

# Лабораторная работа № 4

## Линейная алгебра

Герра Гарсия Паола Валентина

## Содержание

Цель работы .....	2
Задание .....	2
Теоретическое введение .....	2
Выполнение лабораторной работы .....	2
Задания для самостоятельного выполнения .....	5
Выводы .....	10
Список литературы.....	10

## Список иллюстраций

Поэлементные операции над многомерными массивами.....	2
Поэлементные операции над многомерными массивами.....	3
Транспонирование,след,ранг,определитель инверсия матрицы.....	3
Матричное умножение,единичная матрица,скалярное произведение .....	4
Факторизация.Специальные матричные структуры .....	4
Факторизация.Специальные матричные структуры .....	4
Факторизация.Специальные матричные структуры .....	5
Общаялинейная алгебра.....	5
Произведение векторов .....	5
Произведение векторов .....	6
Системы линейных уравнений .....	6
Системы линейных уравнений.....	7
Системы линейных уравнений.....	7
Операции с матрицами .....	8
Операции с матрицами .....	8
Операции с матрицами .....	9
Линейные модели экономики .....	9
Линейные модели экономики .....	10

# 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

Выполним примеры из раздела про поэлементные операции над многомерными массивами (рис. [-@fig:001]-[-@fig:002]).

The screenshot shows a JupyterLab interface with several code cells and their corresponding outputs:

- In [3]:** # Массив 4x3 со случайными целыми числами (от 1 до 20):  
a = rand(1:20,(4,3))
- Out [2]:** 4x3 Matrix{Int64}:  
19 10 14  
5 18 7  
7 16 11  
13 2 20
- In [3]:** # Поэлементная сумма:  
sum(a)  
# Поэлементная сумма по столбцам:  
sum(a,dims=1)  
# Поэлементная сумма по строкам:  
sum(a,dims=2)
- Out [3]:** 4x1 Matrix{Int64}:  
43  
38  
34  
35
- In [4]:** # Поэлементное произведение:  
prod(a)  
# Поэлементное произведение по столбцам:  
prod(a,dims=1)  
# Поэлементное произведение по строкам:  
prod(a,dims=2)
- Out [4]:** 4x1 Matrix{Int64}:  
2660  
630  
1232  
520
- In [5]:** # Подключение пакета Statistics:  
import Pkg  
Pkg.add("Statistics")

Поэлементные операции над многомерными массивами

Jupyter Lab4\_статистический\_анализ Last Checkpoint: hace 19 horas (autosaved)

In [6]: # Вычисление среднего значения массива:  
 mean(a)  
 # Среднее по столбцам:  
 mean(a,dims=1)  
 # Среднее по строкам:  
 mean(a,dims=2)

Out[6]: 4x1 Matrix{Float64}:  
 14.33333333333334  
 10.0  
 11.33333333333334  
 11.666666666666666

**Транспонирование, след, ранг, определитель и инверсия матрицы**

In [7]: # Подключение пакета LinearAlgebra:  
 import Pkg  
 Pkg.add("LinearAlgebra")  
 using LinearAlgebra  
 # Матрица 4x4 со случайными целыми числами (от 1 до 20).  
 b = rand(1:20,(4,4))

Resolving package versions...  
 Updating `~/.julia/environments/v1.12/Project.toml'  
 [37e2ed6] + LinearAlgebra v1.12.0  
 Manifest No packages added to or removed from `~/.julia/environments/v1.12/Manifest.toml`

Out[7]: 4x4 Matrix{Int64}:  
 12 19 9 11  
 14 1 8 10  
 14 13 9 2  
 10 12 2 4

In [8]: # Транспонирование:  
 transpose(b)  
 # След матрицы (сумма диагональных элементов):  
 tr(b)

## Поэлементные операции над многомерными массивами

Выполним примеры из раздела про транспонирование,след,ранг,определитель и инверсия матрицы (рис. [-@fig:003]).

Jupyter Lab4\_статистический\_анализ Last Checkpoint: hace 19 horas (autosaved)

In [9]: # Создание вектора X:  
 X = [2, 4, -5];  
 # Вычисление евклидовой нормы:  
 norm(X)  
 # Вычисление p-нормы:  
 p = 1  
 norm(X,p)

Out[9]: 11.0

In [10]: # Расстояние между двумя векторами X и Y:  
 X = [2, 4, -5];  
 Y = [1, -1, 3];  
 norm(X-Y)  
 # Проверка по базовому определению:  
 sqrt(sum((X-Y).^2))

Out[10]: 9.486832988505138

In [12]: # Угол между двумя векторами:  
 acos(transpose(X)\*Y)/norm(X)\*norm(Y))  
 # Вычисление нормы для двумерной матрицы:  
 d = [5 -4 2; -1 2 3; -2 1 0]

Out[12]: 3x3 Matrix{Int64}:  
 5 -4 2  
 -1 2 3  
 -2 1 0

In [13]: # Вычисление Евклидовой нормы:  
 opnorm(d)  
 # Вычисление p-нормы:  
 p=1  
 opnorm(d,p)  
 # Поворот на 180 градусов:  
 rot180(d)  
 # Переворачивание строк:

## Транспонирование,след,ранг,определитель и инверсия матрицы

Выполним примеры из раздела про матричное умножение,единичная матрица,скалярное произведение (рис. [-@fig:004]).

### Матричное умножение, единичная матрица, скалярное произведение

```
In [14]: # 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  
  
Out[14]: 2x4 Matrix{Int64};  
169 132 114 111  
107 76 64 80  
  
In [15]: # Единичная матрица 3x3:  
Matrix(Int)(I, 3, 3)  
# Скалярное произведение векторов X и Y:  
X = [-1, 4, -5]  
Y = [1, 1, 3]  
dot(X,Y)  
# тоже скалярное произведение:  
X*Y  
  
Out[15]: -17
```

### Матричное умножение, единичная матрица, скалярное произведение

Выполним примеры из раздела про факторизацию и специальные матричные структуры (рис. [-@fig:004]-[-@fig:007]).

Jupyter Lab4\_статистический\_анализ Last Checkpoint: hace 19 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [16]: # 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 l  
# (убеждаемся, что x – единичный вектор):  
A\b  
  
Out[16]: 3-element Vector{Float64}:  
1.0  
0.9999999999999999  
1.0  
  
In [17]: # LU-факторизация:  
Atu = lu(A)  
  
Out[17]: Lu{float64, Matrix{Float64}, Vector{Int64}}  
L factors:  
3x3 Matrix{Float64}:  
1.0 0.0 0.0  
0.608442 1.0 0.0  
0.376969 0.642033 1.0  
U factor:  
3x3 Matrix{Float64}:  
0.626777 0.406645 0.529681  
0.0 0.84989 0.371803  
0.0 0.0 0.493271  
  
In [18]: # Решение СЛАУ через матрицу A:  
A\b  
# Решение СЛАУ через объект факторизации:  
Atu\b  
  
Out[18]: 3-element Vector{Float64}:  
1.0  
0.9999999999999999  
1.0

### Факторизация.Специальные матричные структуры

Jupyter Lab4\_статистический\_анализ Last Checkpoint: hace 19 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [23]: # Симметризация матрицы A:  
Asym = A'  
# Спектральное разложение симметризованной матрицы:  
AsymEig = eigen(Asym)  
# Собственные значения:  
AsymEig.values  
#Собственные векторы:  
AsymEig.vectors  
  
Out[23]: 3x3 Matrix{Float64}:  
-0.79062 0.329228 -0.516264  
0.0999061 -0.762485 -0.639246  
0.664101 0.556979 -0.569944  
  
In [24]: # Проверяем, что получится единичная матрица:  
inv(AsymEig)\*Asym  
  
Out[24]: 3x3 Matrix{Float64}:  
1.0 0.88178e-16 -4.55191e-15  
-1.11022e-15 1.0 2.88658e-15  
-5.6205e-16 5.6205e-16 1.0  
  
In [25]: # Матрица 1000 x 1000:  
n = 1000  
A = randn(n,n)  
  
Out[25]: 1000x1000 Matrix{Float64}:  
-0.0219 0.15419 0.622392 - -1.31355 1.52129 -0.339321  
-0.888235 -0.261957 1.06848 0.622479 1.77182 0.877182  
-0.398774 -0.728938 0.084617 0.932836 0.125069 -0.0754095  
-2.05866 0.447725 1.23766 1.0752 -1.11694 0.677346  
-0.417893 -1.53524 0.629413 1.26807 -0.646668 0.609877  
0.941013 -1.56813 -1.04735 -1.26524 1.02505 0.560005  
-0.930211 -0.540406 1.66641 0.710688 -1.08818 0.966647  
1.04961 -0.583824 1.66641 0.499767 0.583264 0.352806 1.89197  
-0.743467 -2.185, 0.499767 0.583264 0.352806 1.89197  
-0.0766108 -0.671386 0.0586091 -0.417238 -0.0628845 -0.852694  
-0.680944 0.281689 0.861829 - 1.00925 0.278522 1.5634  
-0.43977 -0.760769 1.00000 -0.00000 -0.10101 -1.11111

### Факторизация.Специальные матричные структуры

```

In [29]: import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
# Оценка эффективности выполнения операции по нахождению
# собственных значений симметризованной матрицы:
@time eigvals(Asym);

Resolving package versions...
Installed Compat v1.18.1
Installed BenchmarkTools v1.6.3
Updating `~/.julia/environments/v1.12/Project.toml`
[6e408f9] + BenchmarkTools v1.6.3
Updating `~/.julia/environments/v1.12/Manifest.toml`
[6e408f9] + BenchmarkTools v1.6.3
[34d4218] + Compat v1.18.1
[9abb945] + Profile v1.11.0
Precompiling packages...
1863.5 ms ✓ Compat
1272.5 ms ✓ Compat -> CompatLinearAlgebraExt
4795.0 ms ✓ Profile
1494.1 ms ✓ BenchmarkTools
4 dependencies successfully precompiled in 7 seconds. 50 already precompiled.

67.421 ms (18 allocations: 8.06 MiB)

In [30]: # Оценка эффективности выполнения операции по нахождению
# собственных значений звукомётынной матрицы:
@time eigvals(Asym_noisy);

882.553 ms (27 allocations: 7.99 MiB)

In [31]: # Оценка эффективности выполнения операции по нахождению
# собственных значений звукомётынной матрицы,
# для которой явно указано, что она симметрична:
@time eigvals(Asym_explicit);

65.235 ms (18 allocations: 8.06 MiB)

```

## Факторизация. Специальные матричные структуры

Выполним примеры из раздела про общую линейную алгебру (рис. [-@fig:008]).

```

In [34]: # Матрица с рациональными элементами:
Arational = Matrix{Rational{BigInt}}(rand(1:10, 3, 3))/10
# Единичный вектор:
x = fill!(ones(3))
# Заданный вектор b:
b = Arational*x
Out[34]: 3-element Vector{Rational{BigInt}}:
12//5
3//2
9//5

In [35]: # Решение исходного уравнения получаем с помощью функции l
# (убеждаемся, что x - единичный вектор):
Arational\b
Out[35]: 3-element Vector{Rational{BigInt}}:
1
1
1

In [36]: # LU-разложение:
lu(Arational)
Out[36]: LU{Rational{BigInt}, Matrix{Rational{BigInt}}, Vector{Int64}}
L factor:
3x3 Matrix{Rational{BigInt}}:
 1   0   0
 1   0   0
 5//8 5//8 1
U factor:
3x3 Matrix{Rational{BigInt}}:
 4//5  9//10  7//10
 0   -1//10 -1//2
 0     0    3//8

```

## Общая линейная алгебра

### Задания для самостоятельного выполнения

Зададим вектор  $v$ . Умножим вектор  $v$  скалярно сам на себя и сохраним результат  $\text{vdot}_v$  (рис. [-@fig:009]).

#### Задания для самостоятельного выполнения

```

In [38]: using LinearAlgebra
println(">4.1 Producto de vectores")

#1 vector y operaciones
v = [1, 2, 3, 4, 5]

#producto scalar
dot_v = dot(v, v)
println("1. Producto escalar v.v = $dot_v")

->4.1 Producto de vectores
1. Producto escalar v.v = 55

```

## Произведение векторов

Умножим  $v$  матрично на себя(внешнее произведение), присвоив результат переменной outer\_v (рис. [-@fig:010]).

```
In [39]: #producto exterior
outer_v = v * v'
println("\n2. Producto exterior:")
display(outer_v)
```

## 2. Producto exterior:

```
5x5 Matrix{Int64}:
 1  2  3  4  5
 2  4  6  8  10
 3  6  9  12 15
 4  8  12 16 20
 5  10 15 20 25
```

## Произведение векторов

Решим СЛАУ с двумя неизвестными (рис. [-@fig:011]-[-@fig:012]).

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
#1 sistema 2x2
println("sistema 2x2")

#o) x+y=2, x-y=3
A1 = [1, 1; 1, -1]
b1 = [2, 3]
x1 = A1\b1

→4.4.2 sistema de ecuaciones lineales
sistema 2x2

Out[40]: 2-element Vector{Float64}:
 2.5
 -0.5

In [43]: #sistema b)
A_b = [1, 1; 2, 1]
b_b = pinv(A_b) * b_b
Out[43]: 2-element Vector{Float64}:
 0.999999999999996
 0.999999999999996

In [55]: #Sistema c)
using LinearAlgebra

A_c = [1, 1; 2, 2]
b_c = [2, 5]

# Solución usando pseudoinversa (siempre funciona para matrices singulares)
x_c = pinv(A_c) * b_c

println("nc) solución con pseudoinversa: x=$(round(x_c[1], digits=3)), y=$(round(x_c[2], digits=3))")
println("error: A*x-b = $(A_c*x_c - b_c)")

c) solución con pseudoinversa: x=1.2, y=1.2
error: A*x-b = {0.399999999999999, -0.2000000000000195}
```

## Системы линейных уравнений

```

In [55]: #sistema c
using LinearAlgebra

A_c = [1 1; 2 2]
b_c = [2, 5]

# Solución usando pseudoinversa (siempre funciona para matrices singulares)
x_c = pinv(A_c) * b_c

println("\nnc) solución con pseudoinversa: x=$(round(x_c[1], digits=3)), y=$(round(x_c[2], digits=3))\n")
println("error: A*x-b = $(A_c*x_c - b_c)")

c) solución con pseudoinversa: x=1.2, y=1.2
error: A*x-b = [0.3999999999999999, -0.2000000000000000]

In [59]: # Sistema d
using LinearAlgebra

A_d = [1 1; 2 2; 3 3]
b_d = [1, 2, 3]
x_d = pinv(A_d) * b_d

println("\nd) solución particular: x=$(round(x_d[1], digits=3)), y=$(round(x_d[2], digits=3))\n")
println("verificación: x+y=$(round(x_d[1]+x_d[2], digits=3)) = 1")

d) solución particular: x=0.5, y=0.5
verificación: x+y=1.0 = 1

In [60]: #sistema 3x3
println("\nn2. sistema 3x3:")

#)
A5 = [1 1 1]
b5 = [2]
println("a) x + y + z = 2 -> infinitas soluciones")

```

## Систем лінійних уравнень

Решим СЛАУ с тремя неизвестными (рис. [-@fig:013]).

```

In [61]: 2. sistema 3x3:
a) x + y + z = 2 -> infinitas soluciones

In [62]: #b sistema consistente determinado
A6 = [1 0 1; 0 1 1; 1 1 0]
b6 = [1, 2, 3]
x6 = A6 \ b6
println("b) sistema x+z=2, y+z=2, x=y=3")
println("solucion: x = $(x6[1]), y = $(x6[2]), z = $(x6[3]))")

b) sistema x+z=2, y+z=2, x=y=3
solucion: x = 1.0, y = 2.0, z = -0.0

In [63]: #c) sistema con solución única
A7 = [1 2 3; 4 5 6; 7 8 10]
b7 = [1, 2, 3]
x7 = A7\b7
println("c) sistema 3x3 general")
println("solucion: x = $(x7[1]), y = $(x7[2]), z=$(x7[3]))")

c) sistema 3x3 general
solucion: x = -0.3333333333333337, y = 0.6666666666666666, z=3.1720657846433024e-17

In [72]: # Sistema homogéneo
A8 = Float64[1 1 1; 1 1 1; 1 1 1]
b8 = [0.0, 0.0, 0.0]

x8 = pinv(A8) * b8
println("Sistema homogéneo: solución trivial x = y = z = 0")
println("Solución encontrada: x = $(x8[1]), y = $(x8[2]), z = $(x8[3]))")

Sistema homogéneo: solución trivial x = y = z = 0
Solución encontrada: x = 0.0, y = 0.0, z = 0.0

In [73]: println("-> 4.4.3 operaciones con matrices")
using LinearAlgebra
m1..diagonalizarin de matrices

```

## Систем лінійних уравнень

Приведем матрицы к диагональному виду (рис. [-@fig:014]).

```

In [72]: # Sistema homogéneo
A0 = Float64[1 1 1; 1 1 1; 1 1 1]
b0 = [0.0, 0.0, 0.0]

x0 = pinv(A0) * b0
println("Sistema homogéneo: solución trivial x = y = z = 0")
println("Solución encontrada: x = $(x0[1]), y = $(x0[2]), z = $(x0[3])")

Sistema homogéneo: solución trivial x = y = z = 0
Solución encontrada: x = 0.0, y = 0.0, z = 0.0

In [73]: println("→ 4.4.3 operaciones con matrices")

using LinearAlgebra
#1. diagonalización matrices
println("1. diagonalización de matrices")

#)
M1 = [1 -2; -2 1]
vals1, vecs1 = eigen(M1)
D1 = Diagonal(vals1)
println("Valores propios: $(vals1)")
println("matriz diagonal: $D1")

→ 4.4.3 operaciones con matrices
1. diagonalización de matrices
Valores propios: [-1.0, 3.0]
matriz diagonal: Diagonal([-1.0, 3.0])

In [74]: #b)
M2 = [1 -2; -2 3]
vals2, vecs2 = eigen(M2)
D2 = Diagonal(vals2)
println("Valores propios: $(vals2)")
println("matriz diagonal: $D2")

Valores propios: [-0.2360679774997897, 4.23606797749979]
matriz diagonal: Diagonal([-0.2360679774997897, 4.23606797749979])

```

## Операции с матрицами

Вычислим (рис. [-@fig:015]).

```

b) sqrt([5 -2; -2 5]);
2x2 Matrix{Float64}:
 2.1889 -0.4569
 -0.4569  2.1889

In [79]: #c)
M1_cbrt = M1^(1/3)
println("n\n$c:")
display(round.(M1_cbrt, digits=4))

n
c:
2x2 Matrix{ComplexF64}:
 0.9711+0.433im -0.4711+0.433im
 -0.4711+0.433im  0.9711+0.433im

In [80]: #d)
M4 = [1 2; 2 3]
try
    M4_sqrt = sqrt(M4)
    println("nd) sqrt([1 2; 2 3]):")
    display(round.(M4_sqrt, digits=4))
catch
    println("nd) sqrt([1 2; 2 3]) no tiene raíz real definida positiva")
    M4_sqrt_complex = sqrt(complex(M4))
    println("raíz compleja aproximada:")
    display(round.(M4_sqrt_complex, digits=4))
end

d) sqrt([1 2; 2 3]);
2x2 Matrix{ComplexF64}:
 0.5689+0.3516im  0.9204+0.2173im
 0.9204+0.2173im  1.4893+0.1343im

```

## Операции с матрицами

Найдем собственные значения матрицы A. Создадим диагональную матрицу из собственных значений матрицы A. Создадим нижнедиагональную матрицу из матрицы A. Оценим эффективность выполняемых операций (рис. [-@fig:016]).

```

In [83]: #matriz triangular inferior
L_A = tril(A)
println("matriz triangular inferior")
display(L_A)

matrix triangular inferior
5x5 Matrix{Int64}:
 140   0   0   0   0
 97  106   0   0   0
 74   89  152   0   0
 168  131  144  54   0
 131   36   71  142  36

In [84]: #matriz triangular superior
U_A = triu(A)
println("matriz triangular superior")
display(U_A)

matrix triangular superior
5x5 Matrix{Int64}:
 140   97  74  168  131
   0  106   89  131   36
   0   0  152  144   71
   0   0   0  54  142
   0   0   0   0  36

In [85]: #eficiencia
println("\nEvaluación de eficiencia:")
println("Tamaño de A: $(size(A))")
@time eigen(A);
println("tiempo para descomposición espectral")

Evaluación de eficiencia:
Tamaño de A: (5, 5)
@ time eigen(A); 0.000000 seconds (0.000 CPU)

```

## Операции с матрицами

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.

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 (рис. [-@fig:017]-[-@fig:018]).

```

In [100]: function es_productiva_inversa(A)
    n = size(A, 1)
    E = Matrix{Float64}(I, n, n)

    try
        inversa = inv(E-A)
        return all(inversa .>= -1e-10)
    catch
        return false
    end
end
Out[100]: es_productiva_inversa (generic function with 1 method)

In [101]: function es_productiva_espectral(A)
    vals = eigvals(A)
    return all(abs.(vals) .< 1 + 1e-10)
end
Out[101]: es_productiva_espectral (generic function with 1 method)

In [106]: #1. Matrices del ejercicio 4.4.4.1
using LinearAlgebra

# Matrices a evaluar
matrices_economia = [
    (1 1 2; 3 4), "A = [1 1 2; 3 4]"),
    (0.5*(1 1 2; 3 4)), "A = 0.5*[1 1 2; 3 4]"),
    (0.1*(1 1 2; 3 4)), "A = 0.1*[1 1 2; 3 4]")
]

for (A, nombre) in matrices_economia
    println("\nNombre: $nombre")
    println("Definición: $(es_productiva_inversa(A)))")
    println(" (I-A)^{-1} no negativo: $(es_productiva_inversa(A)))")
    println(" Criterio espectral: $(es_productiva_espectral(A)))")
end

```

## Линейные модели экономики

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Jupyter Лаб4\_статистический\_анализ Last Checkpoint: hace 19 horas (autosaved)
- Toolbar:** File Edit View Insert Cell Kernel Widgets Help
- Status Bar:** Not Trusted Julia 1.12
- Code Cell:**

```
In [113]: # 4. Comparación de criterios
    println("\nMatriz\t\tDefinición\t\t(E-A)^{-1}≥0\t|\lambda|<1")
    println("—60")
    todas_matrices = vcat(matrices_economia, matrices_2)
    nombres_unicos = []
    matrices_unicas = []

    # Eliminar duplicados
    for (A, nombre) in toutes_matrices
        if !(nombre in nombres_unicos)
            push!(nombres_unicos, nombre)
            push!(matrices_unicas, A)
        end
    end

    # Añadir matriz 3x3
    push!(nombres_unicos, "A 3x3")
    push!(matrices_unicas, [0.1 0.2 0.3; 0 0.1 0.2; 0 0.1 0.3])

    for (A, nombre) in zip(matrices_unicas, nombres_unicos)
        def = es_productiva_definicion(A, 50)
        inv_pos = es_productiva_inversa(A)
        esp = es_productiva_espectral(A)

        # Acortar nombre para tabla
        nombre_corto = length(nombre) > 15 ? nombre[1:12] * "..." : nombre
        println("${pad(nombre_corto, 15)}\t$def\t$inv_pos\t$esp")
    end

```
- Output Cell:**

4. Resumen comparativo:

Matriz	Definición	$(E-A)^{-1} \geq 0$	$ \lambda  < 1$
A = 1 2 3 4 1 2 1 2 0	def	def	def

## Линейные модели экономики

## Выводы

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

## Список литературы