Компьютерный практикум по статистическому анализу данных. Лаб №4

Линейная алгебра

Шаповалова Диана Дмитриевна

5 декабря 2024

Российский университет дружбы народов, Москва, Россия

Вводная часть

Цель работы

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

Выполнение работы

Шаповалова Диана Дмитриевна. 1032211220

Лаб №4. Линейная алгебра ¶

4.2.1. Поэлементные операции над многомерными массивами

[1]:
```
a = rand(1:20,(4,3))
```

[1]:
```
4×3 Matrix{Int64}:
  2  12   5
  5   7  11
 20  12   7
  4  16  10
```

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

[2]:
```
111
```

[4]:
```
# Поэлементная сумма по столбцам:
sum(a,dims=1)
```

[4]:
```
1×3 Matrix{Int64}:
 31  47  33
```

[5]:
```
# Поэлементная сумма по строкам:
sum(a,dims=2)
```

[5]:
```
4×1 Matrix{Int64}:
 19
 23
 39
 30
```

[6]:
```
# Поэлементное произведение:
prod(a)
```

[6]:
```
49674240000
```

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

[15]:
```
import Pkg
Pkg.add("LinearAlgebra")
using LinearAlgebra
```

```
  Resolving package versions...
   Updating `C:\Users\dina7\.julia\environments\v1.10\Project.toml`
  [37e2e46d] + LinearAlgebra
  No Changes to `C:\Users\dina7\.julia\environments\v1.10\Manifest.toml`
```

[16]:
```
b = rand(1:20,(4,4))
```

[16]:
```
4×4 Matrix{Int64}:
  9  19  17  1
  5  12   2  1
  3   6   6  1
 20   7  17  1
```

[17]:
```
# Транспонирование:
transpose(b)
```

[17]:
```
4×4 transpose(::Matrix{Int64}) with eltype Int64:
  9   5  3  20
 19  12  6   7
 17   2  6  17
  1   1  1   1
```

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

[18]:
```
28
```

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

[19]:
```
4-element Vector{Int64}:
  9
 12
  6
  1
```

4.2.3. Вычисление нормы векторов и матриц, повороты, вращения

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

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

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

25]:
```
6.708203932499369
```

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

26]:
```
11.0
```

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

27]:
```
9.486832980505138
```

29]:
```julia
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))
```

29]:
```
9.486832980505138
```

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

30]:
```
2.4404307889469252
```

4.2.4. Матричное умножение, единичная матрица, скалярное произведение

[37]:
```
# Матрица 2x3 со случайными целыми значениями от 1 до 10:
A = rand(1:10,(2,3))
```

[37]:
```
2×3 Matrix{Int64}:
 5   4  6
 5  10  7
```

[38]:
```
# Матрица 3x4 со случайными целыми значениями от 1 до 10:
B = rand(1:10,(3,4))
```

[38]:
```
3×4 Matrix{Int64}:
 8   3  7  8
 1   9  1  1
 9  10  3  1
```

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

[39]:
```
2×4 Matrix{Int64}:
  98  111  57  50
 113  175  66  57
```

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

[40]:
```
3×3 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
```

4.2.5. Факторизация. Специальные матричные структуры

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

[43]:
```
3×3 Matrix{Float64}:
 0.0348329  0.375824   0.747843
 0.240031   0.385661   0.968305
 0.360039   0.0386811  0.778836
```

[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}:
 1.1584997099418903
 1.5939970955428544
 1.177556154750584
```

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

[46]:
```
3-element Vector{Float64}:
 0.9999999999999992
 0.999999999999999
 1.0000000000000004
```

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

[47]:
```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
```

Общая линейная алгебра

4.2.6. Общая линейная алгебра

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

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

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

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

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

```
1]: 3-element Vector{Rational{BigInt}}:
      1
      2
     8//5
```

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

```
2]: 3-element Vector{Rational{BigInt}}:
     1
     1
     1
```

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

```
3]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
```

Задания для самостоятельного выполнения

4.4. Задания для самостоятельного выполнения

4.4.1. Произведение векторов

[84]:
```
### 1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результат в dot_v.
```

[85]:
```
v = [1, 2, 3]
```

[85]:
```
3-element Vector{Int64}:
 1
 2
 3
```

[86]:
```
dot_v= dot(v, v)
```

[86]:
```
14
```

[87]:
```
### 2. Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной outer_v
```

[88]:
```
outer_v = v * transpose(v)
```

[88]:
```
3×3 Matrix{Int64}:
 1  2  3
 2  4  6
 3  6  9
```

1. Решить СЛАУ с двумя неизвестными. Пунтк a)

```
[90]: A = [1 1; 1 -1]

[90]: 2×2 Matrix{Int64}:
       1   1
       1  -1

[91]: B = [2; 3]

[91]: 2-element Vector{Int64}:
       2
       3

[92]: A\B

[92]: 2-element Vector{Float64}:
        2.5
       -0.5
```

Пункт b) (имеет бесконечно много решений, т.к. второе уравнение линейно зависимо от первого, поэтому выводит ошибку)

```
[105]: A = [1 1; 2 2]
       B = [2; 4]

[105]: 2-element Vector{Int64}:
        2
        4

[108]: rez = A\ B

SingularException(2)

Stacktrace:
 [1] checknonsingular
   @ c:\users\dina7\appdata\local\programs\julia-1.10.5\share\julia\stdlib\v1.10\LinearAlgebra\src\factorization.jl:68 [inlined]
 [2] checknonsingular
   @ c:\users\dina7\appdata\local\programs\julia-1.10.5\share\julia\stdlib\v1.10\LinearAlgebra\src\factorization.jl:71 [inlined]
 [3] #lu!#158
   @ c:\users\dina7\appdata\local\programs\julia-1.10.5\share\julia\stdlib\v1.10\LinearAlgebra\src\lu.jl:83 [inlined]
 [4] lu!
   @ c:\users\dina7\appdata\local\programs\julia-1.10.5\share\julia\stdlib\v1.10\LinearAlgebra\src\lu.jl:81 [inlined]
 [5] lu(A::Matrix{Int64}, pivot::RowMaximum; check::Bool)
```

Решить СЛАУ с двумя неизвестными

Пункт d)

[111]:
```
A = [1 1; 2 2; 3 3]
B = [1; 2; 3]
```

[111]:
```
3-element Vector{Int64}:
 1
 2
 3
```

[112]:
```
rez = A\ B
```

[112]:
```
2-element Vector{Float64}:
 0.4999999999999999
 0.5
```

Пункт e)

[113]:
```
A = [1 1; 2 1; 1 -1]
B = [2; 1; 3]
```

[113]:
```
3-element Vector{Int64}:
 2
 1
 3
```

[114]:
```
rez = A\ B
```

[114]:
```
2-element Vector{Float64}:
  1.5000000000000004
 -0.9999999999999997
```

Решить СЛАУ с тремя неизвестными

2. Решить СЛАУ с тремя неизвестными. Пункт а)

[117]:
```
A = [1 1 1; 1 -1 -2;]
B = [2; 3]
```

[117]:
```
2-element Vector{Int64}:
 2
 3
```

[118]:
```
rez = A\ B
```

[118]:
```
3-element Vector{Float64}:
  2.2142857142857144
  0.35714285714285704
 -0.5714285714285712
```

Пункт b)

[119]:
```
A = [1 1 1; 2 2 -3; 3 1 1]
B = [2; 4; 1]
```

[119]:
```
3-element Vector{Int64}:
 2
 4
 1
```

[120]:
```
rez = A\ B
```

[120]:
```
3-element Vector{Float64}:
 -0.5
  2.5
  0.0
```

Пункт c) (система имеет бесконечно много решений)

[121]:
```
A = [1 1 1; 1 1 2; 2 2 3]
B = [1; 0; 1]
```

[121]:
```
3-element Vector{Int64}:
```

4.4.3. Операции с матрицами

1. Приведите приведённые ниже матрицы к диагональному виду. Пункт а)

5]:
```
using LinearAlgebra

# Исходная матрица
A = [1 -2;
     -2 1]

# Собственные значения и собственные векторы
eigen_decomp = eigen(A)

# Собственные значения (диагональная матрица)
D = Diagonal(eigen_decomp.values)

# Матрица собственных векторов
P = eigen_decomp.vectors

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

println("\nДиагональная матрица D:")
println(D)

println("\nМатрица собственных векторов P:")
println(P)
```

```
Исходная матрица A:
[1 -2; -2 1]

Диагональная матрица D:
[-1.0 0.0; 0.0 3.0]

Матрица собственных векторов P:
[-0.7071067811865475 -0.7071067811865475; -0.7071067811865475 0.7071067811865475]
```

Вычислите

2. Вычислите. Пункт a)

29]:
```
using LinearAlgebra

# Исходная матрица
A = [1 -2;
     -2 1]

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

# Возводим диагональную матрицу в 10-ю степень
D_10 = D.^10

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

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

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

Пункт b)

31]:
```
using LinearAlgebra

# Исходная матрица
A = [5 -2;
     -2 5]

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

# Проверяем, что собственные значения неотрицательные
if all(eigenvalues .>= 0)
```

Найдите собственные значения матрицы A. Создайте диагональную матрицу из собственных значений матрицы ?. Создайте нижнедиагональную матрицу из матрица ?. Оцените эффективность выполняемых операций.

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

[140]:

```julia
using LinearAlgebra
using BenchmarkTools  # Для оценки времени выполнения

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

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

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

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

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

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

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

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

```
  8.367 μs (11 allocations: 3.00 KiB)
Собственные значения матрицы A:
[-0.2360679774997897, 4.23606797749979]
```

4.4.4. 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 это определение, проверьте, являются ли матрицы продуктивными.

Пункт a)

[45]:
```julia
using LinearAlgebra

# Исходная матрица A
A = [1 2; 3 4]

# Создаем единичную матрицу I (например, с использованием Matrix{Float64})
I_matrix = I(2)  # Единичная матрица размера 2x2

# Матрица I - A
I_minus_A = I_matrix - A

# Проверяем, является ли матрица I - A невырожденной (обратимой)
det_I_minus_A = det(I_minus_A)
println("Определитель матрицы I - A: ", det_I_minus_A)

# Если определитель ненулевой, то матрица I - A обратима, и система имеет единственное решение
if det_I_minus_A != 0
    println("Матрица I - A обратима, система имеет единственное решение.")
else
    println("Матрица I - A вырождена, система не имеет единственного решения.")
end
```

```
Определитель матрицы I - A: -6.0
Матрица I - A обратима, система имеет единственное решение.
```

Пункт b)

Рис. 14: 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.

Пункт а)

```
using LinearAlgebra

# Исходная матрица A
A = [1 2; 3 1]

# Единичная матрица E размером 2x2
E = I(2)

# Матрица E - A
E_minus_A = E - A

# Проверяем, существует ли обратная матрица
if det(E_minus_A) != 0
    # Находим обратную матрицу (E - A)^-1
    inv_E_minus_A = inv(E_minus_A)

    # Проверяем, являются ли все элементы матрицы обратной матрицы неотрицательными
    if all(x -> x >= 0, inv_E_minus_A)
        println("Матрица A является продуктивной.")
    else
        println("Матрица A не является продуктивной.")
    end
else
    println("Матрица E - A вырождена, не существует обратной матрицы.")
end
```

```
Матрица A не является продуктивной.
```


Рис. 15: 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.

Пункт a)

[52]:
```julia
using LinearAlgebra

# Исходная матрица A
A = [1 2; 3 1]

# Находим собственные значения матрицы A
eigenvalues_A = eigen(A).values

# Проверяем, что все собственные значения по модулю меньше 1
if all(abs(eigenvalue) < 1 for eigenvalue in eigenvalues_A)
    println("Матрица A является продуктивной.")
else
    println("Матрица A не является продуктивной.")
end
```

```
Матрица A не является продуктивной.
```

Пункт b)

[54]:
```julia
using LinearAlgebra

# Исходная матрица A
A = [1 2; 3 1]

# Создаем матрицу (1/2) * A
half_A = 0.5 * A

# Находим собственные значения матрицы (1/2) * A
eigenvalues_half_A = eigen(half_A).values

# Проверяем, что все собственные значения по модулю меньше 1
if all(abs(eigenvalue) < 1 for eigenvalue in eigenvalues_half_A)
    println("Матрица (1/2) * A является продуктивной.")
else
    println("Матрица (1/2) * A не является продуктивной.")
end
```

Выводы

Выводы

Мы изучили возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры