

Eliminación Guassiana

Mateo Cumbal

2025-01-08

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import init_printing
init_printing()
```

CONJUNTO DE EJERCICIOS

Ejercicio 1

Para cada uno de los siguientes sistemas lineales, obtenga, de ser posible, una solución con métodos gráficos. Explique los resultados desde un punto de vista geométrico.

a.

$$x_1 + 2x_2 = 0$$

$$x_1 - x_2 = 0$$

```
x1 = np.linspace(-10, 10, 400)

x2_1 = -x1 / 2
x2_2 = x1

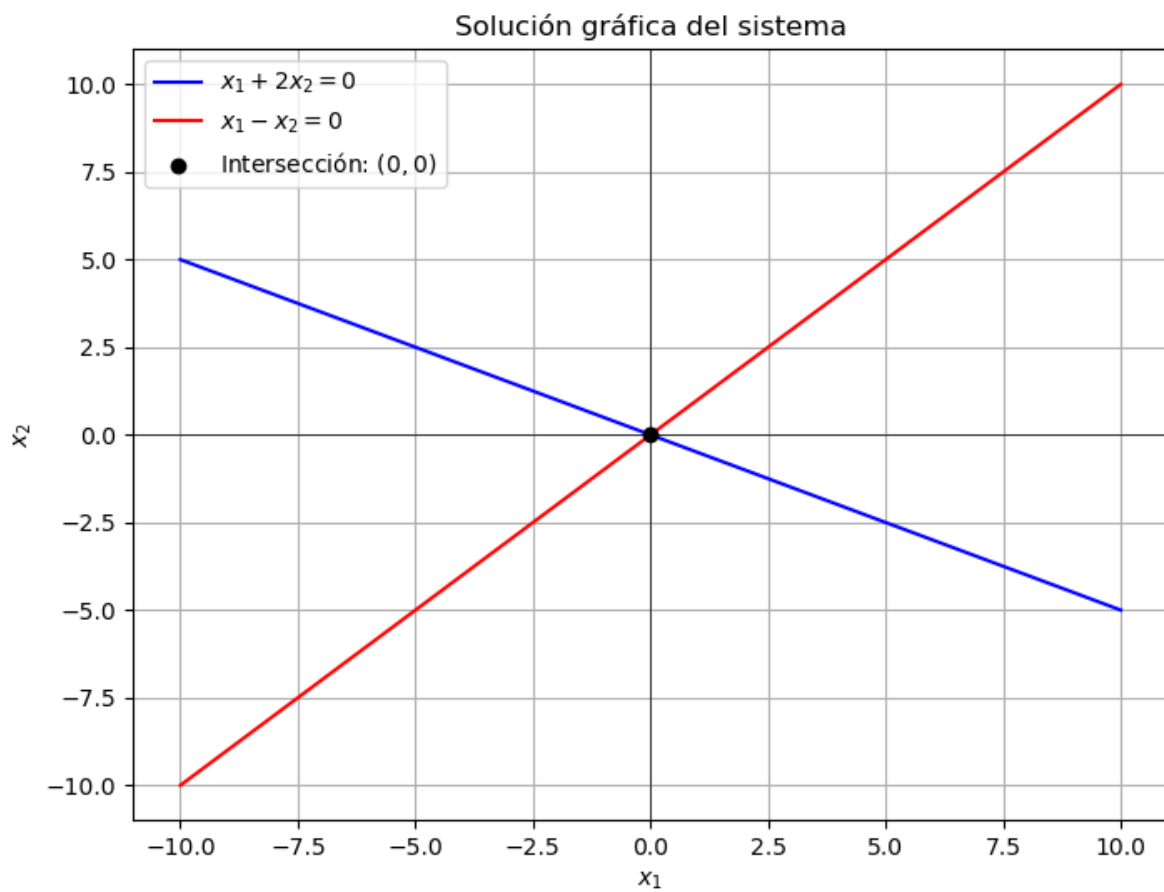
plt.figure(figsize=(8, 6))
plt.plot(x1, x2_1, label=r"$x_1 + 2x_2 = 0$", color="blue")
plt.plot(x1, x2_2, label=r"$x_1 - x_2 = 0$", color="red")
```

```

sol_a = (0, 0)
plt.scatter(*sol_a, color="black", label="Intersección: $(0, 0)$", zorder=5)

plt.axhline(0, color="black", linewidth=0.5)
plt.axvline(0, color="black", linewidth=0.5)
plt.title("Solución gráfica del sistema")
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.grid()
plt.legend()
plt.show()

```



b.

$$x_1 + 2x_2 = 3$$

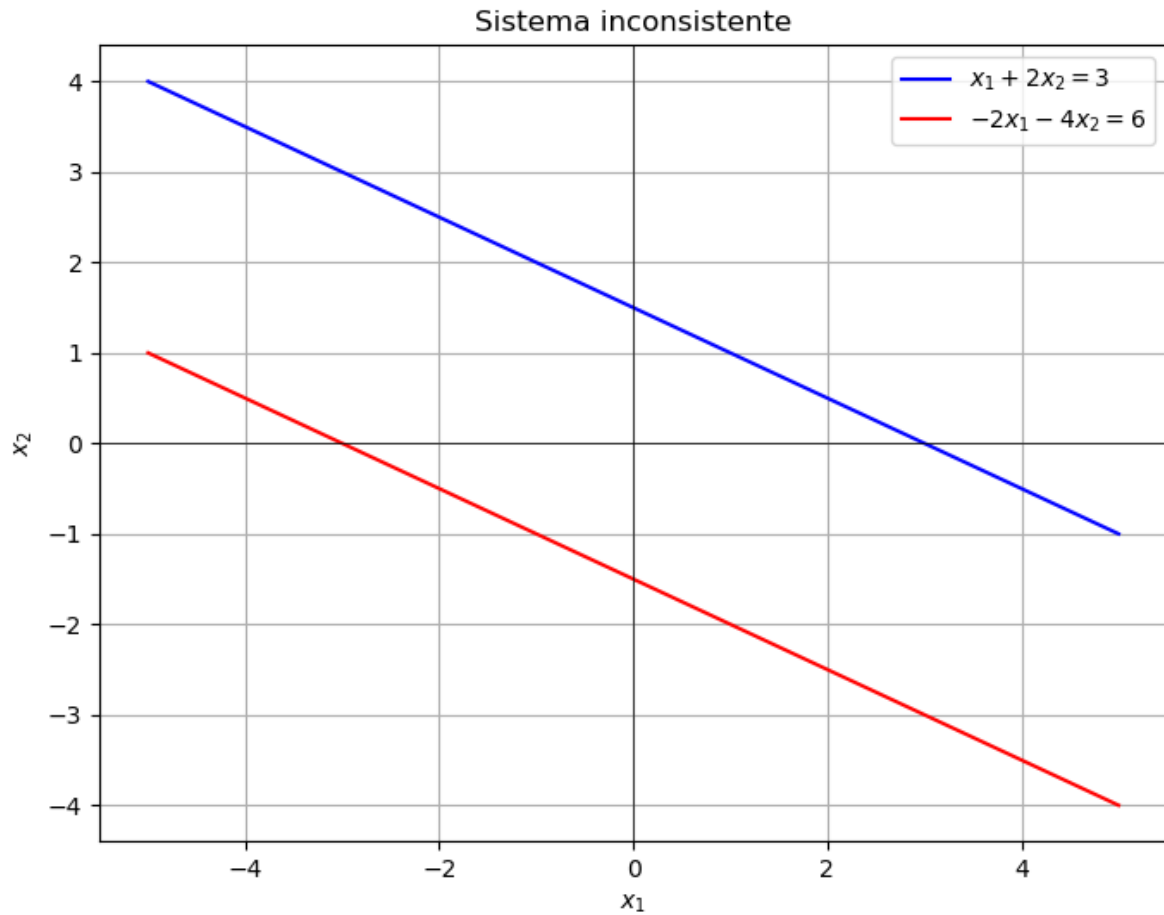
$$-2x_1 - 4x_2 = 6$$

```
x1 = np.linspace(-5, 5, 400)

x2_1 = (3 - x1) / 2
x2_2 = - (6 + 2 * x1) / 4

plt.figure(figsize=(8, 6))
plt.plot(x1, x2_1, label=r"$x_1 + 2x_2 = 3$", color="blue")
plt.plot(x1, x2_2, label=r"$-2x_1 - 4x_2 = 6$", color="red")

plt.title("Sistema inconsistente")
plt.axhline(0, color="black", linewidth=0.5)
plt.axvline(0, color="black", linewidth=0.5)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.grid()
plt.legend()
plt.show()
```



No hay intersección porque las líneas son paralelas. El sistema es **inconsistente**.

c.

$$2x_1 + x_2 = -1$$

$$x_1 + x_2 = 2$$

$$x_1 - 3x_2 = 5$$

```
x1 = np.linspace(-10, 10, 400)
```

```
x2_1 = -1 - 2 * x1
```

```
x2_2 = 2 - x1
```

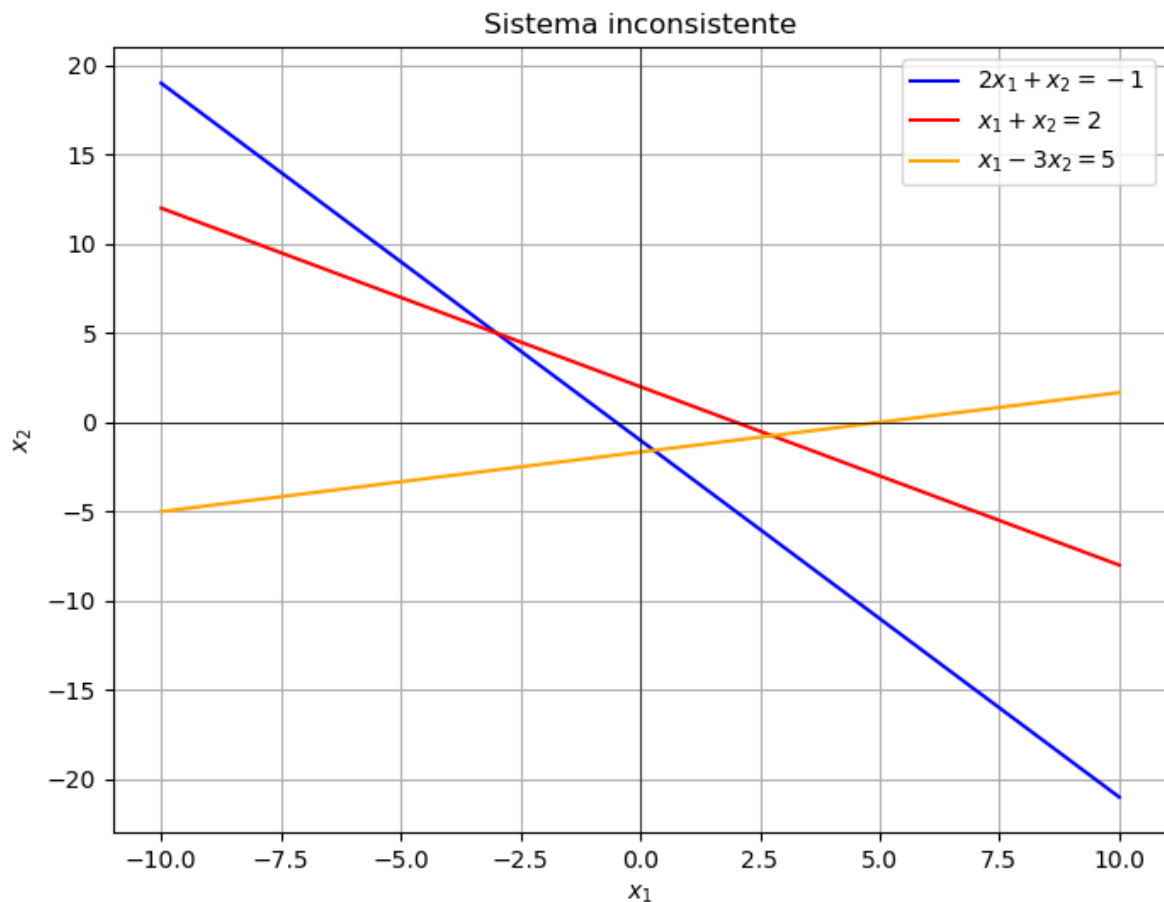
```

x2_3 = - (5 - x1) / 3

plt.figure(figsize=(8, 6))
plt.plot(x1, x2_1, label=r"$2x_1 + x_2 = -1$", color="blue")
plt.plot(x1, x2_2, label=r"$x_1 + x_2 = 2$", color="red")
plt.plot(x1, x2_3, label=r"$x_1 - 3x_2 = 5$", color="orange")

plt.title("Sistema inconsistente")
plt.axhline(0, color="black", linewidth=0.5)
plt.axvline(0, color="black", linewidth=0.5)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.grid()
plt.legend()
plt.show()

```



El sistema es **inconsistente**: no existe un punto en el que las tres funciones itersecan, por lo que no hay solución.

d.

$$2x_1 + x_2 + x_3 = 1$$

$$2x_1 + 4x_2 = 0$$

```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.linspace(-5, 5, 100)
x2 = np.linspace(-5, 5, 100)
X1, X2 = np.meshgrid(x1, x2)

# Definir los planos
X3_1 = 1 - 2 * X1 - X2
X3_2 = 2 * X1 + 4 * X2 + 1

# Resolver el sistema para obtener la recta de soluciones
# Usando x2 como parámetro libre, obtenemos las relaciones entre las variables
t = np.linspace(-6, 6, 100) # Parámetro libre para la recta
x1_line = t
x2_line = -t
x3_line = 3 * t - 1

# Crear una figura en 3D
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

# Graficar los planos
ax.plot_surface(X1, X2, X3_1, alpha=0.5, rstride=100, cstride=100, color='red', label="$2x_1 + x_2 + x_3 = 1$")
ax.plot_surface(X1, X2, X3_2, alpha=0.5, rstride=100, cstride=100, color='blue', label="$2x_1 + 4x_2 = 0$")

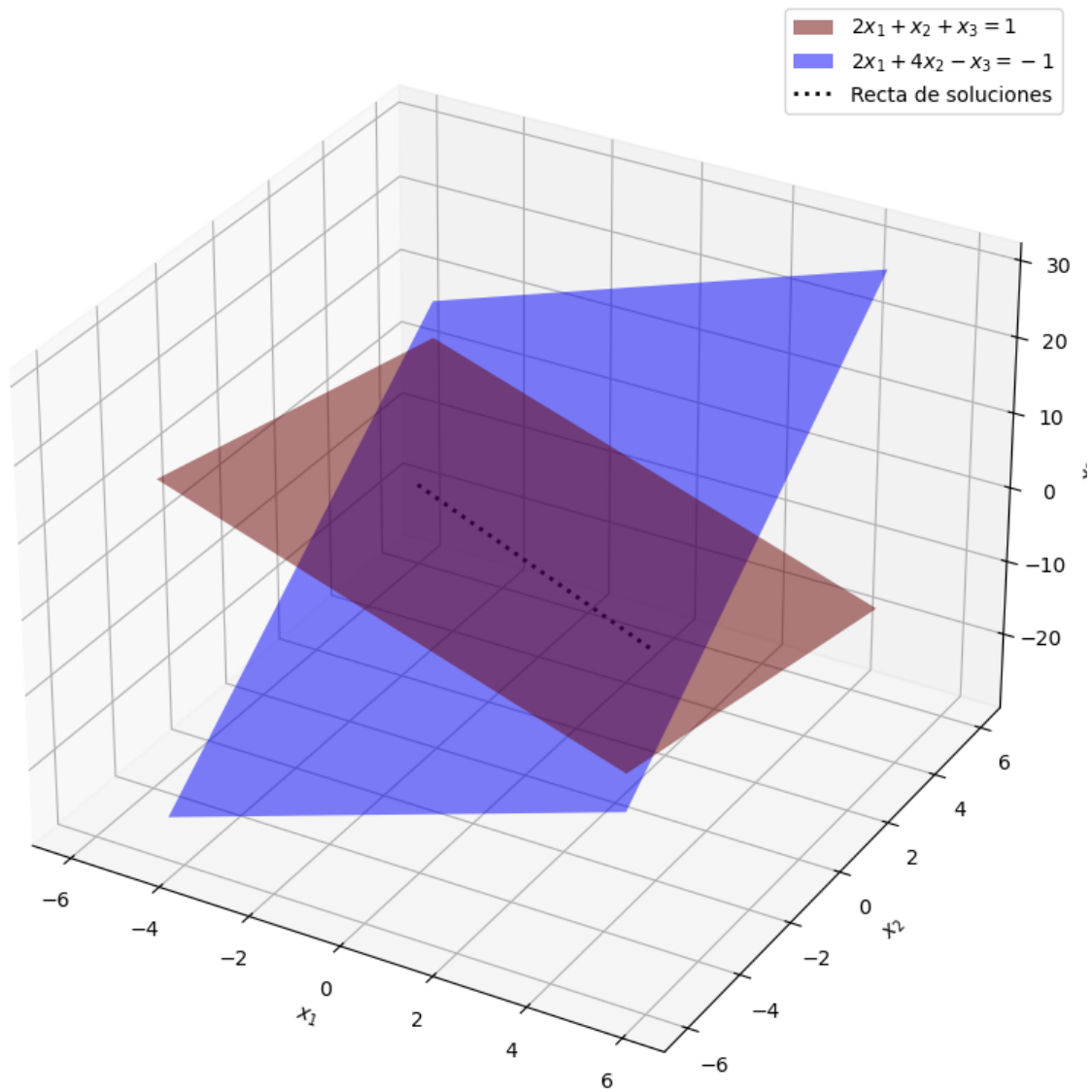
# Graficar la recta de soluciones
ax.plot(x1_line, x2_line, x3_line, color='black', linewidth=2, linestyle=':', label='Recta de soluciones')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')
ax.set_title("Planos y recta de soluciones del sistema")
```

```
ax.legend()
```

```
plt.show()
```

Planos y recta de soluciones del sistema



Ejercicio 2

Utilice la eliminación gaussiana con sustitución hacia atrás y aritmética de redondeo de dos dígitos para resolver los siguientes sistemas lineales. No reordene las ecuaciones. (La solución exacta para cada sistema es $x_1 = -1, x_2 = 1, x_3 = 3$)

```
def eliminacion_gaussiana_no_cambio_fila(A):
    if not isinstance(A, np.ndarray):
        A = np.array(A)
    A = np.round(A, 2)
    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1)."
    n = A.shape[0]

    print("Estado inicial de la matriz:")
    print(A)

    for i in range(0, n - 1):
        if A[i, i] == 0:
            raise ValueError("El pivote es cero, no se permite cambiar filas en este caso.")

        for j in range(i + 1, n):
            m = round(A[j, i] / A[i, i], 2)
            A[j, i:] = np.round(A[j, i:] - m * A[i, i:], 2)

        print(f"Estado de la matriz tras la eliminación en la columna {i}:")
        print(A)

    if A[n - 1, n - 1] == 0:
        raise ValueError("No existe solución única.")

    solucion = np.zeros(n)
    solucion[n - 1] = round(A[n - 1, n] / A[n - 1, n - 1], 2)

    for i in range(n - 2, -1, -1):
        suma = 0
        for j in range(i + 1, n):
            suma += A[i, j] * solucion[j]
        solucion[i] = round((A[i, n] - suma) / A[i, i], 2)

    return solucion
```

a.

$$-x_1 + 4x_2 + x_3 = 8$$

$$\frac{5}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 = 1$$

$$2x_1 + x_2 + 4x_3 = 11$$

```
A = np.array([
    [-1, 4, 1, 8],
    [5/3, 2/3, 2/3, 1],
    [2, 1, 4, 11]
])

solucion = eliminacion_gaussiana_no_cambio_fila(A)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[-1.  4.  1.  8. ]
 [ 1.67  0.67  0.67  1. ]
 [ 2.  1.  4.  11. ]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[-1.  4.  1.  8. ]
 [ 0.  7.35  2.34  14.36]
 [ 0.  9.  6.  27. ]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[-1.  4.  1.  8. ]
 [ 0.  7.35  2.34  14.36]
 [ 0.  0.03  3.15  9.48]]
```

Solución del sistema:

```
[-0.99  1.  3.01]
```

b.

$$4x_1 + 2x_2 - x_3 = -5$$

$$\frac{1}{9}x_1 + \frac{1}{9}x_2 - \frac{1}{3}x_3 = -1$$

$$x_1 + 4x_2 + 2x_3 = 9$$

```

B = np.array([
    [ 4, 2, -1, -5],
    [1/9, 1/9, -1/3, -1],
    [ 1, 4, 2, 9]
])

solucion = eliminacion_gaussiana_no_cambio_fila(B)
print("\nSolución del sistema:")
print(solucion)

```

```

Estado inicial de la matriz:
[[ 4.    2.   -1.   -5. ]
 [ 0.11  0.11 -0.33 -1. ]
 [ 1.    4.    2.    9. ]]
Estado de la matriz tras la eliminación en la columna 0:
[[ 4.000e+00  2.000e+00 -1.000e+00 -5.000e+00]
 [-1.000e-02  5.000e-02 -3.000e-01 -8.500e-01]
 [ 0.000e+00  3.500e+00  2.250e+00  1.025e+01]]
Estado de la matriz tras la eliminación en la columna 1:
[[ 4.000e+00  2.000e+00 -1.000e+00 -5.000e+00]
 [-1.000e-02  5.000e-02 -3.000e-01 -8.500e-01]
 [ 0.000e+00  0.000e+00  2.325e+01  6.975e+01]]

Solución del sistema:
[-1.  1.  3.]

```

Ejercicio 3

Utilice el algoritmo de eliminación gaussiana para resolver, de ser posible, los siguientes sistemas lineales, y determine si se necesitan intercambios de fila.

```

def eliminacion_gaussiana(A):
    if not isinstance(A, np.ndarray):
        A = np.array(A)
    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1)."
    n = A.shape[0]

    print("Estado inicial de la matriz:")
    print(A)

    for i in range(0, n - 1):

```

```

p = None
for pi in range(i, n):
    if A[pi, i] == 0:
        continue

    if p is None:
        p = pi
        continue

    if abs(A[pi, i]) < abs(A[p, i]):
        p = pi

if p is None:
    raise ValueError("No existe solución única.")

if p != i:
    print(f"Intercambiando filas {i} y {p}")
    _aux = A[i, :].copy()
    A[i, :] = A[p, :].copy()
    A[p, :] = _aux
    print(f"Estado de la matriz tras el intercambio:")
    print(A)

for j in range(i + 1, n):
    m = A[j, i] / A[i, i]
    A[j, i:] = A[j, i:] - m * A[i, i:]

print(f"Estado de la matriz tras la eliminación en la columna {i}:")
print(A)

if A[n - 1, n - 1] == 0:
    raise ValueError("No existe solución única.")

solucion = np.zeros(n)
solucion[n - 1] = A[n - 1, n] / A[n - 1, n - 1]

for i in range(n - 2, -1, -1):
    suma = 0
    for j in range(i + 1, n):
        suma += A[i, j] * solucion[j]
    solucion[i] = (A[i, n] - suma) / A[i, i]

```

```
return solucion
```

a.

$$x_1 - x_2 + 3x_3 = 2$$

$$3x_1 - 3x_2 + x_3 = -1$$

$$x_1 + x_2 = 3$$

```
A = np.array([
    [1, -1, 3, 2],
    [3, -3, 1, -1],
    [1, 1, 0, 3]
])

solucion = eliminacion_gaussiana(A)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[ 1 -1  3  2]
 [ 3 -3  1 -1]
 [ 1  1  0  3]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[ 1 -1  3  2]
 [ 0  0 -8 -7]
 [ 0  2 -3  1]]
```

Intercambiando filas 1 y 2

Estado de la matriz tras el intercambio:

```
[[ 1 -1  3  2]
 [ 0  2 -3  1]
 [ 0  0 -8 -7]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 1 -1  3  2]
 [ 0  2 -3  1]
 [ 0  0 -8 -7]]
```

Solución del sistema:

```
[1.1875 1.8125 0.875 ]
```

Dado que el elemento A_{11} es cero en la matriz $A^{(1)}$, se realiza un intercambio de filas entre 1 y 2 para obtener $A^{(1')}$ y seguir con el algoritmo.

b.

$$2x_1 - 1.5x_2 + 3x_3 = 1$$

$$-x_1 + 2x_3 = 3$$

$$4x_1 - 4.5x_2 + 5x_3 = 1$$

```
B = np.array([
    [ 2, -1.5, 3, 1],
    [-1,  0, 2, 3],
    [ 4, -4.5, 5, 1]
])

solucion = eliminacion_gaussiana(B)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[ 2. -1.5  3.  1.]
 [-1.  0.  2.  3.]
 [ 4. -4.5  5.  1.]]
```

Intercambiando filas 0 y 1

Estado de la matriz tras el intercambio:

```
[[ -1.  0.  2.  3.]
 [  2. -1.5  3.  1.]
 [  4. -4.5  5.  1.]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[ -1.  0.  2.  3.]
 [  0. -1.5  7.  7.]
 [  0. -4.5 13. 13.]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ -1.  0.  2.  3.]
 [  0. -1.5  7.  7.]
 [  0.  0. -8. -8.]]
```

Solución del sistema:

```
[-1. -0.  1.]
```

Dado que el elemento A_{11} es cero en la matriz inicial $A^{(0)}$, se realiza un intercambio de filas entre 0 y 1 para obtener $A^{(0')}$ y seguir con el algoritmo.

c.

$$2x_1 = 3$$

$$x_1 + 1.5x_2 = 4.5$$

$$-3x_2 + 0.5x_3 = -6.6$$

$$2x_1 - 2x_2 + x_3 + x_4 = 0.8$$

```
C = np.array([
    [2, 0, 0, 0, 3],
    [1, 1.5, 0, 0, 4.5],
    [0, -3, 0.5, 0, -6.6],
    [2, -2, 1, 1, 0.8]
])

solucion = eliminacion_gaussiana(C)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[ 2.  0.  0.  0.  3.]
 [ 1.  1.5 0.  0.  4.5]
 [ 0. -3.  0.5 0. -6.6]
 [ 2. -2.  1.  1.  0.8]]
```

Intercambiando filas 0 y 1

Estado de la matriz tras el intercambio:

```
[[ 1.  1.5 0.  0.  4.5]
 [ 2.  0.  0.  0.  3.]
 [ 0. -3.  0.5 0. -6.6]
 [ 2. -2.  1.  1.  0.8]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[ 1.  1.5 0.  0.  4.5]
 [ 0. -3.  0.  0. -6.]
 [ 0. -3.  0.5 0. -6.6]
 [ 0. -5.  1.  1. -8.2]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 1.  1.5  0.  0.  4.5]
 [ 0. -3.  0.  0. -6. ]
 [ 0.  0.  0.5  0. -0.6]
 [ 0.  0.  1.  1.  1.8]]
```

Estado de la matriz tras la eliminación en la columna 2:

```
[[ 1.  1.5  0.  0.  4.5]
 [ 0. -3.  0.  0. -6. ]
 [ 0.  0.  0.5  0. -0.6]
 [ 0.  0.  0.  1.  3. ]]
```

Solución del sistema:

```
[ 1.5  2. -1.2  3. ]
```

En este caso, el intercambio de fila NO es estrictamente necesario, solo se lo realiza para encontrar el mejor pivote.

d.

$$2x_1 = 3$$

$$x_1 + 1.5x_2 = 4.5$$

$$-3x_2 + 0.5x_3 = -6.6$$

$$2x_1 - 2x_2 + x_3 + x_4 = 0.8$$

```
D = np.array([
    [1, 1, 0, 1, 2],
    [2, 1, -1, 1, 1],
    [4, -1, -2, 2, 0],
    [3, -1, -1, 2, -3]
])
try:
    solucion = eliminacion_gaussiana(D)
    print("\nSolución del sistema:")
    print(solucion)
except ValueError as e:
    print('\nERROR:', e)
```

Estado inicial de la matriz:

```
[[ 1  1  0  1  2]
 [ 2  1 -1  1  1]
 [ 4 -1 -2  2  0]
 [ 3 -1 -1  2 -3]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[ 1  1  0  1  2]
 [ 0 -1 -1 -1 -3]
 [ 0 -5 -2 -2 -8]
 [ 0 -4 -1 -1 -9]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 1  1  0  1  2]
 [ 0 -1 -1 -1 -3]
 [ 0  0  3  3  7]
 [ 0  0  3  3  3]]
```

Estado de la matriz tras la eliminación en la columna 2:

```
[[ 1  1  0  1  2]
 [ 0 -1 -1 -1 -3]
 [ 0  0  3  3  7]
 [ 0  0  0  0 -4]]
```

ERROR: No existe solución única.

Para este caso, el sistema es **inconsistente**.

Ejercicio 4

Use el algoritmo de eliminación gaussiana y la aritmética computacional de precisión de 32 bits para resolver los siguientes sistemas lineales.

Solamente se especifica el tipo de datos que tiene la matriz, en este caso, es *float32*.

a.

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 = 9$$

$$\frac{1}{3}x_1 + \frac{1}{4}x_2 - \frac{1}{5}x_3 = 8$$

$$\frac{1}{2}x_1 + x_2 + 2x_3 = 8$$


```
A = np.array([
    [1/4, 1/5, 1/6, 9],
    [1/3, 1/4, 1/5, 8],
    [1/2, 1, 2, 8],
], dtype='float32')

solucion = eliminacion_gaussiana(A)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[0.25      0.2      0.16666667 9.      ]
 [0.33333334 0.25     0.2      8.      ]
 [0.5       1.       2.       8.      ]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[ 0.25      0.2      0.16666667 9.      ]
 [ 0.        -0.01666668 -0.02222224 -4.      ]
 [ 0.         0.6      1.6666666  -10.     ]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 2.5000000e-01  2.0000000e-01  1.6666667e-01  9.0000000e+00]
 [ 0.0000000e+00 -1.6666681e-02 -2.2222236e-02 -4.0000000e+00]
 [ 0.0000000e+00 -5.9604645e-08  8.6666673e-01 -1.5399989e+02]]
```

Solución del sistema:

```
[-227.07666725  476.92263902 -177.69216919]
```

b.

$$3.333x_1 + 15920x_2 - 10.333x_3 = 15913$$

$$2.222x_1 + 16.71x_2 + 9.612x_3 = 28.544$$

$$1.5611x_1 + 5.1791x_2 + 1.6852x_3 = 8.4254$$

```
B = np.array([
    [3.333, 15920, 10.333, 15913],
    [2.222, 16.71, 9.612, 28.544],
    [1.5611, 5.1791, 1.6852, 8.4254],
], dtype='float32')
```

```

solucion = eliminacion_gaussiana(B)
print("\nSolución del sistema:")
print(solucion)

```

Estado inicial de la matriz:

```

[[3.3330e+00 1.5920e+04 1.0333e+01 1.5913e+04]
 [2.2220e+00 1.6710e+01 9.6120e+00 2.8544e+01]
 [1.5611e+00 5.1791e+00 1.6852e+00 8.4254e+00]]

```

Intercambiando filas 0 y 2

Estado de la matriz tras el intercambio:

```

[[1.5611e+00 5.1791e+00 1.6852e+00 8.4254e+00]
 [2.2220e+00 1.6710e+01 9.6120e+00 2.8544e+01]
 [3.3330e+00 1.5920e+04 1.0333e+01 1.5913e+04]]

```

Estado de la matriz tras la eliminación en la columna 0:

```

[[1.5611000e+00 5.1791000e+00 1.6852000e+00 8.4253998e+00]
 [0.0000000e+00 9.3383007e+00 7.2133622e+00 1.6551662e+01]
 [0.0000000e+00 1.5908942e+04 6.7350426e+00 1.5895012e+04]]

```

Estado de la matriz tras la eliminación en la columna 1:

```

[[ 1.5611000e+00  5.1791000e+00  1.6852000e+00  8.4253998e+00]
 [ 0.0000000e+00  9.3383007e+00  7.2133622e+00  1.6551662e+01]
 [ 0.0000000e+00  0.0000000e+00 -1.2282113e+04 -1.2302777e+04]]

```

Solución del sistema:

```

[1.00249532 0.99870038 1.0016824 ]

```

c.

$$x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 = \frac{1}{6}$$

$$\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 + \frac{1}{5}x_4 = \frac{1}{7}$$

$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 + \frac{1}{6}x_4 = \frac{1}{8}$$

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 + \frac{1}{7}x_4 = \frac{1}{9}$$

```

C = np.array([
    [1, 1/2, 1/3, 1/4, 1/6],
    [1/2, 1/3, 1/4, 1/5, 1/7],
    [1/3, 1/4, 1/5, 1/6, 1/8],
    [1/4, 1/5, 1/6, 1/7, 1/9],
], dtype='float32')

solucion = eliminacion_gaussiana(C)
print("\nSolución del sistema:")
print(solucion)

```

Estado inicial de la matriz:

```

[[1.          0.5          0.33333334 0.25          0.16666667]
 [0.5         0.33333334 0.25          0.2           0.14285715]
 [0.33333334 0.25         0.2           0.16666667 0.125        ]
 [0.25        0.2         0.16666667 0.14285715 0.11111111]]

```

Intercambiando filas 0 y 3

Estado de la matriz tras el intercambio:

```

[[0.25        0.2         0.16666667 0.14285715 0.11111111]
 [0.5         0.33333334 0.25          0.2           0.14285715]
 [0.33333334 0.25         0.2           0.16666667 0.125        ]
 [1.          0.5          0.33333334 0.25          0.16666667]]

```

Estado de la matriz tras la eliminación en la columna 0:

```

[[ 0.25        0.2         0.16666667 0.14285715 0.11111111]
 [ 0.          -0.06666666 -0.08333334 -0.0857143  -0.07936507]
 [ 0.          -0.01666668 -0.02222224 -0.02380954 -0.02314815]
 [ 0.          -0.3         -0.33333334 -0.3214286  -0.27777778 ]]

```

Intercambiando filas 1 y 2

Estado de la matriz tras el intercambio:

```

[[ 0.25        0.2         0.16666667 0.14285715 0.11111111]
 [ 0.          -0.01666668 -0.02222224 -0.02380954 -0.02314815]
 [ 0.          -0.06666666 -0.08333334 -0.0857143  -0.07936507]
 [ 0.          -0.3         -0.33333334 -0.3214286  -0.27777778 ]]

```

Estado de la matriz tras la eliminación en la columna 1:

```

[[ 2.5000000e-01  2.0000000e-01  1.6666667e-01  1.4285715e-01
   1.1111111e-01]
 [ 0.0000000e+00 -1.6666681e-02 -2.2222236e-02 -2.3809537e-02
  -2.3148149e-02]
 [ 0.0000000e+00  0.0000000e+00  5.5555180e-03  9.5237717e-03
   1.3227440e-02]
 [ 0.0000000e+00  2.9802322e-08  6.6666603e-02  1.0714275e-01
   1.3888860e-01]]

```

Estado de la matriz tras la eliminación en la columna 2:

```
[[ 2.50000000e-01  2.00000000e-01  1.6666667e-01  1.4285715e-01
   1.1111111e-01]
 [ 0.0000000e+00 -1.6666681e-02 -2.2222236e-02 -2.3809537e-02
  -2.3148149e-02]
 [ 0.0000000e+00  0.0000000e+00  5.5555180e-03  9.5237717e-03
   1.3227440e-02]
 [ 0.0000000e+00  2.9802322e-08  0.0000000e+00 -7.1431771e-03
  -1.9841611e-02]]
```

Solución del sistema:

```
[-0.03174083  0.5951854 -2.3808313  2.77770114]
```

d.

$$2x_1 + x_2 - x_3 + x_4 - 3x_5 = 7$$

$$x_1 + 2x_3 - x_4 + x_5 = 2$$

$$-2x_2 - x_3 + x_4 - x_5 = -5$$

$$3x_1 + x_2 - 4x_3 + 5x_5 = 6$$

$$x_1 - x_2 - x_3 - x_4 + x_5 = -3$$

```
D = np.array([
    [2, 1, -1, 1, -3, 7],
    [1, 0, 2, -1, 1, 2],
    [0, -2, -1, 1, -1, -5],
    [3, 1, -4, 0, 5, 6],
    [1, -1, -1, -1, 1, -3],
], dtype='float32')

solucion = eliminacion_gaussiana(D)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[ 2.  1. -1.  1. -3.  7.]
```

```

[ 1.  0.  2. -1.  1.  2.]
[ 0. -2. -1.  1. -1. -5.]
[ 3.  1. -4.  0.  5.  6.]
[ 1. -1. -1. -1.  1. -3.]]
Intercambiando filas 0 y 1
Estado de la matriz tras el intercambio:
[[ 1.  0.  2. -1.  1.  2.]
 [ 2.  1. -1.  1. -3.  7.]
 [ 0. -2. -1.  1. -1. -5.]
 [ 3.  1. -4.  0.  5.  6.]
 [ 1. -1. -1. -1.  1. -3.]]
Estado de la matriz tras la eliminación en la columna 0:
[[ 1.  0.  2. -1.  1.  2.]
 [ 0.  1. -5.  3. -5.  3.]
 [ 0. -2. -1.  1. -1. -5.]
 [ 0.  1. -10.  3.  2.  0.]
 [ 0. -1. -3.  0.  0. -5.]]
Estado de la matriz tras la eliminación en la columna 1:
[[ 1.  0.  2. -1.  1.  2.]
 [ 0.  1. -5.  3. -5.  3.]
 [ 0.  0. -11.  7. -11.  1.]
 [ 0.  0. -5.  0.  7. -3.]
 [ 0.  0. -8.  3. -5. -2.]]
Intercambiando filas 2 y 3
Estado de la matriz tras el intercambio:
[[ 1.  0.  2. -1.  1.  2.]
 [ 0.  1. -5.  3. -5.  3.]
 [ 0.  0. -5.  0.  7. -3.]
 [ 0.  0. -11.  7. -11.  1.]
 [ 0.  0. -8.  3. -5. -2.]]
Estado de la matriz tras la eliminación en la columna 2:
[[ 1.  0.  2. -1.  1.  2.  ]
 [ 0.  1. -5.  3. -5.  3.  ]
 [ 0.  0. -5.  0.  7. -3.  ]
 [ 0.  0.  0.  7. -26.400002  7.6000004]
 [ 0.  0.  0.  3. -16.2  2.8000002]]
Intercambiando filas 3 y 4
Estado de la matriz tras el intercambio:
[[ 1.  0.  2. -1.  1.  2.  ]
 [ 0.  1. -5.  3. -5.  3.  ]
 [ 0.  0. -5.  0.  7. -3.  ]
 [ 0.  0.  0.  3. -16.2  2.8000002]
 [ 0.  0.  0.  7. -26.400002  7.6000004]]

```

Estado de la matriz tras la eliminación en la columna 3:

```
[ [ 1.      0.      2.      -1.      1.      2.      ]
  [ 0.      1.     -5.       3.     -5.      3.      ]
  [ 0.      0.     -5.       0.      7.     -3.      ]
  [ 0.      0.      0.       3.    -16.2    2.8000002]
  [ 0.      0.      0.       0.    11.399998  1.0666666]]
```

Solución del sistema:

```
[1.88304105 2.80701722 0.73099417 1.43859666 0.09356727]
```

Ejercicio 5

Dado el sistema lineal:

$$x_1 - x_2 + \alpha x_3 = -2$$

$$-x_1 + 2x_2 - \alpha x_3 = 3$$

$$\alpha x_1 + x_2 + x_3 = 2$$

$$A^{(0)} = \begin{bmatrix} 1 & -1 & \alpha & -2 \\ -1 & 2 & -\alpha & 3 \\ \alpha & 1 & 1 & 2 \end{bmatrix}$$

$$F1 + F2 \longrightarrow F2$$

$$-\alpha \cdot F1 + F3 \longrightarrow F3$$

$$A^{(1)} = \begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & \alpha + 1 & -\alpha^2 + 1 & 2\alpha + 2 \end{bmatrix}$$

$$-(\alpha + 1) \cdot F2 + F3 \longrightarrow F3$$

$$A^{(2)} = \begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & -\alpha^2 + 1 & \alpha + 1 \end{bmatrix}$$

$$Si : -\alpha^2 + 1 = 0$$

$$\alpha = -1 \quad \text{ó} \quad \alpha = 1$$

$$Si : \alpha + 1 = 0$$

$$\alpha = -1$$

a. Encuentre el valor(es) de α para los que el sistema no tiene soluciones.

Si $\alpha = 1 \Rightarrow rang(A) = 2 \neq rang(A|b) = 3$ lo que hace al sistema **incosistente**.

b. Encuentre el valo(es) de α para los que el sistema tiene un número infinito de soluciones.

Si $\alpha = -1 \Rightarrow rang(A) = 2 = rang(A|b) = 2 \neq n = 3$ lo que hace al sistema **consistente de infinitas soluciones**.

c. Suponga que existe una única solución para una a determinada, encuentre la solución.

Se cumple cuando $rang(A) = rang(A|b) = n$. En este caso, sucederá para:

$$\alpha \in \mathbb{R} - \{-1, 1\}$$

EJERCICIOS APLICADOS

Ejercicio 6

Suponga que en un sistema biológico existen n especies de animales y m fuentes de alimento. Si x_j representa la población de las j -ésimas especies, para cada $j = 1, \dots, n$; b_i representa el suministro diario disponible del i -ésimo alimento y a_{ij} representa la cantidad del i -ésimo alimento.

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

\vdots

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$

representa un equilibrio donde existe un suministro diario de alimento para cumplir con precisión con el promedio diario de consumo de cada especie.

a. Si

$$A = [a_{ij}] = \begin{bmatrix} 1 & 2 & 0 & 3 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$x = (x_j) = [1000, 500, 350, 400], \quad \text{y} \quad b = (b_i) = [3500, 2700, 900].$$

¿Existe suficiente alimento para satisfacer el consumo promedio diario?

Para responder la pregunta, se debe asegurar que el resultado de la multiplicación de las matrices A y x tenga como resultado valores menores o iguales a los de la matriz b . Esto significa que el suministro diario excede o es igual al alimento que se requiere.

```
A = np.array([
    [1, 2, 0, 3],
    [1, 0, 2, 2],
    [0, 0, 1, 1]
])
x = np.array([1000, 500, 350, 400])
b = np.array([3500, 2700, 900])

B = A @ x

print('Ax =', B)
```

Ax = [3200 2500 750]

```
print('b\tB')
for bi, Bi in zip(b, B):
    print(f'{bi} \t{Bi}')
```


b	B
3500	3200
2700	2500
900	750

A partir del resultado, podemos comparar las matrices. Dadas las matrices $b = [b_{ij}]$ y $B = [B_{ij}]$, se puede asegurar que:

$$b_{ij} \geq B_{ij}, \forall i, j$$

Por lo que SÍ existe suficiente alimento para satisfacer el consumo promedio diario.

b. ¿Cuál es el número máximo de animales de cada especie que se podría agregar de forma individual al sistema con el suministro de alimento que cumpla con el consumo?

```
def aumento_maximo_especie(A, b, B):
    filas = A.shape[0]
    columnas = A.shape[1]
    sobrantes = b - B
    maximos = np.zeros((filas, columnas))

    print('Numero máximo de especies a añadir:')
    for j in range(columnas):
        for i in range(filas):
            if A[i, j] != 0:
                maximos[i, j] = sobrantes[i]/A[i, j]
            else:
                maximos[i, j] = 10000
    print(f'- Especie {j+1}: {min(maximos[:, j])}')

```

```
aumento_maximo_especie(A, b, B)
```

Numero máximo de especies a añadir:

- Especie 1: 200.0
- Especie 2: 150.0
- Especie 3: 100.0
- Especie 4: 100.0

c. Si la especie 1 se extingue, ¿qué cantidad de incremento individual de las especies restantes se podría soportar?

```

Ac = np.array([
    [2, 0, 3],
    [0, 2, 2],
    [0, 1, 1]
])
x = np.array([500, 350, 400])
b = np.array([3500, 2700, 900])

B = Ac @ x

print('Ax =', B)

```

Ax = [2200 1500 750]

```
aumento_maximo_especie(Ac, b, B)
```

Numero máximo de especies a añadir:

- Especie 1: 650.0
- Especie 2: 150.0
- Especie 3: 150.0

d. Si la especie 2 se extingue, ¿qué cantidad de incremento individual de las especies restantes se podría soportar?

```

Ad = np.array([
    [0, 3],
    [2, 2],
    [1, 1]
])
x = np.array([350, 400])
b = np.array([3500, 2700, 900])

B = Ad @ x

print('Ax =', B)

```

Ax = [1200 1500 750]

```
aumento_maximo_especie(Ad, b, B)
```

Numero máximo de especies a añadir:

- Especie 1: 150.0
- Especie 2: 150.0

EJERCICIOS TEÓRICOS

Ejercicio 7

Repita el ejercicio 4 con el método Gauss-Jordan.

```
def gauss_jordan(A):
    if not isinstance(A, np.ndarray):
        A = np.array(A)
    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1)."
    n = A.shape[0]

    print("Estado inicial de la matriz:")
    print(A)

    for i in range(0, n):
        p = None
        for pi in range(i, n):
            if A[pi, i] == 0:
                continue

            if p is None:
                p = pi
                continue

            if abs(A[pi, i]) < abs(A[p, i]):
                p = pi

        if p is None:
            raise ValueError("No existe solución única.")

        if p != i:
            print(f"Intercambiando filas {i} y {p}")
            _aux = A[i, :].copy()
```

```

        A[i, :] = A[p, :].copy()
        A[p, :] = _aux
        print(f"Estado de la matriz tras el intercambio:")
        print(A)

    # Normalizar la fila del pivote
    A[i, :] = A[i, :] / A[i, i]

    for j in range(n):
        if j != i:
            m = A[j, i]
            A[j, :] = A[j, :] - m * A[i, :]

    print(f"Estado de la matriz tras la eliminación en la columna {i}:")
    print(A)

solucion = A[:, -1]
return solucion

```

a.

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 = 9$$

$$\frac{1}{3}x_1 + \frac{1}{4}x_2 - \frac{1}{5}x_3 = 8$$

$$\frac{1}{2}x_1 + x_2 + 2x_3 = 8$$

```

A = np.array([
    [1/4, 1/5, 1/6, 9],
    [1/3, 1/4, 1/5, 8],
    [1/2, 1, 2, 8],
], dtype='float32')

solucion = gauss_jordan(A)
print("\nSolución del sistema:")
print(solucion)

```

Estado inicial de la matriz:
 [[0.25 0.2 0.16666667 9.]

```

[0.33333334 0.25      0.2      8.      ]
[0.5      1.      2.      8.      ]]
Estado de la matriz tras la eliminación en la columna 0:
[[ 1.0000000e+00  8.0000001e-01  6.6666669e-01  3.6000000e+01]
 [ 0.0000000e+00 -1.6666681e-02 -2.2222236e-02 -4.0000000e+00]
 [ 0.0000000e+00  6.0000002e-01  1.6666666e+00 -1.0000000e+01]]
Estado de la matriz tras la eliminación en la columna 1:
[[ 1.      0.      -0.3999998 -155.99985 ]
 [ -0.      1.      1.333333  239.9998  ]
 [ 0.      0.      0.8666668 -153.9999  ]]
Estado de la matriz tras la eliminación en la columna 2:
[[ 1.      0.      0.      -227.07668]
 [ -0.      1.      0.      476.9226 ]
 [ 0.      0.      1.      -177.69215]]

Solución del sistema:
[-227.07668  476.9226 -177.69215]

```

b.

$$3.333x_1 + 15920x_2 - 10.333x_3 = 15913$$

$$2.222x_1 + 16.71x_2 + 9.612x_3 = 28.544$$

$$1.5611x_1 + 5.1791x_2 + 1.6852x_3 = 8.4254$$

```

B = np.array([
    [3.333, 15920, 10.333, 15913],
    [2.222, 16.71, 9.612, 28.544],
    [1.5611, 5.1791, 1.6852, 8.4254],
], dtype='float32')

solucion = gauss_jordan(B)
print("\nSolución del sistema:")
print(solucion)

```

```

Estado inicial de la matriz:
[[3.3330e+00 1.5920e+04 1.0333e+01 1.5913e+04]
 [2.2220e+00 1.6710e+01 9.6120e+00 2.8544e+01]
 [1.5611e+00 5.1791e+00 1.6852e+00 8.4254e+00]]

```

Intercambiando filas 0 y 2

Estado de la matriz tras el intercambio:

```
[[1.5611e+00 5.1791e+00 1.6852e+00 8.4254e+00]
 [2.2220e+00 1.6710e+01 9.6120e+00 2.8544e+01]
 [3.3330e+00 1.5920e+04 1.0333e+01 1.5913e+04]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[1.0000000e+00 3.3175967e+00 1.0794952e+00 5.3970919e+00]
 [0.0000000e+00 9.3382998e+00 7.2133622e+00 1.6551662e+01]
 [0.0000000e+00 1.5908942e+04 6.7350426e+00 1.5895012e+04]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 1.0000000e+00  0.0000000e+00 -1.4831798e+00 -4.8318005e-01]
 [ 0.0000000e+00  1.0000000e+00  7.7244920e-01  1.7724493e+00]
 [ 0.0000000e+00  0.0000000e+00 -1.2282114e+04 -1.2302781e+04]]
```

Estado de la matriz tras la eliminación en la columna 2:

```
[[ 1.      0.      0.      1.0024954]
 [ 0.      1.      0.      0.9987003]
 [-0.     -0.      1.      1.0016826]]
```

Solución del sistema:

```
[1.0024954 0.9987003 1.0016826]
```

c.

$$x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 = \frac{1}{6}$$

$$\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 + \frac{1}{5}x_4 = \frac{1}{7}$$

$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 + \frac{1}{6}x_4 = \frac{1}{8}$$

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 + \frac{1}{7}x_4 = \frac{1}{9}$$

```
C = np.array([
    [1, 1/2, 1/3, 1/4, 1/6],
    [1/2, 1/3, 1/4, 1/5, 1/7],
    [1/3, 1/4, 1/5, 1/6, 1/8],
    [1/4, 1/5, 1/6, 1/7, 1/9],
], dtype='float32')

solucion = gauss_jordan(C)
```

```
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[1.      0.5      0.33333334 0.25      0.16666667]
 [0.5     0.33333334 0.25      0.2       0.14285715]
 [0.33333334 0.25     0.2       0.16666667 0.125     ]
 [0.25     0.2       0.16666667 0.14285715 0.11111111]]
```

Intercambiando filas 0 y 3

Estado de la matriz tras el intercambio:

```
[[0.25     0.2       0.16666667 0.14285715 0.11111111]
 [0.5     0.33333334 0.25      0.2       0.14285715]
 [0.33333334 0.25     0.2       0.16666667 0.125     ]
 [1.      0.5      0.33333334 0.25      0.16666667]]
```

Estado de la matriz tras la eliminación en la columna 0:

```
[[ 1.      0.8      0.66666667 0.5714286 0.44444445]
 [ 0.     -0.06666666 -0.08333334 -0.0857143 -0.07936507]
 [ 0.     -0.01666668 -0.02222224 -0.02380954 -0.02314815]
 [ 0.     -0.3      -0.33333334 -0.3214286 -0.27777778 ]]
```

Intercambiando filas 1 y 2

Estado de la matriz tras el intercambio:

```
[[ 1.      0.8      0.66666667 0.5714286 0.44444445]
 [ 0.     -0.01666668 -0.02222224 -0.02380954 -0.02314815]
 [ 0.     -0.06666666 -0.08333334 -0.0857143 -0.07936507]
 [ 0.     -0.3      -0.33333334 -0.3214286 -0.27777778 ]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 1.      0.     -0.3999998 -0.57142836 -0.6666658 ]
 [-0.      1.      1.333333  1.4285711  1.3888878 ]
 [ 0.      0.      0.00555552 0.00952377 0.01322744]
 [ 0.      0.      0.06666657 0.10714275 0.13888857]]
```

Estado de la matriz tras la eliminación en la columna 2:

```
[[ 1.      0.      0.      0.1142875 0.28571594]
 [-0.      1.      0.     -0.857149 -1.7857188 ]
 [ 0.      0.      1.      1.7142905 2.3809555 ]
 [ 0.      0.      0.     -0.00714312 -0.01984157]]
```

Estado de la matriz tras la eliminación en la columna 3:

```
[[ 1.      0.      0.      0.     -0.03174219]
 [-0.      1.      0.      0.      0.5951971 ]
 [ 0.      0.      1.      0.     -2.3808553 ]
 [-0.     -0.     -0.      1.      2.7777152 ]]
```

Solución del sistema:

[-0.03174219 0.5951971 -2.3808553 2.7777152]

d.

$$2x_1 + x_2 - x_3 + x_4 - 3x_5 = 7$$

$$x_1 + 2x_3 - x_4 + x_5 = 2$$

$$-2x_2 - x_3 + x_4 - x_5 = -5$$

$$3x_1 + x_2 - 4x_3 + 5x_5 = 6$$

$$x_1 - x_2 - x_3 - x_4 + x_5 = -3$$

```
D = np.array([
    [2, 1, -1, 1, -3, 7],
    [1, 0, 2, -1, 1, 2],
    [0, -2, -1, 1, -1, -5],
    [3, 1, -4, 0, 5, 6],
    [1, -1, -1, -1, 1, -3],
], dtype='float32')

solucion = gauss_jordan(D)
print("\nSolución del sistema:")
print(solucion)
```

Estado inicial de la matriz:

```
[[ 2.  1. -1.  1. -3.  7.]
 [ 1.  0.  2. -1.  1.  2.]
 [ 0. -2. -1.  1. -1. -5.]
 [ 3.  1. -4.  0.  5.  6.]
 [ 1. -1. -1. -1.  1. -3.]]
```

Intercambiando filas 0 y 1

Estado de la matriz tras el intercambio:

```
[[ 1.  0.  2. -1.  1.  2.]
 [ 2.  1. -1.  1. -3.  7.]
 [ 0. -2. -1.  1. -1. -5.]
 [ 3.  1. -4.  0.  5.  6.]
 [ 1. -1. -1. -1.  1. -3.]]
```


Estado de la matriz tras la eliminación en la columna 0:

```
[[ 1.  0.  2. -1.  1.  2.]
 [ 0.  1. -5.  3. -5.  3.]
 [ 0. -2. -1.  1. -1. -5.]
 [ 0.  1. -10.  3.  2.  0.]
 [ 0. -1. -3.  0.  0. -5.]]
```

Estado de la matriz tras la eliminación en la columna 1:

```
[[ 1.  0.  2. -1.  1.  2.]
 [ 0.  1. -5.  3. -5.  3.]
 [ 0.  0. -11.  7. -11.  1.]
 [ 0.  0. -5.  0.  7. -3.]
 [ 0.  0. -8.  3. -5. -2.]]
```

Intercambiando filas 2 y 3

Estado de la matriz tras el intercambio:

```
[[ 1.  0.  2. -1.  1.  2.]
 [ 0.  1. -5.  3. -5.  3.]
 [ 0.  0. -5.  0.  7. -3.]
 [ 0.  0. -11.  7. -11.  1.]
 [ 0.  0. -8.  3. -5. -2.]]
```

Estado de la matriz tras la eliminación en la columna 2:

```
[[ 1.          0.          0.          -1.          3.8
  0.79999995]
 [ 0.          1.          0.          3.          -12.
  6.          ]
 [ -0.          -0.          1.          -0.          -1.4
  0.6          ]
 [ 0.          0.          0.          7.          -26.4
  7.6000004 ]
 [ 0.          0.          0.          3.          -16.2
  2.8000002 ]]
```

Intercambiando filas 3 y 4

Estado de la matriz tras el intercambio:

```
[[ 1.          0.          0.          -1.          3.8
  0.79999995]
 [ 0.          1.          0.          3.          -12.
  6.          ]
 [ -0.          -0.          1.          -0.          -1.4
  0.6          ]
 [ 0.          0.          0.          3.          -16.2
  2.8000002 ]
 [ 0.          0.          0.          7.          -26.4
  7.6000004 ]]
```

Estado de la matriz tras la eliminación en la columna 3:

```

[[ 1.      0.      0.      0.      -1.6000001  1.7333333]
 [ 0.      1.      0.      0.       4.2000001  3.1999998]
 [ 0.      0.      1.      0.      -1.4        0.6        ]
 [ 0.      0.      0.      1.      -5.4        0.9333334]
 [ 0.      0.      0.      0.      11.4        1.0666666]]

```

Estado de la matriz tras la eliminación en la columna 4:

```

[[1.      0.      0.      0.      0.      1.8830409 ]
 [0.      1.      0.      0.      0.      2.8070173 ]
 [0.      0.      1.      0.      0.      0.73099416]
 [0.      0.      0.      1.      0.      1.4385965 ]
 [0.      0.      0.      0.      1.      0.09356725]]

```

Solución del sistema:

```

[1.8830409  2.8070173  0.73099416  1.4385965  0.09356725]

```