Лабораторная работа #4.

Линейная алгебра

Хохлачева Яна

Содержание

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Поэлементные операции над многомерными массивами

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

...

Операции

```
# Поэлементная сумма:
sum(a)
```

...

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

...

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

...

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

...

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

...


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

...

Для работы со средними значениями можно воспользоваться возможностями пакета Statistics:

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

•••

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

•••

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

•••

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

•••

Рис. 2.2: Использование Statistics

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:

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

•••

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

•••

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

•••

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

•••

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

•••

```
# Ранг матрицы:
rank(b)
```

•••

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

•••

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

•••

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

•••

Рис. 2.3: Операции над матрицами

2.3 Вычисление нормы векторов и матриц, повороты, вращения

Для вычисления нормы используется LinearAlgebra.norm(x).

Евклидова норма:

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

p-норма:

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

```
# Создание вектора X:
X = [2, 4, -5]
•••
# Вычисление евклидовой нормы:
norm(X)
•••
# Вычисление p-нормы:
p = 1
norm(X, p)
•••
•••
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5]
Y = [1, -1, 3]
norm(X-Y)
•••
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))
•••
•••
# Угол между двумя векторами:
acos((transpose(X)*Y)/(norm(X)*norm(Y)))
```

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

Рис. 2.4: Нормы, повороты и вращения

2.4 Матричное умножение, единичная матрица, скалярное произведение

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

•••

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

•••

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

•••

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

•••

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

•••

```
# тоже скалярное произведение:
X'Y
```

•••

Рис. 2.5: Умножение, ед. матрица и скалярное произведение

2.5 Факторизация. Специальные матричные структуры

В математике факторизация (или разложение) объекта — его декомпозиция (например, числа, полинома или матрицы) в произведение других объектов или факторов, которые, будучи перемноженными, дают исходный объект.

Матрица может быть факторизована на произведение матриц специального вида для приложений, в которых эта форма удобна. К специальным видам матриц относят ортогональные, унитарные и треугольные матрицы.

LU-разложение - представление матрицы A в виде произведения двух матриц L и U, L — нижняя треугольная матрица, а U — верхняя треугольная матрица. LU-разложение существует только в том случае, когда матрица A обратима, а все её ведущие (угловые) главные миноры невырождены.

Обращение матрицы A эквивалентно решению линейной системы $AX = I$, где X — неизвестная матрица, I — единичная матрица. Решение X этой системы является обратной матрицей A^{-1}.

LUP-разложение — представление матрицы A в виде произведения $PA = LU$, где матрица L является нижнетреугольной с единицами на главной диагонали, U — верхнетреугольная общего вида матрица, P— матрица перестановок, получаемая из единичной матрицы путём перестановки строк или столбцов.

QR-разложение матрицы — представление матрицы в виде произведения унитарной (или ортогональной) матрицы Q и верхнетреугольной матрицы R. QR-разложение применяется для нахождения собственных векторов и собственных значений матрицы.

Q является ортогональной матрицей, если $Q^TQ = I$, где I — единичная матрица.

Спектральное разложение матрицы A — представление её в виде произведения $A = V\Lambda V^{-1}$, где V— матрица, столбцы которой являются собственными векторами матрицы A, Λ — диагональная матрица с соответствующими собственными значениями на главной диагонали, V^{-1} — матрица, обратная матрице V.

Рассмотрим несколько примеров. Для работы со специальными матричными структурами потребуется пакет LinearAlgebra.

```
# Задаём квадратную матрицу 3x3 со случайными значениями:
A = rand(3, 3)
# Задаём единичный вектор:
x = fill(1.0, 3)
# Задаём вектор b:
b = A*x
```

•••

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

•••

•••

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

•••

Различные части факторизации могут быть извлечены путём доступа к их специальным свойствам:

```
# Матрица перестановок:
Alu.P
```

•••

```
# Вектор перестановок:
Alu.p
```

•••

```
# Матрица L:
Alu.L
```

•••

```
# Матрица U:
Alu.U
```

•••

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

•••

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

•••

•••

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

•••

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

•••

Рис. 2.6: LU-факторизация

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

•••

•••

```
# Матрица Q:
Aqr.Q
```

•••

```
# Матрица R:
Aqr.R
```

•••

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

•••

•••

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

•••

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

•••

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

•••

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

•••

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

•••

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

•••

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

•••

•••

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

•••

```
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)
```

•••

•••

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

•••

Рис. 2.7: QR-факторизация и матрицы большого размера

```
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
```

•••

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

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

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

```
  971.020 ms (13 allocations: 7.92 MiB)
```

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

•••

•••

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

•••

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

•••

•••

```
B = Matrix(A)
```

•••

Рис. 2.8: BenchmarkTools

2.6 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.

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

•••

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

•••

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

•••

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

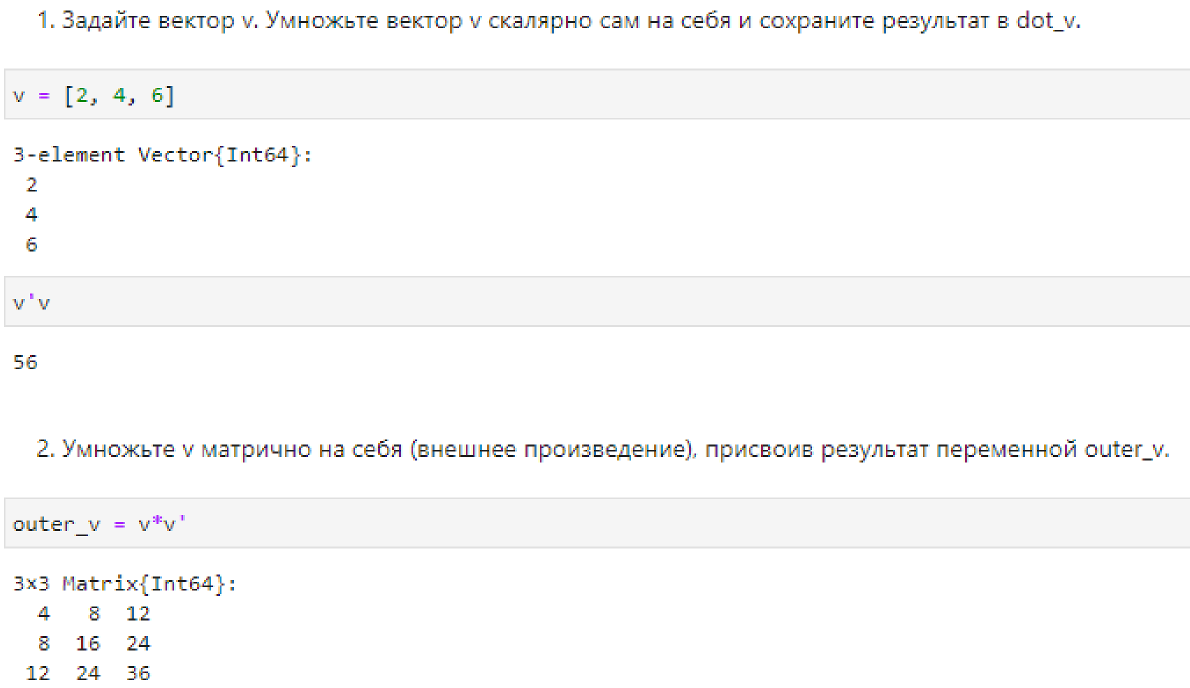
•••

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

•••

Рис. 2.9: Работа с рациональными значениями

3 Задания для самостоятельного выполнения



1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результат в dot_v.

```
v = [2, 4, 6]
```

```
3-element Vector{Int64}:
 2
 4
 6
```

```
v'v
```

```
56
```

2. Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной outer_v.

```
outer_v = v*v'
```

```
3×3 Matrix{Int64}:
  4   8  12
  8  16  24
 12  24  36
```

Рис. 3.1: Задание 1

```
x = [1 1;1 -1]
```

```
2×2 Matrix{Int64}:
 1   1
 1  -1
```

```
y = [2; 3]
```

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

```
res = [1; 1]
res = x\y
```

```
2-element Vector{Float64}:
  2.5
 -0.5
```

b - нет решений

```
x = [1 1;2 2]
```

```
2×2 Matrix{Int64}:
 1  1
 2  2
```

```
y = [2; 4]
```

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

```
res = [1; 1]
res = x\y
```

•••

c - нет решений

```
x = [1 1;2 2]
```

```
2×2 Matrix{Int64}:
 1  1
 2  2
```

```
y = [2; 5]
```

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

```
res = [1; 1]
res = x\y
```

•••

d

```
x = [1 1;2 2; 3 3]
```

```
3×2 Matrix{Int64}:
 1  1
 2  2
 3  3
```

```
y = [1; 2; 3]
```

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

```
res = [1; 1]
res = x\y
```

```
2-element Vector{Float64}:
 0.4999999999999999
 0.5
```

e)

```
x = [1 1;2 1; 1 -1]
y = [2; 1; 3]
res = [1; 1]
res = x\y
```

```
2-element Vector{Float64}:
  1.5000000000000004
 -0.9999999999999997
```

e)

```
x = [1 1;2 1; 1 -1]
y = [2; 1; 3]
res = [1; 1]
res = x\y
```

```
2-element Vector{Float64}:
  1.5000000000000004
 -0.9999999999999997
```

f)

```
x = [1 1; 2 1; 3 2]
y = [2; 1; 3]
res = [1; 1]
res = x\y
```

```
2-element Vector{Float64}:
 -0.9999999999999989
  2.9999999999999982
```

Рис. 3.2: Задание 2 пункт 1

a)

```
x = [1 1 1; 1 -1 -2]
y = [2; 3]
res = [1; 1; 1]
res = x\y
```

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

b)

```
x = [1 1 1; 2 2 -3; 3 1 1]
y = [2; 4; 1]
res = [1; 1; 1]
res = x\y
```

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

c) - нет решений

```
x = [1 1 1; 1 1 2; 2 2 3]
y = [1; 0; 1]
res = [1; 1; 1]
res = x\y
```

•••

d) - нет решений

```
x = [1 1 1; 1 1 2; 2 2 3]
y = [1; 0; 0]
res = [1; 1; 1]
res = x\y
```

•••

Рис. 3.3: Задание 2 пункт 2

a)

```
x = [1 -2; -2 1]
```

```
2×2 Matrix{Int64}:
  1  -2
 -2   1
```

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

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
2-element Vector{Float64}:
 -1.0
  3.0
vectors:
2×2 Matrix{Float64}:
 -0.707107  -0.707107
 -0.707107   0.707107
```

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

```
2-element Vector{Float64}:
 -1.0
  3.0
```

```
Diagonal(AsymEig.values)
```

```
2×2 Diagonal{Float64, Vector{Float64}}:
 -1.0   ⋅
   ⋅   3.0
```

b)

```
y = [1 -2; -2 3]
```

```
2×2 Matrix{Int64}:
  1  -2
 -2   3
```

```
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(y)
# Собственные значения:
AsymEig.values
Diagonal(AsymEig.values)
```

```
2×2 Diagonal{Float64, Vector{Float64}}:
 -0.236068   ⋅
   ⋅        4.23607
```

c)

```
z = [1 -2 0; -2 1 2; 0 2 0]
```

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

```
# Проверка, является ли матрица симметричной:
issymmetric(z)
```

```
true
```

```
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(z)
# Собственные значения:
AsymEig.values
Diagonal(AsymEig.values)
```

```
3×3 Diagonal{Float64, Vector{Float64}}:
 -2.14134   ⋅         ⋅
   ⋅       0.515138   ⋅
   ⋅        ⋅        3.6262
```

Рис. 3.4: Задание 3

a)

```
A = [1 -2; -2 1]
```

```
2×2 Matrix{Int64}:
  1  -2
 -2   1
```

```
A^10
```

```
2×2 Matrix{Int64}:
  29525  -29524
 -29524   29525
```

b)

```
B = [5 -2; -2 5]
```

```
2×2 Matrix{Int64}:
  5  -2
 -2   5
```

```
sqrt(B)
```

```
2×2 Matrix{Float64}:
  2.1889   -0.45685
 -0.45685   2.1889
```

c)

```
C = [1 -2; -2 1]
```

```
2×2 Matrix{Int64}:
  1  -2
 -2   1
```

```
C^(1/3)
```

```
2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
  0.971125+0.433013im  -0.471125+0.433013im
 -0.471125+0.433013im   0.971125+0.433013im
```

d)

```
D = [1 2; 2 3]
```

```
2×2 Matrix{Int64}:
 1  2
 2  3
```

```
sqrt(D)
```

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

Рис. 3.5: Задание 3

```
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]
```

```
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
```

•••

```
# Проверка, является ли матрица симметричной:
issymmetric(A)
```

```
true
```

```
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(A)
# Собственные значения:
AsymEig.values
Diagonal(AsymEig.values)
```

```
5×5 Diagonal{Float64, Vector{Float64}}:
 -128.493     ⋅         ⋅         ⋅         ⋅
     ⋅     -55.8878     ⋅         ⋅         ⋅
     ⋅        ⋅       42.7522     ⋅         ⋅
     ⋅        ⋅         ⋅       87.1611     ⋅
     ⋅        ⋅         ⋅         ⋅      542.468
```

```
lu(A).L
```

```
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
```

```
# Оценка эффективности выполнения операции
@btime AsymEig.values;
```

```
  37.941 ns (1 allocation: 32 bytes)
```

```
# Оценка эффективности выполнения операции
@btime Diagonal(AsymEig.values);
```

```
  202.931 ns (2 allocations: 48 bytes)
```

```
# Оценка эффективности выполнения операции
@btime lu(A).L;
```

```
  764.423 ns (4 allocations: 736 bytes)
```

Рис. 3.6: Задание 3

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. Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части y имеет только неотрицательные элементы $ □_{□}. $ Используя это определение, проверьте, являются ли матрицы продуктивными.

```
A = [1 2 ; 3 4]
y = rand(1:15, 2)
E = Matrix{Int}(I, 2, 2)
A = E - A
x = A\y

decision = "Something"
for i in 1:2
    if x[i] < 0
        decision = "Условие не выполняется"
        break
    else
        decision = "Матрица продуктивная"
    end
end
print(decision)
```

```
Условие не выполняется
```

```
B = (1/2)*[1 2 ; 3 4]
y = rand(1:15, 2)
E = Matrix{Int}(I, 2, 2)
A = E - A
x = A\y

decision = "Something"
for i in 1:2
    if x[i] < 0
        decision = "Условие не выполняется"
        break
    else
        decision = "Матрица продуктивная"
    end
end
print(decision)
```

```
Условие не выполняется
```

```
C = (1/10)*[1 2 ; 3 4]
y = rand(1:15, 2)
E = Matrix{Int}(I, 2, 2)
A = E - A
x = A\y

decision = "Something"
for i in 1:2
    if x[i] < 0
        decision = "Условие не выполняется"
        break
    else
        decision = "Матрица продуктивная"
    end
end
print(decision)
```

```
Условие не выполняется
```

Рис. 3.7: Задание 4.1

4 Вывод

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