Лабораторная работа №4. Компьютерный практикум по статистическому анализу данных

Леснухин Даниил Дмитриевич
Российский университет дружбы народов
Москва

Цель работы

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

Задание

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

Теоретическое введение

Julia – высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений [@julialang]. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia [@juliadoc].

Выполнение лабораторной работы

Выполнение примеров

Выполним примеры из раздела про поэлементные операции над многомерными массивами .

Выполним примеры из раздела про транспонирование,след,ранг,определительи инверсия матрицы

Выполним примеры из раздела про вычисление нормы векторов и матриц, повроты, вращения

Выполним примеры из раздела про матричное умножение, единичную матрицу, скалярное произведение

Выполним примеры из раздела про факторизацию и специальные матричные структуры

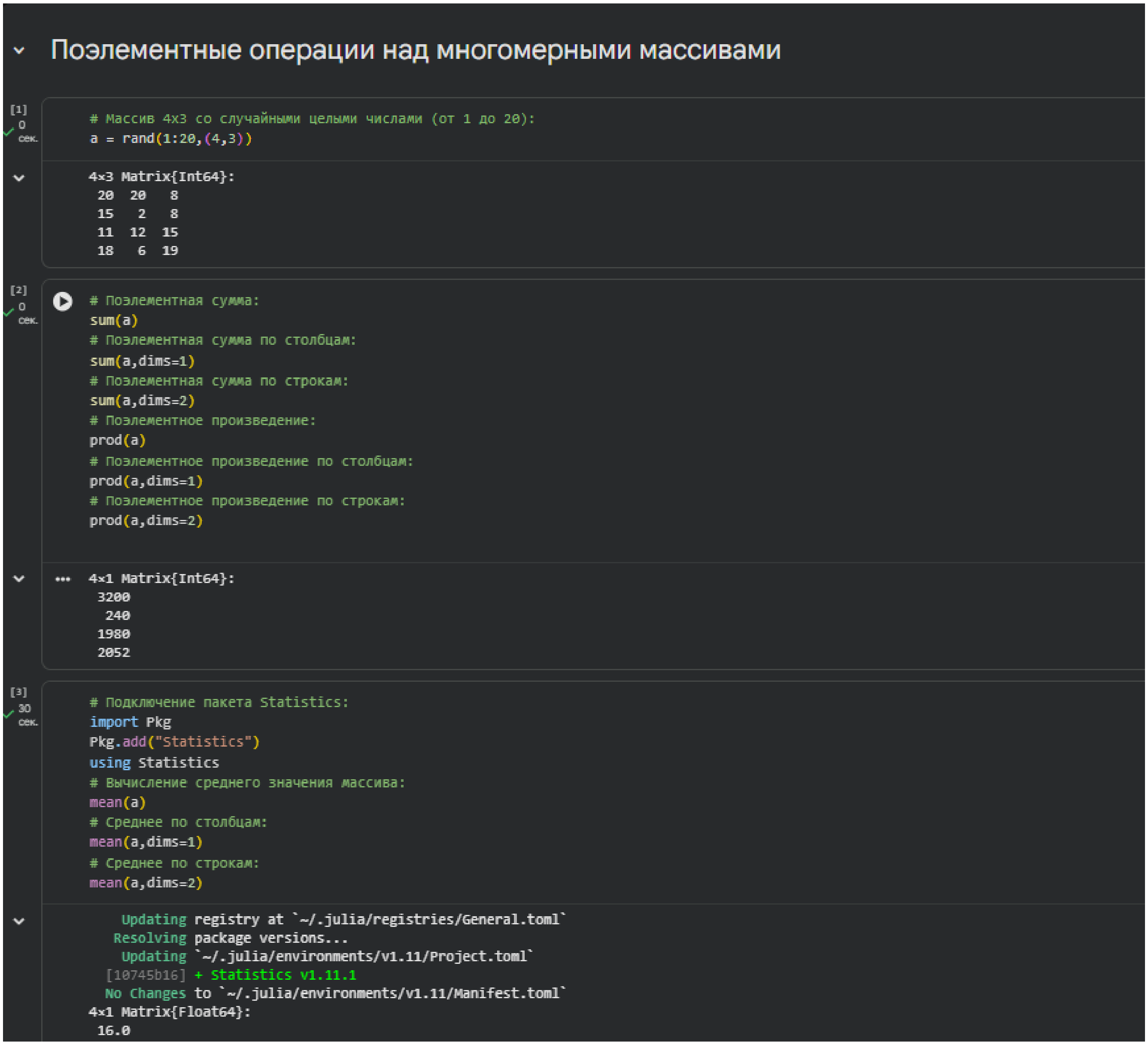
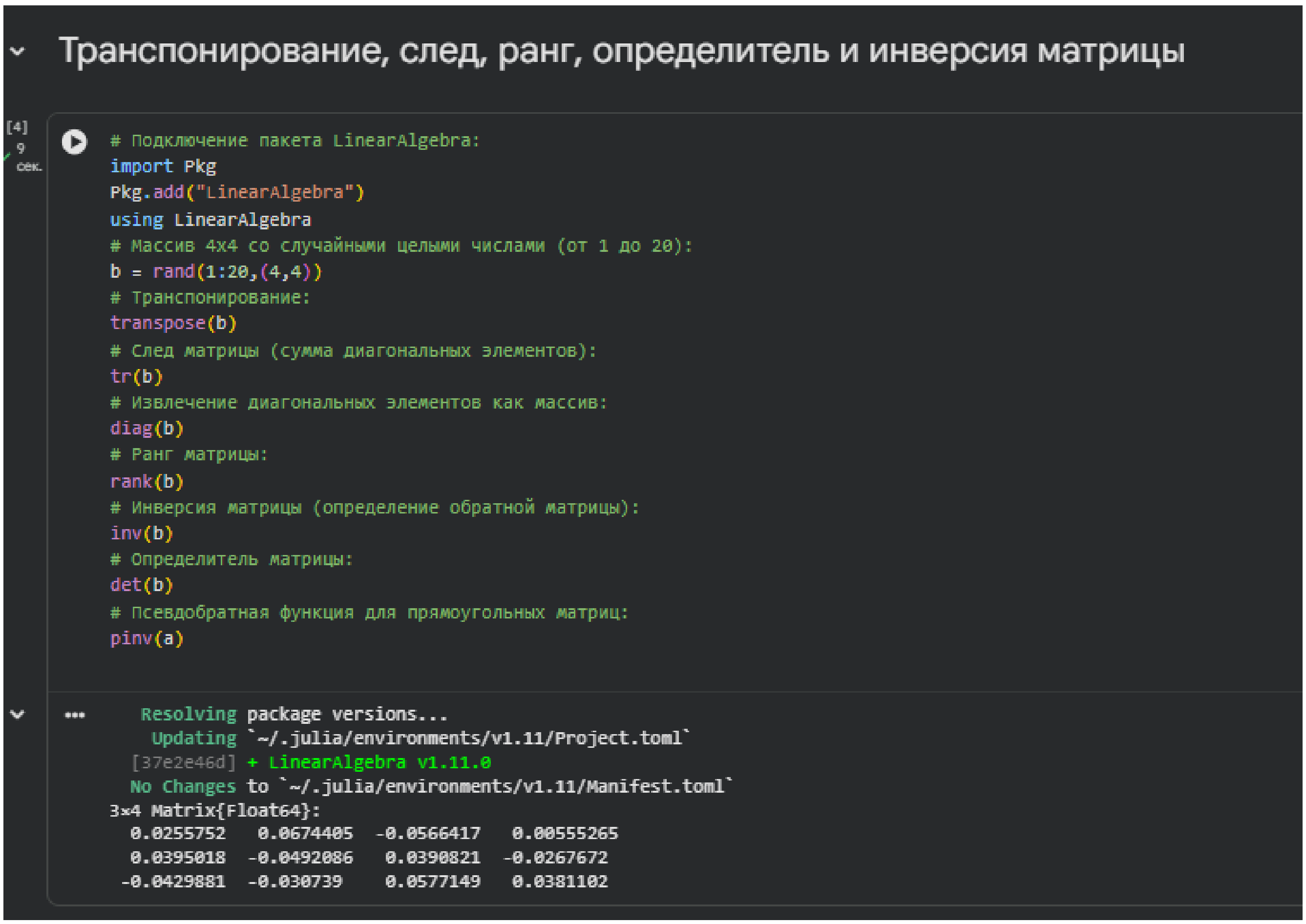


Рис. 1: Поэлементные операции над многомерными массивами



Транспонирование, след, ранг, определитель и инверсия матрицы

```
# 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`
3×4 Matrix{Float64}:
  0.0255752   0.0674405  -0.0566417   0.00555265
  0.0395018  -0.0492086   0.0390821  -0.0267672
 -0.0429881  -0.030739    0.0577149   0.0381102
```

Рис. 2: Транспонирование, след, ранг, определитель и инверсия матрицы

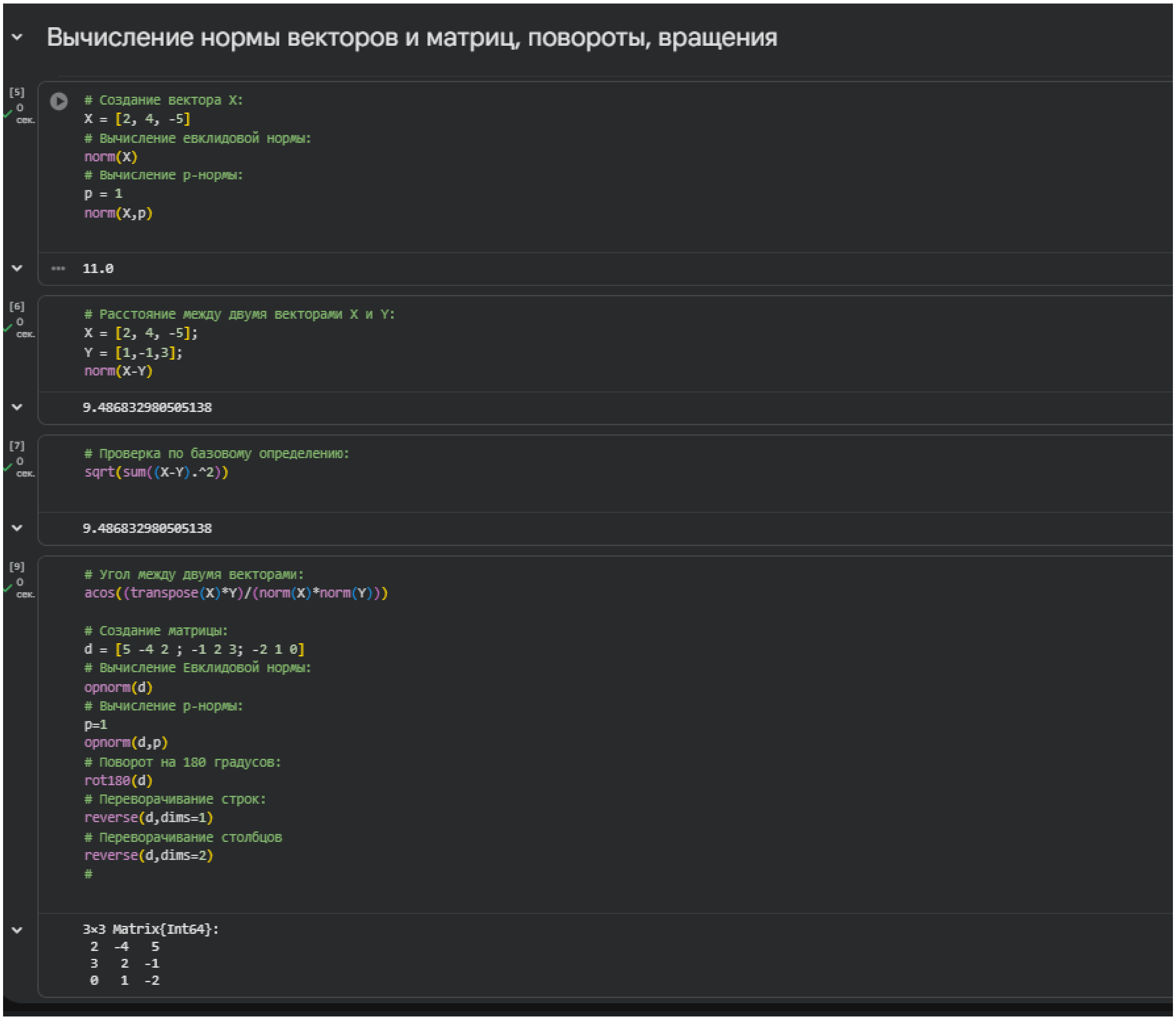


Рис. 3: Вычисление нормы векторов и матриц, повороты, вращения

Матричное умножение, единичная матрица, скалярное произведение

[10]

```
# 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
```

Рис. 4: Матричное умножение,единичная матрица,скалярное произведение

Факторизация. Специальные матричные структуры

[11]

```
# 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]

```
# 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
```

[14]

```
# Решение СЛАУ через матрицу A:
A\b
# Решение СЛАУ через объект факторизации:
Alu\b
# Детерминант матрицы A:
det(A)
```

```
0.020283246786260464
```

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

```
3×3 Matrix{Float64}:
 1.0   0.0          -1.11022e-16
 0.0   1.0          -1.11022e-16
 0.0  -2.77556e-17   1.0
```

```
# 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
```

```
3×3 Matrix{Float64}:
 1.0          -5.77316e-15   1.02141e-14
 2.22045e-15   1.0          -5.9952e-15
 1.77636e-15   1.9984e-15    1.0
```

```
# Матрица 1000 х 1000:
n = 1000
A = randn(n,n)
# Симметризация матрицы:
Asym = A + A'
# 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)
```

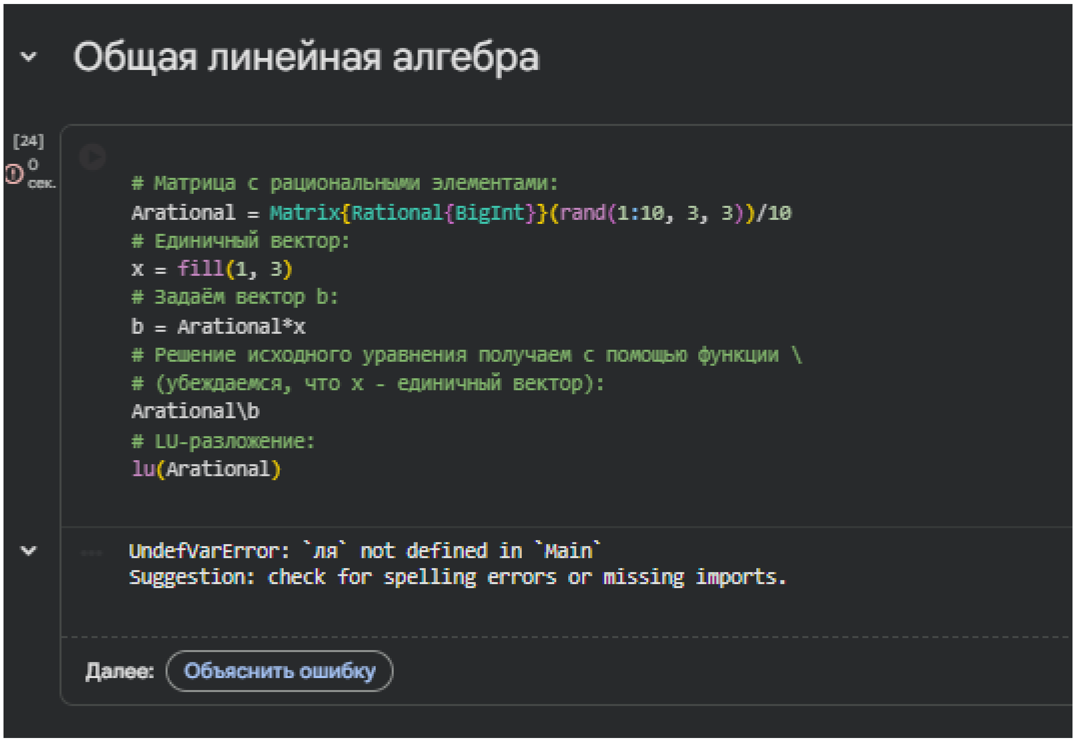
```
false
```

```julia
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 ─ v1.6.3
    Updating `~/.julia/environments/v1.11/Project.toml`
  [6e4b80f9] + BenchmarkTools v1.6.3
    Updating `~/.julia/environments/v1.11/Manifest.toml`
  [6e4b80f9] + BenchmarkTools v1.6.3
  [9abbd945] + Profile v1.11.0
Precompiling project...
   2710.9 ms  ✓ BenchmarkTools
  1 dependency successfully precompiled in 4 seconds. 469 already precompiled.
  123.690 ms (21 allocations: 7.99 MiB)
  703.937 ms (27 allocations: 7.93 MiB)
  123.717 ms (21 allocations: 7.99 MiB)
```

```julia
# 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)
```

```
  709.663 ms (44 allocations: 183.11 MiB)
6.934806147575959
```



Задания для самостоятельного выполнения

Произведение векторов

Теперь перейдем к выполнению заданий для самостоятельной работы.

1. Зададим вектор v. Умножим вектор v скалярно сам на себя и сохраним результат в dot_v.
2. Умножим v матрично на себя (внешнее произведение), присвоив результат переменной outer_v

Системы линейных уравнений

1. Решим СЛАУ с двумя неизвестными

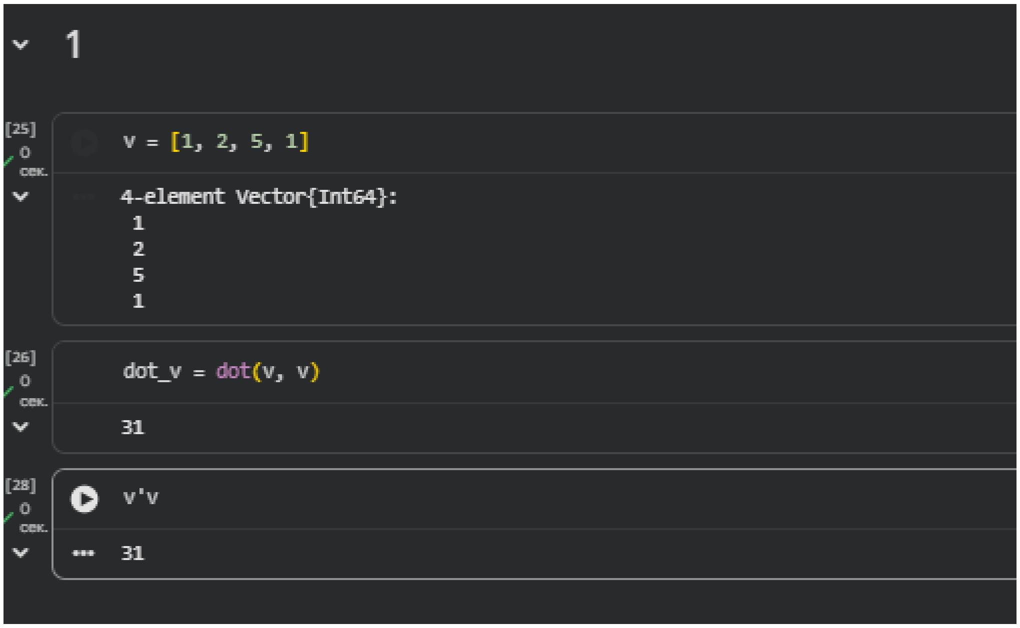
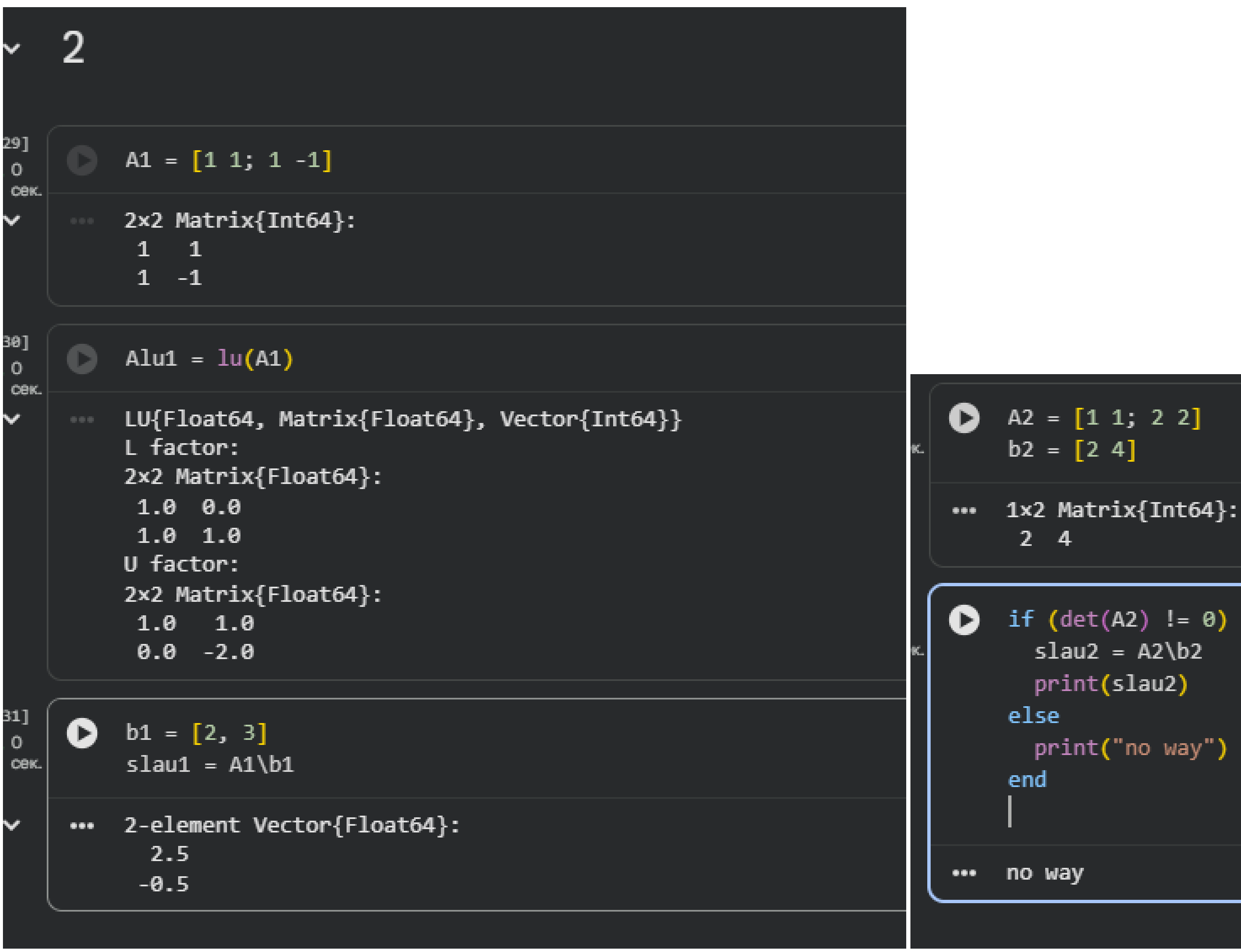


Рис. 5: Произведение векторов



```
A3 = [1 1; 2 2]
b3 = [2 5]

if (det(A3) != 0)
  slau2 = A3\b3
  print(slau3)
else
  print("no way")
end
```

```
no way
```

```
A4 = [1 1; 2 2; 3 3]
b4 = [1, 2, 3]
slau4 = A4\b4
```

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

```
A5 = [1 1; 2 1; 1 -1]
b5 = [2, 1, 3]
slau5 = A5\b5
```

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

Решим СЛАУ с тремя неизвестными).

Операции с матрицами

1. Приведем матрицы к диагональному виду .
2. Вычислим .
3. Найдем собственные значения матрицы A. Создадим диагональную матрицу из собственных значений матрицы A. Создадим нижнедиагональную матрицу из матрицы A. Оценим эффективность выполняемых операций

Линейные модели экономики

Линейная модель 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.

1. Матрица A называется продуктивной, если решение x системы при любой неотрицательной правой части y имеет только неотрицательные элементы x_i. Используя это определение, проверим, являются ли матрицы продуктивными
2. Критерий продуктивности: матрица A является продуктивной тогда и только тогда, когда все элементы матрицы $(E - A)^{-1}$ являются неотрицательными числами. Используя этот

```
[42]
A1 = [1 1 1; 1 -1 2]
b1 = [2, 3]
slau1 = A1\b1
```

```
3-element Vector{Float64}:
 0.7857142857142853
 0.07142857142857145
 1.1428571428571426
```

```
[45]
A2 = [1 1 1; 2 2 -3; 3 1 1]
b2 = [2, 4, 1]
slau2 = A2\b2
```

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

```
[46]
A3 = [1 1 1; 1 1 2; 2 2 3]
b3 = [1, 0, 1]
if (det(A2) != 0)
  slau2 = A2\b2
  print(slau2)
else
  print("no way")
end
```

```
[-0.5, 2.5, 0.0]
```

Рис. 6: Системы линейных уравнений

```
[47]
A = [1 -2; 2 -1]

2×2 Matrix{Int64}:
 1  -2
 2  -1
```

```
[51]
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]
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]
B = [1 -2; -2 3]
Beig = eigen(B)
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
```

Рис. 7: Операции с матрицами

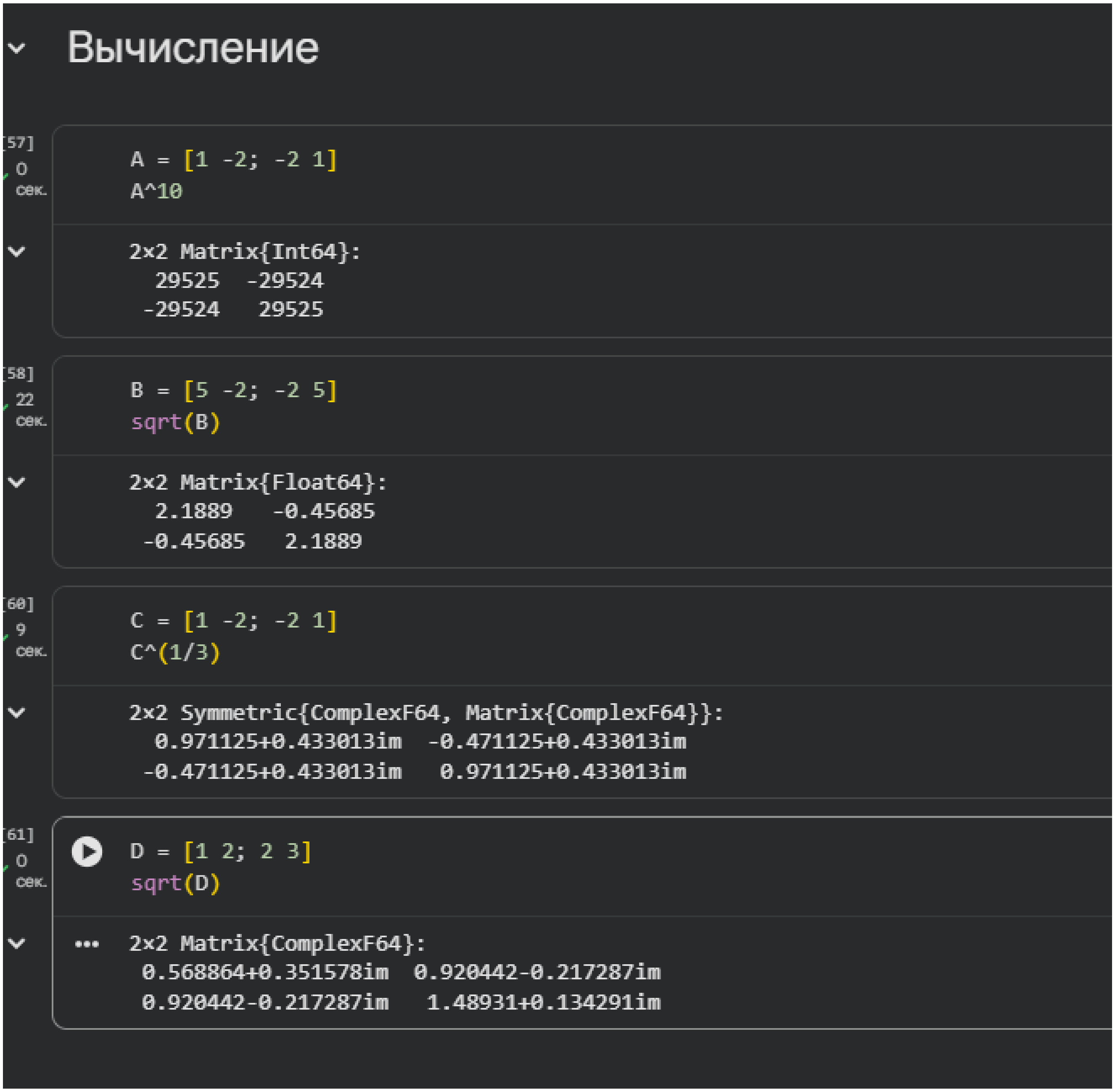


Рис. 8: Операции с матрицами

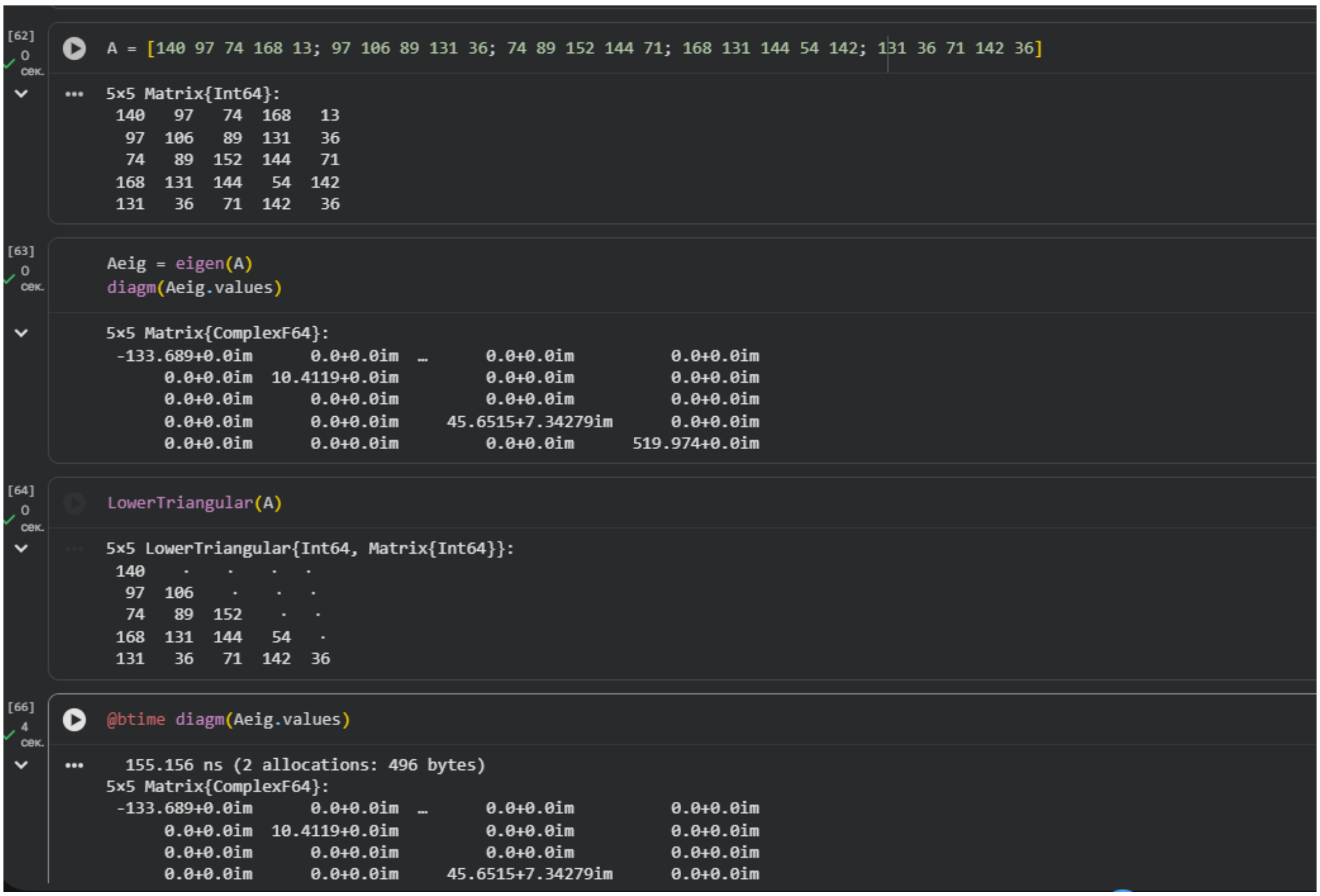


Рис. 9: Операции с матрицами

критерий, проверим, являются ли матрицы продуктивными

Выводы

В процессе выполнения данной лабораторной работы я изучил возможности специализированных пакетов Julia для выполнения и оценки эффективности операций над объектами линейной алгебры.