Лабораторная работа №4

Леснухин Даниил Дмитриевич Российский университет дружбы народов Москва

Цель работы

Изучение возможностей специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.

Задание

1. Используя JupyterLab, повторить примеры
2. Выполнить задания для самостоятельной работы

Теоретическое введение

Julia — высокоуровневый язык с динамической типизацией, предназначенный для математических вычислений.

Синтаксис языка близок к другим математическим языкам, но имеет отличия.

Использовалась официальная документация Julia.

Примеры: поэлементные операции над массивами

Поэлементные операции над многомерными массивами

[1]

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

```
4×3 Matrix{Int64}:
 20  20   8
 15   2   8
 11  12  15
 18   6  19
```

[2]

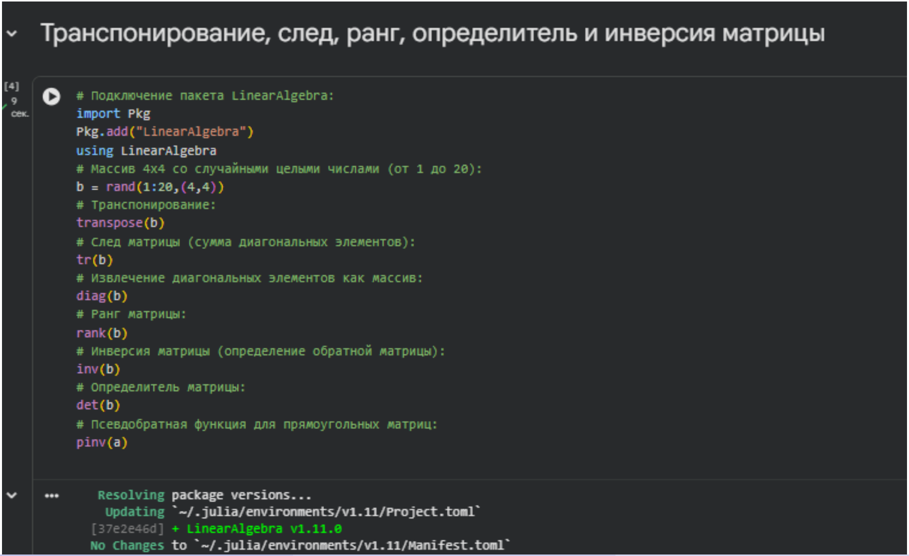
```
# Поэлементная сумма:
sum(a)
# Поэлементная сумма по столбцам:
sum(a,dims=1)
# Поэлементная сумма по строкам:
sum(a,dims=2)
# Поэлементное произведение:
prod(a)
# Поэлементное произведение по столбцам:
prod(a,dims=1)
# Поэлементное произведение по строкам:
prod(a,dims=2)
```

```
4×1 Matrix{Int64}:
 3200
  240
 1980
 2052
```

[3]

```
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics
# Вычисление среднего значения массива:
mean(a)
# Среднее по столбцам:
```

Примеры: транспонирование, след, ранг, определитель, инверсия




```julia
# 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)
```

```
   Resolving package versions...
    Updating `~/.julia/environments/v1.11/Project.toml`
  [37e2e46d] + LinearAlgebra v1.11.0
  No Changes to `~/.julia/environments/v1.11/Manifest.toml`
```

Примеры: нормы, повороты, вращения

Вычисление нормы векторов и матриц, повороты, вращения

[5]

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

```
11.0
```

[6]

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

```
9.486832980505138
```

[7]

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

```
9.486832980505138
```

[9]

```
# 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)
#
```

Примеры: матричное умножение, единичная матрица, скалярное произведение

Матричное умножение, единичная матрица, скалярное произведение

[10]
0 сек.

```
# 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
```

```
-17
```

Примеры: факторизация и специальные матричные структуры

Факторизация. Специальные матричные структуры

[11]
2 сек.

```
# 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
```

```
3-element Vector{Float64}:
 0.9999999999999992
 1.0000000000000009
 1.0
```

[13]
0 сек.

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

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
 1.0        0.0       0.0
 0.630634   1.0       0.0
 0.802078  -0.663093  1.0
U factor:
3×3 Matrix{Float64}:
 0.572395   0.565877  0.232299
 0.0       -0.170278  0.550369
 0.0        0.0       0.208105
```

Задание №1: Произведение векторов

- Задаём вектор v
- Скалярное произведение: dot_v
- Внешнее произведение: outer_v

```
1

[25]
v = [1, 2, 5, 1]

4-element Vector{Int64}:
 1
 2
 5
 1

[26]
dot_v = dot(v, v)

31

[28]
v'v
```

Задание №2: Системы линейных уравнений

- Решение СЛАУ с двумя неизвестными

2

```
A1 = [1 1; 1 -1]
```

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

```
Alu1 = lu(A1)
```

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
2×2 Matrix{Float64}:
 1.0  0.0
 1.0  1.0
U factor:
2×2 Matrix{Float64}:
 1.0   1.0
 0.0  -2.0
```

```
b1 = [2, 3]
```

Задание №3: Операции с матрицами

- Приведение к диагональному виду

```
[47]
0 сек.

A = [1 -2; 2 -1]

2×2 Matrix{Int64}:
 1  -2
 2  -1
```

```
[51]
0 сек.

AsymEig = eigen(A)

Eigen{ComplexF64, ComplexF64, Matrix{ComplexF64}, Vector{ComplexF64}}
values:
2-element Vector{ComplexF64}:
 -1.1102230246251565e-16 - 1.7320508075688772im
 -1.1102230246251565e-16 + 1.7320508075688772im
vectors:
2×2 Matrix{ComplexF64}:
 -0.707107-0.0im       -0.707107+0.0im
 -0.353553-0.612372im  -0.353553+0.612372im
```

```
[53]
0 сек.

display(diagm(AsymEig.values))

2×2 Matrix{ComplexF64}:
 -1.11022e-16-1.73205im           0.0+0.0im
          0.0+0.0im      -1.11022e-16+1.73205im
```

```
[54]
0 сек.

B = [1 -2; -2 3]
Beig = eigen(B)
```

Задание №4: Линейные модели экономики

- Линейная модель: (x - Ax = y)
- Проверка продуктивности матриц
- Критерий: элементы ((E-A)^{-1}) ≥ 0

Выводы

В ходе работы:

- Изучены специализированные пакеты Julia
- Выполнены операции линейной алгебры и матричные вычисления
- Оценена эффективность операций с объектами линейной алгебры