Лабораторная работа №4

Линейная алгебра

Тазаева Анастасия Анатольевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

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

3 Выполнение лабораторной работы

3.1 Поэлементные операции над многомерными массивами

Примеры представлены на рис. 3.1 и 3.2 :

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

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

```
4×3 Matrix{Int64}:
 12  16  11
 14   8   3
  7  20  16
 10  19  10
```

```
# Поэлементная сумма:
sum(a)
```

```
146
```

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

```
1×3 Matrix{Int64}:
 43  63  40
```

```
# Поэлементная сумма по строкам:
sum(a,dims=2)
```

```
4×1 Matrix{Int64}:
 39
 25
 43
 39
```

```
# Поэлементное произведение:
prod(a)
```

```
3020193792000
```

Рис. 3.1: Поэлементные операции над многомерными массивами. Примеры. Часть 1

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

```
1×3 Matrix{Int64}:
 11760  48640  5280
```

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

```
4×1 Matrix{Int64}:
 2112
  336
 2240
 1900
```

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics:

```
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics
```

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

```
# Вычисление среднего значения массива:
mean(a)
```

```
12.166666666666666
```

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

```
1×3 Matrix{Float64}:
 10.75  15.75  10.0
```

```
# Среднее по строкам:
mean(a,dims=2)
```

```
4×1 Matrix{Float64}:
 13.0
  8.333333333333334
 14.333333333333334
 13.0
```

Рис. 3.2: Поэлементные операции над многомерными массивами. Примеры. Часть 2

3.2 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

Примеры представлены на рис. 3.3 и 3.4 :

Для выполнения таких операций над матрицами, как транспонирование, диагонализация, определение следа, ранга, определителя матр

```
# Подключение пакета LinearAlgebra:
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
```

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

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

```
4×4 Matrix{Int64}:
  6  17   3   1
 13  16   7  13
  3  17  16   2
 18   5  17  14
```

```
# Транспонирование:
transpose(b)
```

```
4×4 transpose(::Matrix{Int64}) with eltype Int64:
  6  13   3  18
 17  16  17   5
  3   7  16  17
  1  13   2  14
```

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

```
52
```

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

```
4-element Vector{Int64}:
  6
 16
 16
 14
```

Рис. 3.3: Транспонирование, след, ранг, определитель и инверсия матрицы. Примеры. Часть 1

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

```
4-element Vector{Int64}:
  6
 16
 16
 14
```

```
# Ранг матрицы:
rank(b)
```

```
4
```

```
# Инверсия матрицы (определение обратной матрицы):
inv(b)
```

```
4×4 Matrix{Float64}:
  0.144229   -0.0954939  -0.0807972   0.0899133
  0.0226343   0.0306759   0.0168804  -0.032513
 -0.0317158  -0.0337955   0.0559792   0.0256499
 -0.155009    0.15286     0.0298787  -0.0637088
```

```
# Определитель матрицы:
det(b)
```

```
28849.999999999996
```

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

```
3×4 Matrix{Float64}:
  0.0481157   0.0830063  -0.00431273  -0.0709288
 -0.0786708  -0.0502929  -0.0702102    0.213962
  0.099349    0.0137191   0.138918    -0.235668
```

Рис. 3.4: Транспонирование, след, ранг, определитель и инверсия матрицы. Примеры. Часть 2

3.3 Вычисление нормы векторов и матриц, повороты, вращения

Для вычисления нормы используется LinearAlgebra.norm(x) .

Примеры представлены на рис. 3.5, 3.6 :

Для вычисления нормы используется LinearAlgebra.norm(x).

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

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

```
# Вычисление евклидовой нормы:
norm(X)
```

```
6.708203932499369
```

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

```
11.0
```

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

```
9.486832980505138
```

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

```
9.486832980505138
```

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

```
2.4404307889469252
```

Рис. 3.5: Вычисление нормы векторов и матриц, повороты, вращения. Примеры. Часть 1

```
# 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:
opnorm(d)
```

```
7.147682841795258
```

```
# Вычисление p-нормы:
p=1
opnorm(d,p)
```

```
8.0
```

```
# Поворот на 180 градусов:
rot180(d)
```

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

```
# Переворачивание строк:
reverse(d,dims=1)
```

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

```
# Переворачивание столбцов
reverse(d,dims=2)
```

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

Рис. 3.6: Вычисление нормы векторов и матриц, повороты, вращения. Примеры. Часть 2

3.4 Матричное умножение, единичная матрица, скалярное произведение

Примеры представлены на рис. 3.7 :

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

```
2x3 Matrix{Int64}:
 10  6  9
  9  5  9
```

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

```
3x4 Matrix{Int64}:
 2  5  1  4
 7  1  1  4
 5  4  6  3
```

```
# Произведение матриц A и B:
A*B
```

```
2x4 Matrix{Int64}:
 107  92  70  91
  98  86  68  83
```

```
# Единичная матрица 3x3:
Matrix{Int}(I, 3, 3)
```

```
3x3 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
```

Рис. 3.7: Матричное умножение, единичная матрица, скалярное произведение. Примеры

3.5 Факторизация. Специальные матричные структуры

В математике факторизация (или разложение) объекта — его декомпозиция (например,числа,
Матрица может быть факторизована на произведение матриц специального вида для приложен
Рассмотрим несколько примеров. Для работы со специальными матричными структурами потре

Примеры представлены на рис. 3.8 - 3.14 :

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

```
3x3 Matrix{Float64}:
 0.864238  0.047893  0.296736
 0.111947  0.576321  0.608906
 0.905204  0.412095  0.025299
```

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

```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

```
# Задаём вектор b:
b = A*x
```

```
3-element Vector{Float64}:
 1.2088667026265363
 1.2971737068077844
 1.3425980522157972
```

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

```
3-element Vector{Float64}:
 0.9999999999999999
 1.0000000000000004
 0.9999999999999999
```

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

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3x3 Matrix{Float64}:
 1.0        0.0       0.0
 0.12367    1.0       0.0
 0.954744  -0.657747  1.0
U factor:
3x3 Matrix{Float64}:
 0.905204  0.412095  0.025299
 0.0       0.525357  0.605777
 0.0       0.0       0.67103
```

Рис. 3.8: Факторизация. Специальные матричные структуры. Примеры. Часть 1

```
# Матрица перестановок:
Alu.P
```

```
3×3 Matrix{Float64}:
 0.0  0.0  1.0
 0.0  1.0  0.0
 1.0  0.0  0.0
```

```
# Вектор перестановок:
Alu.p
```

```
3-element Vector{Int64}:
 3
 2
 1
```

```
# Матрица L:
Alu.L
```

```
3×3 Matrix{Float64}:
 1.0        0.0       0.0
 0.12367    1.0       0.0
 0.954744  -0.657747  1.0
```

```
# Матрица U:
Alu.U
```

```
3×3 Matrix{Float64}:
 0.905204  0.412095  0.025299
 0.0       0.525357  0.605777
 0.0       0.0       0.67103
```

```
# Решение СЛАУ через матрицу A:
A\b
```

```
3-element Vector{Float64}:
 0.9999999999999999
 1.0000000000000004
 0.9999999999999999
```

```
# Решение СЛАУ через объект факторизации:
Alu\b
```

```
3-element Vector{Float64}:
 0.9999999999999999
 1.0000000000000004
 0.9999999999999999
```

Рис. 3.9: Факторизация. Специальные матричные структуры. Примеры. Часть 2

```
# Детерминант матрицы A:
det(A)
```

```
-0.31911178560927134
```

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

```
-0.31911178560927134
```

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

```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
R factor:
3×3 Matrix{Float64}:
 -1.25652  -0.381163  -0.276571
  0.0      -0.599146  -0.450881
  0.0       0.0       -0.423879
```

```
# Матрица Q:
Aqr.Q
```

```
3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
```

```
# Матрица R:
Aqr.R
```

```
3×3 Matrix{Float64}:
 -1.25652  -0.381163  -0.276571
  0.0      -0.599146  -0.450881
  0.0       0.0       -0.423879
```

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

```
3×3 Matrix{Float64}:
  1.0          -5.55112e-17  -2.22045e-16
 -5.55112e-17   1.0           0.0
  0.0          -2.77556e-17   1.0
```

Рис. 3.10: Факторизация. Специальные матричные структуры. Примеры. Часть 3

```
# Симметризация матрицы A:
Asym = A + A'
```

```
3×3 Matrix{Float64}:
 1.72848  0.15984  1.20194
 0.15984  1.15264  1.021
 1.20194  1.021    0.0505981
```

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

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 -0.9278506612480992
  1.210394183881841
  2.6491722846568964
vectors:
3×3 Matrix{Float64}:
 -0.360356   0.563092  -0.743687
 -0.388501  -0.815412  -0.42915
  0.848063  -0.134277  -0.5126
```

```
# Собственные значения:
AsymEig.values
```

```
3-element Vector{Float64}:
 -0.9278506612480992
  1.210394183881841
  2.6491722846568964
```

```
#Собственные векторы:
AsymEig.vectors
```

```
3×3 Matrix{Float64}:
 -0.360356   0.563092  -0.743687
 -0.388501  -0.815412  -0.42915
  0.848063  -0.134277  -0.5126
```

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

```
3×3 Matrix{Float64}:
  1.0           7.77156e-16  -1.79717e-15
  1.11022e-15   1.0          -2.31065e-15
 -1.44329e-15  -1.33227e-15   1.0
```

Рис. 3.11: Факторизация. Специальные матричные структуры. Примеры. Часть 4

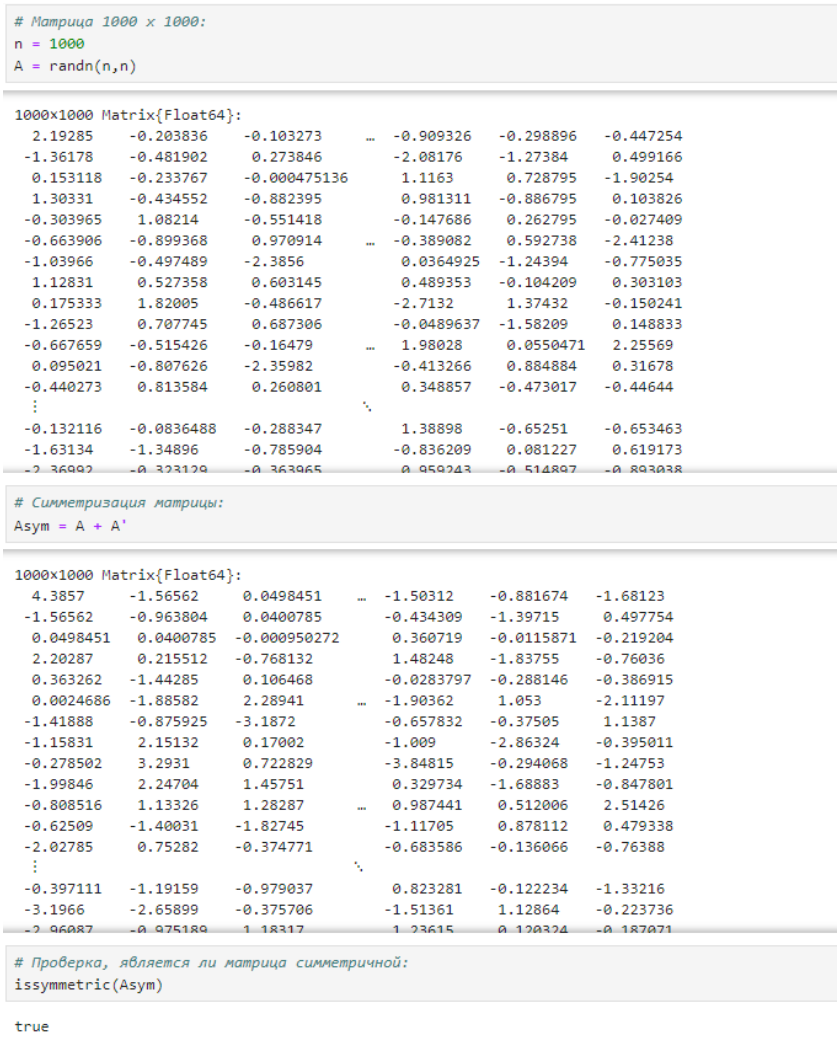


Рис. 3.12: Факторизация. Специальные матричные структуры. Примеры. Часть 5

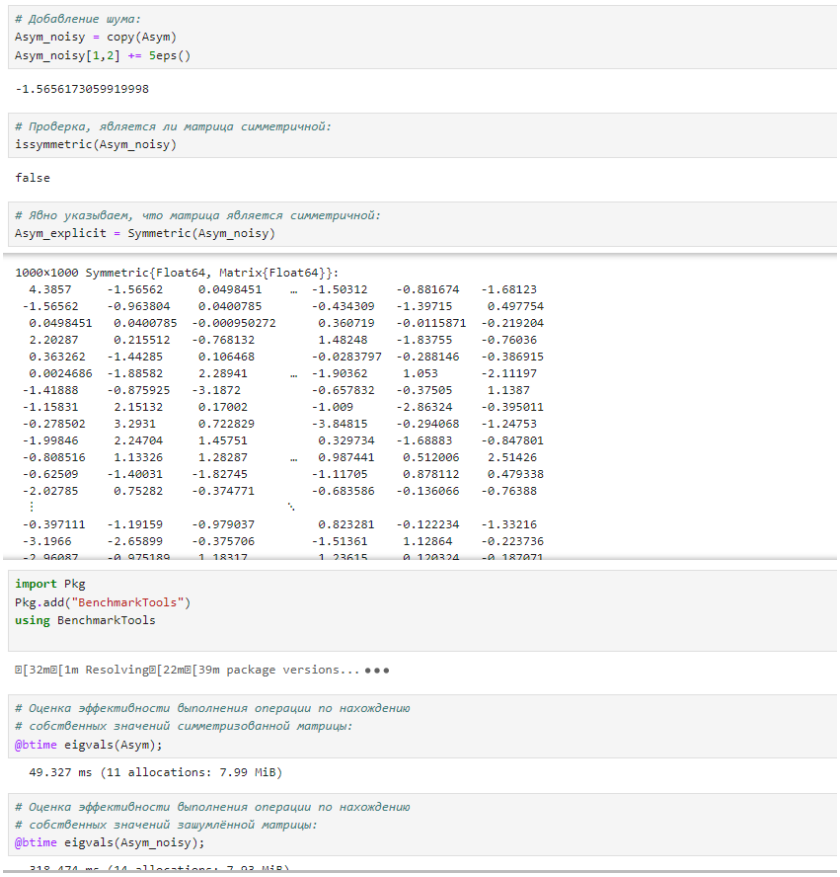


Рис. 3.13: Факторизация. Специальные матричные структуры. Примеры. Часть 6

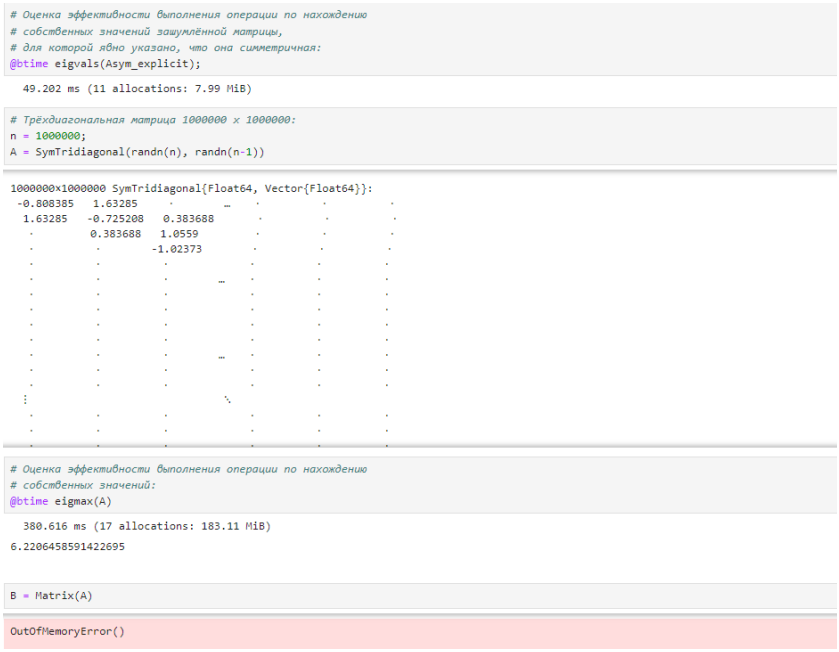


Рис. 3.14: Факторизация. Специальные матричные структуры. Примеры. Часть 7

3.6 Общая линейная алгебра

Обычный способ добавить поддержку числовой линейной алгебры - это обернуть подпрограммы BLAS и LAPACK . Собственно, для матриц с элементами Float32 , Float64 , Complex {Float32} или Complex {Float64} разработчики Julia использовали такое же решение. Однако Julia также поддерживает общую линейную алгебру, что позволяет, например, работать с матрицами и векторами рациональных чисел. Для задания рационального числа используется двойная косая черта: 1//2

Примеры представлены на рис. 3.15 :

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

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

```
# Единичный вектор:
x = fill(1, 3)
```

```
3-element Vector{Int64}:
 1
 1
 1
```

```
# Задаём вектор b:
b = Arational*x
```

```
3-element Vector{Rational{BigInt}}:
 11//5
 11//10
  3//2
```

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

```
3-element Vector{Rational{BigInt}}:
 1
 1
 1
```

```
# LU-разложение:
lu(Arational)
```

```
LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3x3 Matrix{Rational{BigInt}}:
  1       0     0
 7//8     1     0
 3//4  -26//27  1
U factor:
3x3 Matrix{Rational{BigInt}}:
 4//5    1//2    1//5
  0    -27//80   1//8
  0       0     47//54
```

Рис. 3.15: Общая линейная алгебра. Примеры

3.7 Самостоятельная работа

Выполнение заданий можно просмотреть на рис. 3.16 - 3.23 :

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.

```
#вектор v
v = [ 1, 2, 3]
dot_v = v'v
outer_v = v*v'
println("dot_v = ", dot_v)
println("outer_v = ", outer_v)
```

```
dot_v = 14
outer_v = [1 2 3; 2 4 6; 3 6 9]
```

Рис. 3.16: Самостоятельная работа. Задание 1

2. Системы линейных уравнений

1. Решить СЛАУ с двумя неизвестными.
2. Решить СЛАУ с тремя неизвестными.

```
# СЛАУ с 2-мя неизвестными
# если определитель отличен от 0, то находим обратную матрицу. Иначе решения нет
# определить детерминант не квадратного уравнения не удастся
function res_x(A, B)
    if size(A,1) == size(A,2)
        if det(A) != 0
            println(A \ B)
        else
            println("Определитель матрицы равен 0 - решения не существует")
        end
    else
        println(A \ B)
    end
end
println("\nСЛАУ с 2-мя неизвестными")
res_x([1 1; 1 -1], [2; 3]) # решение пункта a)
res_x([1 1; 2 2], [2; 4]) # решение пункта b)
res_x([1 1; 2 2], [2; 5]) # решение пункта c)
res_x([1 1; 2 2; 3 3], [1; 2; 3]) # решение пункта d)
res_x([1 1; 2 1; 1 -1], [2; 1; 3]) # решение пункта e)
res_x([1 1; 2 1; 3 2], [2; 1; 3]) # решение пункта f)
#
println("\nСЛАУ с 3-мя неизвестными")
res_x([1 1 1; 1 -1 -2], [2; 3]) # решение пункта a)
res_x([1 1 1; 2 2 -3; 3 1 1], [2; 4; 1]) # решение пункта b)
res_x([1 1 1; 1 1 2; 2 2 3], [1; 0; 1]) # решение пункта c)
res_x([1 1 1; 1 1 2; 2 2 3], [1; 0; 0]) # решение пункта d)
```

```
СЛАУ с 2-мя неизвестными
[2.5, -0.5]
Определитель матрицы равен 0 - решения не существует
Определитель матрицы равен 0 - решения не существует
[0.4999999999999999, 0.5]
[1.5000000000000004, -0.9999999999999997]
[-0.9999999999999989, 2.9999999999999982]

СЛАУ с 3-мя неизвестными
[2.2142857142857144, 0.35714285714285704, -0.5714285714285712]
[-0.5, 2.5, 0.0]
Определитель матрицы равен 0 - решения не существует
Определитель матрицы равен 0 - решения не существует
```

Рис. 3.17: Самостоятельная работа. Задание 2

3. Операции с матрицами

1. Приведите приведённые ниже матрицы к диагональному виду
2. Вычислите
3. Найдите собственные значения матрицы A. если (условие) Создайте диагональную матрицу из собственных значений матрицы A. Создайте нижнедиагональную матриц

```
# задание 1
function diag_vid(X)
    X_Eig=eigen(X)
    X_diag = diagm(X_Eig.values)
    return X_diag
end
println("Диагональный вид матрицы из пункта a: ")
display(diag_vid([1 -2; -2 1]))
println("Диагональный вид матрицы из пункта b: ")
display(diag_vid([1 -2; -2 3]))
println("Диагональный вид матрицы из пункта c: ")
display(diag_vid([1 -2 0; -2 1 2; 0 2 0]))
```

```
Диагональный вид матрицы из пункта a:
2×2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
Диагональный вид матрицы из пункта b:
2×2 Matrix{Float64}:
 -0.236068  0.0
  0.0       4.23607
Диагональный вид матрицы из пункта c:
3×3 Matrix{Float64}:
 -2.14134  0.0       0.0
  0.0      0.515138  0.0
  0.0      0.0       3.6262
```

Рис. 3.18: Самостоятельная работа. Задание 3.1

```
# задание 2
println("Результат вычисления пункта a: ")
display([1 -2; -2 1]^10)
println("Результат вычисления пункта b: ")
display([5 -2; -2 5]^(1/2))
println("Результат вычисления пункта c: ")
display([1 -2; -2 1]^(1/3))
println("Результат вычисления пункта d: ")
display([1 2; 2 3]^(1/2))
```

```
Результат вычисления пункта a:
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
Результат вычисления пункта b:
2×2 Symmetric{Float64, Matrix{Float64}}:
  2.1889   -0.45685
 -0.45685   2.1889
Результат вычисления пункта c:
2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
  0.971125+0.433013im  -0.471125+0.433013im
 -0.471125+0.433013im   0.971125+0.433013im
Результат вычисления пункта d:
2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
 0.568864+0.351578im  0.920442-0.217287im
 0.920442-0.217287im   1.48931+0.134291im
```

Рис. 3.19: Самостоятельная работа. Задание 3.2

```
# задание 3. собст.знач
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]
#display(A)
println("Собственные значения матрицы A и эффективность выполнения операций: ")
@btime A_Eig = eigen(A)
```

```
Собственные значения матрицы A и эффективность выполнения операций:
  3.538 μs (11 allocations: 3.00 KiB)
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
5-element Vector{Float64}:
 -128.49322764802145
  -55.887784553057
   42.752167279318854
   87.16111477514488
  542.467730146614
vectors:
5×5 Matrix{Float64}:
 -0.147575   0.647178   0.010882    0.548903   -0.507907
 -0.256795  -0.173068   0.834628   -0.239864   -0.387253
 -0.185537   0.239762  -0.422161   -0.731925   -0.440631
  0.819704  -0.247506  -0.0273194   0.0366447  -0.514526
 -0.453805  -0.657619  -0.352577    0.322668   -0.364928
```

```
# задание 3. диаг.матр
println("Диагональная матрица из собственных значений матрицы A и эффективность операции: ")
@btime diagm(A_Eig.values)
```

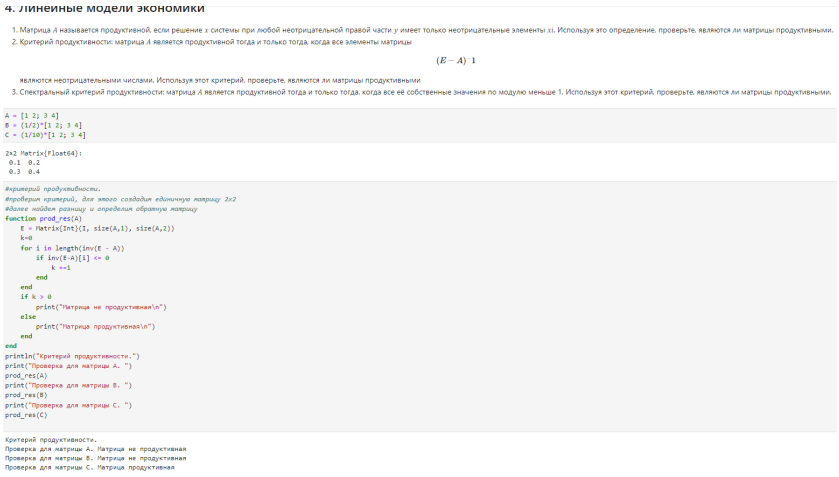
```
Диагональная матрица из собственных значений матрицы A и эффективность операции:
  52.439 ns (1 allocation: 256 bytes)
5×5 Matrix{Float64}:
 -128.493    0.0      0.0      0.0       0.0
    0.0    -55.8878   0.0      0.0       0.0
    0.0      0.0     42.7522   0.0       0.0
    0.0      0.0      0.0     87.1611    0.0
    0.0      0.0      0.0      0.0     542.468
```

Рис. 3.20: Самостоятельная работа. Задание 3.3. Часть 1

```
# задание 3. нижнедиаг.матр
println("Нижнедиагональная матрица из матрицы A и эффективность операции: ")
@btime LowerTriangular(A)
```

```
Нижнедиагональная матрица из матрицы A и эффективность операции:
  113.081 ns (1 allocation: 16 bytes)
5×5 LowerTriangular{Int64, Matrix{Int64}}:
 140    ⋅    ⋅    ⋅   ⋅
  97  106    ⋅    ⋅   ⋅
  74   89  152    ⋅   ⋅
 168  131  144   54   ⋅
 131   36   71  142  36
```

Рис. 3.21: Самостоятельная работа. Задание 3.3. Часть 2



4. Линейные модели экономики

1. Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части y имеет только неотрицательные элементы xi. Используя это определение, проверьте, являются ли матрицы продуктивными.
2. Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрицы

$$(E - A)\ 1$$

являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными
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 = [1 2; 3 4]
B = (1/2)*[1 2; 3 4]
C = (1/10)*[1 2; 3 4]
```

```
2×2 Matrix{Float64}:
 0.1  0.2
 0.3  0.4
```

```
#критерий продуктивности.
#проверим критерий, для этого создадим единичную матрицу 2×2
#далее найдем разницу и определим обратную матрицу
function prod_res(A)
    E = Matrix{Int}(I, size(A,1), size(A,2))
    k=0
    for i in length(inv(E - A))
        if inv(E-A)[i] <= 0
            k +=1
        end
    end
    if k > 0
        print("Матрица не продуктивная\n")
    else
        print("Матрица продуктивная\n")
    end
end
println("Критерий продуктивности.")
print("Проверка для матрицы A. ")
prod_res(A)
print("Проверка для матрицы B. ")
prod_res(B)
print("Проверка для матрицы C. ")
prod_res(C)
```

```
Критерий продуктивности.
Проверка для матрицы A. Матрица не продуктивная
Проверка для матрицы B. Матрица не продуктивная
Проверка для матрицы C. Матрица продуктивная
```

Рис. 3.22: Самостоятельная работа. Задание 4. Часть 1

```
#спектральный критерий продуктивности.
function spectr_prod_res(A)
    k=0
    for i in length(eigen(A).values)
        if abs(eigen(A).values) >= 1
            k +=1
        end
    end
    if k > 0
        print("Матрица не продуктивная\n")
    else
        print("Матрица продуктивная\n")
    end
end
println("Критерий спектральной продуктивности.")
print("Проверка для матрицы A. ")
prod_res(A)
print("Проверка для матрицы B. ")
prod_res(B)
print("Проверка для матрицы C. ")
prod_res(C)
print("Проверка для матрицы D. ")
prod_res([0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3])
```

```
Критерий спектральной продуктивности.
Проверка для матрицы A. Матрица не продуктивная
Проверка для матрицы B. Матрица не продуктивная
Проверка для матрицы C. Матрица продуктивная
Проверка для матрицы D. Матрица продуктивная
```

Рис. 3.23: Самостоятельная работа. Задание 4. Часть 2

4 Выводы

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