Отчет по лабораторной работе 4

Линейная алгебра

Шалыгин Георгий Эдуардович

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 Выполнение лабораторной работы

1. Повторим примеры (fig. 2.7).

```
a = rand(1:20,(4,3))
# Поэлементная сумма:
@show sum(a)
# Поэлементная сумма по столбцам:
@show sum(a,dims=1)
# Поэлементная сумма по строкам:
@show sum(a,dims=2)
# Поэлементное произведение:
@show prod(a)
# Поэлементное произведение по столбцам:
@show prod(a,dims=1)
# Поэлементное произведение по строкам:
@show prod(a,dims=2)
```

```
sum(a) = 133
sum(a, dims = 1) = [48 47 38]
sum(a, dims = 2) = [45; 23; 42; 23;;]
prod(a) = 518192640000
prod(a, dims = 1) = [18900 8160 3360]
prod(a, dims = 2) = [3332; 180; 2400; 360;;]
```

Рис. 2.1: Поэлементные операции над многомерными массивами

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

```
    Updating registry at `D:\julia\depot\registries\General.toml`
   Resolving package versions...
    Updating `D:\julia\depot\environments\v1.8\Project.toml`
  [10745b16] + Statistics
  No Changes to `D:\julia\depot\environments\v1.8\Manifest.toml`
```

```
# Вычисление среднего значения массива:
@show mean(a)
# Среднее по столбцам:
@show mean(a,dims=1)
# Среднее по строкам:
@show mean(a,dims=2)
```

```
mean(a) = 11.083333333333334
mean(a, dims = 1) = [12.0 11.75 9.5]
mean(a, dims = 2) = [15.0; 7.666666666666667; 14.0; 7.66666666666666
7;;]
```

Рис. 2.2: Работа со средними значениями

```
# Массив 4x4 со случайными целыми числами (от 1 до 20):
@show b = rand(1:20,(4,4))
# Транспонирование:
@show transpose(b)
# След матрицы (сумма диагональных элементов):
@show tr(b)
# Извлечение диагональных элементов как массив:
@show diag(b)
# Ранг матрицы:
@show rank(b)
# Инверсия матрицы (определение обратной матрицы):
@show inv(b)
# Определитель матрицы:
@show det(b)
# Псевдобратная функция для прямоугольных матриц:
@show pinv(a)
```

```
b = rand(1:20, (4, 4)) = [16 11 15 6; 8 19 7 18; 13 2 2 15; 20 11 9 14]
transpose(b) = [16 8 13 20; 11 19 2 11; 15 7 2 9; 6 18 15 14]
tr(b) = 51
diag(b) = [16, 19, 2, 14]
rank(b) = 4
inv(b) = [-0.06991215226939972 -0.05209858467545145 -0.07076622742801368 0.17276720351390926; -0.08272327964860907 0.0317227916
0566129 -0.1395802830649097 0.14421669106881405; 0.18301610541727673 0.016837481698389473 0.1259150805270864 -0.234992679355783
33; 0.0472181551976574 0.03867740361151782 0.12981942410932162 -0.1376281112737921]
det(b) = 16392.0
pinv(a) = [0.0008722142328345202 0.07408928069981337 -0.026000493545614924 0.01516163618880728; 0.002310304041117368 0.00356331
4243896793 0.06283771709625009 -0.07935234902184397; 0.027917237506062943 -0.07250333994230612 -0.022449943412442372 0.10235646
75749038]
```

Рис. 2.3: Транспонирование, след, ранг, определитель и инверсия матрицы

```julia
# Создание вектора X:
X = [2, 4, -5]
# Вычисление евклидовой нормы:
@show norm(X)
# Вычисление p-нормы:
p = 1
@show norm(X,p)
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
@show norm(X-Y)
# Проверка по базовому определению:
@show sqrt(sum((X-Y).^2))
```

```
norm(X) = 6.708203932499369
norm(X, p) = 11.0
norm(X - Y) = 9.486832980505138
sqrt(sum((X - Y) .^ 2)) = 9.486832980505138
```

Рис. 2.4: Вычисление нормы векторов и матриц

```julia
# Угол между двумя векторами:
@show acos((transpose(X)*Y)/(norm(X)*norm(Y)))
# Вычисление нормы для двумерной матрицы:
# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
# Вычисление Евклидовой нормы:
@show opnorm(d)
# Вычисление p-нормы:
p=1
@show opnorm(d,p)
# Поворот на 180 градусов:
@show rot180(d)
# Переворачивание строк:
@show reverse(d,dims=1)
# Переворачивание столбцов
@show reverse(d,dims=2)
```

```
acos((transpose(X) * Y) / (norm(X) * norm(Y))) = 2.4404307889469252
opnorm(d) = 7.147682841795258
opnorm(d, p) = 8.0
rot180(d) = [0 1 -2; 3 2 -1; 2 -4 5]
reverse(d, dims = 1) = [-2 1 0; -1 2 3; 5 -4 2]
reverse(d, dims = 2) = [2 -4 5; 3 2 -1; 0 1 -2]
```

Рис. 2.5: Углы, повороты, вращения

```
# Матрица 2х3 со случайными целыми значениями от 1 до 10:
@show A = rand(1:10,(2,3))
# Матрица 3х4 со случайными целыми значениями от 1 до 10:
@show B = rand(1:10,(3,4))
# Произведение матриц А и В:
@show A*B
# Единичная матрица 3х3:
@show Matrix{Int}(I, 3, 3)
# Скалярное произведение векторов X и Y:
X = [2, 4, -5]
Y = [1,-1,3]
@show dot(X,Y)
# тоже скалярное произведение:
@show X'Y
```

```
A = rand(1:10, (2, 3)) = [3 1 4; 3 6 6]
B = rand(1:10, (3, 4)) = [9 6 7 7; 10 2 1 10; 10 6 7 3]
A * B = [77 44 50 43; 147 66 69 99]
Matrix{Int}(I, 3, 3) = [1 0 0; 0 1 0; 0 0 1]
dot(X, Y) = -17
X' * Y = -17
```

Рис. 2.6: Матричное умножение, единичная матрица, скалярное произведение

```
# Задаём квадратную матрицу 3х3 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):
@show A\b
```

```
A \ b = [1.0000000000000007, 0.9999999999999993, 1.0]
```

Рис. 2.7: Решение линейного уравнения

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

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
 1.0        0.0       0.0
 0.800261   1.0       0.0
 0.586608  -0.542824  1.0
U factor:
3×3 Matrix{Float64}:
 0.979145   0.991517   0.3451
 0.0       -0.17578   -0.137436
 0.0        0.0        0.242259
```

Рис. 2.8: LU-факторизация

```
# Матрица перестановок:
@show Alu.P
# Вектор перестановок:
@show Alu.p
# Матрица L:
@show Alu.L
# Матрица U:
@show Alu.U
```

```
Alu.P = [0.0 1.0 0.0; 0.0 0.0 1.0; 1.0 0.0 0.0]
Alu.p = [2, 3, 1]
Alu.L = [1.0 0.0 0.0; 0.8002613627032392 1.0 0.0; 0.5866081989227033
-0.5428244340727237 1.0]
Alu.U = [0.9791451756422701 0.9915173982962635 0.3451004516615477;
0.0 -0.17578046013727233 -0.1374360353122751; 0.0 0.0 0.242258871769
32254]
```

Рис. 2.9: Элементы LU-факторизации

```
# Решение СЛАУ через матрицу A:
@show A\b
# Решение СЛАУ через объект факторизации:
@show Alu\b
# Детерминант матрицы A:
@show det(A)
# Детерминант матрицы A через объект факторизации:
@show det(Alu)
```

```
A \ b = [1.0000000000000007, 0.9999999999999993, 1.0]
Alu \ b = [1.0000000000000007, 0.9999999999999993, 1.0]
det(A) = -0.04169628627108656
det(Alu) = -0.04169628627108656
```

Рис. 2.10: Решение СЛАУ через разложение

```
# QR-факторизация:
Aqr = qr(A)
# Матрица Q:
@show Aqr.Q
# Матрица R:
@show Aqr.R
# Проверка, что матрица Q - ортогональная:
@show Aqr.Q'*Aqr.Q
```

```
Aqr.Q = [-0.4164084903496892 0.6314481315362941 0.6541232501172973;
-0.7098579445606401 0.2237331655078416 -0.6678661311939573; -0.56807
18860398182 -0.7424397132637532 0.35507408305873267]
Aqr.R = [-1.3793536906152442 -1.3366597416793222 -0.540024934569548
3; 0.0 0.19075786727438956 0.30212021045366017; 0.0 0.0 0.1584671605
7149896]
(Aqr.Q)' * Aqr.Q = [1.0000000000000002 1.1102230246251565e-16 1.1102
230246251565e-16; 1.1102230246251565e-16 1.0000000000000004 3.330669
0738754696e-16; 1.1102230246251565e-16 2.220446049250313e-16 1.00000
00000000002]
```

Рис. 2.11: QR-факторизация

```
# Симметризация матрицы A:
@show Asym = A + A'
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
# Собственные значения:
@show AsymEig.values
#Собственные векторы:
@show AsymEig.vectors
# Проверяем, что получится единичная матрица:
@show inv(AsymEig)*Asym
```

```
Asym = A + A' = [1.1487491759347321 1.6561953396524238 1.30287331679
92845; 1.6561953396524238 1.983034796592527 0.9627930558288136; 1.30
28733167992845 0.9627930558288136 0.27746904480779677]
AsymEig.values = [-0.686257491378484, 0.1495517696505394, 3.94595873
9062997]
AsymEig.vectors = [-0.6519294365477903 0.4671431102150852 -0.5972983
545440733; 0.13538681317539314 -0.7033369209262074 -0.69784495876967
78; 0.7460954500469037 0.5358119915050601 -0.3952810255738182]
inv(AsymEig) * Asym = [1.0000000000000067 7.105427357601002e-15 3.33
06690738754696e-15; -8.43769498715119e-15 0.9999999999999907 -4.4408
92098500626e-15; 7.105427357601002e-15 7.105427357601002e-15 1.00000
00000000044]
```

Рис. 2.12: Примеры собственной декомпозиции матрицы ☒

```
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:
Asym = A + A'
# Проверка, является ли матрица симметричной:
@show issymmetric(Asym)
# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
# Проверка, является ли матрица симметричной:
@show issymmetric(Asym_noisy)
# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
@show issymmetric(Asym_explicit)
```

```
issymmetric(Asym) = true
issymmetric(Asym_noisy) = false
issymmetric(Asym_explicit) = true
```

Рис. 2.13: Примеры работы с матрицами большой размерности и специальной структуры

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

  206.682 ms (11 allocations: 7.99 MiB)
  1.221 s (13 allocations: 7.92 MiB)
  191.125 ms (11 allocations: 7.99 MiB)
```

Рис. 2.14: Воспользуемся пакетом BenchmarkTools

```
# 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)

  442.892 ms (17 allocations: 183.11 MiB)

6.60809048391199

B = Matrix(A)

OutOfMemoryError()
```

Рис. 2.15: Использование типов Tridiagonal и SymTridiagonal для хранения трёхдиагональных матриц

```
# Матрица с рациональными элементами:
@show 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):
@show Arational\b
# LU-разложение:
@show lu(Arational)

Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3)) / 10 = Ration
al{BigInt}[2//5 1//5 3//5; 4//5 1//1 7//10; 2//5 3//10 3//5]
Arational \ b = Rational{BigInt}[1//1, 1//1, 1//1]
lu(Arational) = LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vecto
r{Int64}}(Rational{BigInt}[4//5 1//1 7//10; 1//2 -3//10 1//4; 1//2
2//3 1//12], [2, 2, 3], 0)
```

Рис. 2.16: Символьное решение СЛАУ с рациональными коэффициентами

2.1 Задания для самостоятельного выполнения

1. Задайте вектор v. Умножьте вектор v скалярно сам на себя и сохраните результат в dot_v. Умножьте v матрично на себя (внешнее произведение), присвоив результат переменной outer_v.

```
v = [1, 2, 3]
@show dot_v = dot(v, v)
@show outer_v = cross(v, v)
```

```
dot_v = dot(v, v) = 14
outer_v = cross(v, v) = [0, 0, 0]
```

Рис. 2.17: Умножение векторов

2. Решить СЛАУ с двумя и тремя неизвестными.

```
function slau(A, b)
    return pinv(A) *  b
end
```

```
slau (generic function with 1 method)
```

```
@show slau([1 1; 1 -1], [2 3]')
@show slau([1 1; 2 2], [2 4]')
@show slau([1 1; 2 2], [2 5]')
@show slau([1 1; 2 2; 3 3], [1 2 3]')
@show slau([1 1; 2 1; 1 -1], [2 1 3]')
@show slau([1 1; 2 1; 3 2], [2 1 3]')
```

```
slau([1 1; 1 -1], ([2 3])') = [2.499999999999999; -0.5;;]
slau([1 1; 2 2], ([2 4])') = [0.9999999999999996; 0.9999999999999996;;]
slau([1 1; 2 2], ([2 5])') = [1.1999999999999995; 1.1999999999999995;;]
slau([1 1; 2 2; 3 3], ([1 2 3])') = [0.4999999999999999; 0.5;;]
slau([1 1; 2 1; 1 -1], ([2 1 3])') = [1.5000000000000004; -0.9999999999999998;;]
slau([1 1; 2 1; 3 2], ([2 1 3])') = [-0.9999999999999987; 2.9999999999999982;;]
```

Рис. 2.18: Решение СЛАУ с 2-мя неизвестными

```
@show slau([1 1 1; 1 -1 -2], [2 3]')
@show slau([1 1 1; 2 2 -3; 3 1 1], [2 4 1]')
@show slau([1 1 1; 1 1 2; 2 2 3], [1 0 1]')
@show slau([1 1 1; 1 1 2; 2 2 3], [1 0 0]')
```

```
slau([1 1 1; 1 -1 -2], ([2 3])') = [2.2142857142857144; 0.35714285714285676; -0.5714285714285716;;]
slau([1 1 1; 2 2 -3; 3 1 1], ([2 4 1])') = [-0.5000000000000007; 2.4999999999999996; 5.273559366969494e-16;;]
slau([1 1 1; 1 1 2; 2 2 3], ([1 0 1])') = [0.9999999999999994; 1.0000000000000016; -1.0000000000000007;;]
slau([1 1 1; 1 1 2; 2 2 3], ([1 0 0])') = [0.8333333333333326; 0.8333333333333345; -1.0000000000000002;;]
```

Рис. 2.19: Решение СЛАУ с 3-мя неизвестными

3. Приведите матрицы к диагональному виду.

```
@show diag([1 -2; -2 1])
@show diag([1 -2; -2 3])
@show diag([1 -2 0; -2 1 2; 0 2 0])
```

```
diag([1 -2; -2 1]) = [1, 1]
diag([1 -2; -2 3]) = [1, 3]
diag([1 -2 0; -2 1 2; 0 2 0]) = [1, 1, 0]
```

Рис. 2.20: Диагонализация

4. Вычислите степени от матриц.

```
using LinearAlgebra
@show [1 -2; -2 1]^10
@show sqrt([5 -2; -2 5])
@show [5 -2; -2 5]^(1/3)
@show [1 2; 2 3]^(1/2)
```

```
[1 -2; -2 1] ^ 10 = [29525 -29524; -29524 29525]
sqrt([5 -2; -2 5]) = [2.1889010593167337 -0.45685025174785665; -0.45
685025174785665 2.1889010593167337]
[5 -2; -2 5] ^ (1 / 3) = [1.6775903765398983 -0.23534080623249043; -
0.23534080623249043 1.6775903765398983]
[1 2; 2 3] ^ (1 / 2) = ComplexF64[0.5688644810057828 + 0.35157758425
414287im 0.9204420652599258 - 0.2172868967516401im; 0.92044206525992
58 - 0.2172868967516401im 1.489306546265709 + 0.1342906875025027im]
```

Рис. 2.21: Матрицы в степени

5. Найдите собственные значения матрицы A.

```
eigen([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])
```

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
5-element Vector{Float64}:
 -128.49322764802145
  -55.887784553057
   42.752167279318854
   87.16111477514494
  542.4677301466137
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
```

Рис. 2.22: Собственные значения и векторы

6. 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.

```
using SymPy
@syms y1 y2
y = [y1; y2]
A1 = [1 2; 3 4]
@show (Matrix{Int}(I, 2, 2) - A1) \ y
A2 = [1 2; 3 4]/2
@show (Matrix{Int}(I, 2, 2) - A2) \ y
A3 = [1 2; 3 4]/10
@show (Matrix{Int}(I, 2, 2) - A3) \ y
```

```
(Matrix{Int}(I, 2, 2) - A1) \ y = Sym{PyCall.PyObject}[0.5*y1 - 0.333333333333333*y2, -0.5*y1]
(Matrix{Int}(I, 2, 2) - A2) \ y = Sym{PyCall.PyObject}[0.5*y1 - 0.5*y2, -0.75*y1 - 0.25*y2]
(Matrix{Int}(I, 2, 2) - A3) \ y = Sym{PyCall.PyObject}[1.25*y1 + 0.416666666666667*y2, 0.625*y1 + 1.875*y2]

2-element Vector{Sym{PyCall.PyObject}}:
 1.25*y1 + 0.416666666666667*y2
            0.625*y1 + 1.875*y2
```

```
y = [1 1]'
@show (Matrix{Int}(I, 2, 2) - A1) \ y
@show (Matrix{Int}(I, 2, 2) - A2) \ y
```

```
(Matrix{Int}(I, 2, 2) - A1) \ y = [0.16666666666666666; -0.5;;]
(Matrix{Int}(I, 2, 2) - A2) \ y = [-0.0; -1.0;;]
```

Рис. 2.23: Проверка продуктивности

Решим уравнения символьно с помощью SymPy. Для двух первых матриц видим, что существуют отрицательные решения. Ниже приведены примеры такие контрпримеры. Для третьей матрицы отрицательных решений нет.

7. Критерий продуктивности: матрица ⊠ является продуктивной тогда и только тогда, когда все элементы матрица $(E - A)^{-1}$ являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными.

```
@show inv((Matrix{Int}(I, 2, 2) - A1))
@show inv((Matrix{Int}(I, 2, 2) - A2))
@show inv((Matrix{Int}(I, 2, 2) - A3))
```

```
inv(Matrix{Int}(I, 2, 2) - A1) = [0.5 -0.3333333333333333; -0.5 0.0]
inv(Matrix{Int}(I, 2, 2) - A2) = [0.5 -0.5; -0.75 -0.25]
inv(Matrix{Int}(I, 2, 2) - A3) = [1.25 0.4166666666666667; 0.625 1.875]
```

Рис. 2.24: Проверка с помощью критерия

Критерием подтверждаются полученные выше результаты: продуктивная только третья матрица.

8. 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.

```
@show eigen(A1)
@show eigen(A2)
@show eigen(A3)
@show eigen([0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3])
```

```
eigen(A1) = Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}([-0.3722813232690143, 5.372281323269014], [-0.82456484013
23938 -0.4159735579192842; 0.5657674649689923 -0.9093767091321241])
eigen(A2) = Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}([-0.18614066163450715, 2.686140661634507], [-0.8245648401
323938 -0.4159735579192842; 0.5657674649689923 -0.9093767091321241])
eigen(A3) = Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}([-0.03722813232690142, 0.5372281323269015], [-0.824564840
1323938 -0.4159735579192843; 0.5657674649689923 -0.9093767091321241])
eigen([0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]) = Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}([0.0267949192431228, 0.
1, 0.3732050807568877 4], [0.756568226232575 1.0 -0.796750730318148; -0.6140722619430444 0.0 -0.35695904753438806; 0.22476604763
052643 0.0 -0.48761512704267107])
```

Рис. 2.25: Спектральный критерий

Выводы опять подтвердились, также видим, что четвертая матрица продуктивной является.

3 Выводы

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

Список литературы