0.00757323
       -0.0565843   0.0507362   0.0139091   0.0236476
        0.0407696   0.0475014  -0.0299499   0.00562164
```

4.2.3. Вычисление нормы векторов и матриц, повороты, вращения

4.2.3. Вычисление нормы векторов и матриц, повороты, вращения

Для вычисления нормы используется `LinearAlgebra.norm(x)`.
Евклидова норма:

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

p-норма:

$$\|\vec{A}\|_p = \left(\sum_{i=1}^{n} |a_i|^p \right)^{1/p}.$$

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

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

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

[25]: 6.708203932499369

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

[26]: 11.0
```

Евклидово расстояние между двумя векторами $\vec{X}$ и $\vec{Y}$ определяется как $||\vec{X}-\vec{Y}||_2$.

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

[27]: 9.486832980505138
```

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

[28]: 9.486832980505138

Угол между двумя векторами $\vec{X}$ и $\vec{Y}$ определяется как $\cos^{-1}\dfrac{\vec{X}^T\vec{Y}}{||\vec{X}||_2||\vec{Y}||_2}$.

[30]:
```
# Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))
```

[30]: 2.4404307889469252

Вычисление нормы для двумерной матрицы:

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

[31]: 3x3 Matrix{Int64}:
        5  -4  2
       -1   2  3
       -2   1  0

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

[32]: 7.147682841795258

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

[33]: 8.0
```

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

[34]: 3x3 Matrix{Int64}:
       0   1  -2
       3   2  -1
       2  -4   5

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

[35]: 3x3 Matrix{Int64}:
       -2   1  0
       -1   2  3
        5  -4  2

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

[36]: 3x3 Matrix{Int64}:
       2  -4   5
       3   2  -1
       0   1  -2
```

4.2.4. Матричное умножение, единичная матрица, скалярное произведение

4.2.4. Матричное умножение, единичная матрица, скалярное произведение ¶

```
[37]: # Матрица 2х3 со случайными целыми значениями от 1 до 10:
      A = rand(1:10,(2,3))

[37]: 2×3 Matrix{Int64}:
       10  5  10
        5  5   1

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

[38]: 3×4 Matrix{Int64}:
       8  8  2   8
       2  8  2   8
       9  8  6  10
```

4.2.4. Матричное умножение, единичная матрица, скалярное произведение

```
[39]: # Произведение матриц A и B:
      A*B

[39]: 2x4 Matrix{Int64}:
       180  200  90  220
        59   88  26   90

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

[40]: 3x3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1

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

[41]: -17
```

```
[42]: # тоже скалярное произведение:
      X'Y

[42]: -17
```

4.2.5. Факторизация. Специальные матричные структуры

Решение систем линейный алгебраических уравнений $Ax = b$:

[43]:
```
# Задаём квадратную матрицу 3x3 со случайными значениями:
A = rand(3, 3)
```

[43]:
```
3×3 Matrix{Float64}:
 0.267987  0.224647  0.369579
 0.937245  0.983916  0.0429566
 0.181963  0.916723  0.599615
```

[44]:
```
# Задаём единичный вектор:
x = fill(1.0, 3)
```

[44]:
```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

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

[45]: 3-element Vector{Float64}:
       0.8622121186634498
       1.9641175235461295
       1.6983007392856093

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

[46]: 3-element Vector{Float64}:
       1.0000000000000002
       1.0
       1.0
```

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

[47]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3×3 Matrix{Float64}:
       1.0        0.0         0.0
       0.194147   1.0         0.0
       0.28593   -0.0781108   1.0
      U factor:
      3×3 Matrix{Float64}:
       0.937245  0.983916  0.0429566
       0.0       0.725699  0.591275
       0.0       0.0       0.403481

[48]: # Матрица перестановок:
      Alu.P

[48]: 3×3 Matrix{Float64}:
       0.0  1.0  0.0
       0.0  0.0  1.0
       1.0  0.0  0.0
```

```
[50]: # Матрица L:
      Alu.L

[50]: 3x3 Matrix{Float64}:
       1.0        0.0        0.0
       0.194147   1.0        0.0
       0.28593   -0.0781108  1.0

[51]: # Матрица U:
      Alu.U

[51]: 3x3 Matrix{Float64}:
       0.937245  0.983916  0.0429566
       0.0       0.725699  0.591275
       0.0       0.0       0.403481
```

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

[52]:
```
# Решение СЛАУ через матрицу A:
A\b
```

[52]:
```
3-element Vector{Float64}:
 1.0000000000000002
 1.0
 1.0
```

[53]:
```
# Решение СЛАУ через объект факторизации:
Alu\b
```

[53]:
```
3-element Vector{Float64}:
 1.0000000000000002
 1.0
 1.0
```

- Аналогично можно найти детерминант матрицы:

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

[54]: 0.2744307662052006

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

[55]: 0.2744307662052006
```

4.2.5. Факторизация. Специальные матричные структуры

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

[57]:
```julia
# QR-факторизация:
Aqr = qr(A)
```

[57]:
```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
R factor:
3×3 Matrix{Float64}:
 -0.991643  -1.15887   -0.250504
  0.0        0.718307   0.535523
  0.0        0.0       -0.385272
```

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

[58]:
```
# Матрица Q:
Aqr.Q
```

[58]:
```
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
```

[59]:
```
# Матрица R:
Aqr.R
```

[59]:
```
3×3 Matrix{Float64}:
 -0.991643  -1.15887   -0.250504
  0.0        0.718307   0.535523
  0.0        0.0       -0.385272
```

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

[60]: 3x3 Matrix{Float64}:
       1.0          -4.16334e-17  2.22045e-16
       0.0           1.0          0.0
       2.498e-16     0.0          1.0
```

- Примеры собственной декомпозиции матрицы A:

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

[61]: 3x3 Matrix{Float64}:
       0.535973  1.16189   0.551542
       1.16189   1.96783   0.959679
       0.551542  0.959679  1.19923

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

[62]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
      values:
      3-element Vector{Float64}:
       -0.11306455463587163
        0.593888560230713
        3.222211176987134
      vectors:
      3x3 Matrix{Float64}:
        0.876092   -0.219201  -0.429434
       -0.481884   -0.427297  -0.764987
       -0.0158098   0.877136  -0.479981
```

```
[63]: # Собственные значения:
      AsymEig.values

[63]: 3-element Vector{Float64}:
       -0.11306455463587163
        0.593888560230713
        3.2221117698713 4

[64]: #Собственные векторы:
      AsymEig.vectors

[64]: 3×3 Matrix{Float64}:
        0.876092   -0.219201  -0.429434
       -0.481884   -0.427297  -0.764987
       -0.0158098   0.877136  -0.479981

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

[65]: 3×3 Matrix{Float64}:
        1.0          6.66134e-16  4.16334e-16
       -4.996e-16    1.0          2.22045e-16
        6.66134e-16  1.11022e-15  1.0
```

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

[66]:
```
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
```

[66]:
```
1000×1000 Matrix{Float64}:
  0.541853   1.17683    -0.102626    0.711308   …  -0.100048     1.40661
 -1.63016    0.192139   -0.640807    1.27749        0.341348    -0.334139
  0.868226   0.337315    2.07433     0.0740266      0.637573    -0.913898
  0.714988   0.725385   -0.131877    0.19432        0.00760246  -0.426979
 -0.181526  -1.25989     0.194939    0.699478      -0.596407    -0.536494
  0.669015   0.099575   -1.05191     0.0690244  …  -0.132029     0.0727038
  1.05694    0.0697213  -0.251643    0.612429       1.18582      0.456311
  0.560079  -0.198567    0.969559   -0.0749454     -1.98284     -0.310298
 -0.802105  -0.082776    1.07067    -0.43749        2.14436      0.395738
  1.97598   -1.20432    -0.344461    0.784787      -0.0365541   -1.1726
 -1.50265   -1.05075     0.0854441  -1.00694    …  -0.0247736   -0.0620557
  1.36112   -1.71758    -0.0819758   0.844557      -0.65691     -0.888152
  0.972742   0.696585   -0.196527    0.13378        0.205359     0.208161
  ⋮                                             ⋱
 -1.02966    0.507089    0.141884   -1.8208         0.262267     0.0554675
```

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

[67]: 1000×1000 Matrix{Float64}:
        1.08371   -0.453339    0.7656     …   1.67338    2.20554    0.307862
       -0.453339   0.384278   -0.303492       0.766215  -1.33895   -0.958399
        0.7656    -0.303492    4.14866       -1.84645   -0.298946   1.50845
        1.4263     2.00287    -0.0578506      0.969793  -0.387153  -0.635564
        0.850148   0.00813477 -0.0746053      2.81317   -0.0444381  0.121552
       -0.436239   0.409179   -1.33971    …   0.625248  -0.472241   0.302211
        0.202025  -3.16299    -0.479252      -1.15971    1.84215    0.400232
        0.436958   0.00660826  1.04619       -1.05527   -4.38537   -2.03937
       -0.387422   1.32338     1.95016       -1.16502    1.09687   -0.00387839
        2.26474   -1.7631     -2.57739        0.780182   0.952357  -1.93394
       -1.66728   -2.1635      1.27112    …   1.2145    -0.650064   0.222182
        0.570905  -2.4779     -2.43396        1.31673    0.0640703 -1.02218
        1.63754    2.0534     -0.859851      -0.694551   0.351228  -0.888802
        ⋮                                 ⋱
        1.15158    0.250859   -1.744         -0.396391  -0.0547553  0.0219166
```

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

[68]: true
```

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

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

[69]: -0.4533389657531679

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

[70]: false
```

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

[71]:
```julia
# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
```

[71]:
```
1000×1000 Symmetric{Float64, Matrix{Float64}}:
  1.08371   -0.453339    0.7656     …   1.67338    2.20554     0.307862
 -0.453339   0.384278   -0.303492       0.766215  -1.33895    -0.958399
  0.7656    -0.303492    4.14866       -1.84645   -0.298946    1.50845
  1.4263     2.00287    -0.0578506      0.969793  -0.387153   -0.635564
  0.850148   0.00813477 -0.0746053      2.81317   -0.0444381   0.121552
 -0.436239   0.409179   -1.33971    …   0.625248  -0.472241    0.302211
  0.202025  -3.16299    -0.479252      -1.15971    1.84215     0.400232
  0.436958   0.00660826  1.04619       -1.05527   -4.38537    -2.03937
 -0.387422   1.32338     1.95016       -1.16502    1.09687    -0.00387839
  2.26474   -1.7631     -2.57739        0.780182   0.952357   -1.93394
 -1.66728   -2.1635      1.27112    …   1.2145    -0.650064    0.222182
  0.570905  -2.4779     -2.43396        1.31673    0.0640703  -1.02218
  1.63754    2.0534     -0.859851      -0.694551   0.351228   -0.888802
  ⋮                                 ⋱
  1.15158    0.250859   -1.744         -0.396391  -0.0547553   0.0219166
```

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

[72]:
```
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
```

```
   Resolving package versions...
   Installed BenchmarkTools ─ v1.5.0
    Updating `C:\Users\ksudz\.julia\environments\v1.10\Project.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
    Updating `C:\Users\ksudz\.julia\environments\v1.10\Manifest.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
  [9abbd945] + Profile
Precompiling project...
  ✓ BenchmarkTools
  1 dependency successfully precompiled in 3 seconds. 27 already precompiled.
```

[73]:
```
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@btime eigvals(Asym);
```

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

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

        653.783 ms (14 allocations: 7.93 MiB)

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

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

- Далее рассмотрим примеры работы с разряженными матрицами большой размерности. Использование типов Tridiagonal и SymTridiagonal для хранения трёхдиагональных матриц позволяет работать с потенциально очень большими трёхдиагональными матрицами:

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

```
[76]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
        0.299772  -1.63496    ⋅         …   ⋅         ⋅         ⋅
       -1.63496   -0.531184   0.611759      ⋅         ⋅         ⋅
         ⋅         0.611759   1.60504       ⋅         ⋅         ⋅
         ⋅          ⋅        -0.747954      ⋅         ⋅         ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅         ⋅
         ⋅          ⋅          ⋅        …   ⋅         ⋅         ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅         ⋅
         ⋅          ⋅          ⋅            ⋅         ⋅         ⋅
```

```
 .        .        .             .             .           .
 .        .        .        …    .             .           .
 .        .        .           1.07941         .           .
 .        .        .           0.170096  -0.52283          .
 .        .        .          -0.52283    0.587034  -0.376894
 .        .        .             .       -0.376894   2.02015
```

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

        371.107 ms (17 allocations: 183.11 MiB)

[77]: 6.076883425838304
```

4.2.6. Общая линейная алгебра

[78]:
```julia
# Матрица с рациональными элементами:
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
```

[78]:
```
3x3 Matrix{Rational{BigInt}}:
 4//5    1//2    1
 7//10   1       3//10
 7//10   7//10   9//10
```

[79]:
```julia
# Единичный вектор:
x = fill(1, 3)
```

[79]:
```
3-element Vector{Int64}:
 1
 1
 1
```

```
[80]: # Задаём вектор b:
      b = Arational*x

[80]: 3-element Vector{Rational{BigInt}}:
       23//10
         2
       23//10

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

[81]: 3-element Vector{Rational{BigInt}}:
       1
       1
       1
```

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

[82]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
      L factor:
      3x3 Matrix{Rational{BigInt}}:
        1     0     0
       7//8   1     0
       7//8  7//15  1
      U factor:
      3x3 Matrix{Rational{BigInt}}:
       4//5  1//2      1
        0    9//16  -23//40
        0     0      22//75
```

4.4 Задания для самостоятельного выполнения

4.4.1. 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.

```
[87]: v = rand(1:10, 3,3)
      dot_v = v'v

[87]: 3×3 Matrix{Int64}:
       66  35   91
       35  21   64
       91  64  262

[88]: outer_v = v*v'

[88]: 3×3 Matrix{Int64}:
       86   93   97
       93  102  102
       97  102  161
```

4.4.2. Системы линейных уравнений

1. Решить СЛАУ с двумя неизвестными.

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

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

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

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

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

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

[114]:
```
## a)
a = [1 1; 1 -1]
b = [2; 3]
a\b
```

[114]:
```
2-element Vector{Float64}:
  2.5
 -0.5
```

```
•[116]: ## b)
        a = [1 1; 2 2]
        b = [2; 4]
        a\b
        ## бесконечное число решений, вся система линейно зависима

 [116]: 2-element Vector{Int64}:
         2
         4

•[118]: ## c)
        a = [1 1; 2 2]
        b = [2; 5]
        a\b
        ## Нет решений, т.к матрица коэффициентов линейно зависима, а векторы нет

 [118]: 2-element Vector{Int64}:
         2
         5
```

```
•[120]: ## d)
        a = [2 2; 3 3]
        b = [2; 3]
        a\b
        ## бесконечное число решений

 [120]: 2-element Vector{Int64}:
         2
         3

 [121]: ## e)
        a = [1 1; 2 1; 1 -1]
        b = [2; 1; 3]
        a\b

 [121]: 2-element Vector{Float64}:
          1.5000000000000002
         -0.9999999999999999

 [122]: ## f)
        a = [1 1; 2 1; 3 2]
        b = [2; 1; 3]
        a\b

 [122]: 2-element Vector{Float64}:
         -0.9999999999999989
          2.9999999999999982
```

4.4.2. Системы линейных уравнений

2. Решить СЛАУ с тремя неизвестными.

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

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

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

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

[123]:

```
## a)
a = [1 1 1; 1 -1 -2]
b = [2; 3]
a\b
```

[123]:

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

4.4.2. Системы линейных уравнений

```
[124]: ## b)
       a = [1 1 1; 2 2 -3; 3 1 1]
       b = [2; 4; 1]
       a\b

[124]: 3-element Vector{Float64}:
        -0.5
         2.5
         0.0

[126]: ## c)
       a = [1 1 1; 1 1 2; 2 2 3]
       b = [1; 0; 1]
       a\b
       ## Бесконечное число решени

[126]: 3-element Vector{Int64}:
        1
        0
        1

[128]: ## d)
       a = [1 1 1; 1 1 2; 2 2 3]
       b = [1; 0; 0]
       a\b
       ## Нет решений

[128]: 3-element Vector{Int64}:
        1
        0
        0
```

4.4.3. Операции с матрицами ¶

1. Приведите приведённые ниже матрицы к диагональному виду

a) $\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$ b) $\begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}$ c) $\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix}$

```
[32]: ## a)
a = [1 -2; -2 1]
Matrix(Diagonal(eigen(a).values))
```

```
[32]: 2x2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
```

```
[33]: ## b)
a = [1 -2; -2 3]
Matrix(Diagonal(eigen(a).values))
```

```
[33]: 2x2 Matrix{Float64}:
 -0.236068  0.0
  0.0       4.23607
```

```
[134]: ## c)
       a = [1 -2 0; -2 1 2; 0 2 0]
       Matrix(Diagonal(eigen(a).values))
```

```
[134]: 3x3 Matrix{Float64}:
        -2.14134  0.0       0.0
         0.0      0.515138  0.0
         0.0      0.0       3.6262
```

2. Вычислите

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

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

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

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

[136]:
```
## a)
a = [1 -2; -2 3]
a^10
```

[136]:
```
2×2 Matrix{Int64}:
  514229   -832040
 -832040   1346269
```

[138]:
```
## b)
b = [5 -2; -2 5]
b^(1/2)
```

[138]:
```
2×2 Symmetric{Float64, Matrix{Float64}}:
  2.1889   -0.45685
 -0.45685   2.1889
```

```
[139]: ## c)
       c = [1 -2; -2 1]
       c^(1/3)

[139]: 2x2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
         0.971125+0.433013im  -0.471125+0.433013im
        -0.471125+0.433013im   0.971125+0.433013im

[140]: ## d)
       d = [1 2; 2 3]
       d^(1/2)

[140]: 2x2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
        0.568864+0.351578im  0.920442-0.217287im
        0.920442-0.217287im   1.48931+0.134291im
```

4.4.3. Операции с матрицами

3. Найдите собственные значения матрицы A, если

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

- Создайте диагональную матрицу из собственных значений матрицы A. Создайте нижнедиагональную матрицу из матрица A. Оцените эффективность выполняемых операций.

```
[141]: 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]
```

```
[141]: 5×5 Matrix{Int64}:
 140   97   74  168  131
  97  106   89  131   36
  74   89  152  144   71
 168  131  144   54  142
 131   36   71  142   36
```

```
[149]: @btime Matrix(Diagonal(eigvals(A)))

  1.470 μs (12 allocations: 2.86 KiB)

[149]: 5×5 Matrix{Float64}:
 -128.493    0.0       0.0      0.0       0.0
    0.0    -55.8878    0.0      0.0       0.0
    0.0      0.0      42.7522   0.0       0.0
    0.0      0.0       0.0     87.1611    0.0
    0.0      0.0       0.0      0.0     542.468

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

  246.734 ns (4 allocations: 640 bytes)

[151]: 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
```

4.4.4. Линейные модели экономики

4.4.4. Линейные модели экономики

Линейная модель экономики может быть записана как СЛАУ

$$x - Ax = y,$$

где элементы матрицы A и столбца y — неотрицательные числа. По своему смыслу в экономике элементы матрицы A и столбцов x, y не могут быть отрицательными числами.

1. Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части y имеет только неотрицательные элементы x_i. Используя это определение, проверьте, являются ли матрицы продуктивными.
   a) $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
   b) $\frac{1}{2}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
   c) $\frac{1}{10}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

]:
```julia
## a)
a = [1 2; 3 4]
y = rand(0:100000,2)
e = Matrix{Int}(I,2,2)
y\(e-a)
## значение отрицательные -> матрица не продуктивна
```

]:
```
1×2 transpose(::Vector{Float64}) with eltype Float64:
 -2.48707e-5  -3.73589e-5
```

```
•[155]: ## b)
        a = [1/2 1; 3/2 2]
        y = rand(0:100000,2)
        e = Matrix{Int}(I,2,2)
        y\(e-a)
        ## значение отрицательные -> матрица не продуктивна

 [155]: 1×2 transpose(::Vector{Float64}) with eltype Float64:
         -3.42817e-6  -1.91936e-5

 [156]: ## c)
        a = [1/10 2/10; 3/10 4/10]
        y = rand(0:100000,2)
        e = Matrix{Int}(I,2,2)
        y\(e-a)
        ## значения положительные -> матрица продуктивна

 [156]: 1×2 transpose(::Vector{Float64}) with eltype Float64:
         3.14849e-6  2.53702e-6
```

4.4.4. 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.

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

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

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

```
]: ## a)
a = [1 2; 3 1]
e = Matrix{Int}(I,2,2)
inv(e-a)
## все элементы матрицы отрицательные -> матрица не продуктивна
```

```
]: 2x2 Matrix{Float64}:
 -0.0  -0.333333
 -0.5   0.0
```

```
•[159]: ## b)
        a = [1/2 1; 3/2 1/2]
        e = Matrix{Int}(I,2,2)
        inv(e-a)
        ## все элементы матрицы отрицательные -> матрица не продуктивна

 [159]: 2×2 Matrix{Float64}:
         -0.4  -0.8
         -1.2  -0.4

•[160]: ## c)
        a = [1/10 2/10; 3/10 1/10]
        e = Matrix{Int}(I,2,2)
        inv(e-a)
        ## все элементы матрицы не отрицательные -> матрица продуктивна

 [160]: 2×2 Matrix{Float64}:
         1.2  0.266667
         0.4  1.2
```

4.4.4. Линейные модели экономики

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.

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

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

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

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

•[164]:

```
## a)
a = [1 2; 3 1]
eigvals(a)
## Все собственные значения по модулю не меньше 1 -> матрица не продуктивна
```

[164]:

```
2-element Vector{Float64}:
 -1.4494897427831779
  3.4494897427831783
```

```
[166]: ## b)
       b = [1/2 1; 3/2 1/2]
       eigvals(b)
       ## Не все собственные значения по модулю меньше 1 -> матрица не продуктивна

[166]: 2-element Vector{Float64}:
        -0.7247448713915892
         1.724744871391589

•[167]: ## c)
       c = [1/10 2/10; 3/10 1/10]
       eigvals(c)
       ## Все собственные значения по модулю меньше 1 -> матрица продуктивна

[167]: 2-element Vector{Float64}:
        -0.14494897427831785
         0.34494897427831783

•[168]: ## d)
       d = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
       eigvals(d)
       ## Все собственные значения по модулю меньше 1 -> матрица продуктивна

[168]: 3-element Vector{Float64}:
        0.02679491924311228
        0.1
        0.37320508075688774
```

Выводы

Выводы

Использую Jupyter lab повторил примеры из раздела 4.2 и выполнил задания для самостоятельной работы. Изучил возможности специальных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.