Отчёт по лабораторной работе №4

Линейная алгебра

Гань Чжаолун

Цель работы

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

Выполнение лабораторной работы

4.2.1. Поэлементные операции над многомерными массивами

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

```
4×3 Matrix{Int64}:
  3  15  15
  8  15   9
 12  11  17
 11  19  20
```

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

```
Поэлементная сумма = 155
Поэлементная сумма по столбцам = [34 60 61]
Поэлементная сумма по строкам = [33; 32; 40; 50;;]
```

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

```
Поэлементное произведение = 6837961680000
Поэлементное произведение по столбцам = [3168 47025 45900]
Поэлементное произведение по строкам = [675; 1080; 2244; 4180;;]
```

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

```
    Updating registry at `C:\Users\GanZL\.julia\registries\General.toml`
   Resolving package versions...
    Updating `C:\Users\GanZL\.julia\environments\v1.11\Project.toml`
  [10745b16] + Statistics v1.11.1
  No Changes to `C:\Users\GanZL\.julia\environments\v1.11\Manifest.toml`
```

```
println("Вычисление среднего значения массива = ", mean(a))
println("Среднее по столбцам = ", mean(a, dims=1))
println("Среднее по строкам = ", mean(a, dims=2))
```

```
Вычисление среднего значения массива = 12.916666666666666
Среднее по столбцам = [8.5 15.0 15.25]
Среднее по строкам = [11.0; 10.666666666666666; 13.333333333333334; 16.666666666666668;;]
```

Рисунок 0.1. Поэлементные операции над многомерными массивами

4.2.2. Транспонирование, след, ранг, определитель и инверсия матрицы

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

```
   Resolving package versions...
    Updating `C:\Users\GanZL\.julia\environments\v1.11\Project.toml`
  [37e2e46d] + LinearAlgebra v1.11.0
  No Changes to `C:\Users\GanZL\.julia\environments\v1.11\Manifest.toml`
```

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

```
4×4 Matrix{Int64}:
 13  17  17  11
 12  20  16  17
 19   9  10  17
 13  12  13   3
```

```
println("Транспонированная матрица = ", transpose(b))
println("След матрицы (сумма диагональных элементов) = ", tr(b))
println("Извлечение диагональных элементов как массив = ", diag(b))
```

```
Транспонированная матрица = [13 12 19 13; 17 20 9 12; 17 16 10 13; 11 17 17 3]
След матрицы (сумма диагональных элементов) = 46
Извлечение диагональных элементов как массив = [13, 20, 10, 3]
```

```
println("Ранг матрицы = ", rank(b))
```

```
Ранг матрицы = 4
```

```
println("Инверсия матрицы = ")
inv(b)
```

```
Инверсия матрицы =
4×4 Matrix{Float64}:
 -0.19359    0.0504938   0.0475126     0.154462
 -0.432458   0.296628   -0.0603689     0.246879
  0.566983  -0.318241   -0.000558971  -0.272405
  0.111794  -0.0262717   0.0380101    -0.143097
```

```
println("Определитель матрицы = ", det(b))
```

```
Определитель матрицы = 5366.999999999999
```

```
println("Псевдобратная функция для прямоугольных матриц = ")
pinv(a)
```

```
Псевдобратная функция для прямоугольных матриц =
3×4 Matrix{Float64}:
 -0.118411    0.0665795   0.0661416  0.00262701
  0.0184153   0.0948568  -0.0726917  0.00529096
  0.0623009  -0.115529    0.0488468  0.0137428
```

Рисунок 0.2. Транспонирование, след, ранг, определитель и инверсия матрицы

4.2.3. Вычисление нормы векторов и матриц, повороты, вращения

```
# Создание вектора X
X = [2, 4, -5]
```

```
3-element Vector{Int64}:
  2
  4
 -5
```

```
println("Евклидова норма вектора X = ", norm(X))
```

```
Евклидова норма вектора X = 6.708203932499369
```

```
p = 1
println("p-норма вектора X = ", norm(X, p))
```

```
p-норма вектора X = 11.0
```

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

```
Расстояние между векторами X и Y = 9.486832980505138
Расстояние по базовому определению(проверка) = 9.486832980505138
```

```
println("Угол между векторами X и Y = ", acos((transpose(X)*Y)/(norm(X)*norm(Y))))
```

```
Угол между векторами X и Y = 2.4404307889469252
```

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

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

```
println("Евклидовая норма матрицы d = ", opnorm(d))
p = 1
println("p-норма матрицы d = ", opnorm(d, p))
```

```
Евклидовая норма матрицы d = 7.147682841795258
p-норма матрицы d = 8.0
```

```
println("Поворот на 180 градусов матрицы d = ", rot180(d))
println("Переворачивание строк матрицы d = ", reverse(d,dims=1))
println("Переворачивание столбцов матрицы d = ", reverse(d,dims=2))
```

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

Рисунок 0.3. Вычисление нормы векторов и матриц, повороты, вращения

4.2.4. 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))
```

```
3×4 Matrix{Int64}:
 7   6  10  1
 3   5   2  4
 7  10   3  7
```

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

```
2×4 Matrix{Int64}:
 107  156  60  111
  63   91  39   64
```

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

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

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

```
-17
```

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

```
-17
```

Рисунок 0.4. Матричное умножение, единичная матрица, скалярное произведение

4.2.5. 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
```

```
3-element Vector{Float64}:
 1.0
 1.0000000000000004
 0.9999999999999997
```

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

```
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
 1.0       0.0       0.0
 0.33023   1.0       0.0
 0.265203  0.622077  1.0
U factor:
3×3 Matrix{Float64}:
 0.688738  0.356098  0.466056
 0.0       0.663961  0.29521
 0.0       0.0       0.310928
```

```
println("Матрица перестановок = ", Alu.P)
println("Вектор перестановок = ", Alu.p)
```

```
Матрица перестановок = [0.0 1.0 0.0; 0.0 0.0 1.0; 1.0 0.0 0.0]
Вектор перестановок = [2, 3, 1]
```

```
println("Матрица L = ")
Alu.L
```

```
Матрица L =

3×3 Matrix{Float64}:
 1.0       0.0       0.0
 0.33023   1.0       0.0
 0.265203  0.622077  1.0
```

```
println("Матрица U = ")
Alu.U
```

```
Матрица U =

3×3 Matrix{Float64}:
 0.688738  0.356098  0.466056
 0.0       0.663961  0.29521
 0.0       0.0       0.310928
```

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

```
3-element Vector{Float64}:
 1.0
 1.0000000000000004
 0.9999999999999997
```

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

```
3-element Vector{Float64}:
 1.0
 1.0000000000000004
 0.9999999999999997
```

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

```
0.1421861276752491
```

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

```
0.1421861276752491
```

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

```
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
R factor:
3×3 Matrix{Float64}:
 -0.747966  -0.689483  -0.716677
  0.0        0.720956   0.466795
  0.0        0.0       -0.263674
```

```
# Матрица Q:
Aqr.Q
# Матрица R:
Aqr.R
```

```
3×3 Matrix{Float64}:
 -0.747966  -0.689483  -0.716677
  0.0        0.720956   0.466795
  0.0        0.0       -0.263674
```

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

```
3×3 Matrix{Float64}:
  1.0           1.11022e-16  -3.33067e-16
  1.11022e-16  1.0            1.11022e-16
 -1.66533e-16  0.0            1.0
```

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

```
3×3 Matrix{Float64}:
 0.36531   1.19621   0.845613
 1.19621   0.712197  1.24761
 0.845613  1.24761   0.89823
```

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

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 -0.7145637089949742
 -0.19157738289872284
  2.881878721965793
vectors:
3×3 Matrix{Float64}:
  0.62916   -0.596328  -0.498548
 -0.73877   -0.259403  -0.622036
  0.241613   0.759673  -0.603755
```

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

```
3-element Vector{Float64}:
 -0.7145637089949742
 -0.19157738289872284
  2.881878721965793
```

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

```
3×3 Matrix{Float64}:
  0.62916   -0.596328  -0.498548
 -0.73877   -0.259403  -0.622036
  0.241613   0.759673  -0.603755
```

```
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym
```

```
3×3 Matrix{Float64}:
  1.0           2.66454e-15  3.10862e-15
  1.11022e-15   1.0          1.55431e-15
 -3.10862e-15  -2.66454e-15  1.0
```

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

```
1000×1000 Matrix{Float64}:
  0.145678   -2.09796    0.371242   …   1.0364      0.640123    0.605619
  0.528729   -1.3466     0.18581       -0.458951   -0.160147    0.766213
 -1.77042    -1.67603   -1.15056       -0.0874261   0.0629916  -0.135821
 -0.279828    0.0370928 -1.54006       -0.375403    0.698306    0.621639
  1.18356     0.370427   2.4111        -1.48507    -0.668565    0.712572
 -0.267973   -0.652426  -0.43766    …  -0.234668    1.20137    -0.637785
  0.915547    0.539074   1.17664        1.89096    -0.248465   -0.23787
  0.69756    -0.151839  -0.296609      -0.0284107  -0.170259   -0.516867
 -0.282964    2.89712   -0.210701       1.2273      0.438124    2.36076
 -0.538873    0.416527  -0.601241       0.586093    1.70513     0.952744
  0.210923    0.0639703 -1.93127    …  -0.353719    1.09359    -0.623454
  2.59561     1.1324     0.964851      -1.76586     0.111247    0.735038
 -0.496435    1.52672    0.599742       0.589988    1.30489     0.972948
  ⋮                                 ⋱
  0.562163   -0.956142  -3.5368         0.617387    0.205788   -1.71322
  1.78537    -1.53803    0.231132      -0.681622    0.189001    0.914589
 -1.2565     -1.25046    0.908643   …  -0.411648    1.2333     -0.873703
 -1.38878    -0.934201   1.03458       -0.333723    0.0269304  -1.10727
 -1.07837    -0.372949   0.432681      -2.42585    -0.398173    0.772855
  1.13652     0.585185  -1.06978        3.11415     0.036929   -0.726685
  1.95734    -0.442784  -1.05317        0.274      -0.599454    1.06383
  0.464526    0.288391   2.29629    …  -0.599927    0.398798   -1.10519
  0.0566294   0.537454   0.123746      -0.380964    0.0279099   1.03905
  0.820493   -0.296017   1.08016       -0.156861    0.997232    0.649939
 -0.452679    1.26647   -0.860126      -0.322121   -1.18713     0.6931
 -0.098936   -0.0852735 -0.0853568      0.510708    0.567533   -0.287865
```

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

```
1000×1000 Matrix{Float64}:
  0.291357   -1.56923   -1.39918    …   1.85689     0.187444    0.506683
 -1.56923    -2.6932    -1.49022       -0.754968    1.10632     0.68094
 -1.39918    -1.49022   -2.30112        0.99273    -0.797134   -0.221178
 -0.347395    0.276533  -1.34956        1.11688    -0.0298017   0.167306
  3.90814    -0.448152   3.11968       -3.17615    -0.886637    1.756
  1.5434     -0.0254891 -0.466988   …  -0.0558649   1.12509     0.758792
  1.74196     0.892849   0.176108       1.30557    -1.39873     0.449264
 -0.121089   -0.457916  -0.274718       1.71998    -1.0829     -0.285236
  1.38659     3.01964    1.94639        1.76113    -0.480088    4.26256
  0.128627    0.54673    0.0723914      0.528345    2.87871     3.6812
  0.0956949  -0.992318  -1.04406    …  -0.927922    0.904211   -1.13191
  2.96932     2.37411    0.940668      -2.5133      0.919277    1.1746
 -0.648856    2.04737    0.472488       0.63303     2.55265     0.00797444
  ⋮                                 ⋱
  1.44157    -1.63118   -5.64235        0.438064    1.71865    -1.59158
  1.38065    -1.58671    0.580974       0.381833    0.296747    3.72581
 -2.03087    -0.394341   2.17406    …  -0.164631    0.610997   -1.74269
 -2.03675     0.81989    2.05559       -0.387363   -0.529037    0.16605
 -0.691508   -1.13514    1.64349       -1.03618    -0.186936    1.01965
  1.96429     0.726367  -2.43164        3.76243    -0.444916    0.616191
  1.64337     0.306503  -0.242901       1.08283    -0.541516    1.82458
  0.801663   -0.966926   1.27133    …  -0.198313   -0.409904   -0.326065
 -0.970558    1.27757    0.527657      -1.59456     0.405109    2.25622
  1.85689    -0.754968   0.99273       -0.313722    0.675111    1.16065
  0.187444    1.10632   -0.797134       0.675111   -2.37427     1.26063
  0.506683    0.68094   -0.221178       1.16065     1.26063    -0.57573
```

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

```
true
```

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

```
false
```

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

```
1000×1000 Symmetric{Float64, Matrix{Float64}}:
  0.291357   -1.56923    -1.39918    …   1.85689     0.187444    0.506683
 -1.56923    -2.6932     -1.49022       -0.754968    1.10632     0.68094
 -1.39918    -1.49022    -2.30112        0.99273    -0.797134   -0.221178
 -0.347395    0.276533   -1.34956        1.11688    -0.0298017   0.167306
  3.90814    -0.448152    3.11968       -3.17615    -0.886637    1.756
  1.5434     -0.0254891  -0.466988   …  -0.0558649   1.12509     0.758792
  1.74196     0.892849    0.176108       1.30557    -1.39873     0.449264
 -0.121089   -0.457916   -0.274718       1.71998    -1.0829     -0.285236
  1.38659     3.01964     1.94639        1.76113    -0.480088    4.26256
  0.128627    0.54673     0.0723914      0.528345    2.87871     3.6812
  0.0956949  -0.992318   -1.04406    …  -0.927922    0.904211   -1.13191
  2.96932     2.37411     0.940668      -2.5133      0.919277    1.1746
 -0.648856    2.04737     0.472488       0.63303     2.55265     0.00797444
  ⋮                                  ⋱
  1.44157    -1.63118    -5.64235        0.438064    1.71865    -1.59158
  1.38065    -1.58671     0.580974       0.381833    0.296747    3.72581
 -2.03087    -0.394341    2.17406    …  -0.164631    0.610997   -1.74269
 -2.03675     0.81989     2.05559       -0.387363   -0.529037    0.16605
 -0.691508   -1.13514     1.64349       -1.03618    -0.186936    1.01965
  1.96429     0.726367   -2.43164        3.76243    -0.444916    0.616191
  1.64337     0.306503   -0.242901       1.08283    -0.541516    1.82458
  0.801663   -0.966926    1.27133    …  -0.198313   -0.409904   -0.326065
 -0.970558    1.27757     0.527657      -1.59456     0.405109    2.25622
  1.85689    -0.754968    0.99273       -0.313722    0.675111    1.16065
  0.187444    1.10632    -0.797134       0.675111   -2.37427     1.26063
  0.506683    0.68094    -0.221178       1.16065     1.26063    -0.57573
```

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

```
   Resolving package versions...
   Installed BenchmarkTools ─ v1.5.0
    Updating `C:\Users\GanZL\.julia\environments\v1.11\Project.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
    Updating `C:\Users\GanZL\.julia\environments\v1.11\Manifest.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
  [9abbd945] + Profile v1.11.0
Precompiling project...
   2125.6 ms  ✓ BenchmarkTools
  1 dependency successfully precompiled in 2 seconds. 54 already precompiled.
```

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

```
  34.939 ms (21 allocations: 7.99 MiB)
```

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

```
  259.575 ms (27 allocations: 7.93 MiB)
```

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

```
  34.553 ms (21 allocations: 7.99 MiB)
```

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

```
1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}:
 0.245908  0.546101   ⋅          ⋅          …   ⋅          ⋅          ⋅
 0.546101  0.0125396  0.71222    ⋅              ⋅          ⋅          ⋅
  ⋅        0.71222    1.55519   1.33239         ⋅          ⋅          ⋅
  ⋅         ⋅         1.33239   0.0314337       ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅       -0.84155         ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅          …   ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅          …   ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
 ⋮                                          ⋱
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅          …   ⋅          ⋅          ⋅
  ⋅         ⋅          ⋅         ⋅              ⋅          ⋅          ⋅
```

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

```
  319.483 ms (44 allocations: 183.11 MiB)
6.919811236474235
```

Рисунок 0.5.1-0.5.8 Факторизация. Специальные матричные структуры

4.2.6. Общая линейная алгебра

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

```
3×3 Matrix{Rational{BigInt}}:
 3//10  4//5   1//5
  1     1//10   1
 3//10  3//10  2//5
```

```julia
# Единичный вектор:
x = fill(1, 3)
# Задаём вектор b
b = Arational*x
```

```
3-element Vector{Rational{BigInt}}:
 13//10
 21//10
   1
```

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

```
3-element Vector{Rational{BigInt}}:
 1
 1
 1
```

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

```
LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3×3 Matrix{Rational{BigInt}}:
  1       0     0
 3//10    1     0
 3//10  27//77  1
U factor:
3×3 Matrix{Rational{BigInt}}:
 1   1//10     1
 0  77//100  -1//10
 0    0      52//385
```

Рисунок 0.6. Общая линейная алгебра

Задания для самостоятельного выполнения

4.4.1. Произведение векторов

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

```
v = [3, 5, 8, 4]
dot_v = dot(v, v)
```

```
114
```

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

```
4×4 Matrix{Int64}:
  9  15  24  12
 15  25  40  20
 24  40  64  32
 12  20  32  16
```

Рисунок 1.1. Код и результат Задания 1

1.Скалярное произведение (dot_v): Результатом является скаляр, который вычисляется как сумма произведений соответствующих элементов вектора на самих себя. В данном случае результат равен 114.

2.Внешнее произведение (outer_v): Результатом является матрица 4x4, каждый элемент которой представляет собой произведение соответствующих элементов вектора. Полученная матрица является симметричной.

4.4.2. Системы линейных уравнений

A - коэффициенты при неизвестных

b - ответы

```
# first task
A_1 = [1 1; 1 -1]
b_1 = [2; 3]
```

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

```
# Решение уравнения получаем с помощью функции \
A_1\b_1
```

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

```
# second task
A_2 = [1 1; 2 2]
b_2 = [2; 4]
```

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

```
A_2\b_2
```

```
SingularException(2)

Stacktrace:
  [1] checknonsingular
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\factorization.jl:69 [inlined]
  [2] _check_lu_success
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:84 [inlined]
  [3] #lu!#182
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:92 [inlined]
  [4] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:90 [inlined]
  [5] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:89 [inlined]
  [6] _lu
```

```
# third task
A_3 = [1 1; 2 2]
b_3 = [2; 5]
```

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

```
A_3\b_3
```

```
SingularException(2)

Stacktrace:
  [1] checknonsingular
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\factorization.jl:69 [inlined]
  [2] _check_lu_success
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:84 [inlined]
  [3] #lu!#182
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:92 [inlined]
  [4] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:90 [inlined]
  [5] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:89 [inlined]
  [6] _lu
```

```julia
# fourth task
A_4 = [1 1; 2 2; 3 3]
b_4 = [1; 2; 3]
A_4\b_4
```

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

```julia
# fifth task
A_5 = [1 1; 2 1; 1 -1]
b_5 = [2; 1; 3]
A_5\b_5
```

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

```julia
# sixth task
A_6 = [1 1; 2 1; 3 2]
b_6 = [2; 1; 3]
A_6\b_6
```

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

```julia
# seventh task
A_7 = [1 1 1; 1 -1 -2]
b_7 = [2; 3]
A_7\b_7
```

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

```julia
# eighth task
A_8 = [1 1 1; 2 2 -3; 3 1 1]
b_8 = [2; 4; 1]
A_8\b_8
```

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

```
# ninth task
A_9 = [1 1 1; 1 1 2; 2 2 3]
b_9 = [1; 0; 1]
A_9\b_9
```

```
  [1] checknonsingular
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\factorization.jl:69 [inlined]
  [2] _check_lu_success
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:84 [inlined]
  [3] #lu!#182
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:92 [inlined]
  [4] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:90 [inlined]
  [5] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:89 [inlined]
  [6] _lu
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:347 [inlined]
  [7] lu(::Matrix{Int64}; kwargs::@Kwargs{})
    @ LinearAlgebra C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\st
```

```
# tenth task
A_10 = [1 1 1; 1 1 2; 2 2 3]
b_10 = [1; 0; 0]
A_10\b_10
```

```
  [1] checknonsingular
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\factorization.jl:69 [inlined]
  [2] _check_lu_success
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:84 [inlined]
  [3] #lu!#182
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:92 [inlined]
  [4] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:90 [inlined]
  [5] lu!
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:89 [inlined]
  [6] _lu
    @ C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\stdlib\v1.11\LinearAlgebra\src\lu.jl:347 [inlined]
  [7] lu(::Matrix{Int64}; kwargs::@Kwargs{})
    @ LinearAlgebra C:\Users\GanZL\.julia\juliaup\julia-1.11.1+0.x64.w64.mingw32\share\julia\st
```

Рисунок 1.2.1-1.2.4. Код и результат Задания 2

1.Представление СЛАУ в матричной форме: СЛАУ с двумя или несколькими неизвестными можно записать в матричной форме, где матрица коэффициентов переменных (A) умножается на вектор неизвестных (x), что дает вектор значений правой части (b):

$A*x=b$

Здесь A — это матрица коэффициентов, а b — вектор правых частей.

2.Решение через оператор "левого деления" (): В Julia для решения линейных уравнений используется оператор \, который находит вектор неизвестных x:

x = A \ b

Этот оператор находит решение, используя методы линейной алгебры, такие как метод Гаусса или LU-разложение.

4.4.3 Операции с матрицами

Приведите приведенные ниже матрицы к диагональному виду

```
function diagonal_matrices(matrix)
    # Проведем симметризацию матриц
    Asym = matrix + matrix'
    # Спектральное разложение симметризированной матрицы
    AsymEig = eigen(Asym)
    # в итоге приводим матрицу к диагональному виду
    return inv(AsymEig.vectors) * matrix * AsymEig.vectors
end
```

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

```
matrix_1 = [1 -2; -2 1]
diagonal_matrices(matrix_1)
```

```
2×2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
```

```
matrix_2 = [1 -2; -2 3]
diagonal_matrices(matrix_2)
```

```
2×2 Matrix{Float64}:
 -0.236068     4.44089e-16
  2.22045e-16  4.23607
```

```
matrix_3 = [1 -2 0; -2 1 2; 0 2 0]
diagonal_matrices(matrix_3)
```

```
3×3 Matrix{Float64}:
 -2.14134       3.55271e-15  -1.9984e-15
  3.38618e-15   0.515138      1.11022e-16
 -6.66134e-16  -4.44089e-16   3.6262
```

Рисунок 1.3.1 Код и результат Задания 1

1.Симметризация: Первым шагом симметризуем матрицу, чтобы работать с более удобной в вычислениях версией, так как симметричные матрицы имеют вещественные собственные значения.

2.Спектральное разложение: Вычисляем собственные значения и векторы симметризованной матрицы, что позволяет привести исходную матрицу к диагональному виду.

3.Диагонализация: Используем собственные векторы для преобразования матрицы в диагональный вид, что упрощает ее дальнейший анализ и вычисления.

Вычислим матрицы

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

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

```
sqrt([5 -2; -2 5])
```

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

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

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

```
sqrt([1 2; 2 3])
```

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

Рисунок 1.3.2 Код и результат Задания 1

1.Возведение матрицы в степень: Использую оператор ^, чтобы возвести матрицу в нужную степень. Это полезно, когда необходимо работать с многократными линейными преобразованиями.

2.Извлечение корня: Применяю функцию sqrt() для извлечения квадратного корня из матрицы, что работает с симметричными положительно определёнными матрицами. Для кубического корня применяю возведение в степень 1/3.

Найдите собственные значения марицы А

```
# Введем матрицу А
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]
# Найдем собственные значения
eigenvalues = eigvals(A)
```

```
5-element Vector{Float64}:
 -128.49322764802145
  -55.88778455305688
   42.75216727931894
   87.16111477514521
  542.4677301466143
```

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

```
  1.170 μs (15 allocations: 2.55 KiB)
```

```
# Создадим диагональную матрицу из собственных значений
B = zeros(5, 5)
@btime for i in 1:1:5
    B[i, i] = eigenvalues[i]
end
B
```

```
  137.324 ns (5 allocations: 80 bytes)
5×5 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
```

```
# Создадим нижнеждиагональную матрицу из матрицы A
# LU-факторизация:
Alu = lu(A)
@btime Alu.L
```

```
  46.411 ns (2 allocations: 272 bytes)
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
```

Рисунок 1.3.3 Код и результат Задания 1

1.Собственные значения: Для нахождения собственных значений использую eigvals(). Это важный шаг для анализа структуры матрицы и её свойств.

2.Диагональная матрица: Создаю диагональную матрицу из собственных значений, чтобы легко анализировать свойства матрицы в диагонализованной форме.

3.Нижняя треугольная матрица: Использую LU-разложение для выделения нижней треугольной матрицы. Этот метод удобен для анализа, поскольку LU-разложение разбивает матрицу на произведение двух треугольных матриц, что упрощает дальнейшие вычисления.

4.4.4 Линейные модели экономики

```
function productive_matrix(matrix, size)
    ans=""
    # единичная матрица
    E = [1 0; 0 1]
    # зададим любые неотрицательные числа
    Y = rand(0:1000, size)
    # По формуле вычислим x - A*x = y
    S = E - matrix
    # найдем значения x
    X = S\Y
    # теперь проверим есть ли среди x отрциательное число
    for i in 1:1:size
        if X[i] < 0
            ans = "Матрица непродуктивная"
            break
        else
            ans = "Матрица продуктивная"
        end
    end
    return ans
end
```

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

```
matrix_1 = [1 2; 3 4]
productive_matrix(matrix_1, 2)
```

```
"Матрица непродуктивная"
```

```
matrix_2 = ([1 2; 3 4])*(1/2)
productive_matrix(matrix_2, 2)
```

```
"Матрица непродуктивная"
```

```
matrix_3 = ([1 2; 3 4])*(1/10)
productive_matrix(matrix_3, 2)
```

```
"Матрица продуктивная"
```

Рисунок 1.4.1 Код и результат Задания 1

1.Проверка продуктивности через систему уравнений: Сначала формирую матрицу вида E - A и решаю систему линейных уравнений со случайной неотрицательной правой частью. Если среди элементов решения есть отрицательные, матрица считается непродуктивной.

2.Обратная матрица для анализа знаков: Также использую метод нахождения обратной матрицы E - A и анализирую все её элементы на наличие отрицательных значений. Отрицательные элементы свидетельствуют о непродуктивности матрицы.

```
function productive_matrix_2(matrix, size)
    # единичная матрица
    ans = ""
    E = [1 0; 0 1]
    matrix_new = E - matrix
    inv_matrix_new = inv(matrix_new)
    for i in 1:1:size
        for j in 1:1:size
            if inv_matrix_new[i, j] < 0
                ans = "Матрица непродуктивная"
                break
            else
                ans = "Матрица продуктивная"
            end
        end
    end
    return ans
end
```

```
productive_matrix_2 (generic function with 1 method)
```

```
matrix_1 = [1 2; 3 1]
productive_matrix_2(matrix_1, 2)
```

```
"Матрица непродуктивная"
```

```
matrix_2 = ([1 2; 3 1])*(1/2)
productive_matrix_2(matrix_2, 2)
```

```
"Матрица непродуктивная"
```

```
matrix_3 = ([1 2; 3 1])*(1/10)
productive_matrix_2(matrix_3, 2)
```

```
"Матрица продуктивная"
```

Рисунок 1.4.2 Код и результат Задания 1

1.Формирование обратной матрицы: Сначала вычисляю обратную матрицу для (E - A). Продуктивность матрицы A определяется тем, что все элементы обратной матрицы должны быть неотрицательными.

2.Проверка на неотрицательность: Анализирую все элементы обратной матрицы. Если среди них встречаются отрицательные значения, матрица считается непродуктивной. Если все элементы неотрицательные, то матрица является продуктивной.

```
function productive_matrix_3(matrix, size)
    ans=""
    # найдем собственные значения переданной матрицы
    eigenvalues = eigvals(matrix)
    for i in 1:1:size
        if abs(eigenvalues[i]) > 1
            ans = "Матрица непродуктивная"
            break
        else
            ans = "Матрица продуктивная"
        end
    end
    return ans
end
```

```
productive_matrix_3 (generic function with 1 method)
```

```
matrix_1 = [1 2; 3 1]
productive_matrix_3(matrix_1, 2)
```

```
"Матрица непродуктивная"
```

```
matrix_2 = ([1 2; 3 1])*(1/2)
productive_matrix_3(matrix_2, 2)
```

```
"Матрица непродуктивная"
```

```
matrix_3 = ([1 2; 3 1])*(1/10)
productive_matrix_3(matrix_3, 2)
```

```
"Матрица продуктивная"
```

```
matrix_4 = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
productive_matrix_3(matrix_4, 2)
```

```
"Матрица продуктивная"
```

Рисунок 1.4.3 Код и результат Задания 1

1.Спектральный критерий: Сначала нахожу собственные значения матрицы. Продуктивность матрицы A определяется тем, что все её собственные значения по модулю должны быть меньше 1.

2.Проверка собственных значений: Анализирую модуль каждого собственного значения. Если хотя бы одно значение больше или равно 1, то матрица считается непродуктивной. Если все значения меньше 1, то матрица продуктивная.

Вывод

Я изучил возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры. А также изучил основные методы факторизации объектов.