Tarea 9 - Eliminación gaussiana vs Gauss-Jordan

Nombre: Alexis Bautista
Fecha de Entrega: 08 de enero de 2025
Paralelo: GR1CC
Enlace de GitHub: https://github.com/alexis-bautista/Tarea09-MN

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)
$x_1 + 2x_2 = 0,$
$x_1 - x_2 = 0$

Solucion:

$$\begin{bmatrix} 1 & 2 & 0 \\ 1 & -1 & 0 \end{bmatrix}$$

$$-F1 + F2 \rightarrow F2$$

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & -3 & 0 \end{bmatrix}$$

Entonces tenemos el siguiente sistema de ecuaciones:

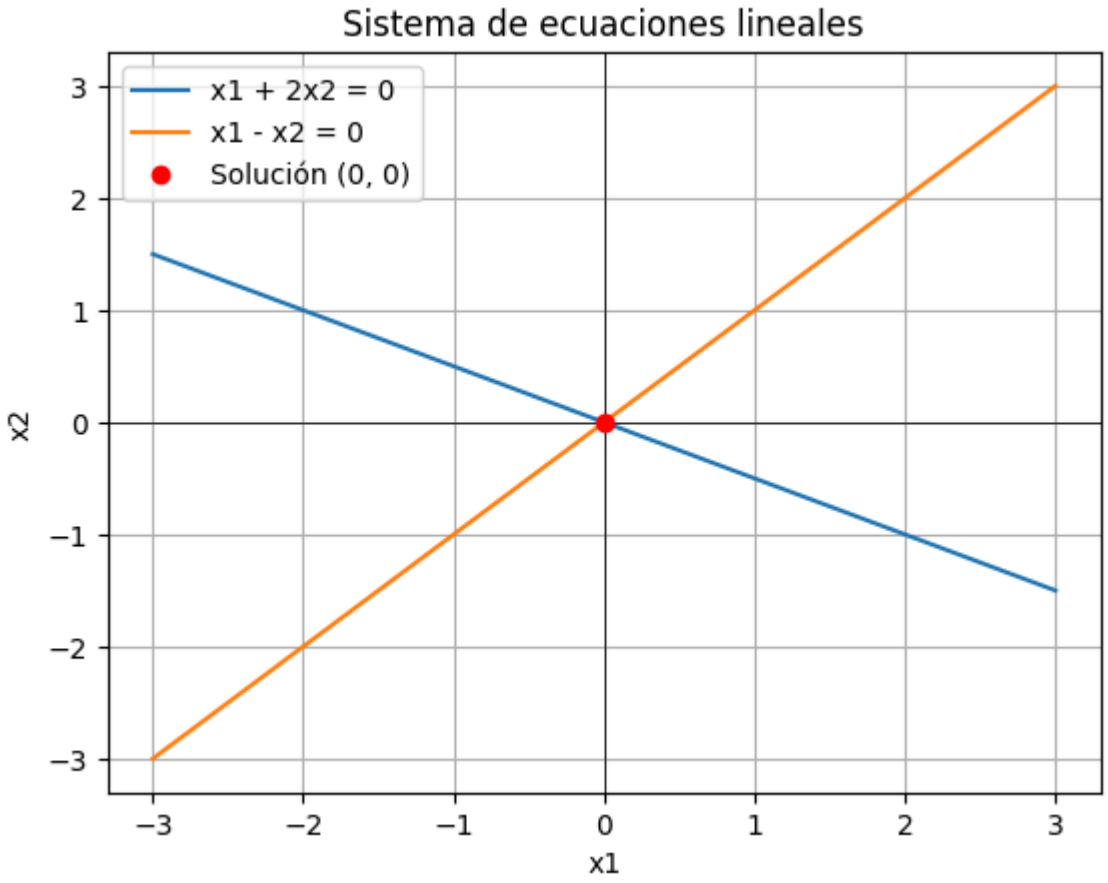
$x_1 + 2x_2 = 0$
$-3x_2 = 0$

Que resolviendo llegamos a:

$x_2 = 0$
$x_1 + 2(0) = 0 \rightarrow x_1 = 0$

In [58]:
```
import numpy as np

import matplotlib.pyplot as plt

x_vals = np.linspace(-3, 3, 100)
y1 = -x_vals / 2  # Recta de la primera ecuación
y2 = x_vals       # Recta de la segunda ecuación

# Graficamos
```

```
plt.axhline(0, color="black", linewidth=0.5)
plt.axvline(0, color="black", linewidth=0.5)

plt.plot(x_vals, y1, label="x1 + 2x2 = 0")
plt.plot(x_vals, y2, label="x1 - x2 = 0")

# Punto de intersección (0, 0)
plt.plot(0, 0, "ro", label="Solución (0, 0)")

plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.title("Sistema de ecuaciones lineales")
plt.grid(True)
plt.show()
```



Explicación

En un sistema de ecuaciones lineales en dos variables, cada ecuación corresponde a una recta en el plano. Hay solución cuando todas las rectas se intersectan en al menos un punto común. En este caso se pudo observar que el sistema de ecuaciones tiene una solución ya que las rectas se intersecan en el punto 0.

b)

$x_1 + 2x_2 = 3,$

$-2x_1 - 4x_2 = 6$

Solución: $\begin{bmatrix} 1 & 2 & 3 \\ -2 & -4 & 6 \end{bmatrix}$

$2F1 + F2 \rightarrow F2$

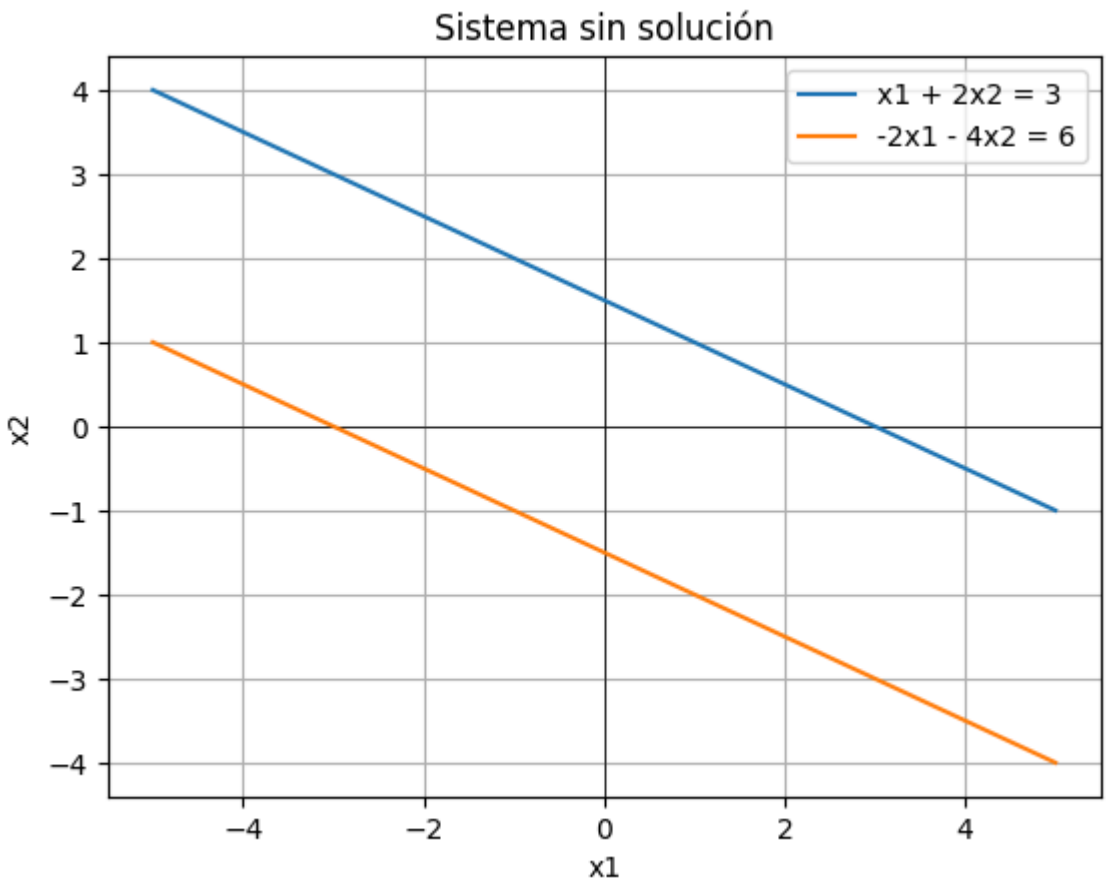
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 12 \end{bmatrix}$$

Entonces tenemos el siguiente sistema de ecuaciones:

$x_1 + 2x_2 = 3$
$0 = 12$

Dado que 0 es diferente de doce podemos concluir que el sistema de ecuaciones no tiene solución

In [59]:
```
x = np.linspace(-5, 5, 100)
y_1 = (3 - x) / 2
y_2 = -(6 + 2*x) / 4

plt.axhline(0, color="black", linewidth=0.5)
plt.axvline(0, color="black", linewidth=0.5)

plt.plot(x, y_1, label="x1 + 2x2 = 3")
plt.plot(x, y_2, label="-2x1 - 4x2 = 6")

plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Sistema sin solución")
plt.legend()
plt.grid(True)
plt.show()
```



Explicación:

En un sistema de ecuacuiones si las rectas son paralelas sin coincidir nunca en ningun punto, podemos decir que es un sistema de ecuaciones sin solución. En este caso en este sistema de dos variables se obtienen dos rectas paralelas por lo que podemos confirmar que es un sistema sin solución.

c)

$2x_1 + x_2 = -1,$

$x_1 - x_2 = 2,$

$x_1 - 3x_2 = 5$

Solución:

$$\begin{bmatrix} 2 & 1 & -1 \\ 1 & -1 & 2 \\ 1 & -3 & 5 \end{bmatrix}$$

$-\frac{1}{2}F1 + F2 \to F2$

$-\frac{1}{2}F1 + F3 \to F3$

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & -\frac{3}{2} & \frac{5}{2} \\ 0 & -\frac{7}{2} & \frac{11}{2} \end{bmatrix}$$

$-\frac{7}{3}F2 + F3 \to F3$

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & -\frac{3}{2} & \frac{5}{2} \\ 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

Entonces tenemos el siguiente sistema de ecuaciones:

$2x_1 + x_2 = -1$

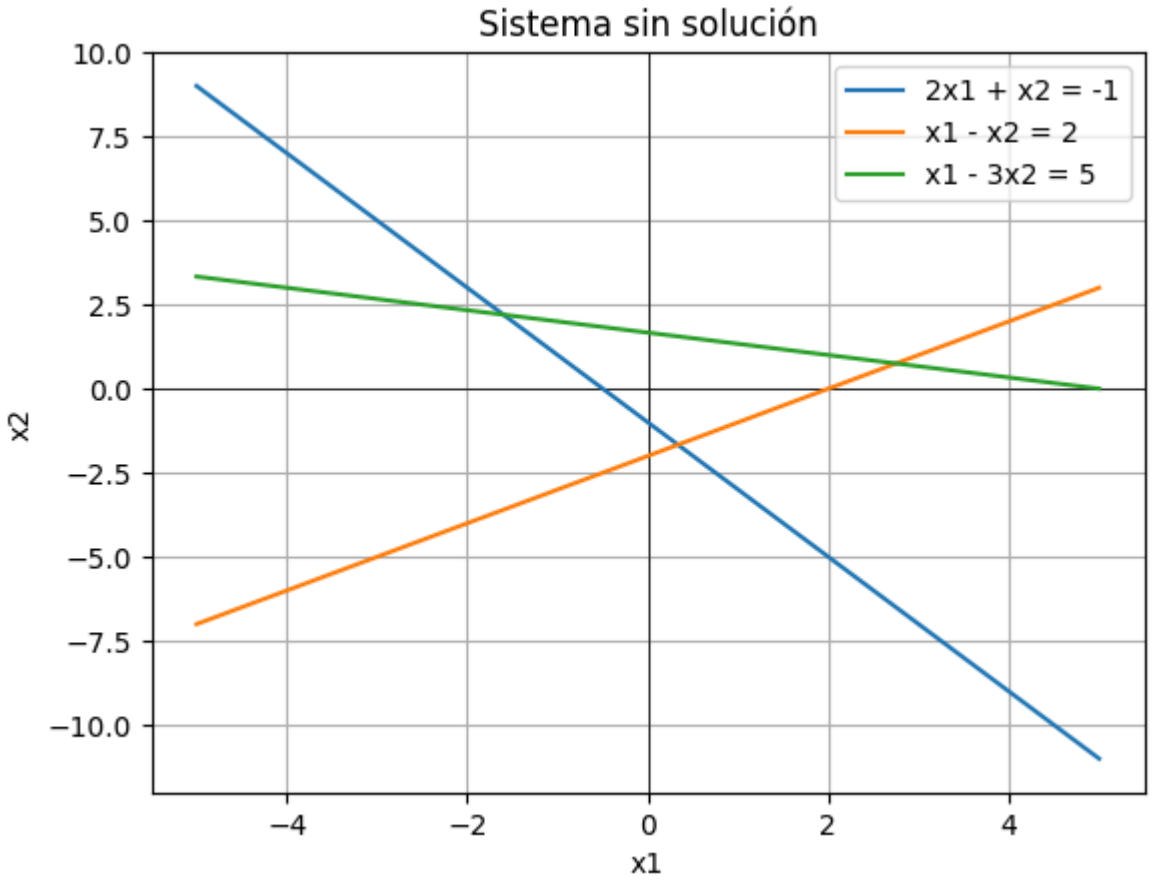
$-\frac{3}{2} = \frac{5}{2}$

$0 = -\frac{1}{3}$

Dado que 0 es diferente de $-\frac{1}{3}$ podemos concluir que el sistema de ecuaciones no tiene solución

In [60]:
```
x = np.linspace(-5, 5, 100)
y_1 = -1 - 2 * x
y_2 = x - 2
y_3 = - (x - 5) / 3

plt.axhline(0, color="black", linewidth=0.5)
plt.axvline(0, color="black", linewidth=0.5)

plt.plot(x, y_1, label="2x1 + x2 = -1")
plt.plot(x, y_2, label="x1 - x2 = 2")
plt.plot(x, y_3, label="x1 - 3x2 = 5")
```

```
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Sistema sin solución")
plt.legend()
plt.grid(True)
plt.show()
```



Explicación

El sistema de ecuaciones lineales no tiene solución, ya que las tres rectas que nunca se intersectan. Por lo tanto podemos concluir que este sistema de ecuaciones no tiene solución.

d)

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

Solución:

$$\begin{bmatrix} 2 & 1 & 1 & 1 \\ 2 & 4 & -1 & -1 \end{bmatrix}$$

$$-F1 + F2 \rightarrow F2$$

$$\begin{bmatrix} 2 & 1 & 1 & 1 \\ 0 & 3 & -2 & -2 \end{bmatrix}$$

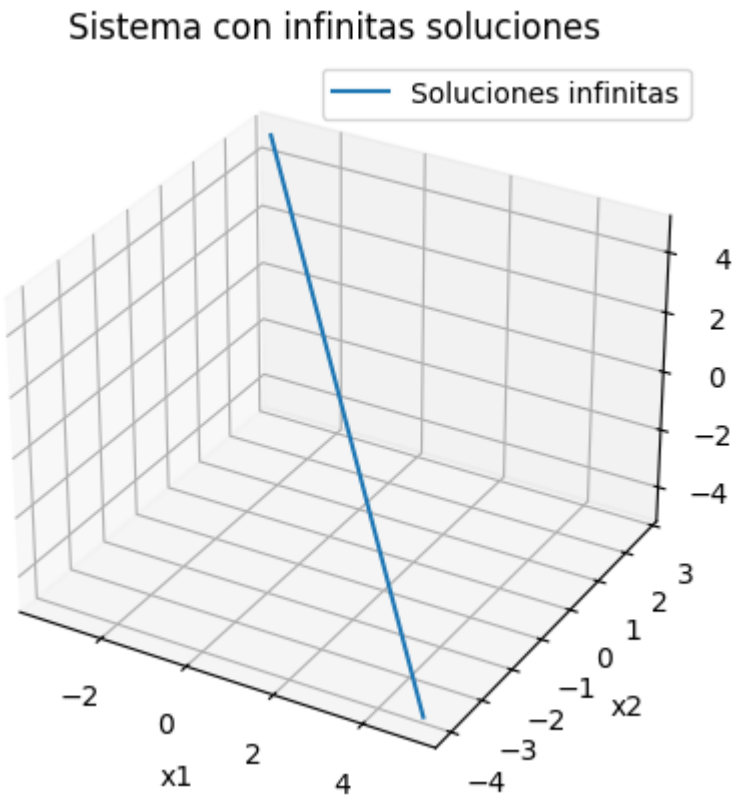
Entonces tenemos el siguiente sistema de ecuaciones:

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

$$x = \begin{bmatrix} 5/6 - 5/6 * x_3 \\ -2/3 + 2/3 * x_3 \\ x_3 \end{bmatrix}$$

Debido a que tenemos dos ecuaciones y tres variables podemos decir que este sistema de ecuaciones tiene infinitas soluciones.

In [61]:
```
t = np.linspace(-5, 5, 100)
x3 = t
x2 = -2/3+(2/3) * t
x1 = 5/6 - (5/6) * t

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x1, x2, x3, label='Soluciones infinitas')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
ax.legend()
plt.title('Sistema con infinitas soluciones')
plt.show()
```



Explicación

Cada ecuación representa la misma linea en el plano es decir contiene infinitos puntos, de ahí que el sistema tenga infinitas soluciones.

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 $x_1 = -1$, $x_2 = 2$, $x_3 = 3$.)

In [62]:
```python
def eliminacion_gaussiana_2dig(A, b):
    # A y b arreglos NumPy de punto flotante
    n = len(b)

    # Función auxiliar para redondear a 2 dígitos
    r2 = lambda x: float(f"{x:.2f}")

    # Eliminación hacia adelante con redondeo
    for i in range(n):
        # Sin intercambio de filas
        pivote = r2(A[i, i])
        for j in range(i + 1, n):
            factor = r2(A[j, i] / pivote)
            b[j] = r2(b[j] - factor * b[i])
            for k in range(i, n):
                A[j, k] = r2(A[j, k] - factor * A[i, k])

    # Sustitución hacia atrás con redondeo
    x = [0.0]*n
    for i in range(n-1, -1, -1):
        s = b[i]
        for j in range(i+1, n):
            s = r2(s - A[i, j]*x[j])
        x[i] = r2(s / A[i, i])
    return x
```

a)

$$-x_1 + 4x_2 + x_3 = 8,$$
$$\frac{5}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 = 1,$$
$$2x_1 + x_2 + 4x_3 = 11$$

In [63]:
```python
A = np.array([
    [-1.0, 4.0, 1.0],
    [5.0/3.0, 2.0/3.0, 2.0/3.0],
    [2.0, 1.0, 4.0]
], dtype=float)

b = np.array([8.0, 1.0, 11.0], dtype=float)

sol = eliminacion_gaussiana_2dig(A, b)
print("Solución aproximada con aritmética de redondeo a 2 dígitos:", sol)
```

```
Solución aproximada con aritmética de redondeo a 2 dígitos: [-0.99, 1.0, 3.01]
```

b)

$$4x_1 + 2x_2 - x_3 = -5,$$
$$\frac{1}{9}x_1 + \frac{1}{9}x_2 - \frac{1}{3}x_3 = -1,$$
$$x_1 + 4x_2 + 2x_3 = 9$$

In [64]:
```python
A = np.array([
    [4.0 , 2.0, -1.0],
    [1.0/9.0, 1.0/9.0, -1.0/3.0],
```

```
    [1.0, 4.0, 2.0]
], dtype=float)

b = np.array([-5.0, -1.0, 9.0], dtype=float)

sol = eliminacion_gaussiana_2dig(A, b)
print("Solución aproximada con aritmética de redondeo a 2 dígitos:", sol)
```

```
Solución aproximada con aritmética de redondeo a 2 dígitos: [-1.0, 1.0, 3.0]
```

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:

In [65]:
```
def eliminacion_gaussiana(a, b):

    n = len(b)
    matriz_aumentada = np.hstack((a, b.reshape(-1, 1)))
    intercambio_filas = False

    for i in range(n):
        # Buscar el pivote máximo en la columna actual
        pivote_maximo = i + np.argmax(np.abs(matriz_aumentada[i:, i]))

        # Si el pivote no está en la fila actual, intercambiar filas
        if pivote_maximo != i:
            matriz_aumentada[[i, pivote_maximo]] = matriz_aumentada[[pivote_maxi
            intercambio_filas = True

        # Verificar si el pivote es cero (sistema singular o dependiente)
        if np.isclose(matriz_aumentada[i, i], 0):
            return None, intercambio_filas

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

        # Eliminar las entradas debajo del pivote
        for j in range(i + 1, n):
            matriz_aumentada[j] -= matriz_aumentada[j, i] * matriz_aumentada[i]

    # Sustitución hacia atrás
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = matriz_aumentada[i, -1] - np.dot(matriz_aumentada[i, i + 1:n], x[

    return x, intercambio_filas
```

a)

$$
\begin{aligned}
x_1 - x_2 + 3x_3 &= 2, \\
3x_1 - 3x_2 + x_3 &= -1, \\
x_1 + x_2 &= 3
\end{aligned}
$$

In [66]:
```
A = np.array([
    [1, -1, 3],
    [3, -3, 1],
    [1, 1, 0]
```

```
], dtype=float)

b = np.array([2, -1, 3], dtype=float)

solucion, intercambios = eliminacion_gaussiana(A, b)

if solucion is not None:
    print("La solución del sistema es:", solucion)
    print("¿Se realizaron intercambios de fila?:", intercambios)
else:
    print("El sistema no tiene solución única o es inconsistente.")
```

```
La solución del sistema es: [1.1875 1.8125 0.875 ]
¿Se realizaron intercambios de fila?: True
```

b)

$$
\begin{aligned}
2x_1 - 1.5x_2 + 3x_3 &= 1, \\
-x_1 + 2x_3 &= 3, \\
4x_1 - 4.5x_2 + 5x_3 &= 1
\end{aligned}
$$

In [67]:
```
A = np.array([
    [2, -1.5, 3],
    [-1, 0, 2],
    [4, -4.5, 5]
], dtype=float)

b = np.array([1, 3, 1], dtype=float)

solucion, intercambios = eliminacion_gaussiana(A, b)

if solucion is not None:
    print("La solución del sistema es:", solucion)
    print("¿Se realizaron intercambios de fila?:", intercambios)
else:
    print("El sistema no tiene solución única o es inconsistente.")
```

```
La solución del sistema es: [-1.  0.  1.]
¿Se realizaron intercambios de fila?: True
```

c)

$$
\begin{aligned}
2x_1 &= 3, \\
x_1 + 1.5x_2 &= 4.5, \\
-3x_2 + 0.5x_3 &= -6.6, \\
2x_1 - 2x_2 + x_3 + x_4 &= 0.8
\end{aligned}
$$

In [68]:
```
A = np.array([
    [2, 0, 0, 0],
    [1, 1.5, 0, 0],
    [0, -3, 0.5, 0],
    [2, -2, 1, 1]
], dtype=float)

b = np.array([3, 4.5, -6.6, 0.8], dtype=float)

solucion, intercambios = eliminacion_gaussiana(A, b)
```

```
if solucion is not None:
    print("La solución del sistema es:", solucion)
    print("¿Se realizaron intercambios de fila?:", intercambios)
else:
    print("El sistema no tiene solución única o es inconsistente.")
```

```
La solución del sistema es: [ 1.5  2.  -1.2  3. ]
¿Se realizaron intercambios de fila?: True
```

d)

$$
\begin{aligned}
x_1 + x_2 + x_4 &= 2,\\
2x_1 + x_2 - x_3 + x_4 &= 1,\\
4x_1 - x_2 - 2x_3 + 2x_4 &= 0,\\
3x_1 - x_2 - x_3 + 2x_4 &= -3.
\end{aligned}
$$

In [69]:
```
A = np.array([
    [1, 1, 0, 1],
    [2, 1, -1, 1],
    [4, -1, -2, 2],
    [3, -1, -1, 2]
], dtype=float)

b = np.array([2, 1, 0, -3], dtype=float)

solucion, intercambios = eliminacion_gaussiana(A, b)

if solucion is not None:
    print("La solución del sistema es:", solucion)
    print("¿Se realizaron intercambios de fila?:", intercambios)
else:
    print("El sistema no tiene solución única o es inconsistente.")
```

```
El sistema no tiene solución única o 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.

In [70]:
```
def eliminacion_gausiana_32bits(A, b):
    # Convert arrays to 32-bit floats
    A = A.astype(np.float32)
    b = b.astype(np.float32)
    n = len(b)

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

    # Sustitución hacia atrás
    x = np.zeros(n, dtype=np.float32)
    for i in range(n-1, -1, -1):
        s = b[i] - np.dot(A[i, i+1:], x[i+1:])
```

```
            x[i] = s / A[i, i]
        return x
```

a)

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 = 9,$$
$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 = 8,$$
$$\frac{1}{2}x_1 + x_2 + 2x_3 = 8.$$

In [71]:
```
# Sistema de ecuaciones
A = np.array([
    [1/4, 1/5, 1/6],
    [1/3, 1/4, 1/5],
    [1/2, 1,   2  ]
    ], dtype=np.float32)
b = np.array([9, 8, 8], dtype=np.float32)

# Resolver y mostrar la solución
solucion = eliminacion_gausiana_32bits(A, b)
print("Solución con precisión 32-bit:", solucion)
```

```
Solución con precisión 32-bit: [-227.07666  476.92264 -177.69217]
```

b)

$$3.333x_1 + 15920x_2 - 10.333x_3 = 15913,$$
$$2.222x_1 + 16.71x_2 + 9.612x_3 = 28.544,$$
$$1.5611x_1 + 5.1791x_2 + 1.6852x_3 = 8.4254.$$

In [72]:
```
# Sistema de ecuaciones
A = np.array([
    [3.333, 15920, -10.333],
    [2.222, 16.71, 9.612],
    [1.5611, 5.1791, 1.6852  ]
    ], dtype=np.float32)
b = np.array([15913, 28.544, 8.4254], dtype=np.float32)

# Resolver y mostrar la solución
solucion = eliminacion_gausiana_32bits(A, b)
print("Solución con precisión 32-bit:", solucion)
```

```
Solución con precisión 32-bit: [0.99970937 1.0000001  1.0001061 ]
```

c)

$$x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 = \frac{1}{6},$$
$$\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 + \frac{1}{5}x_4 = \frac{1}{7},$$
$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 + \frac{1}{6}x_4 = \frac{1}{8},$$
$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 + \frac{1}{7}x_4 = \frac{1}{9}.$$

In [73]:
```
# Sistema de ecuaciones
A = np.array([
    [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]
    ], dtype=np.float32)
b = np.array([1/6, 1/7, 1/8, 1/9], dtype=np.float32)

# Resolver y mostrar la solución
solucion = eliminacion_gausiana_32bits(A, b)
print("Solución con precisión 32-bit:", solucion)
```

```
Solución con precisión 32-bit: [-0.03174768  0.5952596  -2.3810065   2.7778137 ]
```

d)

$$
\begin{aligned}
2x_1 + x_2 - x_3 + x_4 - 3x_5 &= 7,\\
x_1 + 2x_3 - x_4 + x_5 &= 2,\\
-2x_2 - x_3 + x_4 - x_5 &= -5,\\
3x_1 + x_2 - 4x_3 + 5x_5 &= 6,\\
x_1 - x_2 - x_3 - x_4 + x_5 &= -3.
\end{aligned}
$$

In [74]:
```
# Sistema de ecuaciones
A = np.array([
    [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]
    ], dtype=np.float32)
b = np.array([7, 2, -5, 6, -3], dtype=np.float32)

# Resolver y mostrar la solución
solucion = eliminacion_gausiana_32bits(A, b)
print("Solución con precisión 32-bit:", solucion)
```

```
Solución con precisión 32-bit: [1.8830409  2.8070176  0.7309942  1.4385967  0.093
56731]
 [1.8830409  2.8070176  0.7309942  1.4385967  0.09356731]
```

Ejercicio 5

Dado el sistema lineal:

$$
\begin{aligned}
x_1 - x_2 + \alpha x_3 &= -2,\\
-x_1 + 2x_2 - \alpha x_3 &= 3,\\
\alpha x_1 + x_2 + x_3 &= 2.
\end{aligned}
$$

$$
\begin{bmatrix}
1 & -1 & \alpha & -2\\
-1 & 2 & -\alpha & 3\\
\alpha & 1 & 1 & 2
\end{bmatrix}
$$

$F1 + F2 \to F2$

$-\alpha F1 + F3 \to F3$

$$\begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & \alpha+1 & -\alpha^2+1 & -2\alpha+2 \end{bmatrix}$$

$-(\alpha+1)F2+F3 \to F3$

$$\begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & -\alpha^2+1 & \alpha+1 \end{bmatrix}$$

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

cuando $-\alpha^2+1=0$

$$\begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & \alpha+1 \end{bmatrix}$$

si $\alpha = 1$

$$\begin{cases} x_1 - x_2 + x_3 = -2 \\ x_2 = 1 \\ 0 = 2 \end{cases}$$

Debido a que 0 es diferente a 2 el sistema de ecuaciones no tiene solucion cuando $\alpha = 1$

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

cuando $-\alpha^2+1=0$

$$\begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & \alpha+1 \end{bmatrix}$$

si $\alpha = -1$

$$\begin{cases} x_1 - x_2 - x_3 = -2 \\ x_2 = 1 \\ 0 = 0 \end{cases}$$

Cuando $\alpha = -1$ el sistema de ecuaciones tiene infinitas soluciones

$$x = \begin{bmatrix} -1-\alpha x_3 \\ 1 \\ x_3 \end{bmatrix}$$

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

Existe una unica solucion cuando cuando $-\alpha^2+1 \neq 0, 1y-1$

$$\begin{bmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & -\alpha^2+1 & \alpha+1 \end{bmatrix}$$

$$\begin{cases} x_1 - x_2 + \alpha x_3 = -2 \\ x_2 = 1 \\ (-\alpha^2+1)x_3 = \alpha + 1 \end{cases}$$

Donde la solucion general es:

$$x = \begin{bmatrix} \frac{1}{a-1} \\ 1 \\ \frac{-1}{a-1} \end{bmatrix}$$

Supongamos que x=2, obtendriamos la solucion:

$$x = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

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 $j = 1, \ldots, n$; b_i; representa el suministro diario disponible del i-ésimo alimento y a_{ij} representa la cantidad del i-ésimo alimento.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m. \end{aligned}$$

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

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

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

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?

In [75]:
```
# DATOS DEL PROBLEMA
A = np.array([
    [1, 2, 0, 3],
    [1, 0, 2, 2],
    [0, 0, 1, 1]
], dtype=float)

x = np.array([1000, 500, 350, 400], dtype=float)  # Poblaciones actuales
b = np.array([3500, 2700, 900], dtype=float)       # Suministros disponibles

# PARTE a) Verificar si el suministro b alcanza
def tiene_suministro_suficiente(A, x, b):
    consumo = A.dot(x)  # consumo de cada alimento
    return np.all(consumo <= b)

consumo_actual = A @ x
suficiente = tiene_suministro_suficiente(A, x, b)

print("a) Verificando consumo vs. suministro:")
print("   Consumo calculado:", consumo_actual)
print("   Suministro       :", b)
print("   ¿Hay suficiente alimento?", "Sí" if suficiente else "No")
print()

# PARTE b) Máximo aumento individual de cada especie
def max_incremento_individual(A, x, b):
    consumo_base = A @ x
    m, n = A.shape  # m alimentos, n especies
    maximos = []

    for j in range(n):
       # Buscar cuánto podemos aumentar la j-ésima especie
       # sin exceder el suministro en cada alimento i:
       posibles = []
       for i in range(m):
          if A[i,j] > 0:
             # Debemos mantener consumo_base[i] + A[i,j]*k <= b[i]
             restantes_i = b[i] - consumo_base[i]
             k_i = restantes_i / A[i,j]
             posibles.append(k_i)
          # si A[i,j] == 0, no impone restricción para este alimento
       if len(posibles) == 0:
          # Si A[i,j] = 0 para todos i, no hay consumo adicional => infinito
          # Para este ejemplo, no ocurre, pero lo manejamos:
          maximos.append(np.inf)
       else:
          maximos.append(np.floor(min(posibles)))  # o min(posibles) si se desea
    return maximos

print("b) Cálculo del número máximo de animales que se podría agregar a cada esp
inc_indiv = max_incremento_individual(A, x, b)
for j, k in enumerate(inc_indiv, start=1):
```

```
    print(f"   Especie {j}: se pueden agregar hasta {k} animales (aprox).")
print()

# PARTE c) Si la especie 1 se extingue, ¿qué incremento individual se soporta?
#        Para las otras especies (2, 3, 4).
x_extinta = x.copy()
x_extinta[0] = 0  # especie 1 extinta
consumo_extinto = A @ x_extinta

print("c) Extinguiendo la especie 1 => x_1=0.")
print("   Nuevas poblaciones:", x_extinta)
print("   Nuevo consumo:", consumo_extinto)
print("   Verificamos incrementos individuales para especies 2, 3 y 4:")

inc_extinto = max_incremento_individual(A, x_extinta, b)

for j in range(1, len(x)):  # mostrar sólo especies 2..4
    print(f"   Especie {j+1}: se pueden agregar hasta {inc_extinto[j]} animales
print()

# PARTE d) Si la especie 2 se extingue
x_extinta2 = x.copy()
x_extinta2[1] = 0
consumo_extinta2 = A @ x_extinta2

print("d) Extinguiendo la especie 2 => x_2=0.")
print("   Nuevas poblaciones:", x_extinta2)
print("   Nuevo consumo:", consumo_extinta2)
print("   Verificamos incrementos individuales para especies 1, 3 y 4:")

inc_extinto2 = max_incremento_individual(A, x_extinta2, b)
for j in [0, 2, 3]:
    print(f"   Especie {j+1}: se pueden agregar hasta {inc_extinto2[j]} animales
print()
```

```
a) Verificando consumo vs. suministro:
   Consumo calculado: [3200. 2500.  750.]
   Suministro       : [3500. 2700.  900.]
   ¿Hay suficiente alimento? Sí

b) Cálculo del número máximo de animales que se podría agregar a cada especie (in
dividualmente):
   Especie 1: se pueden agregar hasta 200.0 animales (aprox).
   Especie 2: se pueden agregar hasta 150.0 animales (aprox).
   Especie 3: se pueden agregar hasta 100.0 animales (aprox).
   Especie 4: se pueden agregar hasta 100.0 animales (aprox).

c) Extinguiendo la especie 1 => x_1=0.
   Nuevas poblaciones: [  0. 500. 350. 400.]
   Nuevo consumo: [2200. 1500.  750.]
   Verificamos incrementos individuales para especies 2, 3 y 4:
   Especie 2: se pueden agregar hasta 650.0 animales (aprox).
   Especie 3: se pueden agregar hasta 150.0 animales (aprox).
   Especie 4: se pueden agregar hasta 150.0 animales (aprox).

d) Extinguiendo la especie 2 => x_2=0.
   Nuevas poblaciones: [1000.    0.  350.  400.]
   Nuevo consumo: [2200. 2500.  750.]
   Verificamos incrementos individuales para especies 1, 3 y 4:
   Especie 1: se pueden agregar hasta 200.0 animales (aprox).
   Especie 3: se pueden agregar hasta 100.0 animales (aprox).
   Especie 4: se pueden agregar hasta 100.0 animales (aprox).
```

EJERCICIOS TEÓRICOS

Ejercicio 7

Repita el ejercicio 4 con el método Gauss-Jordan:

Use el método Gauss-Jordan y la aritmética computacional de precisión de 32 bits para resolver los siguientes sistemas lineales.

In [76]:
```python
def gauss_jordan_32bits(A, b):

    A = np.array(A, dtype=np.float32)
    b = np.array(b, dtype=np.float32)

    matriz_aumentada = np.hstack([A, b.reshape(-1, 1)])

    # Aplicar eliminación Gauss-Jordan
    n = matriz_aumentada.shape[0]
    for i in range(n):
        # Hacer el pivote igual a 1
        pivot = matriz_aumentada[i, i]
        matriz_aumentada[i] = matriz_aumentada[i] / pivot

        # Hacer ceros en la columna del pivote
        for j in range(n):
            if i != j:
                factor = matriz_aumentada[j, i]
                matriz_aumentada[j] = matriz_aumentada[j] - factor * matriz_aume
```

```
    solucion = matriz_aumentada[:, -1]

    return solucion
```

a)

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 = 9,$$
$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 = 8,$$
$$\frac{1}{2}x_1 + x_2 + 2x_3 = 8.$$

In [77]:
```
A = [[1/4, 1/5, 1/6],
     [1/3, 1/4, 1/5],
     [1/2, 1, 2]]
b = [9, 8, 8]

solucion = gauss_jordan_32bits(A, b)
print("La solución del sistema es:", solucion)
```

```
La solución del sistema es: [-227.07668  476.9226  -177.69215]
```

b)

$$3.333x_1 + 15920x_2 - 10.333x_3 = 15913,$$
$$2.222x_1 + 16.71x_2 + 9.612x_3 = 28.544,$$
$$1.5611x_1 + 5.1791x_2 + 1.6852x_3 = 8.4254.$$

In [78]:
```
A = [[3.333, 15920, -10.333],
     [2.222, 16.71, 9.612],
     [1.5611, 5.1791, 1.6852]]
b = [15913, 28.544, 8.4254]

solucion = gauss_jordan_32bits(A, b)
print("La solución del sistema es:", solucion)
```

```
La solución del sistema es: [0.9998865 1.0000001 1.0001063]
```

c)

$$x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 = \frac{1}{6},$$
$$\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 + \frac{1}{5}x_4 = \frac{1}{7},$$
$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 + \frac{1}{6}x_4 = \frac{1}{8},$$
$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 + \frac{1}{7}x_4 = \frac{1}{9}.$$

In [79]:
```
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]
```

```
solucion = gauss_jordan_32bits(A, b)
print("La solución del sistema es:", solucion)
```

```
La solución del sistema es: [-0.03174722  0.5952536  -2.380991    2.7778032 ]
```

d)

$$
\begin{aligned}
2x_1 + x_2 - x_3 + x_4 - 3x_5 &= 7,\\
x_1 + 2x_3 - x_4 + x_5 &= 2,\\
-2x_2 - x_3 + x_4 - x_5 &= -5,\\
3x_1 + x_2 - 4x_3 + 5x_5 &= 6,\\
x_1 - x_2 - x_3 - x_4 + x_5 &= -3.
\end{aligned}
$$

In [80]:
```
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]

solucion = gauss_jordan_32bits(A, b)
print("La solución del sistema es:", solucion)
```

```
La solución del sistema es: [1.8830409  2.8070173  0.73099416 1.4385965  0.093567
24]
```