Лабораторная работа №4

Линейная алгебра

Ким Реачна[1]

28 ноября, 2023, Москва, Россия

[1]Российский Университет Дружбы Народов

Цели и задачи

Цель лабораторной работы

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

Задание

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

Процесс выполнения лабораторной работы

Поэлементные операции над многомерными массивами

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

# Поэлементная сумма:
println("Поэлементная сумма: ", sum(a))
# Поэлементная сумма по столбцам:
println("Поэлементная сумма по столбцам: ", sum(a,dims=1))
# Поэлементная сумма по строкам:
println("Поэлементная сумма по строкам: ", sum(a,dims=2))

# Поэлементное произведение:
println("Поэлементное произведение: ", prod(a))
# Поэлементное произведение по столбцам:
println("Поэлементное произведение по столбцам: ", prod(a,dims=1))
# Поэлементное произведение по строкам:
println("Поэлементное произведение по строкам: ", prod(a,dims=2))
```

```
Поэлементная сумма: 167
Поэлементная сумма по столбцам: [58 60 49]
Поэлементная сумма по строкам: [38; 42; 41; 46;;]
Поэлементное произведение: 28477329408000
Поэлементное произведение по столбцам: [36720 45900 16896]
Поэлементное произведение по строкам: [1920; 2448; 1800; 3366;;]
```

```
[2]: # Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
```

```
   Updating registry at `C:\Users\Reachna\.julia\registries\General.toml`
  Resolving package versions...
   Updating `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
 [10745b16] + Statistics v1.9.0
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
```

```
[3]: using Statistics
# Вычисление среднего значения массива:
println("Среднего значения массива: ", mean(a))
# Среднее по столбцам:
println("Среднее по столбцам: ", mean(a,dims=1))
# Среднее по строкам:
println("Среднее по строкам: ", mean(a,dims=2))
```

```
Среднего значения массива: 13.916666666666666
Среднее по столбцам: [14.5 15.0 12.25]
Среднее по строкам: [12.666666666666666; 14.0; 13.666666666666666; 15.333333333333334;;]
```

Рис. 1: Примеры с поэлементными операциями над многомерными массивами

Транспонирование, след, ранг, определитель и инверсия матрицы

```
[4]: # Подключение пакета LinearAlgebra:
     import Pkg
     Pkg.add("LinearAlgebra")
```

```
   Resolving package versions...
    Updating `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
  [37e2e46d] + LinearAlgebra
  No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
```

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

```
[5]: 4×4 Matrix{Int64}:
      17   8  1  16
       8  15  8  17
       2  15  5  20
       6  19  4   3
```

```
[6]: # Транспонирование:
     println("Транспонирование: ", transpose(b))
     # След матрицы (сумма диагональных элементов):
     println("След матрицы (сумма диагональных элементов): ", tr(b))
     # Извлечение диагональных элементов как массив:
     println("Извлечение диагональных элементов как массив: ", diag(b))
     # Ранг матрицы:
     println("Ранг матрицы: ", rank(b))
```

```
Транспонирование: [17 8 2 6; 8 15 15 19; 1 8 5 4; 16 17 20 3]
След матрицы (сумма диагональных элементов): 40
Извлечение диагональных элементов как массив: [17, 15, 5, 3]
Ранг матрицы: 4
```

```
[7]: # Инверсия матрицы (определение обратной матрицы):
     println("Инверсия матрицы: ")
     inv(b)
```

```
Инверсия матрицы:
```

```
[7]: 4×4 Matrix{Float64}:
       0.0488384     0.032779   -0.0680218   0.00725977
      -0.000791975  -0.0590901   0.0407427   0.0674498
      -0.0815734     0.247052   -0.136836   -0.0526663
       0.0161035    -0.0207233   0.0604541  -0.0381468
```

```
[8]: # Определитель матрицы:
     println("Определитель матрицы: ", det(b))
```

```
Определитель матрицы: 22728.000000000004
```

```
[9]: # Псевдобратная функция для прямоугольных матриц:
     println("Псевдобратная функция для прямоугольных матриц: ")
     pinv(a)
```

```
Псевдобратная функция для прямоугольных матриц:
```

```
[9]: 3×4 Matrix{Float64}:
       0.0599931  -0.0750363    0.0543877  -0.00778499
      -0.113035    0.08467     -0.0099743   0.0466985
       0.0879067   0.00491167  -0.0325707  -0.0263336
```

Рис. 2: Примеры с транспонированием, трассировкой, ранжированием, определителем и матричной инверсией

Вычисление нормы векторов и матриц, повороты, вращения

[10]:
```
# Создание вектора X:
X = [2, 4, -5]
# Вычисление евклидовой нормы:
println("Евклидовой нормы: ", norm(X))
# Вычисление p-нормы:
p = 1
println("p-нормы: ", norm(X,p))
```

```
Евклидовой нормы: 6.708203932499369
p-нормы: 11.0
```

[11]:
```
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
println("Расстояние между двумя векторами X и Y: ", norm(X-Y))
# Проверка по базовому определению:
println("Расстояние по базовому определению: ", sqrt(sum((X-Y).^2)))
```

```
Расстояние между двумя векторами X и Y: 9.486832980505138
Расстояние по базовому определению: 9.486832980505138
```

[12]:
```
# Угол между двумя векторами:
println("Угол между двумя векторами: ", acos((transpose(X)*Y)/(norm(X)*norm(Y))))
```

```
Угол между двумя векторами: 2.4404307889469252
```

[13]:
```
# Создание матрицы:
d = [5 -4 2 ; -1 2 3; -2 1 0]
```

[13]:
```
3×3 Matrix{Int64}:
  5  -4  2
 -1   2  3
 -2   1  0
```

[14]:
```
# Вычисление Евклидовой нормы:
println("Евклидовой нормы: ", opnorm(d))
# Вычисление p-нормы:
p=1
println("p-нормы: ", opnorm(d,p))
```

```
Евклидовой нормы: 7.147682841795258
p-нормы: 8.0
```

[15]:
```
# Поворот на 180 градусов:
println("Поворот на 180 градусов", rot180(d))
# Переворачивание строк:
println("Переворачивание строк: ", reverse(d,dims=1))
# Переворачивание столбцов
println("Переворачивание столбцов: ", reverse(d,dims=2))
```

```
Поворот на 180 градусов[0 1 -2; 3 2 -1; 2 -4 5]
Переворачивание строк: [-2 1 0; -1 2 3; 5 -4 2]
Переворачивание столбцов: [2 -4 5; 3 2 -1; 0 1 -2]
```

Рис. 3: Примеры с вычислением нормы векторов и матриц, повороты, вращения

Матричное умножение, единичная матрица, скалярное произведение

```
[17]: # 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

[17]: 2×4 Matrix{Int64}:
       73  37  18   66
       87  60  31  117

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

[18]: 3×3 Matrix{Int64}:
       1  0  0
       0  1  0
       0  0  1

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

[19]: -17

[20]: # тоже скалярное произведение:
      X'Y

[20]: -17
```

Рис. 4: Примеры с матричным умножением, единичной матрицей, скалярным произведением

Факторизация. Специальные матричные структуры

```
[21]: # 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

[21]: 3-element Vector{Float64}:
       0.9999999999999998
       1.0
       1.0

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

[22]: LU{Float64, Matrix{Float64}, Vector{Int64}}
      L factor:
      3×3 Matrix{Float64}:
       1.0       0.0        0.0
       0.615141  1.0        0.0
       0.321768  0.0235118  1.0
      U factor:
      3×3 Matrix{Float64}:
       0.909254  0.606996  0.188041
       0.0       0.57954   0.63612
       0.0       0.0       0.861268

[23]: # Матрица перестановок:
      Alu.P

[23]: 3×3 Matrix{Float64}:
       0.0  1.0  0.0
       0.0  0.0  1.0
       1.0  0.0  0.0

[24]: # Вектор перестановок:
      Alu.p

[24]: 3-element Vector{Int64}:
       2
       3
       1

[25]: # Матрица L:
      Alu.L

[25]: 3×3 Matrix{Float64}:
       1.0       0.0        0.0
       0.615141  1.0        0.0
       0.321768  0.0235118  1.0
```

Рис. 5: Примеры с LU-факторизацией

Факторизация. Специальные матричные структуры

```
[26]: # Матрица U:
      Alu.U

[26]: 3×3 Matrix{Float64}:
       0.909254  0.606996  0.188041
       0.0       0.57954   0.63612
       0.0       0.0       0.861268

[27]: # Решение СЛАУ через матрицу A:
      A\b

[27]: 3-element Vector{Float64}:
       0.9999999999999998
       1.0
       1.0

[28]: # Решение СЛАУ через объект факторизации:
      Alu\b

[28]: 3-element Vector{Float64}:
       0.9999999999999998
       1.0
       1.0

[29]: # Детерминант матрицы A:
      det(A)

[29]: 0.45384413687575187

[30]: # Детерминант матрицы A через объект факторизации:
      det(Alu)

[30]: 0.45384413687575187

[31]: # QR-факторизация:
      Aqr = qr(A)

[31]: LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
      Q factor:
      3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
       -0.264319  -0.12994   -0.955642
       -0.821459  -0.488837   0.293673
       -0.505313   0.862644   0.0224688
      R factor:
      3×3 Matrix{Float64}:
       -1.10688  -1.03538   -0.781953
        0.0       0.498166   0.434889
        0.0       0.0       -0.823064
```

Рис. 6: QR-факторизация

Факторизация. Специальные матричные структуры

```
[32]: # Матрица Q:
      Aqr.Q

[32]: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
       -0.264319  -0.12994   -0.955642
       -0.821459  -0.488837   0.293673
       -0.505313   0.862644   0.0224688

[33]: # Матрица R:
      Aqr.R

[33]: 3×3 Matrix{Float64}:
       -1.10688  -1.03538   -0.781953
        0.0       0.498166   0.434889
        0.0       0.0       -0.823064

[34]: # Проверка, что матрица Q - ортогональная:
      Aqr.Q'*Aqr.Q

[34]: 3×3 Matrix{Float64}:
        1.0  -1.66533e-16   0.0
        0.0   1.0          -4.44089e-16
        0.0  -3.33067e-16   1.0

[35]: # Симметризация матрицы A:
      Asym = A + A'

[35]: 3×3 Matrix{Float64}:
       0.585137  1.11819  1.49605
       1.11819   1.21399  1.14097
       1.49605   1.14097  1.50358

[36]: # Спектральное разложение симметризованной матрицы:
      AsymEig = eigen(Asym)

[36]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
      values:
      3-element Vector{Float64}:
       -0.5696462945958927
        0.2280379804830429
        3.6443212274452446
      vectors:
      3×3 Matrix{Float64}:
        0.844286   0.127328  -0.520546
       -0.215406  -0.808803  -0.54721
       -0.490694   0.574131  -0.655434

[37]: # Собственные значения:
      AsymEig.values

[37]: 3-element Vector{Float64}:
       -0.5696462945958927
        0.2280379804830429
        3.6443212274452446
```

Рис. 7: Симметризация матрицы, Спектральное разложение

Задания для самостоятельного выполнения

```
[79]: # Task3
using BenchmarkTools
A = [140 97 74 168 131; 97 106 89 131 36; 74 89 152 144 7; 168 131 144 54 142; 131 36 71 142 36]

eigen_value = eigvals(A)
```

```
[79]: 5-element Vector{Float64}:
 -133.26347908274784
  -46.31722914150222
   34.7455206405183
  100.90643294309879
  531.9287546406343
```

```
[80]: @btime eigvals(A);
```

```
  4.157 μs (12 allocations: 2.38 KiB)
```

```
[81]: B = zeros(5, 5)
@btime for i in 1:5
    B[i, i] = eigen_value[i]
end
B
```

```
  227.292 ns (5 allocations: 80 bytes)
[81]: 5×5 Matrix{Float64}:
 -133.263    0.0      0.0       0.0      0.0
    0.0    -46.3172   0.0       0.0      0.0
    0.0      0.0     34.7455    0.0      0.0
    0.0      0.0      0.0     100.906    0.0
    0.0      0.0      0.0       0.0    531.929
```

```
[82]: Alu = lu(A)
@btime Alu.L
```

```
  110.108 ns (1 allocation: 256 bytes)
[82]: 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
```

Рис. 8: Задания для самостоятельного выполнения

Выводы по проделанной работе

Вывод

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