

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей

Отчет по лабораторной работе № 4

Дисциплина: Практикум по компьютерному моделированию

Студент: Логинов Сергей Андреевич

Группа: НФИбд-01-18

МОСКВА 2021г

Цель работы

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

Ход работы

Поэлементные операции над многомерными массивами

```
In [3]: a = rand(1:20,(4,3))
```

```
Out[3]: 4x3 Matrix{Int64}:
 3  18  20
 11 12  4
 16 15  19
 11  5  11
```

```
In [4]: sum(a)
```

```
Out[4]: 145
```

```
In [5]: sum(a, dims = 1)
```

```
Out[5]: 1x3 Matrix{Int64}:
 41 50 54
```

```
In [6]: sum(a, dims = 2)
```

```
Out[6]: 4x1 Matrix{Int64}:
 41
 27
 50
 27
```

```
In [7]: prod(a)
```

```
Out[7]: 1573178112000
```

```
In [8]: prod(a, dims = 1)
```

```
Out[8]: 1x3 Matrix{Int64}:
 5808 16200 16720
```

```
In [9]: prod(a, dims = 2)
```

```
Out[9]: 4x1 Matrix{Int64}:
 1080
 528
 4560
```

Для работы со средними значениями можно воспользоваться возможностями пакеты Statistics:

```
In [10]: using Statistics  
mean(a)
```

```
Out[10]: 12.08333333333334
```

```
In [11]: mean(a, dims = 1)
```

```
Out[11]: 1×3 Matrix{Float64}:  
10.25 12.5 13.5
```

```
In [12]: mean(a, dims = 2)
```

```
Out[12]: 4×1 Matrix{Float64}:  
13.66666666666666  
9.0  
16.66666666666668  
9.0
```

Транспонирование, след, ранг, определитель и инверсия матрицы

```
In [13]: using LinearAlgebra  
b = rand(1:20, (4,4))
```

```
Out[13]: 4x4 Matrix{Int64}:  
2 6 5 19  
5 10 10 17  
2 3 1 10  
6 5 6 20
```

```
In [14]: transpose(b)
```

```
Out[14]: 4x4 transpose(::Matrix{Int64}) with eltype Int64:  
2 5 2 6  
6 10 3 5  
5 10 1 6  
19 17 10 20
```

```
In [15]: tr(b)
```

```
Out[15]: 33
```

```
In [16]: rank(b)
```

```
Out[16]: 4
```

```
In [17]: inv(b)
```

```
Out[17]: 4x4 Matrix{Float64}:  
-0.338182 0.0521212 0.246061 0.153939  
-0.123636 0.162424 0.441212 -0.241212  
0.138182 0.0145455 -0.512727 0.112727  
0.0909091 -0.0606061 -0.030303 0.030303
```

```
In [18]: det(b)
```

```
Out[18]: 824.9999999999999
```

```
In [19]: pinv(b)
```

```
Out[19]: 4x4 Matrix{Float64}:  
-0.338182 0.0521212 0.246061 0.153939  
-0.123636 0.162424 0.441212 -0.241212  
0.138182 0.0145455 -0.512727 0.112727  
0.0909091 -0.0606061 -0.030303 0.030303
```

```
In [20]: x = [2, 4, -5]
```

```
Out[20]: 3-element Vector{Int64}:
 2
 4
-5
```

```
In [21]: norm(x)
```

```
Out[21]: 6.708203932499369
```

```
In [22]: p = 1
norm(x, p)
```

```
Out[22]: 11.0
```

```
In [23]: y = [1, -1, 3]
```

```
Out[23]: 3-element Vector{Int64}:
 1
-1
 3
```

```
In [24]: norm(x-y)
```

```
Out[24]: 9.486832980505138
```

```
In [25]: sqrt(sum((x-y).^2))
```

```
Out[25]: 9.486832980505138
```

```
In [26]: acos((transpose(x)*y)/(norm(x)*norm(y)))
```

```
Out[26]: 2.4404307889469252
```

```
In [27]: d = [5 -4 2; -1 2 3; -2 1 0]
```

```
Out[27]: 3x3 Matrix{Int64}:
 5  -4   2
 -1   2   3
 -2   1   0
```

```
In [28]: opnorm(d)
```

```
Out[28]: 7.147682841795258
```

```
In [29]: opnorm(d, p)
```

```
Out[29]: 8.0
```

```
In [30]: rot180(d)
```

```
Out[30]: 3x3 Matrix{Int64}:
```

```
 0  1  -2  
 3  2  -1  
 2  -4   5
```

```
In [31]: reverse(d, dims = 1)
```

```
Out[31]: 3x3 Matrix{Int64}:
```

```
-2  1  0  
-1  2  3  
 5  -4  2
```

```
In [32]: reverse(d, dims = 2)
```

```
Out[32]: 3x3 Matrix{Int64}:
```

```
 2  -4   5  
 3   2  -1  
 0   1  -2
```

Матричное умножение, единичная матрица, скалярное произведение

```
In [33]: a = rand(1:10,(2,3))
```

```
Out[33]: 2x3 Matrix{Int64}:
 10  6  1
 7  1  9
```

```
In [34]: b = rand(1:10, (3,4))
```

```
Out[34]: 3x4 Matrix{Int64}:
 2  9  10  9
 5  3  2  1
 7  9  10  1
```

```
In [35]: a * b
```

```
Out[35]: 2x4 Matrix{Int64}:
 57  117  122  97
 82  147  162  73
```

```
In [36]: Matrix{Int}(I, 3, 3)
```

```
Out[36]: 3x3 Matrix{Int64}:
 1  0  0
 0  1  0
 0  0  1
```

```
In [37]: dot(x,y)
```

```
Out[37]: -17
```

```
In [38]: x'y
```

```
Out[38]: -17
```

Факторизация. Специальные матричные структуры


```
In [39]: a = rand(3, 3)
```

```
Out[39]: 3x3 Matrix{Float64}:
 0.383701  0.479133  0.937988
 0.242557  0.356388  0.204184
 0.332105  0.60312   0.828479
```

```
In [40]: x = fill(1.0, 3)
```

```
Out[40]: 3-element Vector{Float64}:
1.0
1.0
1.0
```

```
In [41]: b = a * x
```

```
Out[41]: 3-element Vector{Float64}:
1.8008225110584517
0.803128730182046
1.76370388198425
```

```
In [42]: a\!b
```

```
Out[42]: 3-element Vector{Float64}:
1.0000000000000013
0.9999999999999997
0.9999999999999997
```

```
In [43]: alu = lu(a)
```

```
Out[43]: LU{Float64, Matrix{Float64}}
L factor:
3x3 Matrix{Float64}:
 1.0      0.0      0.0
 0.865532  1.0      0.0
 0.632151  0.283967  1.0
U factor:
3x3 Matrix{Float64}:
 0.383701  0.479133  0.937988
 0.0        0.188415  0.0166199
 0.0        0.0        -0.393486
```

```
In [44]: alu.p
```

```
Out[44]: 3-element Vector{Int64}:
1
3
2
```

```
In [45]: alu.P
```

```
Out[45]: 3x3 Matrix{Float64}:
 1.0  0.0  0.0
 0.0  0.0  1.0
 0.0  1.0  0.0
```

```
In [46]: alu.L
```

```
Out[46]: 3x3 Matrix{Float64}:
```

1.0	0.0	0.0
0.865532	1.0	0.0
0.632151	0.283967	1.0

```
In [47]: alu.U
```

```
Out[47]: 3x3 Matrix{Float64}:
```

0.383701	0.479133	0.937988
0.0	0.188415	0.0166199
0.0	0.0	-0.393486

```
In [48]: alu\b
```

```
Out[48]: 3-element Vector{Float64}:
```

1.000000000000013
0.9999999999999997
0.9999999999999997

```
In [49]: det(a)
```

```
Out[49]: 0.02844705823026136
```

```
In [50]: det(alu)
```

```
Out[50]: 0.02844705823026136
```

```
In [51]: aqr = qr(a)
```

```
Out[51]: LinearAlgebra.QRCompactWY{Float64, Matrix{
```

Q factor:

3x3 LinearAlgebra.QRCompactWYQ{Float64, Ma
-0.682191 0.642838 0.348389
-0.431248 0.031032 -0.9017
-0.590458 -0.765373 0.256053

R factor:

3x3 Matrix{Float64}:
-0.562453 -0.836669 -1.21712
0.0 -0.142548 -0.0247847
0.0 0.0 0.354806

```
In [52]: aqr.Q
```

```
Out[52]: 3x3 LinearAlgebra.QRCompactWYQ{Float64, Ma
```

-0.682191 0.642838 0.348389
-0.431248 0.031032 -0.9017
-0.590458 -0.765373 0.256053

```
In [53]: aqr.R
```

```
Out[53]: 3x3 Matrix{Float64}:
```

-0.562453 -0.836669 -1.21712
0.0 -0.142548 -0.0247847
0.0 0.0 0.354806

```
0.0      -0.142548  -0.024784/  
0.0       0.0       0.354806
```

```
In [54]: aqr.Q' * aqr.Q
```

```
Out[54]: 3x3 Matrix{Float64}:  
1.0      -5.55112e-17  2.77556e-17  
-5.55112e-17  1.0      2.77556e-17  
2.77556e-17  2.77556e-17  1.0
```

```
In [55]: asym = a + a'
```

```
Out[55]: 3x3 Matrix{Float64}:  
0.767401  0.72169   1.27009  
0.72169   0.712777  0.807304  
1.27009   0.807304  1.65696
```

```
In [56]: asymeig = eigen(asym)
```

```
Out[56]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
-0.1660972427481388
 0.24971917284771097
 3.0535130796212915
vectors:
3x3 Matrix{Float64}:
 0.842602   0.00961202   0.538451
 -0.257373  -0.871081    0.418303
 -0.473055   0.491046    0.7315
```

```
In [57]: asymeig.values
```

```
Out[57]: 3-element Vector{Float64}:
-0.1660972427481388
 0.24971917284771097
 3.0535130796212915
```

```
In [58]: asymeig.vectors
```

```
Out[58]: 3x3 Matrix{Float64}:
 0.842602   0.00961202   0.538451
 -0.257373  -0.871081    0.418303
 -0.473055   0.491046    0.7315
```

```
In [59]: inv(asymeig)*asym
```

```
Out[59]: 3x3 Matrix{Float64}:
 1.0          1.33227e-15  1.77636e-15
 -3.10862e-15 1.0          -2.66454e-15
 1.44329e-15  8.04912e-16 1.0
```

```
In [60]: n = 1000
a = randn(n, n)
```

```
Out[60]: 1000x1000 Matrix{Float64}:
 -0.735694   1.15196     2.90918     ... -0.0937169   0.284659
 0.313566
 -0.147111  -0.962118   0.248893
 0.568515
 -0.608051  -0.273287   0.229771
 0.39958
 -0.615477   0.531506   -0.104054
 0.994792
```

```
In [252]: asym = a + a';
```

```
In [253]: issymmetric(asym)
```

Out[253]: true

```
In [254]: asym_noisy = copy(asym);
```

```
In [255]: asym_noisy[1, 2] += 5eps()
```

Out [255]: 0.09283039471965582

```
In [256]: issymmetric(asym_noisy)
```

Out[256]: `false`

```
In [257]: asym_explicit = Symmetric(asym_noisy);
```

```
In [67]: using BenchmarkTools  
@btime eigvals(asym);
```

261.348 ms (11 allocations: 7.99 MiB)

```
In [68]: @btime eigvals(asym_noisy);
```

1.533 s (13 allocations: 7.92 MiB)

```
In [69]: @btime eigvals(asym explicit);
```

269.346 ms (11 allocations: 7.99 MiB)

```
In [70]: n = 1000000;
```

```
a = SymTridiagonal(randn(n), randn(n - 1))
```

Out[70]: 1000000×1000000 SymTridiagonal{Float64, Vector{Float64}}

0.993578 -0.524191

-0.524191 1.30134 0.0346142

0.0346142 -0.424008

0.584861

Общая линейная алгебра

```
In [73]: arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
```

```
Out[73]: 3x3 Matrix{Rational{BigInt}}:  
 3//5  1//10  1//1  
 3//5  1//1   1//1  
 4//5  3//5   7//10
```

```
In [74]: x = fill(1, 3)
```

```
Out[74]: 3-element Vector{Int64}:  
 1  
 1  
 1
```

```
In [75]: b = arational * x
```

```
Out[75]: 3-element Vector{Rational{BigInt}}:  
 17//10  
 13//5  
 21//10
```

```
In [76]: arational\b
```

```
Out[76]: 3-element Vector{Rational{BigInt}}:  
 1//1  
 1//1  
 1//1
```

```
In [77]: lu(arational)
```

```
Out[77]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}}  
 L factor:  
 3x3 Matrix{Rational{BigInt}}:  
 1//1  0//1  0//1  
 3//4  1//1  0//1  
 3//4 -7//11 1//1  
 U factor:  
 3x3 Matrix{Rational{BigInt}}:  
 4//5  3//5   7//10  
 0//1  11//20  19//40  
 0//1  0//1   171//220
```

Задания для самостоятельного выполнения

1. Произведение векторов

1. Задайте вектор v . Умножьте данный вектор скалярно на себя и сохраните результат

2. Умножьте вектор матрично на себя

```
In [78]: v = [1; 2; 3]
```

```
Out[78]: 3-element Vector{Int64}:
1
2
3
```

```
In [79]: dot_v = dot(v, v)
```

```
Out[79]: 14
```

```
In [81]: outer_v = v * v'
```

```
Out[81]: 3x3 Matrix{Int64}:
1 2 3
2 4 6
3 6 9
```

2. Системы линейных уравнений

1. Решить СЛАУ с двумя неизвестными

a) $\begin{cases} x + y = 2, \\ x - y = 3. \end{cases}$

b) $\begin{cases} x + y = 2, \\ 2x + 2y = 4. \end{cases}$

c) $\begin{cases} x + y = 2, \\ 2x + 2y = 5. \end{cases}$

d) $\begin{cases} x + y = 1, \\ 2x + 2y = 2, \\ 3x + 3y = 3. \end{cases}$

e) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ x - y = 3. \end{cases}$

f) $\begin{cases} x + y = 2, \\ 2x + y = 1, \\ 3x + 2y = 3. \end{cases}$

Создадим функцию для решения СЛАУ путем деления:

```
In [89]: function slu(x, y)
    result = x \ y
    return result
end
```

```
Out[89]: slu (generic function with 1 method)
```

Решим уравнения:

```
In [90]: # a)
slu([1 1; 1 -1], [2; 3])
```

```
Out[90]: 2-element Vector{Float64}:
 2.5
 -0.5
```

```
In [96]: # b)
```

```
In [98]: # c)
```

```
In [99]: # d)
slu([1 1; 2 2; 3 3], [1; 2; 3])
```

```
Out[99]: 2-element Vector{Float64}:
 0.4999999999999999
 0.5
```

```
In [259]: # e)
```

```
In [105]: # f)
slu([1 1; 2 1; 3 2], [2; 1; 3])
```

```
Out[105]: 2-element Vector{Float64}:
 -0.9999999999999989
 2.999999999999982
```

Уравнение b имеет решение (1; 1), но функция его не находит. Уравнения c и e не имеют решений.

2. Решить СЛАУ с тремя неизвестными

- a)
$$\begin{cases} x + y + z = 2, \\ x - y - 2z = 3. \end{cases}$$
- b)
$$\begin{cases} x + y + z = 2, \\ 2x + 2y - 3z = 4, \\ 3x + y + z = 1. \end{cases}$$
- c)
$$\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 1. \end{cases}$$
- d)
$$\begin{cases} x + y + z = 1, \\ x + y + 2z = 0, \\ 2x + 2y + 3z = 0. \end{cases}$$

```
In [106]: # a
    slu([1 1 1; 1 -1 -2], [2; 3])
```

```
Out[106]: 3-element Vector{Float64}:
 2.2142857142857144
 0.35714285714285704
 -0.5714285714285712
```

```
In [107]: # b
    slu([1 1 1; 2 2 -3; 3 1 1], [2; 4; 1])
```

```
Out[107]: 3-element Vector{Float64}:
 -0.5
 2.5
 0.0
```

```
In [108]: # c
    slu([1 1 1; 1 1 2; 2 2 3], [1; 0; 1])
```

```
SingularException(2)
```

```
Stacktrace:
 [1] checknonsingular
   @ /Users/julia/buildbot/worker/package_julia/stdlib/v1.6/LinearAlgebra/src/factor.jl
 [2] checknonsingular
   @ /Users/julia/buildbot/worker/package_julia/stdlib/v1.6/LinearAlgebra/src/factor.jl
 [3] #lu!#136
   @ /Users/julia/buildbot/worker/package_julia/stdlib/v1.6/LinearAlgebra/src/lu.jl:8
```

3. Операции с матрицами

Приведите матрицы к диагональному виду

a)

$$\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}$$

b)

$$\begin{pmatrix} 1 & -2 \\ -2 & 3 \end{pmatrix}$$

c)

$$\begin{pmatrix} 1 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 0 \end{pmatrix}$$

Напишем функцию `get_diagonal_matrix`, которая будет возвращать диагональную матрицу из собственных значений матрицы, переданной в функцию в качестве аргумента. Используем спектральное разложение:

```
In [115]: function get_diagonal_matrix(matrix)
    spectr = eigen(matrix)
    return diagm(spectr.values)
end
```

```
Out[115]: get_diagonal_matrix (generic function with 1 method)
```

```
In [121]: # a)
get_diagonal_matrix([1 -2; -2 1])
```

```
Out[121]: 2x2 Matrix{Float64}:
 -1.0  0.0
  0.0  3.0
```

```
In [122]: # b)
get_diagonal_matrix([1 -2; -2 3])
```

```
Out[122]: 2x2 Matrix{Float64}:
 -0.236068  0.0
  0.0        4.23607
```

Задание с) решим подробно и с проверкой. Спектральное разложение - представление матрицы A в виде $A = VSV^{-1}$, где V - собственные векторы матрицы A, S - диагональная матрица из собственных значений. В ходе проверки должны получить исходную матрицу.

```
In [124]: # c)
spectr = eigen([1 -2 0; -2 1 2; 0 2 0])

Out[124]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
-2.1413361156553594
 0.5151380471280724
 3.6261980685272936
vectors:
3x3 Matrix{Float64}:
 0.421859  0.717093  0.554808
 0.6626    0.173846 -0.728518
-0.618866  0.674948 -0.401808
```

```
In [126]: diagm(spectr.values)
```

```
Out[126]: 3x3 Matrix{Float64}:
-2.14134  0.0      0.0
 0.0       0.515138  0.0
 0.0       0.0       3.6262
```

```
In [127]: spectr.vectors * diagm(spectr.values) * inv(spectr.vectors)
```

```
Out[127]: 3x3 Matrix{Float64}:
 1.0          -2.0   -1.11022e-16
 -2.0          1.0     2.0
 -2.22045e-16  2.0   5.88418e-15
```

В итоге получили исходную матрицу (на месте нулей числа, очень близкие к нулю)

Вычислите

a)

$$\begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}^{10}$$

b)

$$\sqrt{\begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}}$$

c)

$$\sqrt[3]{\begin{pmatrix} 5 & -2 \\ -2 & 5 \end{pmatrix}}$$

d)

$$\sqrt{\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}}$$

```
In [129]: # a)
[1 -2; -2 3]^10
```

```
Out[129]: 2×2 Matrix{Int64}:
 514229 -832040
 -832040 1346269
```

```
In [130]: # b)
[5 -2; -2 5]^(1/2)
```

```
Out[130]: 2×2 Symmetric{Float64, Matrix{Float64}}:
 2.1889 -0.45685
 -0.45685 2.1889
```

```
In [131]: # c)
[5 -2; -2 5]^(1/3)
```

```
Out[131]: 2×2 Symmetric{Float64, Matrix{Float64}}:
 1.67759 -0.235341
 -0.235341 1.67759
```

```
In [132]: # d)
[1 2; 2 3]^(1/2)
```

```
Out[132]: 2×2 Symmetric{ComplexF64, Matrix{ComplexF64}}:
 0.568864+0.351578im 0.920442-0.217287im
 0.920442-0.217287im 1.48931+0.134291im
```

Найдите собственные значения матрицы A. Создайте диагональную матрицу из собственных значений матрицы A. Создайте нижнедиагональную матрицу из матрицы A. Оцените эффективность выполняемых операций.

$$A = \begin{pmatrix} 140 & 97 & 74 & 168 & 131 \\ 97 & 106 & 89 & 131 & 36 \\ 74 & 89 & 152 & 144 & 71 \\ 168 & 131 & 144 & 54 & 142 \\ 131 & 36 & 71 & 142 & 36 \end{pmatrix}$$

Воспользуемся функцией нахождения диагональной матрицы и проведем замер затраченных ресурсов:

```
In [133]: a = [140 97 74 168 131; 97 106 89 131 36; 74 89 152 144 71; 168 131
```

```
Out[133]: 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

```
In [137]: @btime get_diagonal_matrix(a)
```

```
8.500 μs (12 allocations: 3.52 KiB)
```

```
Out[137]: 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

Создадим нижнедиагональную матрицу. Пользуемся LU-разложением, результатом будет матрица L:

```
In [138]: @btime alu = lu(a)  
alu.L
```

```
416.247 ns (3 allocations: 448 bytes)
```

```
Out[138]: 5x5 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

4. Линейные модели экономики:

Линейная модель экономики 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.

а) 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.

$$\begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$$

$$\frac{1}{2} \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$$

$$\frac{1}{10} \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$$

Представим x - Ax как $x(E - A)$, где E - единичная матрица. $E - A$ назовем s . Зададим произвольные положительные значения для y . Домножим наше уравнение $xs = y$ на s^{-1} . Получим $x = ys^{-1}$

Первая матрица:

```
In [139]: # 1)
a = [1 2; 3 4]
```

```
Out[139]: 2×2 Matrix{Int64}:
 1 2
 3 4
```

```
In [140]: e = [1 0; 0 1]
```

```
Out[140]: 2×2 Matrix{Int64}:
 1 0
 0 1
```

```
In [141]: s = e - a
```

```
Out[141]: 2×2 Matrix{Int64}:
 0 -2
 -3 -3
```

```
In [167]: y = [0.00001; 0.00002]
```

```
Out[167]: 2-element Vector{Float64}:
 1.0e-5
 2.0e-5
```

```
In [168]: x = y' * inv(s)
```

```
Out[168]: 1×2 adjoint(::Vector{Float64}) with eltype Float64:
 -5.0e-6  -3.33333e-6
```

x отрицательный, матрица непродуктивная

Вторая матрица:

```
In [169]: # 2)
b = [1 2; 3 4]*(1/2)
```

```
Out[169]: 2x2 Matrix{Float64}:
0.5 1.0
1.5 2.0
```

```
In [170]: s1 = e - b
```

```
Out[170]: 2x2 Matrix{Float64}:
0.5 -1.0
-1.5 -1.0
```

```
In [171]: x1 = y' * inv(s1)
```

```
Out[171]: 1x2 adjoint(::Vector{Float64}) with eltype Float64:
-1.0e-5 -1.0e-5
```

х отрицательный, матрица непродуктивная

Третья матрица:

```
In [172]: # 3)
c = [1 2; 3 4]*(1/10)
```

```
Out[172]: 2x2 Matrix{Float64}:
0.1 0.2
0.3 0.4
```

```
In [173]: s2 = e - c
```

```
Out[173]: 2x2 Matrix{Float64}:
0.9 -0.2
-0.3 0.6
```

```
In [174]: x2 = y' * inv(s2)
```

```
Out[174]: 1x2 adjoint(::Vector{Float64}) with eltype Float64:
2.5e-5 4.16667e-5
```

Здесь х положительный, матрица продуктивна.

b) Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрицы

$$(E - A)^{-1}$$

являются неотрицательными числами. Используя этот критерий, проверьте, являются ли матрицы продуктивными:

$$\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

$$\frac{1}{2} \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

$$\frac{1}{10} \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

```
In [175]: a = [1 2; 3 1]
```

```
Out[175]: 2x2 Matrix{Int64}:
```

$$\begin{matrix} 1 & 2 \\ 3 & 1 \end{matrix}$$

```
In [176]: e = [1 0; 0 1]
```

```
Out[176]: 2x2 Matrix{Int64}:
```

$$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

```
In [177]: inv(e - a)
```

```
Out[177]: 2x2 Matrix{Float64}:
```

$$\begin{matrix} -0.0 & -0.333333 \\ -0.5 & 0.0 \end{matrix}$$

```
In [178]: b = [1 2; 3 1] * (1/2)
```

```
Out[178]: 2x2 Matrix{Float64}:
```

$$\begin{matrix} 0.5 & 1.0 \\ 1.5 & 0.5 \end{matrix}$$

```
In [179]: inv(e - b)
```

```
Out[179]: 2x2 Matrix{Float64}:
```

$$\begin{matrix} -0.4 & -0.8 \\ -1.2 & -0.4 \end{matrix}$$

```
In [180]: c = a * (1/10)
```

```
Out[180]: 2x2 Matrix{Float64}:
```

$$\begin{matrix} 0.1 & 0.2 \\ 0.3 & 0.1 \end{matrix}$$

```
In [181]: inv(e - c)
```

```
Out[181]: 2x2 Matrix{Float64}:
```

$$\begin{matrix} 1.2 & 0.266667 \\ 0.4 & 1.2 \end{matrix}$$

Видим, что продуктивна только третья матрица

с) 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.

$$\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

$$\frac{1}{2} \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

$$\frac{1}{10} \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0 & 0.1 & 0.2 \\ 0 & 0.1 & 0.3 \end{pmatrix}$$

```
In [183]: eigen(a)
```

```
Out[183]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
2-element Vector{Float64}:
-1.4494897427831779
 3.4494897427831783
vectors:
2×2 Matrix{Float64}:
-0.632456  0.632456
 0.774597  0.774597
```

```
In [184]: b = a * (1/2)
```

```
Out[184]: 2×2 Matrix{Float64}:
 0.5  1.0
 1.5  0.5
```

```
In [185]: eigen(b)
```

```
Out[185]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
2-element Vector{Float64}:
-0.7247448713915892
 1.724744871391589
vectors:
2×2 Matrix{Float64}:
-0.632456  0.632456
 0.774597  0.774597
```

```
In [186]: c = a * (1/10)
```

```
Out[186]: 2×2 Matrix{Float64}:
 0.1  0.2
 0.3  0.1
```

```
In [187]: eigen(c)
```

```
Out[187]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
2-element Vector{Float64}:
-0.14494897427831785
 0.34494897427831783
vectors:
2×2 Matrix{Float64}:
-0.632456  0.632456
 0.774597  0.774597
```

```
In [188]: d = [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3]
```

```
Out[188]: 3×3 Matrix{Float64}:
 0.1  0.2  0.3
 0.0  0.1  0.2
 0.0  0.1  0.3
```

```
In [189]: eigen(d)
```

```
Out[189]: Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 0.02679491924311228
 0.1
 0.37320508075688774
vectors:
3x3 Matrix{Float64}:
 0.756568  1.0  -0.796751
 -0.614072  0.0  -0.356959
  0.224766  0.0  -0.487615
```

Видим, что продуктивны две матрицы - третья и четвертая.

Вывод

В данной лабораторной работе углубились в элементы линейной алгебры и освоили способы оценки производительности в Julia.