

Лабораторная работа № 4

Линейная алгебра

Сунгурова Мариян

07 декабря 2024

Российский университет дружбы народов, Москва, Россия

Информация

- Сунгурова Мариян Мухсиновна
- НКНбд-01-21
- Российский университет дружбы народов

Вводная часть

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

1. Используя JupyterLab, повторите примеры.
2. Выполните задания для самостоятельной работы.

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia.

Выполнение лабораторной работы

Выполнение лабораторной работы

Выполним примеры из раздела про поэлементные операции над многомерными массивами (рис. (fig:001?)-(fig:002?)).

Поэлементные операции над многомерными массивами

```
a = rand(1:20,(4, 3))  
# 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)  
  
[2] ✓ 0.4s  
... 4×1 Matrix{Int64}:  
2800  
2448  
1960  
216  
  
# Подключение пакета Statistics:  
import Pkg  
Pkg.add("Statistics")  
using Statistics  
  
[3] ✓ 46.1s  
... Updating registry at `C:\Users\HP\.julia\registries\General.toml`  
Resolving package versions...  
No Changes to `C:\Users\HP\.julia\environments\v1.10\Project.toml`  
No Changes to `C:\Users\HP\.julia\environments\v1.10\Manifest.toml`
```

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

Выполнение лабораторной работы

Выполним примеры из раздела про транспонирование, след, ранг, определитель и инверсия матрицы (рис. (fig:003?)).

```
Транспонирование, след, ранг, определитель и инверсия матрицы

# 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)

# Псевдообратная функция для прямоугольных матриц:
pinv(a)

[5] ✓ 14.4s

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

3x4 Matrix{Float64}:
-0.0322146  -0.00484066  0.110415  -0.0682695
 0.0192571  -0.247257   0.31043   -0.0439828
 0.0297387   0.179488  -0.264614  0.0882101
```

Выполнение лабораторной работы

Выполним примеры из раздела про вычисление нормы векторов и матриц, повороты и вращения (рис. (fig:003?) - (fig:004?)).

```
Вычисление нормы векторов и матриц, повороты, вращения

% Создание вектора X:
X = [2, 4, -5];
% Вычисление евклидовой нормы:
norm(X)
% Вычисление p-нормы:
p = 1;
norm(X,p)

01 ✓ ok
-- 11.0

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

01 ✓ ok
-- 9.40612090645338

% Проверка по базису определена:
norm(norm(X-Y), '2')
01 ✓ ok
-- 9.40612090645338

% Угол между двумя векторами:
angle((1/norm(X))*X, (1/norm(Y))*Y)
01 ✓ ok
-- 2.4404387800468252

% Вычисление нормы для диагональной матрицы:
% Создание матрицы:
A = [5 -4 2 ; -1 2 3 ; -2 3 0];
01 ✓ ok
-- 3x3 Matrix [1x364]:
5 -4 2
-1 2 3
-2 3 0
```

Рис. 3: Нормы векторов и матриц, повороты и вращения

Выполнение лабораторной работы

Выполним примеры из раздела про матричное умножение, единичная матрица, скалярное произведение (рис. (fig:005?)).

```
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
0.0s

2x4 Matrix{Int64}:
203 193 138 141
111  88 100  70

# 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
0.3s

-17
```

Рис. 5: Матричное умножение, единичная матрица, скалярное произведение

Выполнение лабораторной работы

Выполним примеры из раздела про факторизацию и специальные матричные структуры (рис. (fig:006?)-(fig:008?)).

Факторизация. Специальные матричные структуры

```
# 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
```

[19] ✓ 24s

```
... 3-element Vector{Float64}:
 1.0000000000000002
 0.9999999999999988
 1.0000000000000013
```

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

[20] ✓ 0.5s

```
... LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3x3 Matrix{Float64}:
 1.0      0.0      0.0
 0.275261 1.0      0.0
 0.422045 0.394016 1.0
U factor:
3x3 Matrix{Float64}:
 0.900552 0.219409 0.102875
 0.0      0.524966 0.448109
 0.0      0.0      0.0439755
```

```
# Матрица перестановок:
```

```
Alu.P
```

```
# Вектор перестановок:
```

```
Alu.p
```

Выполнение лабораторной работы

```
# Решение СЛАУ через матрицу A:  
A\b  
# Решение СЛАУ через объект факторизации:  
A\b  
✓ 0.0s  
  
3-element Vector{Float64}:  
 1.0000000000000002  
 0.9999999999999988  
 1.0000000000000013  
  
# Детерминант матрицы A:  
det(A)  
# Детерминант матрицы A через объект факторизации:  
det(A\b)  
✓ 0.0s  
  
0.020789842464535  
  
# QR-факторизация:  
Aqr = qr(A)  
✓ 0.4s  
  
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}  
Q factor: 3x3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}  
R factor:  
3x3 Matrix{Float64}:  
-1.00841  -0.452695  -0.308471  
 0.0      -0.524902  -0.458847  
 0.0      0.0       0.0392766  
  
# Матрица Q:  
Aqr.Q  
# Матрица R:  
Aqr.R  
# Проверка, что матрица Q - ортогональная:  
Aqr.Q'*Aqr.Q  
✓ 0.2s  
  
3x3 Matrix{Float64}:  
 1.0      0.0      0.0  
 8.32667e-17  1.0     -1.11022e-16  
-5.55112e-17 -5.55112e-17  1.0
```

Выполнение лабораторной работы

```
# Симметризация матрицы A:
Asym = A + A'
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
# Собственные значения:
AsymEig.values
# Собственные векторы:
AsymEig.vectors
# Проверим, что получится единичная матрица:
Inv(AsymEig)*Asym
✓ 2.6s

3x3 Matrix(Float64):
 1.0      4.61853e-14  -8.88178e-14
 1.06581e-13  1.0      -4.9738e-13
-1.56319e-13 -4.26326e-13  1.0

# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:
Asym = A + A'
# Проверка, является ли матрица симметричной:
issymmetric(Asym)
✓ 0.3s

true

# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)
✓ 0.0s

false

# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
✓ 0.5s

1000x1000 Symmetric{Float64, Matrix{Float64}}:
 2.28795      0.0103949  0.544118  - 0.658061  -1.23449  -1.69088
 0.0103949  -1.71543  0.52566  -0.725694  0.679579  -2.03899
 0.544118   0.52566  1.17008  -1.15947  0.760688  -2.17183
 0.0820233  0.336604  0.948103  1.72966  1.85208  0.0995547
 0.143169   0.640678  0.30084  -0.111429  -1.88651  -1.42945
-0.134493   2.18698  -1.10928  -1.98653  0.707359  3.77757
 1.58568    1.49646  0.773682  1.97325  -0.710145  -2.9542
```

Выполнение лабораторной работы

Далее для оценки эффективности выполнения операций над матрицами большой размерности и специальной структуры воспользуемся пакетом BenchmarkTools

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

# 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);

✓ 2m 56.7s

Resolving package versions...
Installed BenchmarkTools - v1.5.0
Updating `C:\Users\HP\.julia\environments\v1.10\Project.toml`
[6e4b80f9] + BenchmarkTools v1.5.0
Updating `C:\Users\HP\.julia\environments\v1.10\Manifest.toml`
[6e4b80f9] ↑ BenchmarkTools v1.4.0 ⇒ v1.5.0
Precompiling project...
✓ BenchmarkTools
✓ MathOptInterface
✓ Optim
✓ DiffEqNoiseProcess
✓ StochasticDiffEq
✓ DifferentialEquations
6 dependencies successfully precompiled in 128 seconds. 311 already precompiled.
193.655 ms (11 allocations: 7.99 MiB)
706.772 ms (14 allocations: 7.93 MiB)
194.802 ms (11 allocations: 7.99 MiB)

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


Выполним примеры из раздела про общую линейную алгебру (рис. (fig:010?)).

```
^ Общая линейная алгебра

# 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)

[32] ✓ 6.7s

... LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3×3 Matrix{Rational{BigInt}}:
 1  0  0
2//7 1  0
 1 7//34 1
U factor:
3×3 Matrix{Rational{BigInt}}:
7//10 1//10 1//2
 0 34//35 11//70
 0  0 23//340
```

Задания для самостоятельного выполнения

Зададим вектор v . Умножим вектор v скалярно сам на себя и сохраним результат в dot_v . Затем умножим v матрично на себя (внешнее произведение), присвоив результат переменной outer_v . (рис. (fig:011?)).

Задания для самостоятельного выполнения

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

```
v = [1, 2, 3, 4]
dot_v = v*v
```

[33] ✓ 0.0s

...

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

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

[34] ✓ 0.1s

...

```
4x4 Matrix{Int64}:
 1  2  3  4
 2  4  6  8
 3  6  9  12
 4  8  12 16
```

Решить СЛАУ с двумя неизвестными.

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

Задания для самостоятельного выполнения

Решим СЛАУ с двумя неизвестными (рис. (fig:012?)-(fig:013?)).

```
A = [ 1 1; 2 2 ]
b = [ 2 ; 4 ]

if det(A)~=0
    println("Нет решений")
else
    println(A\b)
end
✓ 0.0s
Нет решений

A = [ 1 1; 2 2 ]
b = [ 2 ; 5 ]

if det(A)~=0
    println("Нет решений")
else
    println(A\b)
end
✓ 0.0s
Нет решений

A = [ 1 1; 2 2; 3 3 ]
b = [ 1 ; 2; 3 ]

println(A\b)
✓ 0.0s
[0.4999999999999999, 0.5]

A = [ 1 1; 2 1; 1 -1 ]
b = [ 2; 1; 3 ]

println(A\b)
✓ 0.0s
[1.5000000000000004, -0.9999999999999997]
```

Рис. 12: Системы линейных уравнений

Задания для самостоятельного выполнения

Решим СЛАУ с тремя неизвестными (рис. (fig:014?)).

```
A = [ 1 1; 2 1; 3 2 ]  
b = [ 2; 1; 3 ]  
  
println(A\b)  
✓ 0.0s  
[-0.99999999999999989, 2.99999999999999982]
```

Решить СЛАУ с тремя неизвестными.

```
A = [ 1 1 1; 1 -1 -2 ]  
b = [ 2; 3 ]  
  
println(A\b)  
✓ 0.0s  
[2.2142857142857144, 0.35714285714285704, -0.5714285714285712]
```

```
A = [ 1 1 1; 2 2 -3; 3 1 1 ]  
b = [ 2; 4; 1 ]  
  
println(A\b)  
✓ 0.0s  
[-0.5, 2.5, 0.0]
```

```
A = [ 1 1 1; 1 1 2; 2 2 3 ]  
b = [ 1; 0; 1 ]  
if det(A) == 0  
    println("Нет решений")  
else  
    println(A\b)  
end  
✓ 0.0s  
Нет решений
```

```
A = [ 1 1 1; 1 1 2; 2 2 3 ]  
b = [ 1; 0; 0 ]
```

Задания для самостоятельного выполнения

Приведем матрицы к диагональному виду (рис. (fig:015?)).

```
Операции с матрицами. Приведите приведенные ниже матрицы к диагональному виду

A = [ 1 -2 ; -2 1 ]
Aeig = eigen(A)
A_d = diag(Aeig.values)
display(A_d)

A = [ 1 -2 ; -2 3 ]
Aeig = eigen(A)
A_d = diag(Aeig.values)
display(A_d)

A = [ 1 -2 0; -2 1 2; 0 2 0 ]
Aeig = eigen(A)
A_d = diag(Aeig.values)
display(A_d)

In [ ]: ✓ 0/0
-- 2x2 Matrix(Float64):
--1.0  0.0
--0.0  3.0

-- 2x2 Matrix(Float64):
--0.236068  0.0
--0.0      4.23607

-- 3x3 Matrix(Float64):
--2.14134  0.0  0.0
--0.0     0.515138  0.0
--0.0     0.0      3.6262

Вычислите

display([ 1 -2; -2 1]^10)

display([ 5 -2; -2 5]^0.5)

display([ 1 -2; -2 1]^(1/3))

display([ 1 2; 2 3]^0.5)

In [ ]: ✓ 0/0
-- 2x2 Matrix(Int64):
--29525  -29524
--29524  29525
```

Задания для самостоятельного выполнения

Вычислим (рис. (fig:016?)).

Операции с матрицами. Приведите приведенные ниже матрицы к диагональному

```
A = [ 1 -2 ; -2 1 ]
Aeig = eigen(A)
A_d = diag(Aeig.values)
display(A_d)

A = [ 1 -2 ; -2 3 ]
Aeig = eigen(A)
A_d = diag(Aeig.values)
display(A_d)

A = [ 1 -2 0; -2 1 2; 0 2 0 ]
Aeig = eigen(A)
A_d = diag(Aeig.values)
display(A_d)
```

2x2 Matrix(Float64):
-1.0 0.0
0.0 3.0

2x2 Matrix(Float64):
-0.236068 0.0
0.0 4.23607

3x3 Matrix(Float64):
-2.14134 0.0 0.0
0.0 0.515138 0.0
0.0 0.0 3.6262

Вычислите

```
display([ 1 -2; -2 1]^10)
display([ 5 -2; -2 5]^0.5)
display([ 1 -2; -2 1]^(1/3))
display([ 1 2; 2 3]^0.5)
```

2x2 Matrix{Int64}:
29525 -29524
-29524 29525

Задания для самостоятельного выполнения

Найдем собственные значения матрицы A. Создадим диагональную матрицу из собственных значений матрицы A. Создадим нижнедиагональную матрицу из матрицы A. Оценим эффективность выполняемых операций (рис. (fig:017?)).

```
Найдите собственные значения матрицы 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]
✓ 0.0s

5x5 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

Aeig = eigen(A)
display(diag(Aeig.values))
✓ 0.0s

5x5 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

LowerTriangular(A)
✓ 0.3s

5x5 LowerTriangular{Int64, Matrix{Int64}}:
140  - - - -
 97 106 - - -
 74  89 152 - -
168 131 144 54 -
131  36  71 142 36
```

Линейная модель 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 x_i . Используя это определение, проверим, являются ли матрицы продуктивными (рис. (fig:018?)-(fig:019?)).

Задания для самостоятельного выполнения

```
Линейные модели экономики

A = [1 2; 3 4]
B = (1/2)*[1 2; 3 4]
C = (1/10)*[1 2; 3 4]
D = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
E = Matrix{I, 2, 2}

7) ✓ 0.0s
2x2 Matrix{Bool}:
 1  0
 0  1

inv(E-A) #не продуктивна
8) ✓ 0.0s
2x2 Matrix{Float64}:
 0.5 -0.333333
-0.5  0.0

inv(E-B) #не продуктивна
9) ✓ 0.2s
2x2 Matrix{Float64}:
 0.5 -0.5
-0.75 -0.25

inv(E-C) #продуктивна
10) ✓ 0.0s
2x2 Matrix{Float64}:
 1.25  0.416667
 0.625 1.875

Aeig = eigen(A)
abs.(Aeig.values).c1 #не продуктивна
11) ✓ 0.0s
2-element BitVector:
 1
 0
```

Рис. 18: Линейные модели экономики

Выводы

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