Лабораторная работа №4

Линейная алгебра

Лебедева Ольга Андреевна

Содержание

Список иллюстраций

Цель работы

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

Задачи

- Изучение возможностей языка программирования Julia для выполнения операций линейной алгебры. Закрепление навыков работы с векторами и матрицами
- Решение систем линейных уравнений.
- Приведение матриц к диагональному виду.
- Вычисление собственных значений и векторов матриц.
- Оценка эффективности выполнения операций с использованием встроенных библиотек.

Объект и предмет исследования

Объектом исследования являются векторы и матрицы, их математические свойства и операции с ними.

Исследуются также программные средства (функции языка Julia) для работы с линейной алгеброй.

Условные обозначения и термины

Вектор — одномерный массив чисел, представляющий величины с направлением.

Матрица — двумерный массив чисел, используемый для представления систем уравнений или линейных преобразований.

Скалярное произведение — сумма произведений соответствующих элементов двух векторов.

Внешнее произведение — операция, создающая матрицу из двух векторов.

Продуктивность матрицы — свойство, определяющее возможность решения линейной модели $(E-A)x=y$ с неотрицательными значениями.

Техническое оснащение и выбранные методы проведения работы

Основные вычисления реализованы в среде Jupyter Notebook с применением библиотеки LinearAlgebra для операций с матрицами, спектрального разложения и решения СЛАУ.

Теоретическое введение

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения[1].

Задание

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

Выполнение лабораторной работы.

Самостоятельное выполнение

Задание См. рис. 1, См. рис. 2, См. рис. 3

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`.

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

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

4.4.3. Операции с матрицами

1. Приведите приведённые ниже матрицы к диагональному виду

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

Рис. 1: Задание_1

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

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

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

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.

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

Рис. 2: Задание_2

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

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

Рис. 3: Задание_3

Выполним задание 1: См. рис. 4

```
[2]: using LinearAlgebra

     v = [1, 2, 3]

     # Скалярное произведение (dot_v
     dot_v = dot(v, v)
     # Внешнее произведение (outer_v)
     outer_v = v * transpose(v)

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

[4]: dot_v

[4]: 14

[8]: outer_v

[8]: 3×3 Matrix{Int64}:
      1  2  3
      2  4  6
      3  6  9
```

Рис. 4: Задание 1_1

Для выполнения задания был задан вектор □=[1,2,3]. Затем с помощью функции dot было вычислено скалярное произведение, а матричное произведение было получено умножением вектора □ его транспонированный вариант □′.

Выполним задание 2: См. рис. 5, См. рис. 6

```
[186]: using LinearAlgebra

# Универсальная функция для решения СЛАУ
function solve_system(A, b)
    try
        x = A \ b
        println("Решение: ", x)
    catch e
        if e isa SingularException
            println("Система вырождена (бесконечное множество решений или несовместна).")
        else
            rethrow(e)
        end
    end
end

# СЛАУ с двумя неизвестными
systems_2x2 = [
    ([1 1; 1 -1], [2; 3]),    # (a)
    ([1 1; 2 2], [2; 4]),     # (b)
    ([1 1; 2 2], [2; 5]),     # (c)
    ([1 1; 2 2; 3 3], [1; 2; 3]), # (d)
    ([1 1; 2 1; 1 -1], [2; 1; 3]), # (e)
    ([1 1; 2 1; 3 2], [2; 1; 3])   # (f)
]

for (A, b) in systems_2x2
    solve_system(A, b)
    println()
end
```

```
Решение: [2.5, -0.5]

Система вырождена (бесконечное множество решений или несовместна).

Система вырождена (бесконечное множество решений или несовместна).

Решение: [0.5, 0.5]

Решение: [1.5000000000000004, -0.9999999999999997]

Решение: [-0.9999999999999994, 2.999999999999999]
```

Рис. 5: Задание 2_1

```
[17]: # СЛАУ с тремя неизвестными
systems_3x3 = [
    ([1 1 1; 1 -1 -2], [2; 3]),             # (a)
    ([1 1 1; 2 2 -3; 3 1 1], [2; 4; 1]),    # (b)
    ([1 1 1; 1 1 2; 2 2 3], [1; 0; 1]),     # (c)
    ([1 1 1; 1 1 2; 2 2 3], [1; 0; 0])      # (d)
]

println("Решения СЛАУ с тремя неизвестными:")
for (A, b) in systems_3x3
    solve_system(A, b)
    println()
end
```

```
Решения СЛАУ с тремя неизвестными:
Решение: [2.214285714285715, 0.35714285714285715, -0.5714285714285711]

Решение: [-0.5, 2.5, 0.0]

Система вырождена (бесконечное множество решений или несовместна).

Система вырождена (бесконечное множество решений или несовместна).
```

Рис. 6: Задание 2_2

Системы могут быть: - Совместными (имеющими одно или несколько решений). - Несовместными (не имеющими решений).

Для решения СЛАУ использовался оператор обратного слэша , который автоматически выбирает оптимальный метод для нахождения решения.

Выполним задание 3.1-3.2: См. рис. 7, См. рис. 8

```
[40]: using LinearAlgebra

# функция для приведения матрицы к диагональному виду
function diagonalize_matrix(A)
    eig = eigen(A)
    D = Diagonal(eig.values)  # Диагональная матрица
    P = eig.vectors           # Матрица собственных векторов
    println("Матрица A:")
    display(A)
    println("Диагональная матрица D:")
    display(D)
    println("Матрица собственных векторов P:")
    display(P)
end

println("Приведение матриц к диагональному виду:")

# Задачи из пункта (a), (b), (c)
matrices = [
    [1 -2; -2 1],                # (a)
    [1 -2; -2 3],                # (b)
    [1 -2 0; -2 1 2; 0 2 0]      # (c)
]

for A in matrices
    diagonalize_matrix(A)
    display()
end
```

Рис. 7: Задание 3_1

```
Приведение матриц к диагональному виду:
Матрица A:
2×2 Matrix{Int64}:
  1  -2
 -2   1
Диагональная матрица D:
2×2 Diagonal{Float64, Vector{Float64}}:
 -1.0   ⋅
   ⋅   3.0
2×2 Matrix{Float64}:
 -0.707107  -0.707107
 -0.707107   0.707107
Матрица собственных векторов P:
Матрица A:
2×2 Matrix{Int64}:
  1  -2
 -2   3

Диагональная матрица D:
2×2 Diagonal{Float64, Vector{Float64}}:
 -0.236068   ⋅
   ⋅        4.23607

Матрица собственных векторов P:
2×2 Matrix{Float64}:
 -0.850651  -0.525731
 -0.525731   0.850651

Матрица A:
3×3 Matrix{Int64}:
  1  -2  0
 -2   1  2
  0   2  0

3×3 Diagonal{Float64, Vector{Float64}}:
 -2.14134   ⋅         ⋅
   ⋅       0.515138   ⋅
   ⋅        ⋅        3.6262
Диагональная матрица D:
Матрица собственных векторов P:
3×3 Matrix{Float64}:
  0.421859  0.717093   0.554808
  0.6626    0.173846  -0.728518
 -0.618866  0.674948  -0.401808
```

Рис. 8: Задание 3_2

- Приведение к диагональному виду выполнялось через спектральное разложение: были найдены собственные значения и собственные векторы матрицы.
- Собственные значения формируют диагональную матрицу, а собственные векторы — матрицу преобразования.

Выполним задание 3.2: См. рис. 9

```
[42]: using LinearAlgebra

# Функция для возведения матрицы в степень (с поддержкой комплексных чисел)
function matrix_power(A, p)
    eig = eigen(A)
    D = Diagonal((eig.values .+ 0im) .^ p)   # Приведение к комплексному типу
    P = eig.vectors
    return real(P * D * inv(P))  # Берём только вещественную часть
end

# Функция для извлечения корня (квадратного или кубического)
function matrix_root(A, n)
    return matrix_power(A, 1/n)
end

println("Возведение матриц в степень и извлечение корней:")

# Матрицы из задания
matrices_powers = [
    ([1 -2; -2 1], 10),                   # (a)
    ([5 -2; -2 5], 0.5),                  # (b) Квадратный корень
    ([1 -2; -2 1], 1/3),                  # (c) Кубический корень
    ([1 2; 2 3], 0.5)                     # (d) Квадратный корень
]

for (A, p) in matrices_powers
    println("Матрица:")
    println(A)
    println("Результат для степени ", p, ":")
    println(matrix_power(A, p))
    println()
end
```

```
Возведение матриц в степень и извлечение корней:
Матрица:
[1 -2; -2 1]
Результат для степени 10:
[29525.0 -29524.0; -29524.0 29525.0]

Матрица:
[5 -2; -2 5]
Результат для степени 0.5:
[2.188901059316734 -0.45685025174785676; -0.45685025174785676 2.188901059316734]

Матрица:
[1 -2; -2 1]
Результат для степени 0.3333333333333333:
[0.971124785153704 -0.47112478515370404; -0.47112478515370404 0.971124785153704]

Матрица:
[1 2; 2 3]
Результат для степени 0.5:
[0.5688644810057829 0.920442065259926; 0.920442065259926 1.4893065462657094]
```

Рис. 9: Задание 3.2

- Возведение в степень или извлечение корня реализовывалось через спектральное разложение, что позволило работать с матрицами, включая те, которые имеют отрицательные собственные значения.
- Спектральный метод позволяет вычислять дробные степени матрицы, если преобразовать собственные значения в комплексный тип.

Выполним задание 3.3: См. рис. 10,

```
[29]: using LinearAlgebra

# Задаём матрицу A
A = [
    140  97  74  168 131;
     97 106  89  131  36;
     74  89 152  144  71;
    168 131 144   54 142;
    131  36  71  142  36
]

# Нахождение собственных значений и векторов
eig = eigen(A)

# Собственные значения
eigenvalues = eig.values
println("Собственные значения матрицы A:")
display(eigenvalues)

# Создание диагональной матрицы
D = Diagonal(eigenvalues)
println("\nДиагональная матрица из собственных значений:")
display(D)

# Создание нижнетреугольной матрицы
L = tril(A)
println("\nНижнетреугольная матрица:")
display(L)
```

```
Собственные значения матрицы A:
5-element Vector{Float64}:
 -128.49322764802136
  -55.88778455305702
   42.75216727931888
   87.16111477514494
  542.4677301466138

Диагональная матрица из собственных значений:
5×5 Diagonal{Float64, Vector{Float64}}:
 -128.493     ⋅         ⋅         ⋅         ⋅
     ⋅     -55.8878     ⋅         ⋅         ⋅
     ⋅        ⋅       42.7522     ⋅         ⋅
     ⋅        ⋅         ⋅       87.1611     ⋅
     ⋅        ⋅         ⋅         ⋅       542.468

Нижнетреугольная матрица:
5×5 Matrix{Int64}:
 140    0    0    0   0
  97  106    0    0   0
  74   89  152    0   0
 168  131  144   54   0
 131   36   71  142  36
```

Рис. 10: Задание 3.3

- Были найдены собственные значения заданной матрицы, на основе которых была сформирована диагональная матрица.
- Также была создана нижнетреугольная матрица с элементами исходной матрицы.

Выполним задание 4: См. рис. 11, См. рис. 12

```
[37]: using LinearAlgebra

# Функция для проверки продуктивности с использованием (E - A)^-1 и спектрального критерия
function check_productivity(A, b)
    try
        x = A \ b
        println("Система решаема, продуктивная матрица.")
    catch
        println("Система не решаема, матрица не продуктивна.")
    end

    I_matrix = Matrix{Float64}(I, size(A, 1), size(A, 2))  # Единичная матрица
    try
        inv_E_minus_A = inv(I_matrix - A)
        println("Продуктивная матрица через (E - A)^-1.")
    catch
        println("Матрица не продуктивна через (E - A)^-1.")
    end

    eigvals = eigen(A).values
    if all(abs.(eigvals) .< 1)
        println("Продуктивная матрица через спектральный критерий.")
    else
        println("Матрица не продуктивна через спектральный критерий.")
    end
end

# Задаём матрицы для проверки
matrices = [
    [1 2; 3 4],         # (a)
    [1 2; 3 4] * 1/2,   # (b)
    [1 2; 3 4] * 1/10,  # (c)
    [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]  # Матрица d для спектрального критерия
]

println("Проверка продуктивности матриц:")

# Проверка для каждой матрицы
for A in matrices
    display(A)  # Используем display для вывода матрицы
    b = [1.0, 2.0]  # Вектор правых частей для СЛАУ
    println("\nПроверка продуктивности матрицы:")
    check_productivity(A, b)
    println()
end
```

Рис. 11: Задание 4

```
Проверка продуктивности матриц:
2×2 Matrix{Float64}:
 1.0  2.0
 3.0  4.0

Проверка продуктивности матрицы:
Система решаема, продуктивная матрица.
Продуктивная матрица через (E - A)^-1.
Матрица не продуктивна через спектральный критерий.
2×2 Matrix{Float64}:
 0.5  1.0
 1.5  2.0


Проверка продуктивности матрицы:
Система решаема, продуктивная матрица.
Продуктивная матрица через (E - A)^-1.
Матрица не продуктивна через спектральный критерий.
2×2 Matrix{Float64}:
 0.1  0.2
 0.3  0.4


Проверка продуктивности матрицы:
Система решаема, продуктивная матрица.
Продуктивная матрица через (E - A)^-1.
3×3 Matrix{Float64}:
 0.1  0.2  0.3
 0.0  0.1  0.2
 0.0  0.1  0.3

Продуктивная матрица через спектральный критерий.


Проверка продуктивности матрицы:
Система не решаема, матрица не продуктивна.
Продуктивная матрица через (E - A)^-1.
Продуктивная матрица через спектральный критерий.
```

Рис. 12: Задание 4_1

Линейная модель экономики реализовывалась через проверку продуктивности матриц.

Было проверено три критерия: - Возможность решения системы. - Наличие обратной матрицы (E−A)^−1. - Спектральный критерий (все собственные значения по модулю меньше 1).

Выполнение лабораторной работы.

Повторение примеров

Повторим примеры из раздела 4.2: См. рис. 13, См. рис. 14, См. рис. 15, См. рис. 16, См. рис. 17, См. рис. 18, См. рис. 19, См. рис. 20, См. рис. 21, См. рис. 22, См. рис. 23, См. рис. 24, См. рис. 25, См. рис. 26

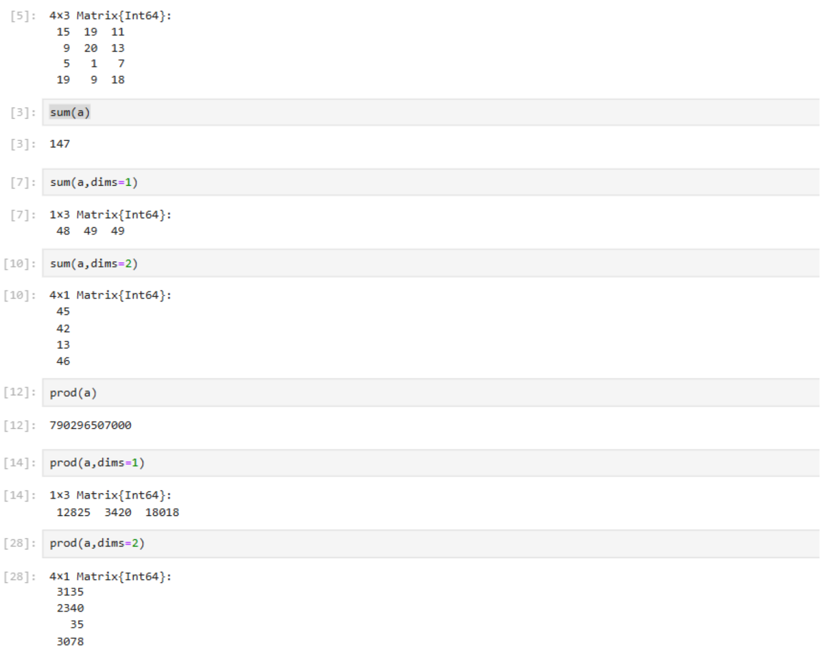


Рис. 1: Повторение примеров_1

```
[29]: import Pkg
      Pkg.add("Statistics")

   Resolving package versions...
  No Changes to `C:\Users\user\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\user\.julia\environments\v1.11\Manifest.toml`

[25]: using Statistics
      mean(a)

[25]: 12.166666666666666

[33]: mean(a,dims=1)

[33]: 1×3 Matrix{Float64}:
       12.0  12.25  12.25

[35]: mean(a,dims=2)

[35]: 4×1 Matrix{Float64}:
       15.0
       14.0
        4.333333333333333
       15.333333333333334
```

Рис. 2: Повторение примеров_2

```
      using LinearAlgebra

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

[39]: b = rand(1:20,(4,4))

[39]: 4×4 Matrix{Int64}:
       20   7  20  12
        7   3  15   2
       17   5  12   7
       13  12   5  11

[41]: transpose(b)

[41]: 4×4 transpose(::Matrix{Int64}) with eltype Int64:
       20   7  17  13
        7   3   5  12
       20  15  12   5
       12   2   7  11

[43]: tr(b)

[43]: 46

[45]: diag(b)

[45]: 4-element Vector{Int64}:
       20
        3
       12
       11

[47]: rank(b)

[47]: 4

[49]: inv(b)

[49]: 4×4 Matrix{Float64}:
       -0.0847369  -0.0293454   0.186664   -0.0210106
       -0.156798    0.14447     0.0277826   0.127105
        0.0369856   0.0722348  -0.0630318  -0.0133704
        0.254384   -0.155756   -0.222261   -0.0168432

[51]: det(b)

[51]: -5759.000000000002

[53]: pinv(a)

[53]: 3×4 Matrix{Float64}:
        0.103743   -0.112739   -0.0485776   0.0369153
        0.0336345   0.0360712  -0.0235894  -0.0374321
       -0.118892    0.0969558   0.0815425   0.0264774
```

Рис. 3: Повторение примеров_3

```
[55]: X = [2, 4, -5]

[55]: 3-element Vector{Int64}:
        2
        4
       -5

[57]: norm(X)

[57]: 6.708203932499369

[59]: p = 1
      norm(X,p)

[59]: 11.0

[61]: X = [2, 4, -5];
      Y = [1,-1,3];
      norm(X-Y)

[61]: 9.486832980505138

[63]: sqrt(sum((X-Y).^2))

[63]: 9.486832980505138

[65]: acos((transpose(X)*Y)/(norm(X)*norm(Y)))

[65]: 2.4404307889469252
```

Рис. 4: Повторение примеров_4

```
[67]: d = [5 -4 2 ; -1 2 3; -2 1 0]

[67]: 3×3 Matrix{Int64}:
        5  -4  2
       -1   2  3
       -2   1  0

[69]: opnorm(d)

[69]: 7.147682841795258

[71]: p=1
      opnorm(d,p)

[71]: 8.0

[73]: rot180(d)

[73]: 3×3 Matrix{Int64}:
       0   1  -2
       3   2  -1
       2  -4   5

[75]: reverse(d,dims=1)

[75]: 3×3 Matrix{Int64}:
       -2   1  0
       -1   2  3
        5  -4  2

[77]: reverse(d,dims=2)

[77]: 3×3 Matrix{Int64}:
       2  -4   5
       3   2  -1
       0   1  -2
```

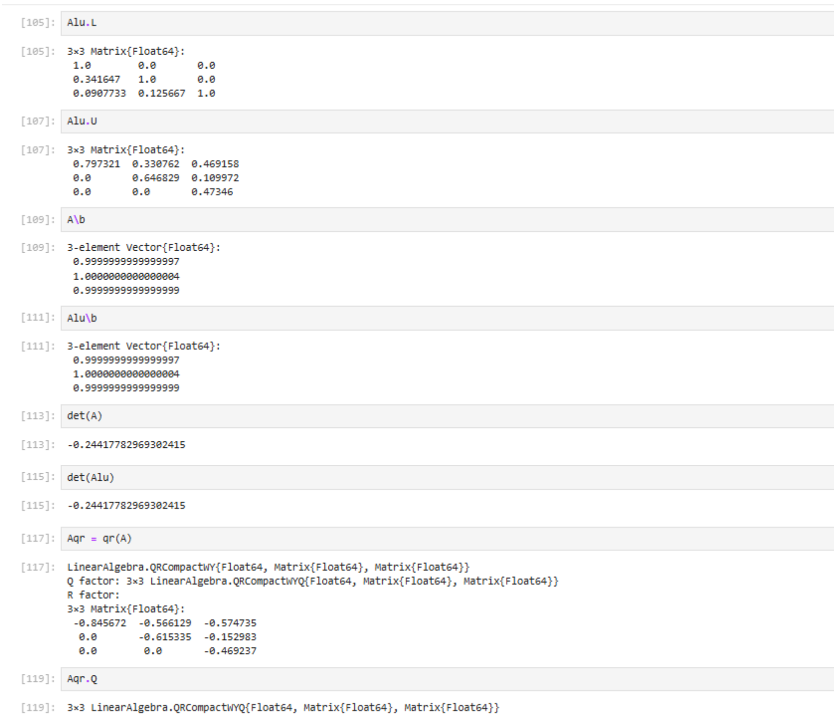
Рис. 5: Повторение примеров_5

```
[79]: A = rand(1:10,(2,3))

[79]: 2×3 Matrix{Int64}:
       6  1  10
       4  2   2

[81]: B = rand(1:10,(3,4))

[81]: 3×4 Matrix{Int64}:
        2  10  1  4
        5   3  5  6
       10   6  6  5

[83]: A*B

[83]: 2×4 Matrix{Int64}:
       117  123  71  80
        38   58  26  38

[85]: Matrix{Int}(I, 3, 3)

[85]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1

[87]: X = [2, 4, -5]
      Y = [1,-1,3]
      dot(X,Y)

[87]: -17

[89]: X'Y

[89]: -17
```

Рис. 6: Повторение примеров_6

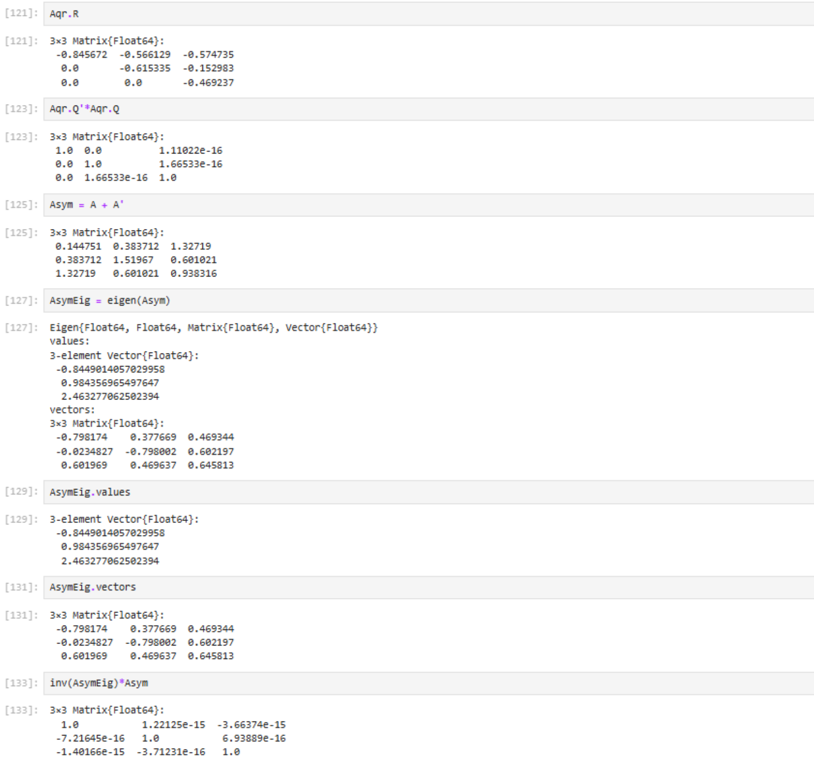
```
[91]: A = rand(3, 3)

[91]: 3×3 Matrix{Float64}:
       0.0723754  0.11131   0.529867
       0.272403   0.759833  0.270259
       0.797321   0.330762  0.469158

[93]: x = fill(1.0, 3)

[93]: 3-element Vector{Float64}:
       1.0
       1.0
       1.0

[95]: b = A*x

[95]: 3-element Vector{Float64}:
       0.713552275158978
       1.3024942180850514
       1.5972403112861897

[97]: A\b

[97]: 3-element Vector{Float64}:
       0.9999999999999997
       1.0000000000000004
       0.9999999999999999

[99]: Alu = lu(A)

[99]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3×3 Matrix{Float64}:
       1.0        0.0       0.0
       0.341647   1.0       0.0
       0.0907733  0.125667  1.0
      U factor:
      3×3 Matrix{Float64}:
       0.797321  0.330762  0.469158
       0.0       0.646829  0.109972
       0.0       0.0       0.47346

[101]: Alu.P

[101]: 3×3 Matrix{Float64}:
        0.0  0.0  1.0
        0.0  1.0  0.0
        1.0  0.0  0.0

[103]: Alu.p

[103]: 3-element Vector{Int64}:
        3
        2
        1
```

Рис. 7: Повторение примеров_7



```
[105]: Alu.L

[105]: 3×3 Matrix{Float64}:
        1.0        0.0       0.0
        0.341647   1.0       0.0
        0.0907733  0.125667  1.0

[107]: Alu.U

[107]: 3×3 Matrix{Float64}:
        0.797321  0.330762  0.469158
        0.0       0.646829  0.109972
        0.0       0.0       0.47346

[109]: A\b

[109]: 3-element Vector{Float64}:
        0.9999999999999997
        1.0000000000000004
        0.9999999999999999

[111]: Alu\b

[111]: 3-element Vector{Float64}:
        0.9999999999999997
        1.0000000000000004
        0.9999999999999999

[113]: det(A)

[113]: -0.24417782969302415

[115]: det(Alu)

[115]: -0.24417782969302415

[117]: Aqr = qr(A)

[117]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
       Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
       R factor:
       3×3 Matrix{Float64}:
        -0.845672  -0.566129  -0.574735
         0.0       -0.615335  -0.152983
         0.0        0.0       -0.469237

[119]: Aqr.Q

[119]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
```

Рис. 8: Повторение примеров_8



```
[121]: Aqr.R

[121]: 3×3 Matrix{Float64}:
        -0.845672  -0.566129  -0.574735
         0.0       -0.615335  -0.152983
         0.0        0.0       -0.469237

[123]: Aqr.Q'*Aqr.Q

[123]: 3×3 Matrix{Float64}:
        1.0  0.0          1.11022e-16
        0.0  1.0          1.66533e-16
        0.0  1.66533e-16  1.0

[125]: Asym = A + A'

[125]: 3×3 Matrix{Float64}:
        0.144751  0.383712  1.32719
        0.383712  1.51967   0.601021
        1.32719   0.601021  0.938316

[127]: AsymEig = eigen(Asym)

[127]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
       values:
       3-element Vector{Float64}:
        -0.8449014057029958
         0.984356965497647
         2.463277062502394
       vectors:
       3×3 Matrix{Float64}:
        -0.798174    0.377669  0.469344
        -0.0234827  -0.798002  0.602197
         0.601969    0.469637  0.645813

[129]: AsymEig.values

[129]: 3-element Vector{Float64}:
        -0.8449014057029958
         0.984356965497647
         2.463277062502394

[131]: AsymEig.vectors

[131]: 3×3 Matrix{Float64}:
        -0.798174    0.377669  0.469344
        -0.0234827  -0.798002  0.602197
         0.601969    0.469637  0.645813

[133]: inv(AsymEig)*Asym

[133]: 3×3 Matrix{Float64}:
         1.0           1.22125e-15  -3.66374e-15
        -7.21645e-16   1.0           6.93889e-16
        -1.40166e-15  -3.71231e-16   1.0
```

Рис. 9: Повторение примеров_9

```
[135]: n = 1000
       A = randn(n,n)

[135]: 1000×1000 Matrix{Float64}:
  0.818062  -1.31053    0.197059   -0.687266  …  -0.75303    0.619482
 -0.79846   -0.476185   0.223528    1.7031       -1.74524   -0.129694
  1.19669   -0.333333  -1.01427     1.17748      -0.11606    0.137839
  0.642393   1.06849   -0.768377   -1.37397       1.98862    0.589615
  0.643972   0.231627  -0.12132    -0.747265     -0.291886   1.43754
 -0.484867  -0.069728  -0.477       0.490485  …   0.218898  -0.655217
  0.178721   0.886854  -0.240787   -0.31513       0.685642  -1.27405
  1.03945    0.93114    0.11651    -1.89176       0.283703  -0.27624
  0.153836  -0.506654   0.0803646  -0.721281      0.550615  -0.00312386
  0.25755    1.03278   -0.0337994   1.85153      -1.5547     1.35997
 -0.623845   0.251447  -1.59541     0.347122  …   1.45416    0.180682
 -0.154458   1.6175    -0.936875   -1.05089       0.0616891  1.18805
 -1.08119    1.54088    0.170194   -0.295595     -0.261444   0.170179
  ⋮                                           ⋱
  1.08306   -0.267086  -0.916292    0.139997     -0.604159   0.456711
 -1.29808   -1.03677    1.65929    -0.495838     -0.768464   1.10486
 -1.05473    0.176077  -0.25693     0.673892  …   0.533003   0.790909
 -0.766099   0.989135  -0.934655    0.613167      0.271493   0.924831
  0.555442  -0.820835  -0.163331    1.22256      -1.76078    0.989671
  0.281597  -2.92211    1.99519     0.385542     -2.60477   -0.105077
  0.317395   0.283298  -0.355746    0.582111      0.519174   0.513723
 -0.519221   0.0403315  0.0680095   0.136166  …   0.611306   0.650782
  0.667242   0.820936   0.798518    1.0457       -0.77417   -0.349142
  0.727814  -0.411155  -1.11169     1.42106      -1.4594    -0.458001
  1.33996   -1.09815   -0.709763   -0.889041     -1.54782   -0.315028
  0.361632  -0.878243  -1.06891    -1.30454       0.742708   1.30256

[137]: Asym = A + A'

[137]: 1000×1000 Matrix{Float64}:
  1.63612    -2.10899    1.39375   …  -0.891124    0.586926   0.981113
 -2.10899    -0.95237   -0.109805      0.878565   -2.84339   -1.00794
  1.39375    -0.109805  -2.02853      -0.502246   -0.825824  -0.931067
 -0.0448727   2.77159    0.409099      1.3049      1.09957   -0.714926
  0.239746    0.779642   1.08499      -1.84741     0.82332    0.992099
  1.18116     0.734446  -1.17177   …   1.71244     0.016373   0.586781
 -0.200147   -0.296418  -0.706104     -1.36245     1.1645    -1.37789
  2.06549     1.87127    0.402672      1.56407    -0.577685   0.430895
  0.209      -0.701503  -0.517886      1.30498     0.840221  -0.895598
  1.10362     0.221068   1.50526      -0.204233   -2.76873    0.955965
 -1.1257     -0.264242  -1.04453   …   0.412988    1.86593    0.0513228
 -0.925704    3.96027   -1.83935      -0.57001     0.101924   1.60281
 -0.982017    0.91361   -0.386693      1.27554    -0.405095   0.869273
  ⋮                                 ⋱
  1.14622    -2.88828   -2.08458       1.00203    -2.31005   -3.29735
 -1.36476    -1.36348    3.68086       0.907519    0.274666   0.16838
  1.00911    -0.177245  -1.59013   …   0.729634    0.386251   0.96611
 -1.9119      0.905417  -0.312312     -1.13246     1.01813    2.1639
  0.170478   -0.207089   0.786903      2.46693    -1.47897   -0.641391
 -1.02793    -3.02517    2.68489       0.184872   -1.83073   -0.615466
 -0.379141    1.39551   -1.34908       1.33808     0.577196   0.155574
 -0.145907    1.52608    0.138885  …   0.502388    0.97309    0.608095
  1.30025     3.01674    0.463508      0.0689384  -1.88534   -0.972294
 -0.891124    0.878565  -0.502246     -0.993957   -2.0127    -0.377036
  0.586926   -2.84339   -0.825824     -2.0127     -3.09564    0.427679
```

Рис. 10: Повторение примеров_10

```
[139]: issymmetric(Asym)

[139]: true

[141]: Asym_noisy = copy(Asym)
       Asym_noisy[1,2] += 5eps()

[141]: -2.108994287183597

[143]: issymmetric(Asym_noisy)

[143]: false

[146]: Asym_explicit = Symmetric(Asym_noisy)

[146]: 1000×1000 Symmetric{Float64, Matrix{Float64}}:
  1.63612    -2.10899    1.39375   …  -0.891124    0.586926   0.981113
 -2.10899    -0.95237   -0.109805      0.878565   -2.84339   -1.00794
  1.39375    -0.109805  -2.02853      -0.502246   -0.825824  -0.931067
 -0.0448727   2.77159    0.409099      1.3049      1.09957   -0.714926
  0.239746    0.779642   1.08499      -1.84741     0.82332    0.992099
  1.18116     0.734446  -1.17177   …   1.71244     0.016373   0.586781
 -0.200147   -0.296418  -0.706104     -1.36245     1.1645    -1.37789
  2.06549     1.87127    0.402672      1.56407    -0.577685   0.430895
  0.209      -0.701503  -0.517886      1.30498     0.840221  -0.895598
  1.10362     0.221068   1.50526      -0.204233   -2.76873    0.955965
 -1.1257     -0.264242  -1.04453   …   0.412988    1.86593    0.0513228
 -0.925704    3.96027   -1.83935      -0.57001     0.101924   1.60281
 -0.982017    0.91361   -0.386693      1.27554    -0.405095   0.869273
  ⋮                                 ⋱
  1.14622    -2.88828   -2.08458       1.00203    -2.31005   -3.29735
 -1.36476    -1.36348    3.68086       0.907519    0.274666   0.16838
  1.00911    -0.177245  -1.59013   …   0.729634    0.386251   0.96611
 -1.9119      0.905417  -0.312312     -1.13246     1.01813    2.1639
  0.170478   -0.207089   0.786903      2.46693    -1.47897   -0.641391
 -1.02793    -3.02517    2.68489       0.184872   -1.83073   -0.615466
 -0.379141    1.39551   -1.34908       1.33808     0.577196   0.155574
 -0.145907    1.52608    0.138885  …   0.502388    0.97309    0.608095
  1.30025     3.01674    0.463508      0.0689384  -1.88534   -0.972294
 -0.891124    0.878565  -0.502246     -0.993957   -2.0127    -0.377036
  0.586926   -2.84339   -0.825824     -2.0127     -3.09564    0.427679
  0.981113   -1.00794   -0.931067     -0.377036    0.427679   2.60511
```

Рис. 11: Повторение примеров_11

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

   Resolving package versions...
   Installed BenchmarkTools ─ v1.5.0
    Updating `C:\Users\user\.julia\environments\v1.11\Project.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
    Updating `C:\Users\user\.julia\environments\v1.11\Manifest.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
  [9abbd945] + Profile v1.11.0
Precompiling project...
   2000.1 ms  ✓ BenchmarkTools
  1 dependency successfully precompiled in 4 seconds. 54 already precompiled.

[152]: using BenchmarkTools
       @btime eigvals(Asym);

  53.050 ms (21 allocations: 7.99 MiB)

[154]: @btime eigvals(Asym_noisy);

  445.836 ms (27 allocations: 7.93 MiB)

[155]: @btime eigvals(Asym_explicit);

  49.995 ms (21 allocations: 7.99 MiB)

[157]: n = 1000000;
       A = SymTridiagonal(randn(n), randn(n-1))

[157]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
 -1.17194   0.752829   ⋅         …  ⋅           ⋅          ⋅
  0.752829  1.07153   0.23193       ⋅           ⋅          ⋅
   ⋅        0.23193   1.41457       ⋅           ⋅          ⋅
   ⋅         ⋅       -0.582606      ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅         …  ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅         …  ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
  ⋮                              ⋱
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅         …  ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅            ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅         …  ⋅           ⋅          ⋅
   ⋅         ⋅         ⋅             1.99383    ⋅          ⋅
   ⋅         ⋅         ⋅           -0.45        0.00395637  ⋅
   ⋅         ⋅         ⋅             0.00395637 -0.309151  -1.08358
   ⋅         ⋅         ⋅            ⋅          -1.08358    0.153842
```

Рис. 12: Повторение примеров_12

```
[158]: @btime eigmax(A)

  523.621 ms (44 allocations: 183.11 MiB)

[158]: 6.535517052271077

[162]: B = Matrix(A)

OutOfMemoryError()

Stacktrace:
 [1] GenericMemory
   @ .\boot.jl:516 [inlined]
 [2] new_as_memoryref
   @ .\boot.jl:535 [inlined]
 [3] Array
   @ .\boot.jl:582 [inlined]
 [4] Matrix{Float64}(M::SymTridiagonal{Float64, Vector{Float64}})
   @ LinearAlgebra C:\Users\user\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\tridiag.jl:136
 [5] (Matrix)(M::SymTridiagonal{Float64, Vector{Float64}})
   @ LinearAlgebra C:\Users\user\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\tridiag.jl:147
 [6] top-level scope
   @ In[162]:1
```

Рис. 13: Повторение примеров_13

```
[164]: Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10

[164]: 3×3 Matrix{Rational{BigInt}}:
        3//5   1//10   1
        1//5   3//10  3//10
        9//10  3//10  3//10

[166]: x = fill(1, 3)

[166]: 3-element Vector{Int64}:
        1
        1
        1

[168]: b = Arational*x

[168]: 3-element Vector{Rational{BigInt}}:
        17//10
         4//5
         3//2

[170]: Arational\b

[170]: 3-element Vector{Rational{BigInt}}:
        1
        1
        1

[172]: lu(Arational)

[172]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
       L factor:
       3×3 Matrix{Rational{BigInt}}:
         1      0    0
        2//9    1    0
        2//3  -3//7  1
       U factor:
       3×3 Matrix{Rational{BigInt}}:
        9//10  3//10  3//10
         0     7//30  7//30
         0      0     9//10
```

Рис. 14: Повторение примеров_13

Полученные результаты

1. Лабораторная работа позволила изучить базовые операции линейной алгебры, такие как умножение векторов, работа с матрицами, спектральное разложение и решение систем уравнений.
2. Реализация на Julia показала эффективность использования встроенных функций и библиотек для работы с линейной алгеброй.

Заключение

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

Библиографическая справка

[1] Julia: https://ru.wikipedia.org/wiki/Julia