# Лабораторная работа № 4

## Линейная алгебра

Старовойтов Е. С.

7 декабря 2024

## Информация

### Докладчик


- Старовойтов Егор Сергеевич
- студент кафедры ТВиК
- Российский университет дружбы народов
- 1032212281@pfur.ru


## Вводная часть

### Цели и задачи

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

1. Используя Jupyter Lab, повторите примеры из раздела 3.2.
2. Выполните задания для самостоятельной работы (раздел 3.4).

### Результаты

Поставленные боевые задачи были выполнены, все цели достигнуты.

## Выполнение лабораторной работы

lab4.ipynb

Code | Notebook | Julia 1.11.0-rc3

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

[1]:
```
4×3 Matrix{Int64}:
  8  14   7
 14  15  18
 10   2  20
  6   8  11
```

[2]:
```
# Поэлементная сумма:
sum(a)
# Поэлементная сумма по столбцам:
sum(a,dims=1)
# Поэлементная сумма по строкам:
sum(a,dims=2)
```

[2]:
```
4×1 Matrix{Int64}:
 29
 47
 32
 25
```

[3]:
```
# Поэлементное произведение:
prod(a)
# Поэлементное произведение по столбцам:
prod(a,dims=1)
# Поэлементное произведение по строкам:
prod(a,dims=2)
```

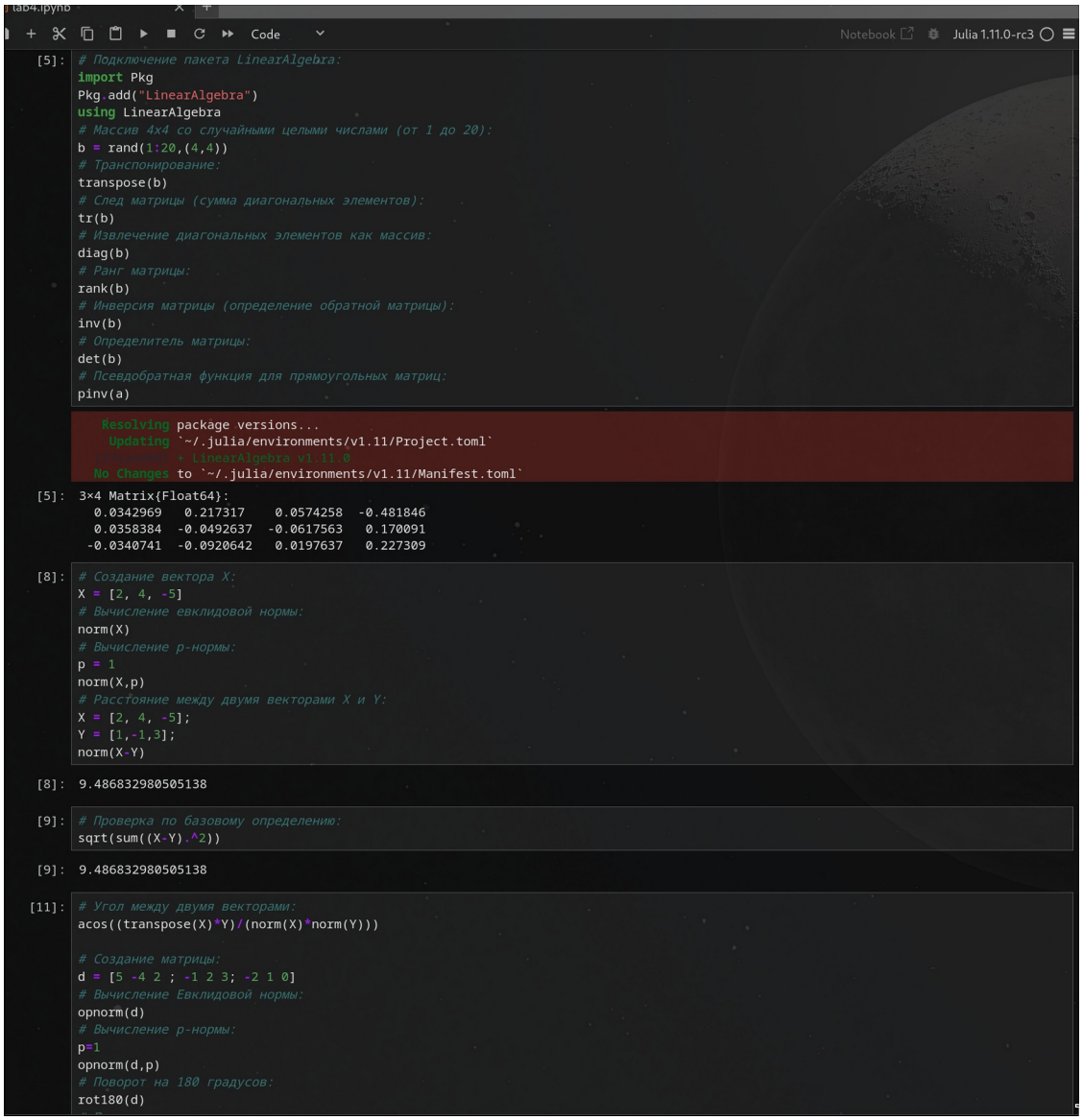
[3]:
```
4×1 Matrix{Int64}:
   784
  3780
   400
   528
```

[4]:
```
# Подключение пакета Statistics:
import Pkg
Pkg.add("Statistics")
using Statistics
# Вычисление среднего значения массива:
mean(a)
# Среднее по столбцам:
mean(a,dims=1)
# Среднее по строкам:
mean(a,dims=2)
```

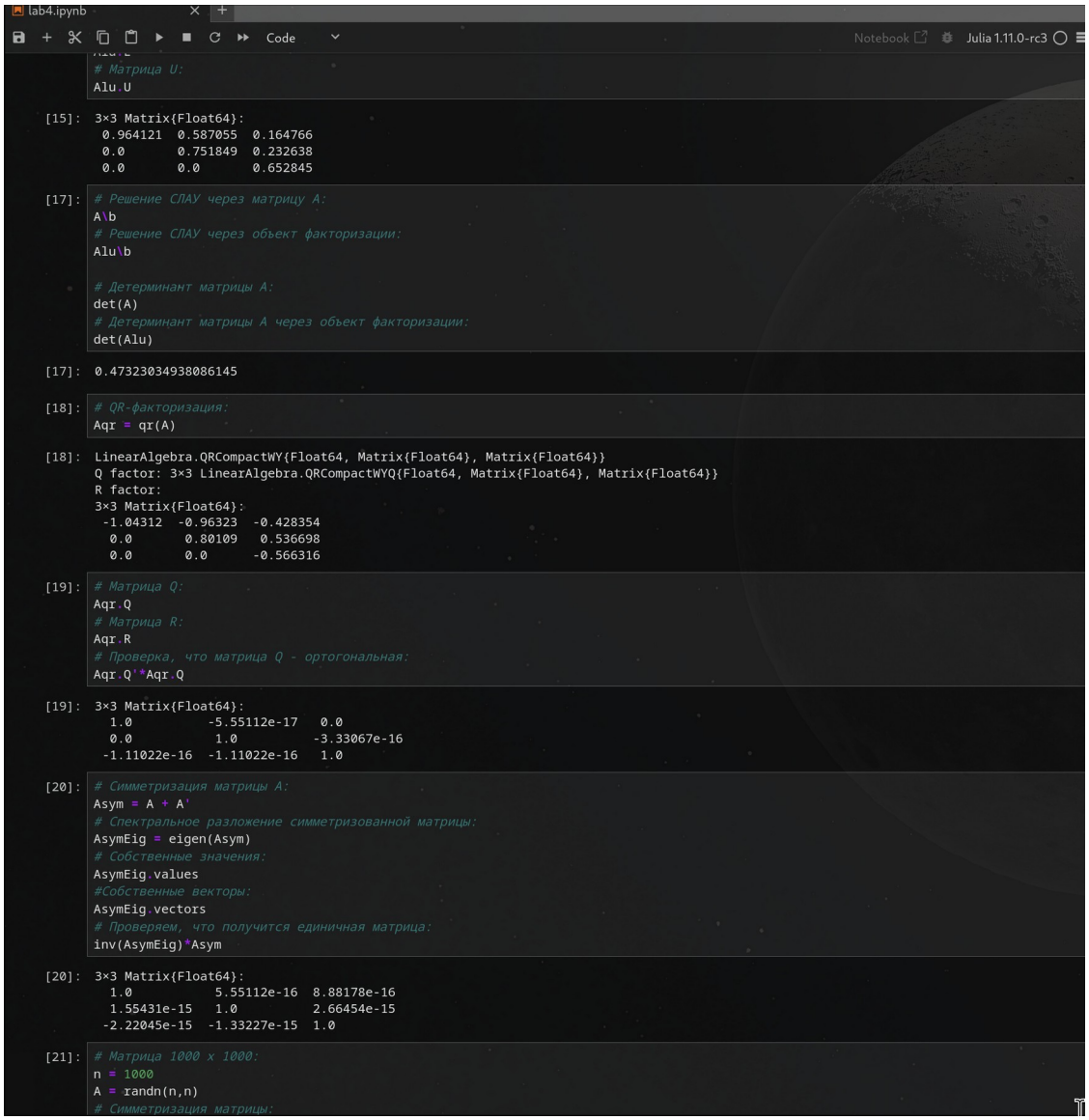
```
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
    Updating `~/.julia/environments/v1.11/Project.toml`
  [10745b16] + Statistics v1.11.1
  No Changes to `~/.julia/environments/v1.11/Manifest.toml`
```

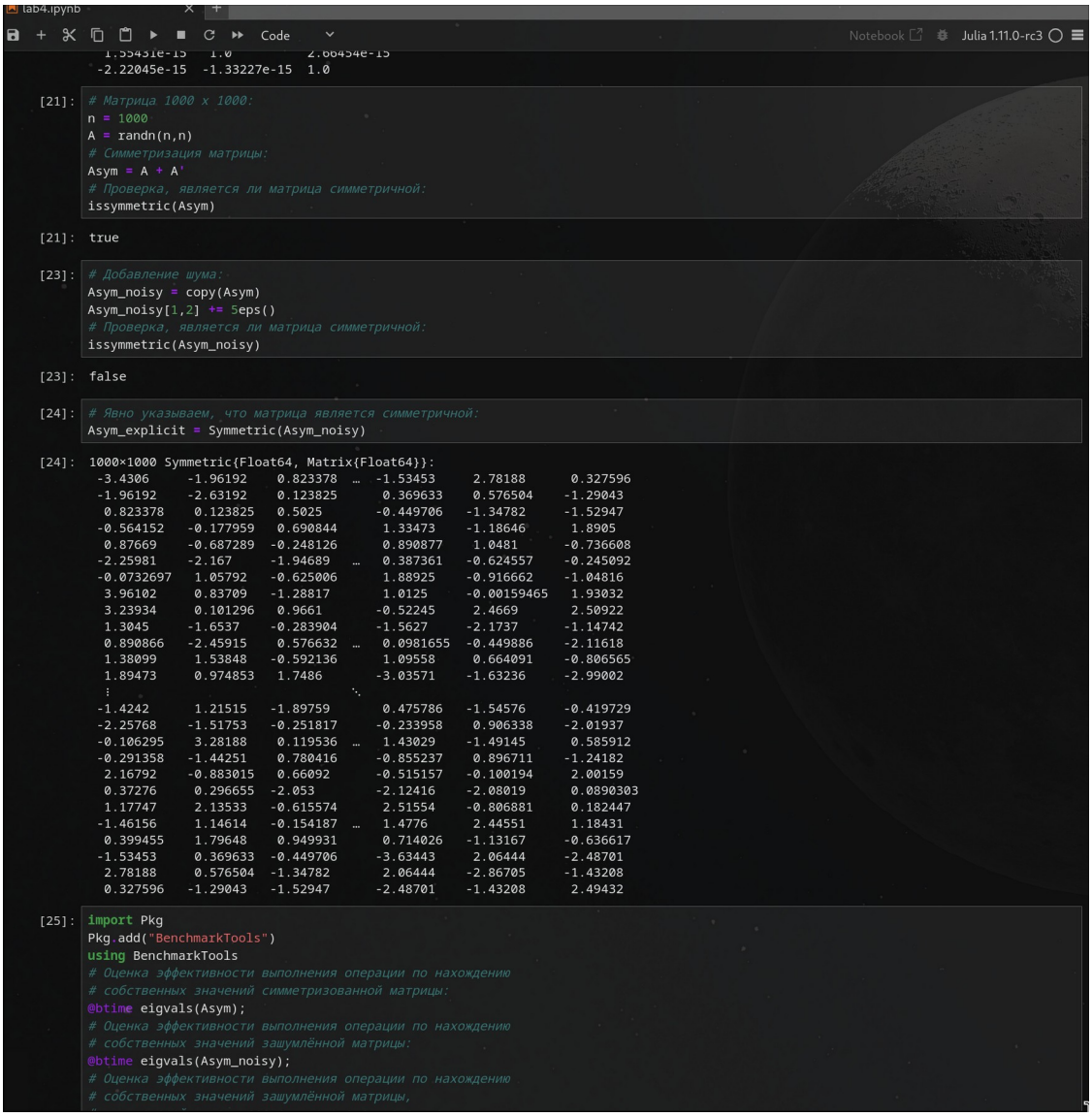
[4]:
```
4×1 Matrix{Float64}:
  9.666666666666666
 15.666666666666666
 10.666666666666666
  8.333333333333334
```

[5]:
```
# 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))
```


lab4.ipynb
Code
Notebook
Julia 1.11.0-rc3
[5]:
# 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`
+ LinearAlgebra v1.11.0
No Changes to `~/.julia/environments/v1.11/Manifest.toml`
[5]:
3×4 Matrix{Float64}:
0.0342969 0.217317 0.0574258 -0.481846
0.0358384 -0.0492637 -0.0617563 0.170091
-0.0340741 -0.0920642 0.0197637 0.227309
[8]:
# Создание вектора X:
X = [2, 4, -5]
# Вычисление евклидовой нормы:
norm(X)
# Вычисление p-нормы:
p = 1
norm(X,p)
# Расстояние между двумя векторами X и Y:
X = [2, 4, -5];
Y = [1,-1,3];
norm(X-Y)
[8]:
9.486832980505138
[9]:
# Проверка по базовому определению:
sqrt(sum((X-Y).^2))
[9]:
9.486832980505138
[11]:
# 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)


Code
Notebook
Julia 1.11.0-rc3
# Переворачивание строк:
reverse(d,dims=1)
# Переворачивание столбцов
reverse(d,dims=2)
[11]:
3×3 Matrix{Int64}:
2 -4 5
3 2 -1
0 1 -2
[12]:
# 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
[12]:
-17
[13]:
# 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
[13]:
3-element Vector{Float64}:
0.9999999999999999
1.0000000000000002
1.0
[14]:
Alu = lu(A)
[14]:
LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
3×3 Matrix{Float64}:
1.0 0.0 0.0
0.331592 1.0 0.0
0.246226 0.570689 1.0
U factor:
3×3 Matrix{Float64}:
0.964121 0.587055 0.164766
0.0 0.751849 0.232638
0.0 0.0 0.652845
[15]:
# Матрица перестановок:
Alu.P
# Вектор перестановок:
Alu.p
# Матрица L:
Alu.L


lab4.ipynb
Code
Notebook
Julia 1.11.0-rc3
# Матрица U:
Alu.U
[15]:
3×3 Matrix{Float64}:
0.964121 0.587055 0.164766
0.0 0.751849 0.232638
0.0 0.0 0.652845
[17]:
# Решение СЛАУ через матрицу A:
A\b
# Решение СЛАУ через объект факторизации:
Alu\b
# Детерминант матрицы A:
det(A)
# Детерминант матрицы A через объект факторизации:
det(Alu)
[17]:
0.47323034938086145
[18]:
# QR-факторизация:
Aqr = qr(A)
[18]:
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor: 3×3 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}
R factor:
3×3 Matrix{Float64}:
-1.04312 -0.96323 -0.428354
0.0 0.80109 0.536698
0.0 0.0 -0.566316
[19]:
# Матрица Q:
Aqr.Q
# Матрица R:
Aqr.R
# Проверка, что матрица Q - ортогональная:
Aqr.Q'*Aqr.Q
[19]:
3×3 Matrix{Float64}:
1.0 -5.55112e-17 0.0
0.0 1.0 -3.33067e-16
-1.11022e-16 -1.11022e-16 1.0
[20]:
# Симметризация матрицы A:
Asym = A + A'
# Спектральное разложение симметризованной матрицы:
AsymEig = eigen(Asym)
# Собственные значения:
AsymEig.values
#Собственные векторы:
AsymEig.vectors
# Проверяем, что получится единичная матрица:
inv(AsymEig)*Asym
[20]:
3×3 Matrix{Float64}:
1.0 5.55112e-16 8.88178e-16
1.55431e-15 1.0 2.66454e-15
-2.22045e-15 -1.33227e-15 1.0
[21]:
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:


tab4.ipynb
Code
Notebook
Julia 1.11.0-rc3
1.55431e-15 1.0 2.66454e-15
-2.22045e-15 -1.33227e-15 1.0
[21]:
# Матрица 1000 x 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:
Asym = A + A'
# Проверка, является ли матрица симметричной:
issymmetric(Asym)
[21]:
true
[23]:
# Добавление шума:
Asym_noisy = copy(Asym)
Asym_noisy[1,2] += 5eps()
# Проверка, является ли матрица симметричной:
issymmetric(Asym_noisy)
[23]:
false
[24]:
# Явно указываем, что матрица является симметричной:
Asym_explicit = Symmetric(Asym_noisy)
[24]:
1000×1000 Symmetric{Float64, Matrix{Float64}}:
-3.4306 -1.96192 0.823378 … -1.53453 2.78188 0.327596
-1.96192 -2.63192 0.123825 0.369633 0.576504 -1.29043
0.823378 0.123825 0.5025 -0.449706 -1.34782 -1.52947
-0.564152 -0.177959 0.690844 1.33473 -1.18646 1.8905
0.87669 -0.687289 -0.248126 0.890877 1.0481 -0.736608
-2.25981 -2.167 -1.94689 … 0.387361 -0.624557 -0.245092
-0.0732697 1.05792 -0.625006 1.88925 -0.916662 -1.04816
3.96102 0.83709 -1.28817 1.0125 -0.00159465 1.93032
3.23934 0.101296 0.9661 -0.52245 2.4669 2.50922
1.3045 -1.6537 -0.283904 -1.5627 -2.1737 -1.14742
0.890866 -2.45915 0.576632 … 0.0981655 -0.449886 -2.11618
1.38099 1.53848 -0.592136 1.09558 0.664091 -0.806565
1.89473 0.974853 1.7486 -3.03571 -1.63236 -2.99002
⋮ ⋱
-1.4242 1.21515 -1.89759 0.475786 -1.54576 -0.419729
-2.25768 -1.51753 -0.251817 -0.233958 0.906338 -2.01937
-0.106295 3.28188 0.119536 … 1.43029 -1.49145 0.585912
-0.291358 -1.44251 0.780416 -0.855237 0.896711 -1.24182
2.16792 -0.883015 0.66092 -0.515157 -0.100194 2.00159
0.37276 0.296655 -2.053 -2.12416 -2.08019 0.0890303
1.17747 2.13533 -0.615574 2.51554 -0.806881 0.182447
-1.46156 1.14614 -0.154187 … 1.4776 2.44551 1.18431
0.399455 1.79648 0.949931 0.714026 -1.13167 -0.636617
-1.53453 0.369633 -0.449706 -3.63443 2.06444 -2.48701
2.78188 0.576504 -1.34782 2.06444 -2.86705 -1.43208
0.327596 -1.29043 -1.52947 -2.48701 -1.43208 2.49432
[25]:
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);
```

```
   Resolving package versions...
   Installed BenchmarkTools ─ v1.5.0
    Updating `~/.julia/environments/v1.11/Project.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
    Updating `~/.julia/environments/v1.11/Manifest.toml`
  [6e4b80f9] + BenchmarkTools v1.5.0
  [de0858da] + Printf v1.11.0
Precompiling project...
    747.6 ms  ✓ BenchmarkTools
  1 dependency successfully precompiled in 1 seconds. 54 already precompiled.
  25.796 ms (21 allocations: 7.99 MiB)
  173.437 ms (27 allocations: 7.93 MiB)
  25.535 ms (21 allocations: 7.99 MiB)
```

[26]:
```
# Трёхдиагональная матрица 1000000 x 1000000:
n = 1000000;
A = SymTridiagonal(randn(n), randn(n-1))
# Оценка эффективности выполнения операции по нахождению
# собственных значений:
@btime eigmax(A)
```

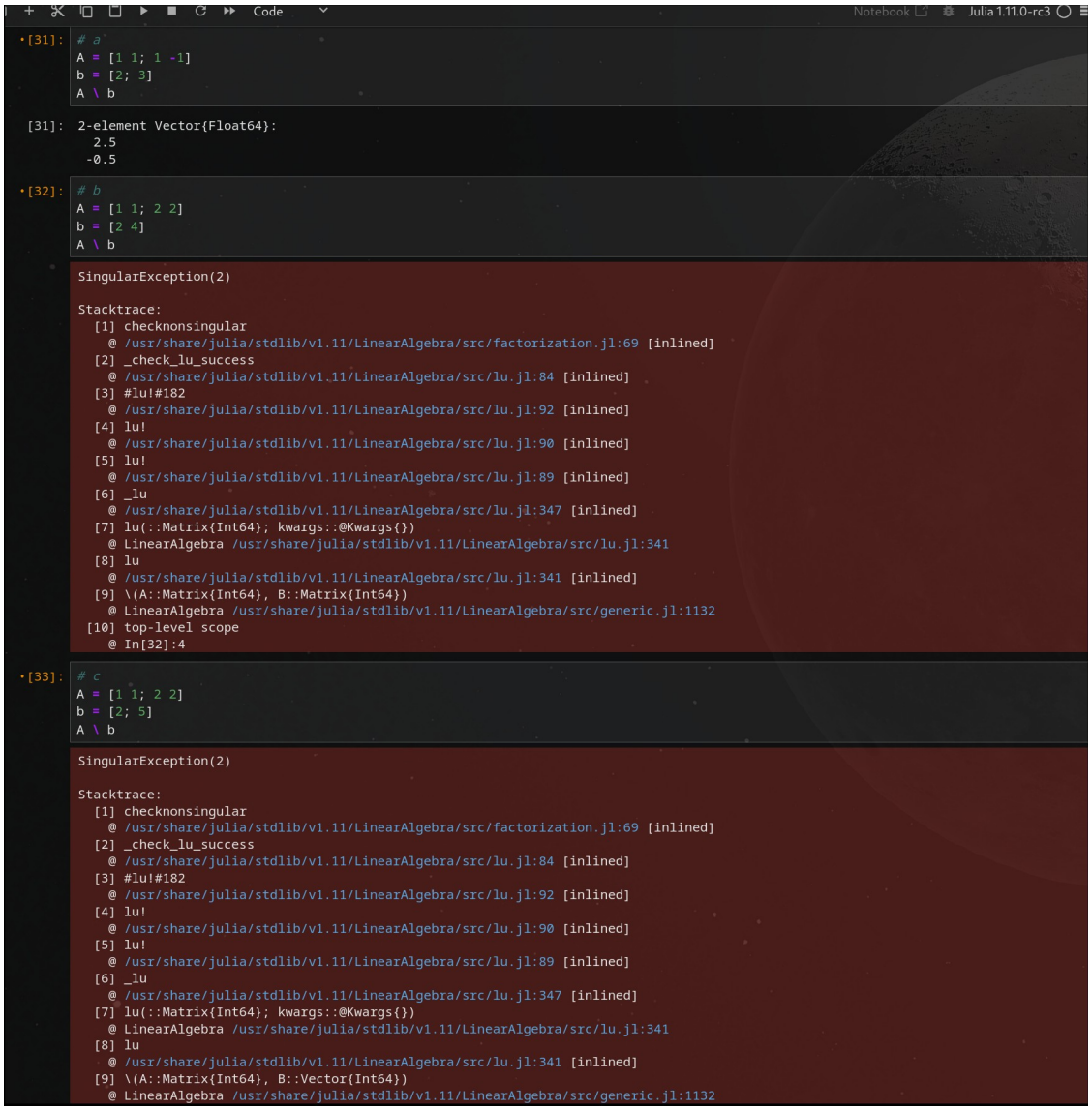
```
  289.920 ms (44 allocations: 183.11 MiB)
```

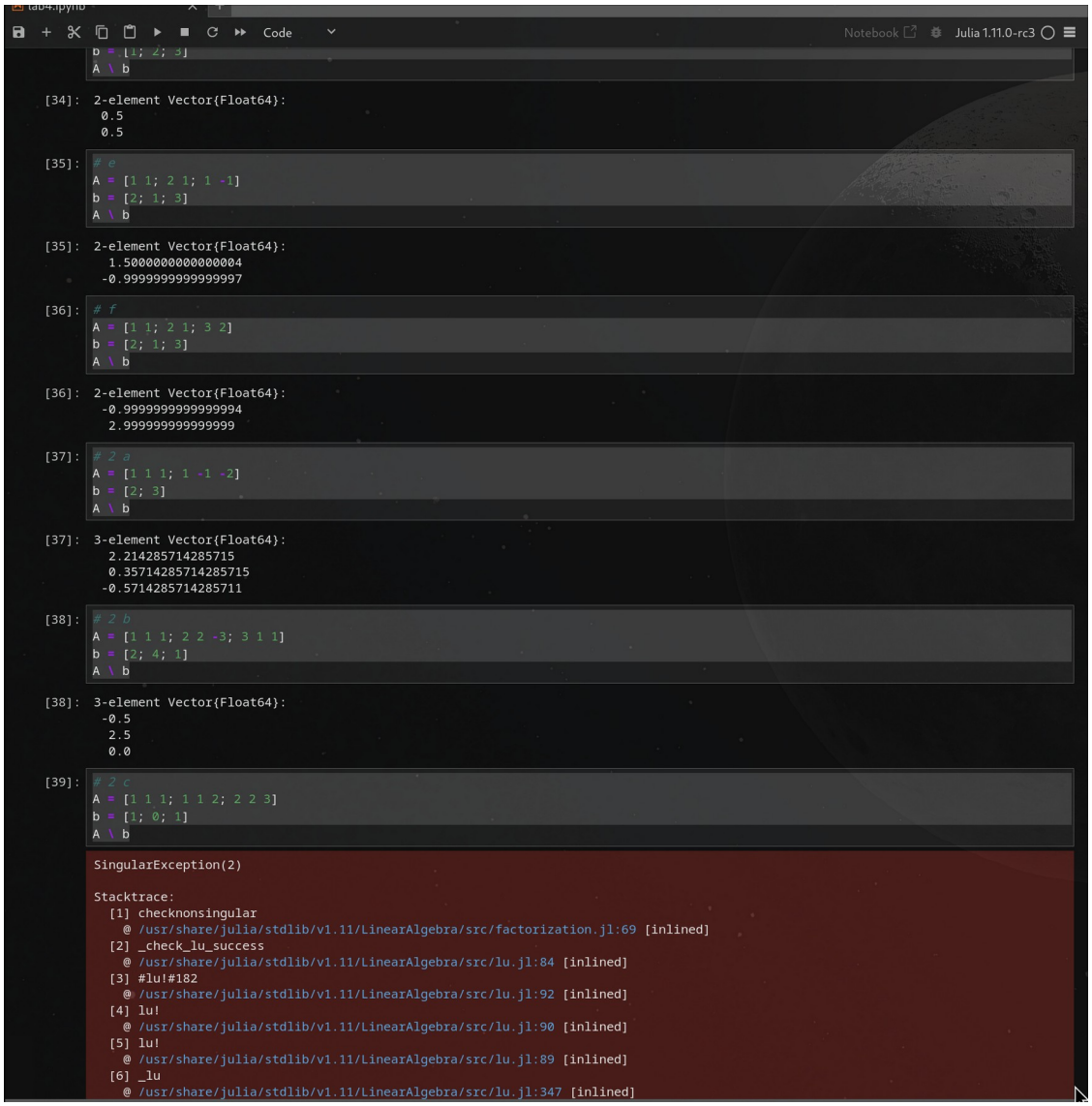
[26]:
```
6.6264726147269055
```

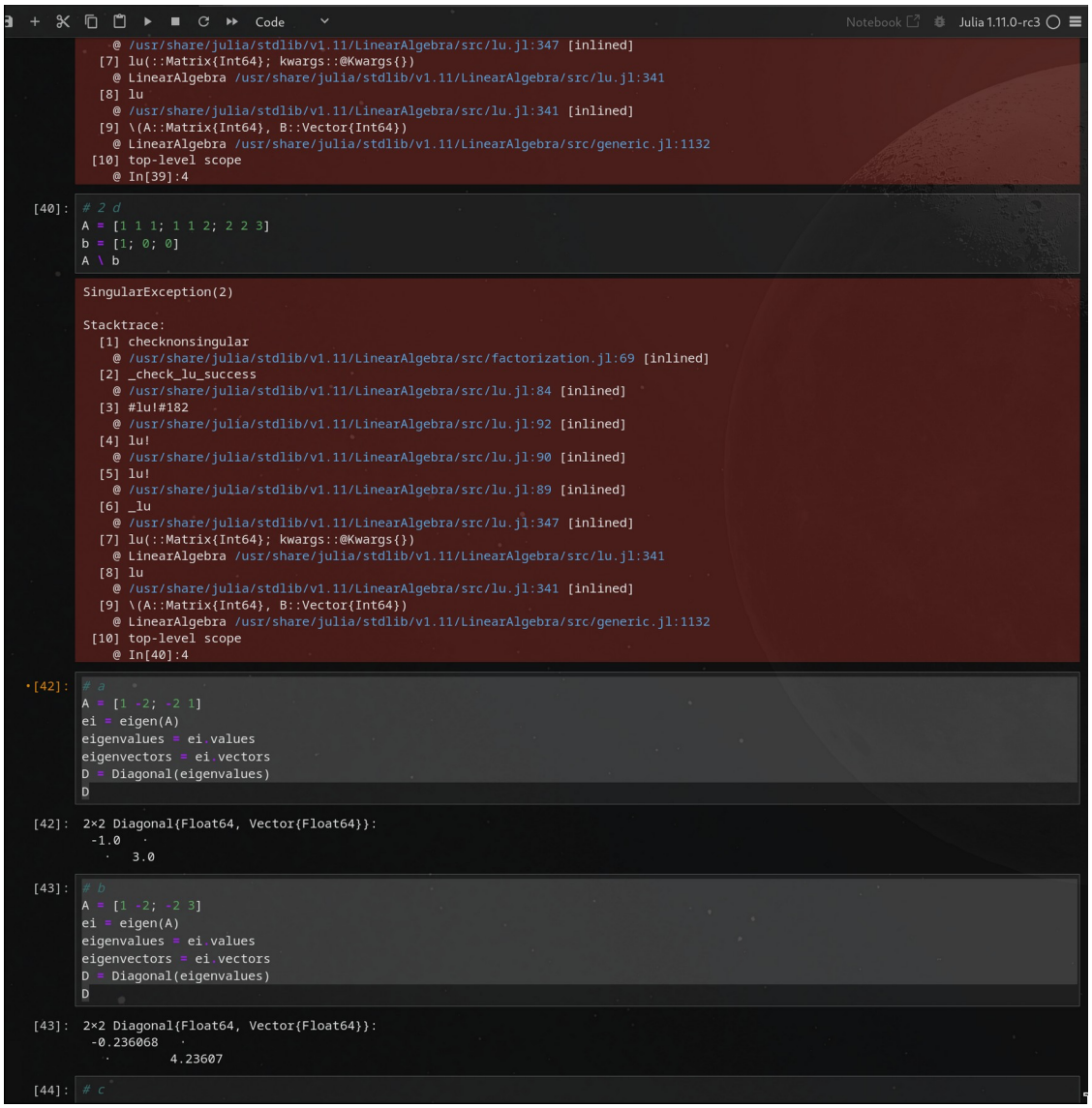
[27]:
```
# Матрица с рациональными элементами:
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
# Единичный вектор:
x = fill(1, 3)
# Задаём вектор b:
b = Arational*x
# Решение исходного уравнения получаем с помощью функции \
# (убеждаемся, что x - единичный вектор):
Arational\b
# LU-разложение:
lu(Arational)
```

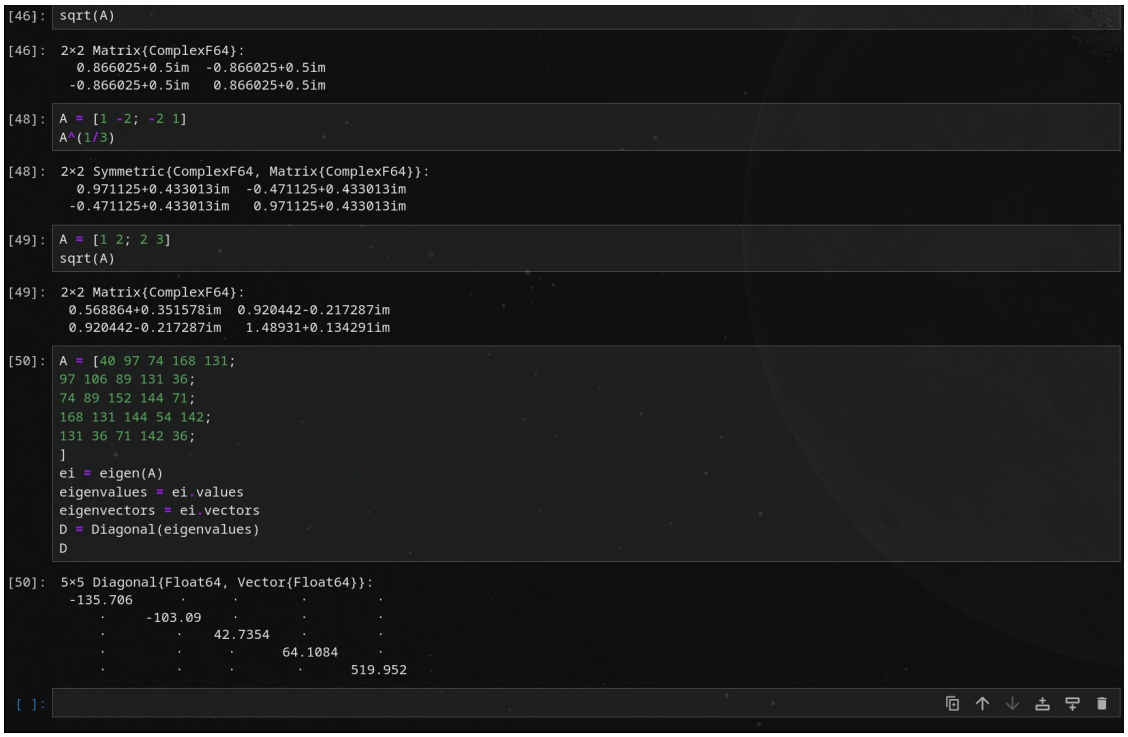
```
SingularException(3)

Stacktrace:
  [1] checknonsingular
    @ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/factorization.jl:69 [inlined]
  [2] _check_lu_success
    @ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:84 [inlined]
  [3] generic_lufact!(A::Matrix{Rational{BigInt}}, pivot::RowMaximum; check::Bool, allowsingular::Bool)
    @ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:214
  [4] generic_lufact!
    @ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:152 [inlined]
  [5] lu! (repeats 2 times)
    @ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:151 [inlined]
  [6] _lu
    @ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:347 [inlined]
  [7] lu(::Matrix{Rational{BigInt}}; kwargs::@Kwargs{})
    @ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341
  [8] lu
    @ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341 [inlined]
  [9] \(A::Matrix{Rational{BigInt}}, B::Vector{Rational{BigInt}})
    @ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/generic.jl:1132
 [10] top-level scope
    @ In[27]:9
```

[31]:
```
#
```


Code
Notebook
Julia 1.11.0-rc3
•[31]:
# a
A = [1 1; 1 -1]
b = [2; 3]
A \ b
[31]:
2-element Vector{Float64}:
2.5
-0.5
•[32]:
# b
A = [1 1; 2 2]
b = [2 4]
A \ b
SingularException(2)
Stacktrace:
[1] checknonsingular
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/factorization.jl:69 [inlined]
[2] _check_lu_success
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:84 [inlined]
[3] #lu!#182
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:92 [inlined]
[4] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:90 [inlined]
[5] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:89 [inlined]
[6] _lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:347 [inlined]
[7] lu(::Matrix{Int64}; kwargs::@Kwargs{})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341
[8] lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341 [inlined]
[9] \(A::Matrix{Int64}, B::Matrix{Int64})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/generic.jl:1132
[10] top-level scope
@ In[32]:4
•[33]:
# c
A = [1 1; 2 2]
b = [2; 5]
A \ b
SingularException(2)
Stacktrace:
[1] checknonsingular
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/factorization.jl:69 [inlined]
[2] _check_lu_success
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:84 [inlined]
[3] #lu!#182
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:92 [inlined]
[4] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:90 [inlined]
[5] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:89 [inlined]
[6] _lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:347 [inlined]
[7] lu(::Matrix{Int64}; kwargs::@Kwargs{})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341
[8] lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341 [inlined]
[9] \(A::Matrix{Int64}, B::Vector{Int64})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/generic.jl:1132


tab4.ipynb
Code
Notebook
Julia 1.11.0-rc3
b = [1; 2; 3]
A \ b
[34]:
2-element Vector{Float64}:
0.5
0.5
[35]:
# e
A = [1 1; 2 1; 1 -1]
b = [2; 1; 3]
A \ b
[35]:
2-element Vector{Float64}:
1.5000000000000004
-0.9999999999999997
[36]:
# f
A = [1 1; 2 1; 3 2]
b = [2; 1; 3]
A \ b
[36]:
2-element Vector{Float64}:
-0.9999999999999994
2.999999999999999
[37]:
# 2 a
A = [1 1 1; 1 -1 -2]
b = [2; 3]
A \ b
[37]:
3-element Vector{Float64}:
2.214285714285715
0.3571428571428571
-0.5714285714285711
[38]:
# 2 b
A = [1 1 1; 2 2 -3; 3 1 1]
b = [2; 4; 1]
A \ b
[38]:
3-element Vector{Float64}:
-0.5
2.5
0.0
[39]:
# 2 c
A = [1 1 1; 1 1 2; 2 2 3]
b = [1; 0; 1]
A \ b
SingularException(2)
Stacktrace:
[1] checknonsingular
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/factorization.jl:69 [inlined]
[2] _check_lu_success
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:84 [inlined]
[3] #lu!#182
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:92 [inlined]
[4] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:90 [inlined]
[5] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:89 [inlined]
[6] _lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:347 [inlined]


Code
Notebook
Julia 1.11.0-rc3
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:347 [inlined]
[7] lu(::Matrix{Int64}; kwargs::@Kwargs{})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341
[8] lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341 [inlined]
[9] \(A::Matrix{Int64}, B::Vector{Int64})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/generic.jl:1132
[10] top-level scope
@ In[39]:4
[40]:
# 2 d
A = [1 1 1; 1 1 2; 2 2 3]
b = [1; 0; 0]
A \ b
SingularException(2)
Stacktrace:
[1] checknonsingular
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/factorization.jl:69 [inlined]
[2] _check_lu_success
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:84 [inlined]
[3] #lu!#182
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:92 [inlined]
[4] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:90 [inlined]
[5] lu!
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:89 [inlined]
[6] _lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:347 [inlined]
[7] lu(::Matrix{Int64}; kwargs::@Kwargs{})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341
[8] lu
@ /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/lu.jl:341 [inlined]
[9] \(A::Matrix{Int64}, B::Vector{Int64})
@ LinearAlgebra /usr/share/julia/stdlib/v1.11/LinearAlgebra/src/generic.jl:1132
[10] top-level scope
@ In[40]:4
[42]:
# a
A = [1 -2; -2 1]
ei = eigen(A)
eigenvalues = ei.values
eigenvectors = ei.vectors
D = Diagonal(eigenvalues)
D
[42]:
2×2 Diagonal{Float64, Vector{Float64}}:
-1.0 ·
· 3.0
[43]:
# b
A = [1 -2; -2 3]
ei = eigen(A)
eigenvalues = ei.values
eigenvectors = ei.vectors
D = Diagonal(eigenvalues)
D
[43]:
2×2 Diagonal{Float64, Vector{Float64}}:
-0.236068 ·
· 4.23607
[44]:
# c




## Выводы

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

## Список литературы