Informe Tarea 9

Anthony Goyes

2024-07-27

Tabla de Contenidos

Conjunto de ejercicios tarea 9

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.

Literal a

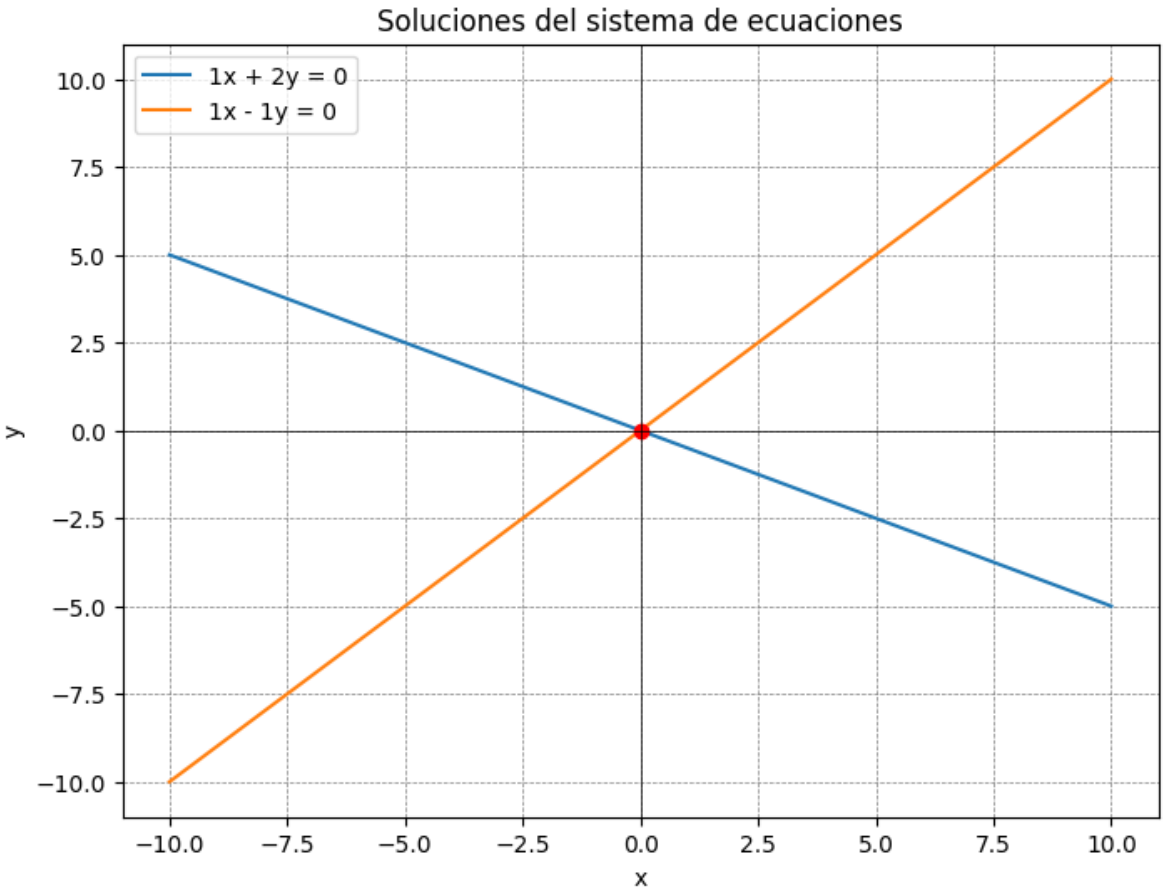
$$x_1 + 2x_2 = 0$$

$$x_1 - x_2 = 0$$

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

A = [
```

```
    [1, 2],
    [1, -1]
]

b = [0, 0]

# Resolver el sistema de ecuaciones
x = np.linalg.solve(A, b)
print(f"Solución: {x}")

# Crear un rango de valores para las gráficas
x_values = np.linspace(-10, 10, 400)
y1_values = (b[0] - A[0][0] * x_values) / A[0][1]
y2_values = (b[1] - A[1][0] * x_values) / A[1][1]

# Configurar la gráfica
plt.figure(figsize=(8, 6))
plt.plot(x_values, y1_values, label='1x + 2y = 0')
plt.plot(x_values, y2_values, label='1x - 1y = 0')

# Mostrar la solución como un punto
plt.plot(x[0], x[1], 'ro')  # 'ro' es el estilo para un punto rojo

# Añadir leyenda y etiquetas
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
plt.title('Soluciones del sistema de ecuaciones')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Solución: [ 0. -0.]
```



Literal b

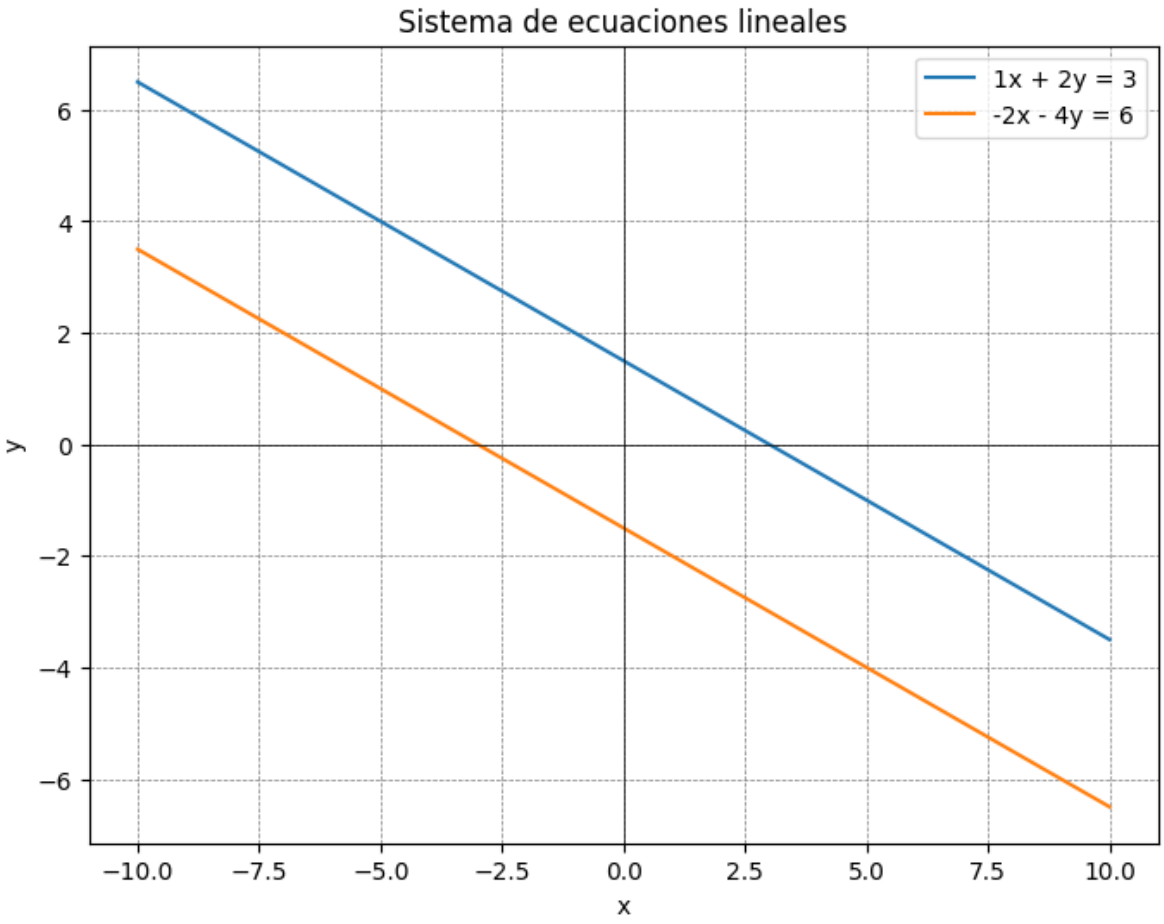
$$x_1 + 2x_2 = 3$$

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

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

A = [
    [1, 2],
    [-2, -4]
]
```

```
b = [3, 6]

try:
    x = np.linalg.solve(A, b)
    print(f"Solución: {x}")
except np.linalg.LinAlgError as e:
    print(f"Error al resolver el sistema de ecuaciones: {e}")

# Crear un rango de valores para las gráficas
x_values = np.linspace(-10, 10, 400)
y1_values = (b[0] - A[0][0] * x_values) / A[0][1]
y2_values = (b[1] - A[1][0] * x_values) / A[1][1]

# Configurar la gráfica
plt.figure(figsize=(8, 6))
plt.plot(x_values, y1_values, label='1x + 2y = 3')
plt.plot(x_values, y2_values, label='-2x - 4y = 6')

# Añadir leyenda y etiquetas
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.title('Sistema de ecuaciones lineales')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Error al resolver el sistema de ecuaciones: Singular matrix
```



Literal c

$$2x_1 + x_2 = -1$$

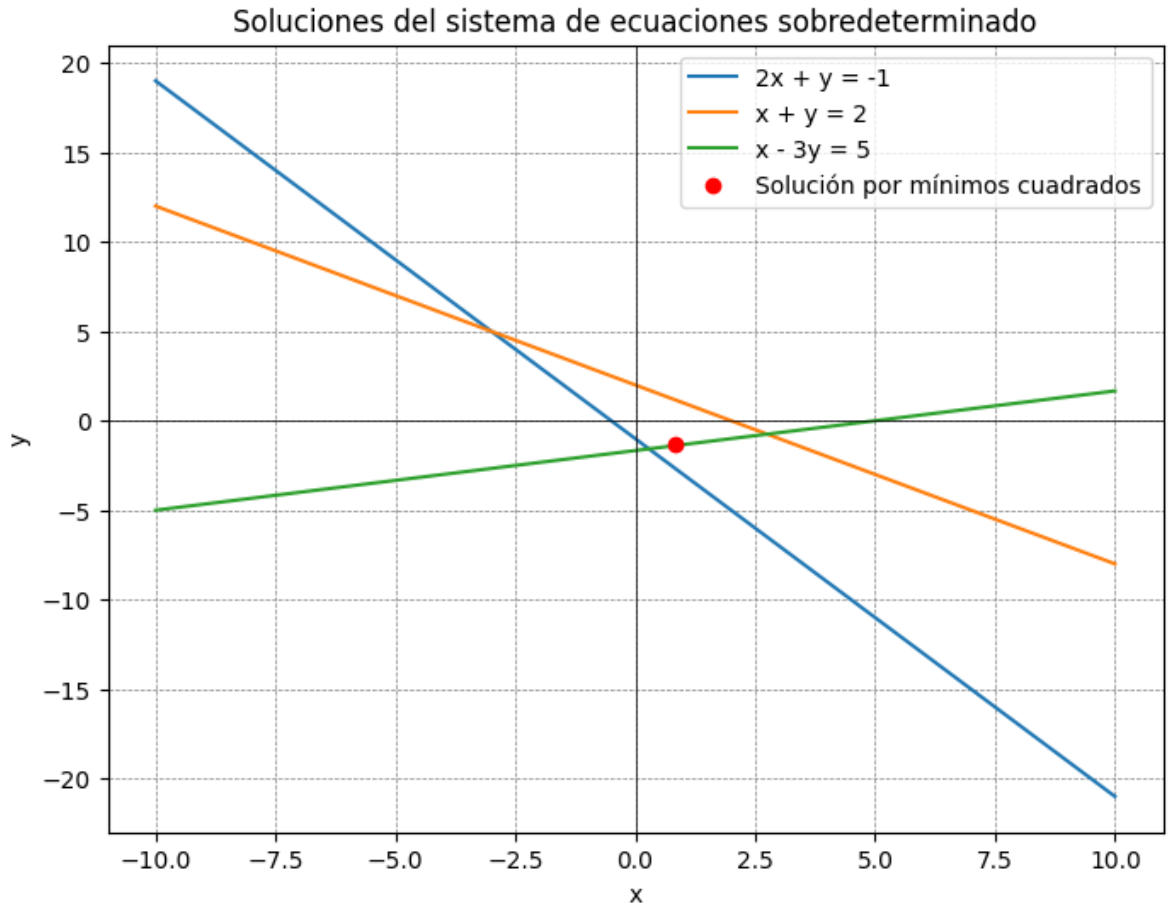
$$x_1 + x_2 = 2$$

$$x_1 - 3x_2 = 5$$

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

A = [
```

```
    [2, 1],
    [1, 1],
    [1, -3]
]

b = [-1, 2, 5]

# Resolver el sistema de ecuaciones usando mínimos cuadrados
x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
print(f"Solución por mínimos cuadrados: {x}")

# Crear un rango de valores para las gráficas
x_values = np.linspace(-10, 10, 400)
y1_values = (b[0] - A[0][0] * x_values) / A[0][1]
y2_values = (b[1] - A[1][0] * x_values) / A[1][1]
y3_values = (b[2] - A[2][0] * x_values) / A[2][1]

# Configurar la gráfica
plt.figure(figsize=(8, 6))
plt.plot(x_values, y1_values, label='2x + y = -1')
plt.plot(x_values, y2_values, label='x + y = 2')
plt.plot(x_values, y3_values, label='x - 3y = 5')

# Mostrar la solución como un punto
plt.plot(x[0], x[1], 'ro', label='Solución por mínimos cuadrados')  # 'ro' es el estilo para

# Añadir leyenda y etiquetas
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.title('Soluciones del sistema de ecuaciones sobredeterminado')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Solución por mínimos cuadrados: [ 0.83333333 -1.27272727]
```



Literal d)

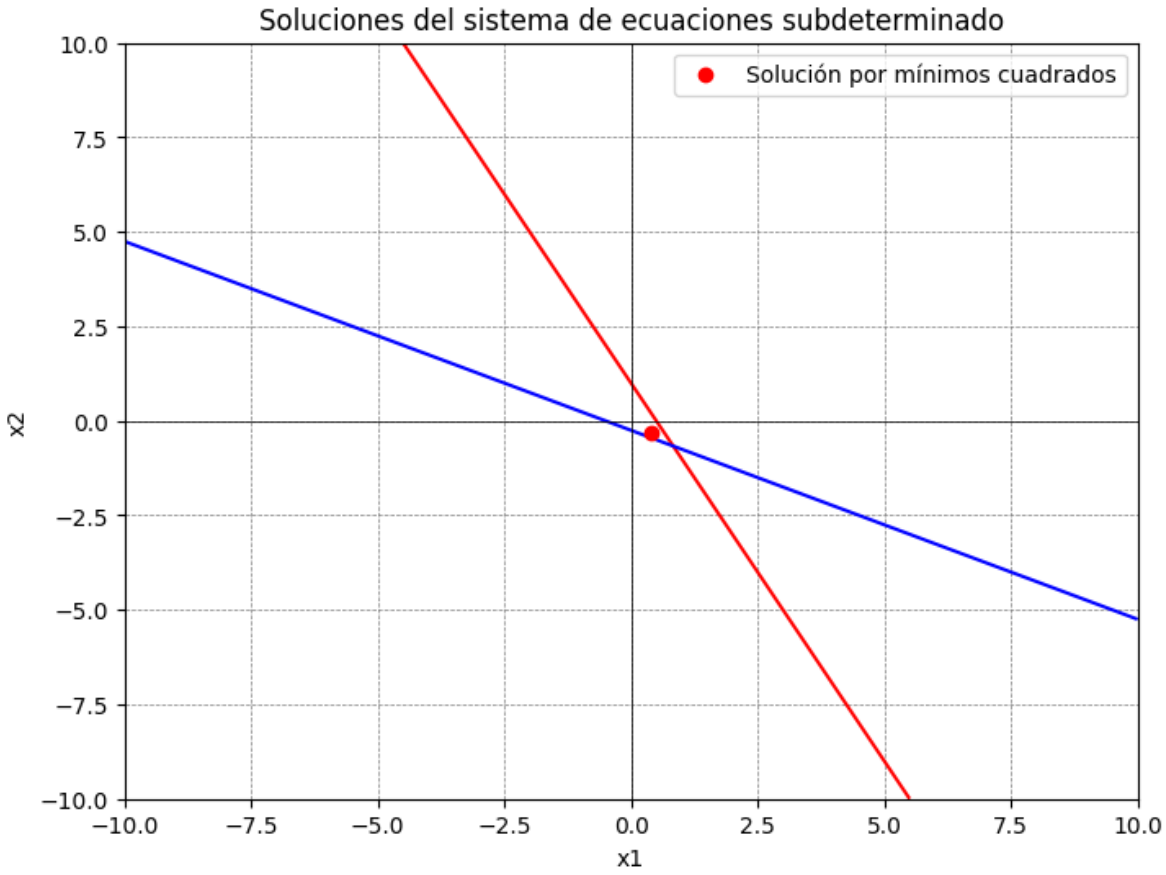
$$2x_1 + x_2 + x_3 = 1$$

$$2x_1 + 4x_2 - x_3 = -1$$

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

A = [
    [2, 1, 1],
    [2, 4, -1]
]
```

```
b = [1, -1]

# Resolver el sistema de ecuaciones usando mínimos cuadrados
x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
print(f"Solución por mínimos cuadrados: {x}")

# Crear un rango de valores para las gráficas
x1_values = np.linspace(-10, 10, 400)
x2_values = np.linspace(-10, 10, 400)
x3_values = (b[0] - A[0][0] * x1_values - A[0][1] * x2_values) / A[0][2]

# Configurar la gráfica
plt.figure(figsize=(8, 6))
X1, X2 = np.meshgrid(x1_values, x2_values)
Z1 = (b[0] - A[0][0] * X1 - A[0][1] * X2) / A[0][2]
Z2 = (b[1] - A[1][0] * X1 - A[1][1] * X2) / A[1][2]

# Graficar las superficies de las ecuaciones
plt.contour(X1, X2, Z1, levels=[0], colors='r', label='2x1 + x2 + x3 = 1')
plt.contour(X1, X2, Z2, levels=[0], colors='b', label='2x1 + 4x2 - x3 = -1')

# Mostrar la solución como un punto
plt.plot(x[0], x[1], 'ro', label='Solución por mínimos cuadrados')  # 'ro' es el estilo para

# Añadir leyenda y etiquetas
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.title('Soluciones del sistema de ecuaciones subdeterminado')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```

```
Solución por mínimos cuadrados: [ 0.38961039 -0.31168831  0.53246753]
```



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 $1 = -1, 2 = 2, 3 = 3$.)

Literal a

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

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

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

```python
import numpy as np

def round_to_two_digits(matrix):
    return np.round(matrix, decimals=2)

def gaussian_elimination(A, b):
    n = len(A)
    A = round_to_two_digits(np.array(A, dtype=float))
    b = round_to_two_digits(np.array(b, dtype=float))

    # Eliminación hacia adelante
    for i in range(n):
        for j in range(i+1, n):
            factor = A[j][i] / A[i][i]
            factor = round(factor, 2)
            for k in range(i, n):
                A[j][k] -= factor * A[i][k]
                A[j][k] = round(A[j][k], 2)
            b[j] -= factor * b[i]
            b[j] = round(b[j], 2)
    return A, b

def back_substitution(A, b):
    n = len(A)
    x = np.zeros(n)

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

    return x

# Literal a
A_a = [
    [-1, 4, 1],
    [5/3, 2/3, -2/3],
    [2, 1, 4]
]
```

```
b_a = [8, 1, 11]

A_a, b_a = gaussian_elimination(A_a, b_a)
x_a = back_substitution(A_a, b_a)

print("Solución para el sistema a:", x_a)
```

```
Solución para el sistema a: [0.7  1.68 1.98]
```

Literal b

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

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

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

```
# Literal b
A_b = [
    [4, 2, -1],
    [1/9, 1/9, -1/3],
    [1, 4, 2]
]

b_b = [-5, -1, 9]

A_b, b_b = gaussian_elimination(A_b, b_b)
x_b = back_substitution(A_b, b_b)

print("Solución para el sistema b:", x_b)
```

```
Solución para el sistema b: [-1.  1.  3.]
```

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

```python
import numpy as np

def eliminacion_gaussiana(A: np.ndarray) -> np.ndarray:
    A = np.array(A, dtype=float)
    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1)."
    n = A.shape[0]

    try:
        for i in range(0, n - 1):  # loop por columna

            # --- encontrar pivote
            p = None  # default, first element
            for pi in range(i, n):
                if A[pi, i] == 0:
                    # must be nonzero
                    continue

                if p is None:
                    # first nonzero element
                    p = pi
                    continue

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

            if p is None:
                # no pivot found.
                print(f"No se encontró pivote en la columna {i}. La matriz en este punto es:"
                print(A)
                raise ValueError("No existe solución única.")

            if p != i:
                # swap rows
                print(f"Intercambiando filas {i} y {p}")
                A[[i, p]] = A[[p, i]]  # Swap rows

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

        if A[n - 1, n - 1] == 0:
```

```
            print(f"El elemento pivote en la última fila es cero. La matriz en este punto es
            print(A)
            raise ValueError("No existe solución única.")

        # --- Sustitución hacia atrás
        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

    except ValueError as e:
        print("Error en la eliminación gaussiana:", e)
        return None
```

Literal a

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

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

$$x_1 + x_2 = 3$$

```
A = [
    [1, -1, 3, 2],
    [3, -3, 1, -1],
    [1, 1, 0, 3]
]

x = eliminacion_gaussiana(A)
if x is not None:
    print("Solución:", x)
else:
    print("No se pudo encontrar una solución única para el sistema de ecuaciones.")
```

```
Intercambiando filas 1 y 2
Solución: [1.1875 1.8125 0.875 ]
```

Literal b

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

$$-x_1 + 2x_3 = 3$$

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

```python
A = [
    [2, -1.5, 3, 1],
    [-1, 0, 2, 3],
    [4, -4.5, 5, 1]
]

x = eliminacion_gaussiana(A)
if x is not None:
    print("Solución:", x)
else:
    print("No se pudo encontrar una solución única para el sistema de ecuaciones.")
```

```
Intercambiando filas 0 y 1
Solución: [-1. -0.  1.]
```

Literal c

$$2x_1 = 3$$

$$x_1 + 1.5x_2 = 4.5$$

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

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

```
A = [
    [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]
]

x = eliminacion_gaussiana(A)
if x is not None:
    print("Solución:", x)
else:
    print("No se pudo encontrar una solución única para el sistema de ecuaciones.")
```

```
Intercambiando filas 0 y 1
Solución: [ 1.5  2.  -1.2  3. ]
```

Literal d

$$x_1 + x_2 + x_4 = 2$$

$$2x_1 + x_2 - x_3 + x_4 = 1$$

$$4x_1 - x_2 - 2x_3 + 2x_4 = 0$$

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

```
A = [
    [1, 1, 0, 1, 2],
    [2, 1, -1, 1, 1],
    [4, -1, -2, 2, 0],
    [3, -1, -1, 2, -3]
]

x = eliminacion_gaussiana(A)
if x is not None:
    print("Solución:", x)
else:
    print("No se pudo encontrar una solución única para el sistema de ecuaciones.")
```

```
El elemento pivote en la última fila es cero. La matriz en este punto es:
[[ 1.  1.  0.  1.  2.]
 [ 0. -1. -1. -1. -3.]
 [ 0.  0.  3.  3.  7.]
 [ 0.  0.  0.  0. -4.]]
Error en la eliminación gaussiana: No existe solución única.
No se pudo encontrar una solución única para el sistema de ecuaciones.
```

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

Literal a

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

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

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

```python
def eliminacion_gaussiana(A: np.ndarray) -> 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]

    for i in range(0, n - 1):  # loop por columna

        # --- encontrar pivote
        p = None  # default, first element
        for pi in range(i, n):
            if A[pi, i] == 0:
                # must be nonzero
                continue

            if p is None:
                # first nonzero element
                p = pi
                continue
```

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

        if p is None:
            # no pivot found.
            raise ValueError("No existe solución única.")

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

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

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

    # --- Sustitución hacia atrás
    solucion = np.zeros(n, dtype=np.float32)
    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 = np.array([[1/4, 1/5, 1/6, 9],
              [1/3, 1/4, 1/5, 8],
              [1/2, 1, 2, 8]], dtype=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
[-227.07666  476.92264 -177.69217]
```

Literal b

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

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

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

```python
A = 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=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
Intercambiando filas 0 y 2
[1.0024955 0.9987004 1.0016824]
```

Literal c

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

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

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

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

```
A = 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=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
Intercambiando filas 0 y 3
Intercambiando filas 1 y 2
[-0.03174075  0.5951853  -2.3808312   2.7777011 ]
```

Literal d

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

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

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

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

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

```
A = 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=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
Intercambiando filas 0 y 1
Intercambiando filas 2 y 3
Intercambiando filas 3 y 4
[1.8830411  2.807017   0.73099416 1.4385967  0.09356727]
```

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

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

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

```
import numpy as np

def eliminacion_gaussiana(A):
    A = np.array(A, dtype=float)
    n = A.shape[0]

    for i in range(n):
        # Pivote
        max_row = i + np.argmax(np.abs(A[i:, i]))
        if A[max_row, i] == 0:
            continue  # no pivot in this column, pass to next column

        # Intercambio de filas
        A[[i, max_row]] = A[[max_row, i]]

        # Normalizar fila pivote
        A[i] = A[i] / A[i, i]

        # Eliminar otras filas
        for j in range(i + 1, n):
            A[j] = A[j] - A[j, i] * A[i]

    return A
```

```python
def analizar_sistema(alpha):
    A = np.array([
        [1, -1, alpha],
        [-1, 2, -alpha],
        [alpha, 1, 1]
    ])

    b = np.array([-2, 3, 2])
    Ab = np.hstack([A, b.reshape(-1, 1)])  # Matriz aumentada

    Ab_reducida = eliminacion_gaussiana(Ab.copy())
    A_reducida = eliminacion_gaussiana(A.copy())

    rango_A = np.linalg.matrix_rank(A_reducida)
    rango_Ab = np.linalg.matrix_rank(Ab_reducida)

    if rango_A < rango_Ab:
        print(f"a) No tiene solución para alpha = {alpha}")
    elif rango_A == rango_Ab == 3:
        print(f"c) Tiene una única solución para alpha = {alpha}")
        x = np.linalg.solve(A, b)
        print(f"Solución: {x}")
    elif rango_A == rango_Ab < 3:
        print(f"b) Tiene infinitas soluciones para alpha = {alpha}")
    else:
        print(f"Condición no contemplada para alpha = {alpha}")

# Análisis para diferentes valores de alpha
alphas = [1, -1/2, 2]

for alpha in alphas:
    analizar_sistema(alpha)
```

```
a) No tiene solución para alpha = 1
c) Tiene una única solución para alpha = -0.5
Solución: [-0.66666667  1.          0.66666667]
c) Tiene una única solución para alpha = 2
Solución: [ 1.  1. -1.]
```

6. Suponga que en un sistema biológico existen n especies de animales y m fuentes de alimento. Si representa la población de las j-ésimas especies, para cada = 1, , ; ; representa el suministro diario disponible del i-ésimo alimento y representa la cantidad del i-ésimo alimento.

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

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$x = (x_j) = (1000, 500, 350, 400)$ y $b = (b_i) = (3500, 2700, 900)$.

a.¿Existe suficiente alimento para satisfacer el promedio consumo 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?

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

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

```
import numpy as np

# Definir la matriz A y los vectores x y b
A = np.array([
    [1, 2, 0, 3],
    [0, 0, 2, 2],
    [0, 0, 1, 1]
])

x = np.array([1000, 500, 350, 400])
b = np.array([3500, 2700, 900])

# Calcular Ax para verificar si se cumple la condición de consumo
Ax = np.dot(A, x)
```

```python
# a) Verificar si hay suficiente alimento para satisfacer el consumo promedio diario
suficiente_alimento = np.all(Ax <= b)
print("a) ¿Existe suficiente alimento para satisfacer el consumo promedio diario?:", suficien

# b) Calcular el máximo número de animales que se podrían agregar de forma individual
incrementos = []
for i in range(A.shape[1]):
    # Crear una copia del vector x
    x_temp = x.copy()
    max_increment = float('inf')
    for j in range(A.shape[0]):
        if A[j, i] != 0:
            max_increment = min(max_increment, (b[j] - np.dot(A[j], x_temp)) / A[j, i])
    incrementos.append(max_increment)

print("b) Incrementos posibles para cada especie:", incrementos)

# c) Si la especie 1 se extingue, ¿qué cantidad de incremento individual de las especies rest
A_ext1 = A[:, 1:]
x_ext1 = x[1:]
incrementos_ext1 = np.dot(np.linalg.pinv(A_ext1), b - np.dot(A_ext1, x_ext1))
print("c) Incremento individual de las especies restantes si la especie 1 se extingue:", incr

# d) Si la especie 2 se extingue, ¿qué cantidad de incremento individual de las especies rest
A_ext2 = np.delete(A, 1, axis=1)
x_ext2 = np.delete(x, 1)
incrementos_ext2 = np.dot(np.linalg.pinv(A_ext2), b - np.dot(A_ext2, x_ext2))
print("d) Incremento individual de las especies restantes si la especie 2 se extingue:", incr
```

```
a) ¿Existe suficiente alimento para satisfacer el consumo promedio diario?: True
b) Incrementos posibles para cada especie: [np.float64(300.0), np.float64(150.0), np.float64
c) Incremento individual de las especies restantes si la especie 1 se extingue: [125.8823529
d) Incremento individual de las especies restantes si la especie 2 se extingue: [ 97.2727272
```

7. Repita el ejercicio 4 con el método Gauss-Jordan.

Literal a

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

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

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

```python
import numpy as np

def gauss_jordan_32bit(a, b):
    """
    Resuelve el sistema de ecuaciones lineales Ax = b usando el método de Gauss-Jordan con pr
    :param a: matriz de coeficientes A
    :param b: vector de términos independientes b
    :return: vector solución x
    """
    # Convertir A y b a matrices de precisión de 32 bits
    a = np.array(a, dtype=np.float32)
    b = np.array(b, dtype=np.float32)
    n = len(b)
    augmented_matrix = np.hstack([a, b.reshape(-1, 1)])

    # Realiza el método de Gauss-Jordan
    for i in range(n):
        # Encuentra el pivote
        max_row = np.argmax(np.abs(augmented_matrix[i:n, i])) + i
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

        # Normaliza la fila del pivote
        augmented_matrix[i] = augmented_matrix[i] / augmented_matrix[i, i]

        # Elimina los elementos por encima y por debajo del pivote
        for j in range(n):
            if j != i:
                augmented_matrix[j] -= augmented_matrix[i] * augmented_matrix[j, i]

    # Extrae la solución
    x = augmented_matrix[:, -1]
    return x

# Datos del sistema
A = [
    [1/4, 1/5, 1/6],
```

```
    [1/3, 1/4, 1/5],
    [1/2, 1, 2]
]
B = [9, 8, 8]

# Resolver el sistema
solucion = gauss_jordan_32bit(A, B)
print("La solución es:", solucion)
```

```
La solución es: [-227.07693  476.92322 -177.69237]
```

Literal b

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

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

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

```
import numpy as np

def gauss_jordan_32bit(a, b):
    """
    Resuelve el sistema de ecuaciones lineales Ax = b usando el método de Gauss-Jordan con pr
    :param a: matriz de coeficientes A
    :param b: vector de términos independientes b
    :return: vector solución x
    """
    # Convertir A y b a matrices de precisión de 32 bits
    a = np.array(a, dtype=np.float32)
    b = np.array(b, dtype=np.float32)
    n = len(b)
    augmented_matrix = np.hstack([a, b.reshape(-1, 1)])

    # Realiza el método de Gauss-Jordan
    for i in range(n):
        # Encuentra el pivote
        max_row = np.argmax(np.abs(augmented_matrix[i:n, i])) + i
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]
```

```
        # Normaliza la fila del pivote
        augmented_matrix[i] = augmented_matrix[i] / augmented_matrix[i, i]

        # Elimina los elementos por encima y por debajo del pivote
        for j in range(n):
            if j != i:
                augmented_matrix[j] -= augmented_matrix[i] * augmented_matrix[j, i]

    # Extrae la solución
    x = augmented_matrix[:, -1]
    return x

# Datos del sistema
A = [
    [3.333, 15920, 10.333],
              [2.222, 16.71, 9.612],
              [1.5611, 5.1791, 1.6852]
]
B = [15913, 28.544, 8.4254]

# Resolver el sistema
solucion = gauss_jordan_32bit(A, B)
print("La solución es:", solucion)
```

```
La solución es: [1.0023832  0.99870026 1.0017889 ]
```

Literal c

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

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

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

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

```python
import numpy as np

def gauss_jordan_32bit(a, b):
    """
    Resuelve el sistema de ecuaciones lineales Ax = b usando el método de Gauss-Jordan con pr
    :param a: matriz de coeficientes A
    :param b: vector de términos independientes b
    :return: vector solución x
    """
    # Convertir A y b a matrices de precisión de 32 bits
    a = np.array(a, dtype=np.float32)
    b = np.array(b, dtype=np.float32)
    n = len(b)
    augmented_matrix = np.hstack([a, b.reshape(-1, 1)])

    # Realiza el método de Gauss-Jordan
    for i in range(n):
        # Encuentra el pivote
        max_row = np.argmax(np.abs(augmented_matrix[i:n, i])) + i
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

        # Normaliza la fila del pivote
        augmented_matrix[i] = augmented_matrix[i] / augmented_matrix[i, i]

        # Elimina los elementos por encima y por debajo del pivote
        for j in range(n):
            if j != i:
                augmented_matrix[j] -= augmented_matrix[i] * augmented_matrix[j, i]

    # Extrae la solución
    x = augmented_matrix[:, -1]
    return x

# Datos del sistema
A = [
    [1, 1/2, 1/3, 1/4],
              [1/2, 1/3, 1/4, 1/5],
              [1/3, 1/4, 1/5, 1/6],
              [1/4, 1/5, 1/6, 1/7]
]
B= [1/6, 1/7, 1/8, 1/9]
# Resolver el sistema
```

```
solucion = gauss_jordan_32bit(A, B)
print("La solución es:", solucion)
```

```
La solución es: [-0.03174686  0.5952499  -2.380982    2.7777972 ]
```

Literal d

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

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

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

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

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

```
import numpy as np

def gauss_jordan_32bit(a, b):
    """
    Resuelve el sistema de ecuaciones lineales Ax = b usando el método de Gauss-Jordan con pr
    :param a: matriz de coeficientes A
    :param b: vector de términos independientes b
    :return: vector solución x
    """
    # Convertir A y b a matrices de precisión de 32 bits
    a = np.array(a, dtype=np.float32)
    b = np.array(b, dtype=np.float32)
    n = len(b)
    augmented_matrix = np.hstack([a, b.reshape(-1, 1)])

    # Realiza el método de Gauss-Jordan
    for i in range(n):
        # Encuentra el pivote
        max_row = np.argmax(np.abs(augmented_matrix[i:n, i])) + i
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]
```

```
        # Normaliza la fila del pivote
        augmented_matrix[i] = augmented_matrix[i] / augmented_matrix[i, i]

        # Elimina los elementos por encima y por debajo del pivote
        for j in range(n):
            if j != i:
                augmented_matrix[j] -= augmented_matrix[i] * augmented_matrix[j, i]

    # Extrae la solución
    x = augmented_matrix[:, -1]
    return x

# Datos del sistema
A = [
    [2, 1, -1, 1, -3],
               [1, 0, 2, -1, 1],
               [0, -2, -1, 1, -1],
               [3, 1, -4, 0, 5],
               [1, -1, -1, -1, 1]
]
B= [7, 2, -5, 6, -3]
# Resolver el sistema
solucion = gauss_jordan_32bit(A, B)
print("La solución es:", solucion)
```

```
La solución es: [1.8830408  2.8070176  0.73099405 1.4385962  0.09356717]
```