UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMÁTICA

Materia: Investigación Operativa I

Estudiantes:	Univ. Aparicio Quispe Jorge Mauricio	**INF-155**	6794525
	Univ. Espinoza Medina Sharam	**INF-155**	13344621
	Univ. Guachalla Choque Brian	**INF-155**	9089128
	Univ. Medrano Callisaya Ivar Pedro	**DAT-244**	11067652
	Univ. Tolay Candia Marioly Giselle	**SIS-243**	8329706

Docente: Lic. Mario Amilcar Miranda Gonzales

Fecha: 02/12/24

La Paz - Bolivia

1. INTRODUCCIÓN

El éxito académico depende de una adecuada distribución del tiempo entre diferentes materias, especialmente cuando los recursos y el tiempo son limitados. Este trabajo se centra en la optimización del tiempo de estudio de un estudiante universitario, quien debe alcanzar puntajes mínimos en dos o más materias para aprobar, así también debe tener un equilibrio en su tiempo de estudio en relación de su descanso diario. A través del uso de modelos matemáticos y técnicas de optimización, se busca maximizar en un ejemplo las horas adecuadas para estudiar y descansar todos los días, y en el otro ejemplo maximizar el puntaje total cumpliendo con restricciones como el tiempo disponible y los requerimientos mínimos de cada materia.

En este contexto, se implementarán tres métodos fundamentales en la investigación operativa: el **Método Simplex**, el **Método de Penalizaciones** y el **Método Gráfico**. Estos métodos serán aplicados a los dos casos planteados y programados en Python para garantizar una solución eficiente, escalable y fácilmente comprensible. Además, la implementación en Python permitirá comparar las soluciones obtenidas con cada técnica, evidenciando su aplicabilidad en problemas reales.

2. JUSTIFICACIÓN

En el ámbito académico, la planificación eficiente del tiempo es esencial para lograr los objetivos deseados. Los estudiantes a menudo enfrentan restricciones de tiempo y deben priorizar sus esfuerzos en función de las metas establecidas. Este trabajo justifica el uso de herramientas de optimización porque permiten identificar las mejores estrategias para distribuir el tiempo de estudio de manera racional y efectiva, asegurando el cumplimiento de objetivos académicos.

Además, la implementación de estos métodos en Python aporta un enfoque práctico y adaptable que facilita el entendimiento y la aplicación de técnicas de optimización. Python, como lenguaje versátil y ampliamente utilizado en investigación operativa, ofrece herramientas potentes para resolver problemas complejos. La inclusión del **Método Simplex**, el **Método de Penalizaciones** y el **Método Gráfico** no solo ilustra la teoría subyacente, sino que también resalta cómo estas técnicas pueden aplicarse a problemas cotidianos, optimizando recursos y maximizando resultados.

3. DESARROLLO

MÉTODO SIMPLEX

Fue dado por George-Dantzig en el año 1957, surge como consecuencia de que el método gráfico es poco útil.

Este método permite resolver el modelo de programación lineal de forma matemática y aritmética.

Condición de Optimidad: Con esta condición se garantiza, con cada proceso iterativo debe mejorar el valor de la función objetivo, en el caso de maximización incrementarse, y en el caso de minimización reducirse.

Condición de factibilidad: Con esta condición se garantiza, con cada proceso de iteración se obtenga una nueva solución factible, es decir que no viole restricciones.

Método Simplex Básico

Permite resolver el modelo de programación lineal canónicos por naturaleza, en el cual se tiene toda la base canónica completa, por lo cual no es necesario utilizar o hacer variables artificiales.

Solución básica inicial. Para el método simplex básico, la solución básica inicial siempre estará formada por puras variables de holgura *s*, las cuales forman bases canónicas completas.

Pasos para el método Simplex Básico

1. Colocar el modelo en su forma estándar, adicionado a cada restricción de la forma menor o igual una variable de holgura, con su subíndice correspondiente al número de restricción, adicionando todas las variables de holguras utilizadas en la función objetiva con coeficiente 0.

2. Introducir el modelo estándar a una tabla tipo simplex, la función objetivo con signo cambiado, y las restricciones con sus propios signos o tal como quedaron definidos.

3. (**Condición de optimidad**) o variable de entrada es aquella variable no básica con coeficientes más negativos en la función objetivo en el caso de maximización y con coeficientes más positivos en la función objetivo en el caso de minimización, en caso de empates considerar importancia en variables de entrada.

4. (**Condición de factibilidad**) la variable de salida es una variable básica que se elige de la siguiente manera:

Realizar los cocientes entre los coeficientes del vector *b* de recursos o disponibilidades, dividido el vector de coeficientes tecnológicos correspondiente a la variable de entrada, no existiendo división entre 0 o negativos, la variable de salida es aquella correspondiente al menor cociente en caso de empates considerar importancia entre las variables de salida.

5. Una vez conocidas las variables de entrada y salida, la casilla de intersección de columna y fila se pivotiza, aplicando el inverso multiplicativo, y se logra un vector canónico columna, aplicando inversos aditivos, de esta forma se logra una nueva solución posible hacer factible y óptimo.

6. De no tenerse una solución óptima y factible repetir los pasos **3, 4** y **5** hasta conseguir una solución óptima y factible.

El proceso iterativo termina cuando se tengan puros coeficientes positivos o nulas en la función objetivo el caso de maximización, y puros negativos o nulos en el caso de minimización.

Método Simplex avanzado: Permite resolver modelos de programación lineal con cualquier tipo de restricción, haciendo uso de variables artificiales *R*, por no tenerse la base canónica completa requerida, para lo cual se presentan dos técnicas de solución la técnica en M y la técnica de dos fases.

Técnica en M

Llamado también método de penalizaciones, por ser *M* una penalización es decir un número inmensamente grande y positivo.

Pasos para la técnica en M

1. Colocar el modelo en su forma estándar de la siguiente manera.

   A cada restricción de la forma menor o igual sumar una variable de holgura, a cada restricción de la forma y para el restar una variable superflua *s* y sumar una variable artificial *R* simultáneamente y a cada restricción de tipo ecuación sumar una variable artificial única, todas las variables, holguras o superfluas *s* se adicionan en la función objetivo con coeficiente 0 y todas las variables artificiales *R* tienen coeficientes de en la función objetivo en el caso de maximización, y tiene coeficiente de las variables artificiales *R* en el caso de minimización.

2. Introducir a una tabla tipo Simplex, el modelo estándar normalmente (la función objetivo con signo cambiado), y las restricciones con su propio signo o tal como quedaron definidos.

3. (**Condición de optimidad**) reponer la base canónica en forma inmediata mediante operaciones elementales, la variable de entrada es aquella variable no básica con coeficiente más negativo en la función objetivo para el caso de maximización, y con coeficiente más positivo en la función objetivo para el caso de minimización, en caso de empates considerar importante los casos de entrada.

4. (**Condición de factibilidad**) la variable de salida se elige de la siguiente manera:

- Realizar los cocientes entre los coeficientes entre los coeficientes del vector *b* de recursos o disponibilidades, dividiendo el vector de coeficientes tecnológicos correspondientes a la variable de entrada, no existiendo la división entre 0 y números negativos, la variable de salida es aquella correspondiente a la de menor cociente, en caso de empates considerar importancia entre las variables de salida.

5. Una vez conocida las variables de entrada y salida, la casilla de intersección de columna y fila se pivotiza aplicando el inverso multiplicativo, y se logra un vector canónico columna, aplicando inversos aditivos de esta forma se logra una nueva solución base, haciendo posible hacer óptima y factible.

6. Consiste en repetir los pasos **3, 4** y **5** del simplex básico normalmente considerando que *M* es una penalización es decir un número inmensamente grande.

El proceso iterativo termina cuando se tenga puro coeficientes positivos o nulas en la función objetivo en el caso de maximización, y puros coeficientes negativos o nulos en la función objetivo en el caso de minimización.

La técnica en M reconoce que un problema no tiene solución, siempre que en la tabla óptima aparezca alguna variable artificial con valor distinto de 0.

Técnica de dos Fases

En forma similar a la técnica en M, resuelve problemas de programación lineal con cualquier tipo de restricción, haciendo uso de variables artificiales *R* por no existir una base canónica completa.

Para aplicar la técnica en dos fases se debe aplicar las siguientes fases.

1era Fase. Crearse un nuevo modelo de programación lineal, asociado al original, cuya función objetivo es de tipo minimización de de suma de variables artificiales *R* a utilizar, sujeta a las restricciones de la técnica en *M*, este modelo se introduce a una

tabla de tipo Simplex normalmente reponer la base canónica en forma inmediata mediante operaciones elementales y aplicar los siguientes pasos del Simplex Básico para el caso de minimización normalmente.

La solución óptima en esta primera fase debe ser , para que el modelo original tenga solución y pueda pasar a la 2da fase.

2da Fase. De la tabla óptima obtenida en la primera fase eliminar todas las columnas correspondientes a las variables artificiales *R* y reemplazar la función objetivo del modelo original, introduciendo con signo cambiado, reponer la base canónica de forma inmediata mediante operaciones elementales y aplicar los siguientes pasos del método Simplex Básicos normalmente si el modelo original es de tipo maximización, en esta fase se maximiza, si el modelo original es de tipo minimizar, entonces la fase se minimiza.

La solución óptima obtenida en esta 2da Fase será la solución óptima del modelo original.

Importancia de variables de Entrada.

Será prioridad variables de decisión , en segunda instancia variables de holgura o superfluas *s* y en última instancia variables artificiales *R*, en caso de empates entre variables con el mismo tipo de equivalencia romper arbitrariamente.

Importancia entre variables de Salida.

Se da prioridad a variables artificiales *R*, en segunda instancia las variables *s* holguras o superfluas, y en última instancia las variables de decisión , empates entre variables del mismo tipo de equivalencia elegir arbitrariamente.

EJEMPLO 1

Contexto:

Un estudiante universitario está planificando su día para maximizar su productividad. Quiere distribuir su tiempo entre dos actividades: **estudiar** y **descansar**, asegurándose de cumplir con ciertas restricciones de tiempo para cada actividad, así como un tiempo total limitado, el estudiante solo tiene 10 horas para estas actividades ya que las otras 14 horas las usa para trabajar y pasar clases en la universidad.

Datos del problema

- **Restricciones:**
  - El tiempo total entre estudiar y descansar no debe superar **10 horas**.
  - El tiempo dedicado a estudiar no puede superar **6 horas**.
  - El tiempo dedicado a descansar no puede superar las **8 horas**.
- **Puntajes por hora:**
  - Cada hora dedicada a estudiar aporta **5 puntos**.
  - Cada hora dedicada a descansar aporta **8 puntos**.

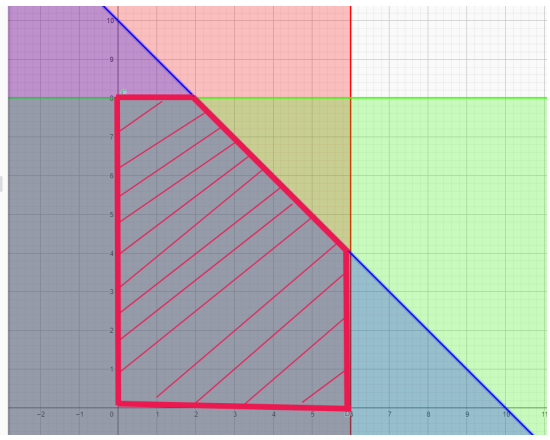
Objetivo:
Maximizar la productividad total.

Planteamiento del problema

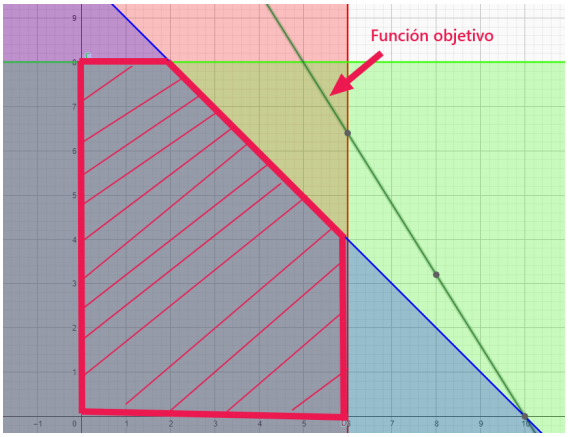
- **Variables de decisión:**
  - x_1: Horas dedicadas a estudiar ($x_1 \geq 0$).
  - x_2: Horas dedicadas a descansar ($x_2 \geq 0$).
- **Función objetivo (maximizar la productividad):**
  $Z = 5x_1 + 8x_2$
- **Restricciones:**
  - $x_1 \leq 6$ (Tiempo dedicado a estudiar).
  - $x_2 \leq 8$ (Tiempo dedicado a descansar).
  - $x_1 + x_2 \leq 10$ (Tiempo total).
  - $x_1, x_2 \geq 0$.

Resolución del problema mediante los métodos seleccionados:

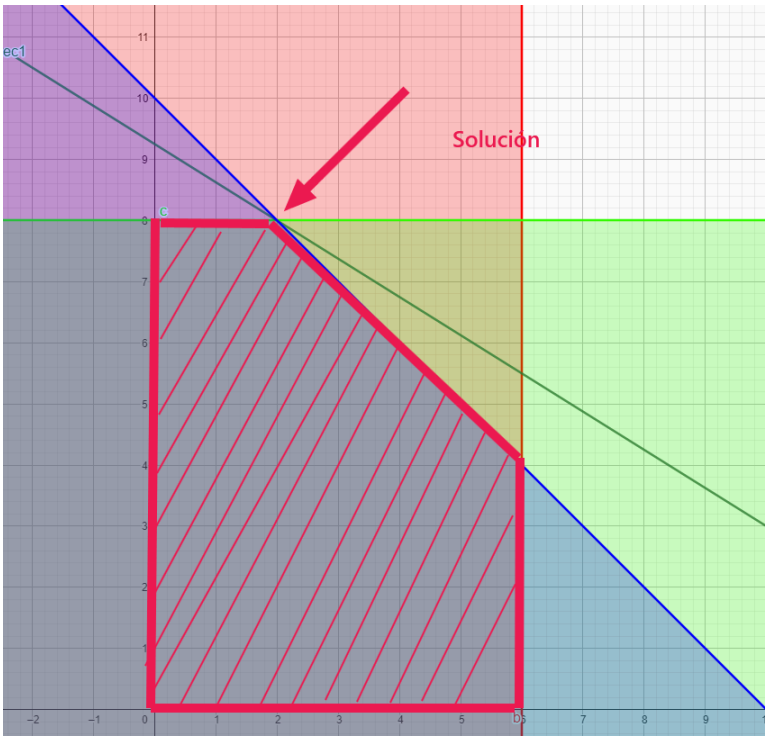
- **Método Gráfico**:
  - Restricciones



- Función objetivo



- Solución



Resolver el sistema:

$$x_1 + x_2 = 10$$

$$x_2 = 8$$

$$x_1 + 8 = 10$$

$$x_1 = 2$$

Reemplazando:

$$Z = 5x_1 + 8x_2 = 5(2) + 8(8) = 10 + 64 = 74$$

Respuesta: El número de horas que el estudiante debe estudiar en un día es de 2 horas, y el número de horas de descanso que debe tener es de 8 horas, alcanzando una puntuación de productividad total de 74.

- **Método Gráfico en Python:**

Código:

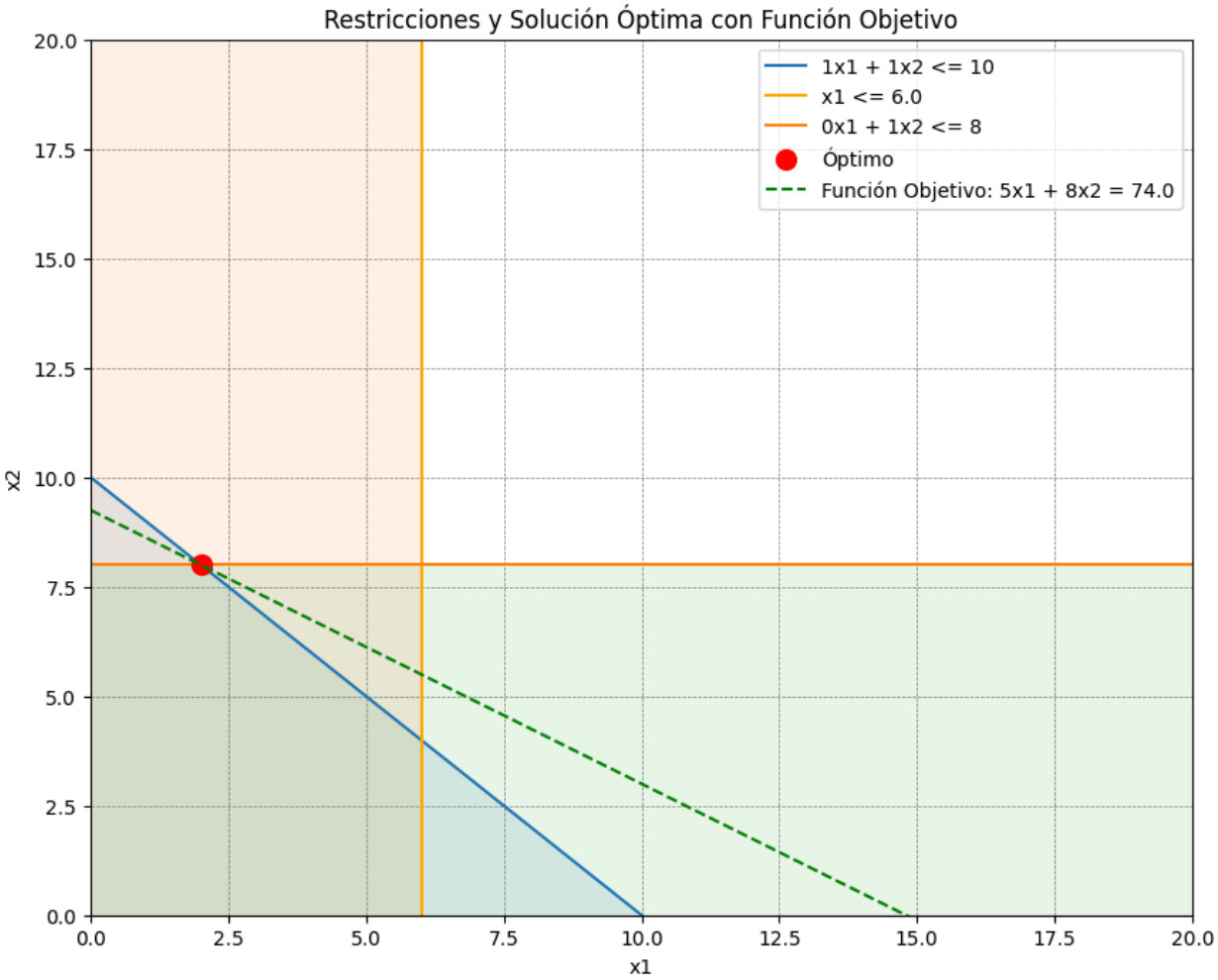
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog
def resolver_y_graficar_con_funcion_objetivo(A, b, c, restricciones, x1_bounds=(0,
None), x2_bounds=(0, None)):
    """
    Resuelve y grafica problemas de programación lineal con restricciones de tipo <=,
>=, =,
    y grafica la función objetivo basada en los valores óptimos encontrados.
    :param A: Matriz de coeficientes de las restricciones.
    :param b: Vector de constantes del lado derecho de las restricciones.
    :param c: Vector de coeficientes de la función objetivo.
    :param restricciones: Lista de tipos de restricciones ('<=', '>=', '=').
    :param x1_bounds: Límite inferior y superior para x1.
    :param x2_bounds: Límite inferior y superior para x2.
    """
    x1 = np.linspace(0, 100, 1000)  # Rango para graficar x1
    x2 = np.linspace(0, 100, 1000)  # Rango para graficar x2
    plt.figure(figsize=(10, 8))
    # Variables auxiliares para restricciones
    A_ub, b_ub, A_eq, b_eq = [], [], [], []
    # Procesar las restricciones
    for i in range(len(restricciones)):
        if restricciones[i] == '<=':
            A_ub.append(A[i])
            b_ub.append(b[i])
            if A[i][1] != 0:  # Restricción no horizontal
                y = (b[i] - A[i][0] * x1) / A[i][1]
                plt.plot(x1, y, label=f'{A[i][0]}x1 + {A[i][1]}x2 <= {b[i]}',
linestyle='-')
                plt.fill_between(x1, 0, y, where=(y >= 0), alpha=0.1)
```

```
            else:  # Restricción vertical (x1 = constante)
                x_vert = np.full_like(x2, b[i] / A[i][0])
                plt.plot(x_vert, x2, color='orange', linestyle='-', label=f'x1 <=
{b[i] / A[i][0]}')
                plt.fill_betweenx(x2, 0, x_vert, where=(x_vert >= 0), alpha=0.1)
        elif restricciones[i] == '>=':
            A_ub.append(-np.array(A[i]))
            b_ub.append(-b[i])
            if A[i][1] != 0:  # Restricción no horizontal
                y = (b[i] - A[i][0] * x1) / A[i][1]
                plt.plot(x1, y, label=f'{A[i][0]}x1 + {A[i][1]}x2 >= {b[i]}',
linestyle='-')
                plt.fill_between(x1, y, 100, where=(y >= 0), alpha=0.1)
            else:  # Restricción vertical (x1 = constante)
                x_vert = np.full_like(x2, b[i] / A[i][0])
                plt.plot(x_vert, x2, color='orange', linestyle='-', label=f'x1 >=
{b[i] / A[i][0]}')
                plt.fill_betweenx(x2, x_vert, 100, where=(x_vert >= 0), alpha=0.1)
        elif restricciones[i] == '=':
            A_eq.append(A[i])
            b_eq.append(b[i])
            if A[i][1] != 0:  # Restricción no horizontal
                y = (b[i] - A[i][0] * x1) / A[i][1]
                plt.plot(x1, y, '--', label=f'{A[i][0]}x1 + {A[i][1]}x2 = {b[i]}')
            else:  # Restricción vertical (x1 = constante)
                x_vert = np.full_like(x2, b[i] / A[i][0])
                plt.plot(x_vert, x2, '--', label=f'x1 = {b[i] / A[i][0]}')
    # Convertir las restricciones en arrays
    A_ub = np.array(A_ub) if A_ub else None
    b_ub = np.array(b_ub) if b_ub else None
    A_eq = np.array(A_eq) if A_eq else None
    b_eq = np.array(b_eq) if b_eq else None
    # Resolver el problema
    res = linprog(-c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=[x1_bounds,
x2_bounds], method='highs')
    # Mostrar solución óptima
    if res.success:
        vertice_optimo = res.x
        valor_optimo = -res.fun
        print(f'Solución óptima: x1 = {vertice_optimo[0]:.2f}, x2 =
{vertice_optimo[1]:.2f}')
        print(f'Valor máximo de la función objetivo: Z = {valor_optimo:.2f}')
        # Agregar el vértice óptimo al gráfico
        plt.plot(vertice_optimo[0], vertice_optimo[1], 'ro', markersize=10,
label='Óptimo')
        # Graficar la función objetivo pasando por el punto óptimo
        if c[1] != 0:  # Función objetivo no vertical
```

```
            z_opt = valor_optimo
            y_obj = (z_opt - c[0] * x1) / c[1]
            plt.plot(x1, y_obj, '--', color='green', label=f'Función Objetivo:
{c[0]}x1 + {c[1]}x2 = {z_opt}')
        else:  # Función objetivo vertical
            x_obj = z_opt / c[0]
            plt.axvline(x=x_obj, linestyle='--', color='green', label=f'Función
Objetivo: x1 = {x_obj}')
    else:
        print("No se encontró una solución factible.")
    # Configuración del gráfico
    plt.xlim(0, 20)
    plt.ylim(0, 20)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Restricciones y Solución Óptima con Función Objetivo')
    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.legend()
    plt.show()
# Ejemplo de uso
c = np.array([5, 8]) # Vector c de coeficientes de la funcion objetivo
# Matriz A de coeficientes de las restricciones
A = np.array([[1, 1],
              [1, 0],
              [0, 1]])
restricciones = ['<=', '<=', '<='] # tipo de restricciones
#vector b de las restricciones
b = np.array([10,
              6,
              8])
resolver_y_graficar_con_funcion_objetivo(A, b, c, restricciones)
```

Resultado y gráfica:

```
Solución óptima: x1 = 2.00, x2 = 8.00
Valor máximo de la función objetivo: Z = 74.00
```



- **Método Simplex Básico:**

$$Max\ Z\ = 5x_1 + 8x_2$$

S.a.

- $x_1 \leq 6$
- $x_2 \leq 8$
- $x_1 + x_2 \leq 10$

$$x \geq 0$$

Volver a su forma estándar.

$$Max\ Z\ = 5x_1 + 8x_2 + 0s_1 + 0s_2 + 0s_3$$

S.A.

$x_1 + s_1 = 6$

$x_2 + s_2 = 8$

$x_1 + x_2 + s_3 = 10$

$x \geq 0; s \geq 0$

	VB	z	x_1	x_2 ↓	s_1	s_2	s_3	Solución		
	z	1	-5	-8	0	0	0	0		
	s_1	0	1	0	1	0	0	6	6/0=ind	
(1/1)	s_2	0	0	1	0	1	0	8 →	8/1=8	
	s_3	0	1	1	0	0	1	10	10/1=10	
	z	1	-5 ↓	0	0	8	0	64		R1=R1+8R3
	s_1	0	1	0	1	0	0	6	6/1=6	
	x_2	0	0	1	0	1	0	8	8/0=ind	
(1/1)	s_3	0	1	0	0	-1	1	2 →	2/1=2	R4=R4-R3
	z	1	0	0	0	3	5	74		R1=R1+5R4
	s_1	0	0	0	1	1	-1	4		R2=R2-R4
	x_2	0	0	1	0	1	0	8		
	x_1	0	1	0	0	-1	1	2		

$x_1 = 2, x_2 = 8$

$s_1 = 4, s_2 = 0, s_3 = 0$

$Max\, Z = 5x_1 + 8x_2$

$Max\, Z = (5 * 2) + (8 * 8)$

$Max\, Z = 74$

- **Método Simplex Básico en Python:**

Código:

```
import numpy as np
def simplex_basico(c, A, b, tipo="max"):
    """
    Método Simplex Básico para maximizar o minimizar problemas de programación lineal.
```

```
    :param c: Vector de coeficientes de la función objetivo.
    :param A: Matriz de coeficientes de las restricciones.
    :param b: Vector de constantes del lado derecho de las restricciones.
    :param tipo: Indica si se va a maximizar ("max") o minimizar ("min").
    :return: Solución óptima, valor óptimo de la función objetivo y detalles de cada
variable.
    """
    # Convertir a maximización si es necesario
    if tipo == "max":
        c = -c  # Invertir signos para maximización
    # Crear la tabla inicial
    num_variables = len(c)
    num_restricciones = len(b)
    # Agregar variables de holgura
    I = np.eye(num_restricciones)  # Matriz identidad para las variables de holgura
    A = np.hstack([A, I])  # Agregar las variables de holgura a la matriz de
restricciones
    c = np.hstack([c, np.zeros(num_restricciones)])  # Extender la función objetivo
con coeficientes 0 para las holguras
    # Inicializar la tabla simplex
    tabla = np.hstack([A, b.reshape(-1, 1)])  # Combinar A y b en una tabla
    c_z = np.hstack([c, 0])  # Extender c con el término constante 0
    # Inicializar variables básicas
    variables_basicas = list(range(num_variables, num_variables + num_restricciones))
# Variables de holgura
    # Iterar hasta alcanzar una solución óptima
    while True:
        # Calcular los costos relativos (c_j - z_j)
        z_j = np.dot(c_z[variables_basicas], tabla[:, :-1])
        costos_relativos = c_z[:-1] - z_j
        # Verificar si la solución es óptima
        if all(costos_relativos >= 0):
            break
        # Determinar la variable entrante (más negativo en costos relativos)
        columna_entrante = np.argmin(costos_relativos)
        # Verificar si el problema es no acotado
        if all(tabla[:, columna_entrante] <= 0):
            raise ValueError("El problema es no acotado.")
        # Determinar la variable saliente (prueba de razón mínima)
        razones = np.divide(tabla[:, -1], tabla[:, columna_entrante],
out=np.full_like(tabla[:, -1], np.inf), where=(tabla[:, columna_entrante] > 0))
        fila_saliente = np.argmin(razones)
        # Actualizar la tabla simplex (realizar pivoteo)
        tabla[fila_saliente, :] /= tabla[fila_saliente, columna_entrante]
        for i in range(len(tabla)):
            if i != fila_saliente:
```

```
                tabla[i, :] -= tabla[i, columna_entrante] * tabla[fila_saliente, :]
        # Actualizar variables básicas
        variables_basicas[fila_saliente] = columna_entrante
    # Obtener la solución óptima
    solucion = np.zeros(len(c_z) - 1)
    for i, var in enumerate(variables_basicas):
        if var < len(solucion):
            solucion[var] = tabla[i, -1]
    # Valor óptimo de la función objetivo
    valor_optimo = np.dot(c, solucion)
    # Ajustar el signo del valor óptimo si fue maximización
    if tipo == "max":
        valor_optimo = -valor_optimo
    # Crear detalles de interpretación
    detalles = {}
    for i in range(len(solucion)):
        if i < num_variables - num_restricciones:
            detalles[f'x{i+1}'] = solucion[i]
        else:
            detalles[f's{i-num_variables+1}'] = solucion[i]
    return solucion, valor_optimo, detalles
# Ejemplo de uso
c = np.array([5, 8])  # Coeficientes de la función objetivo
A = np.array([[1, 1],
              [1, 0],
              [0, 1]])  # Restricciones
b = np.array([10, 6, 8])  # Lado derecho de las restricciones
# Maximización
solucion_max, valor_max, detalles_max= simplex_basico(c, A, b, tipo="max")
print("Maximización: Solución =", solucion_max)
print("Valor óptimo =", valor_max)
print("Detalles de cada variable:")
for var, valor in detalles_max.items():
    print(f"{var}: {valor}")
# Minimización
#solucion_min, valor_min,detalles_max = simplex_basico(c, A, b, tipo="min")
#print("Minimización: Solución =", solucion_min)
#print("Valor óptimo =", valor_min)
#print("Detalles de cada variable:")
#for var, valor in detalles_max.items():
#    print(f"{var}: {valor}")
```

Resultados:

```
Maximización: Solución = [2. 8. 0. 4. 0.]
Valor óptimo = 74.0
Detalles de cada variable:
```

```
s-1: 2.0
s0: 8.0
s1: 0.0
s2: 4.0
s3: 0.0
```

EJEMPLO 2:

Contexto:
Un estudiante universitario tiene dos materias principales: **Matemáticas** y **Física**, y debe distribuir su tiempo de estudio semanal para maximizar su puntaje total, asegurándose de aprobar ambas materias. Cada materia tiene diferentes requisitos mínimos para aprobar, y el estudiante cuenta con restricciones de tiempo y recursos. Además, la complejidad de las materias afecta cuánto tiempo consume cada hora de estudio.

Datos del Problema:

- **Requisitos mínimos para aprobar:**
  - Matemáticas: Se necesitan al menos **51 puntos**.
  - Física: Se necesitan al menos **51 puntos**.
- **Puntaje por hora de estudio:**
  - Cada hora de Matemáticas aporta **7 puntos**.
  - Cada hora de Física aporta **4 puntos**.
- **Restricción de tiempo total:**
  El estudiante tiene **60 unidades de tiempo disponibles** semanalmente, donde:
  - Cada hora de Matemáticas consume **3 unidades de tiempo**.
  - Cada hora de Física consume **2 unidades de tiempo**.
- **Restricción de tiempo por materia:**
  - Matemáticas: Mínimo **8 horas** y máximo **12 horas**.
  - Física: Mínimo **13 horas**.

Objetivo:
Determinar cuántas horas debe dedicar a cada materia para maximizar su puntaje total y garantizar que apruebe ambas materias.

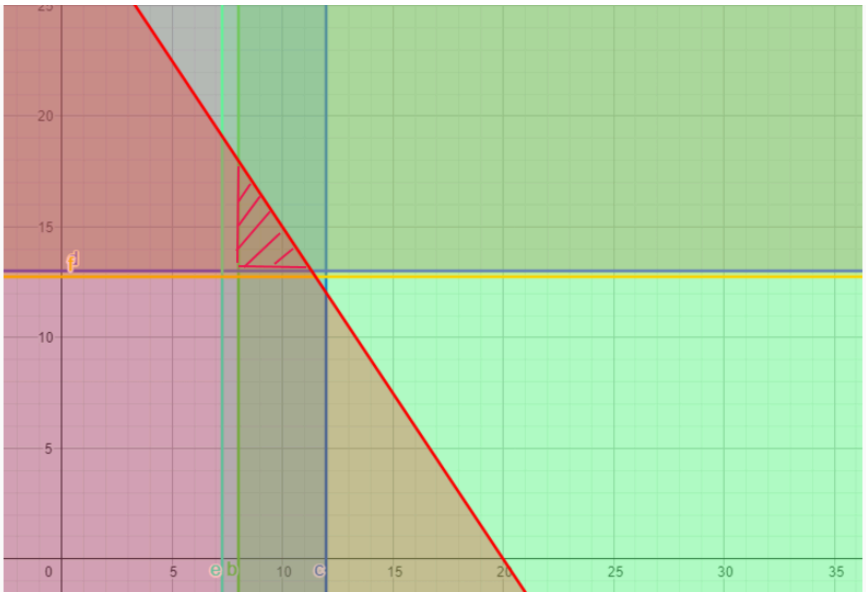
Planteamiento del problema:

- Variables de decisión:
  - x_1: Horas de estudio en Matemáticas.
  - x_2: Horas de estudio en Física.
- Función objetivo: Maximizar el puntaje total, $Z = 7x_1 + 4x_2$.

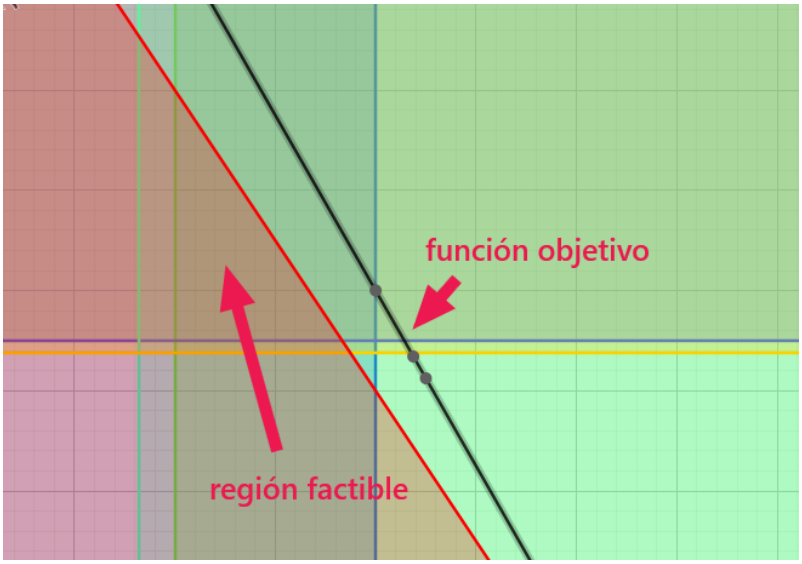
- Restricciones:
  - $3x_1 + 2x_2 \leq 60$ (tiempo disponible).
  - $x_1 \geq 8,\ x_1 \leq 12$ (horas para Matemáticas).
  - $x_2 \geq 13$ (horas para Física).
  - $7x_1 \geq 51$ (puntaje mínimo para Matemáticas).
  - $4x_2 \geq 51$ (puntaje mínimo para Física).
  - $x_1, x_2 \geq 0$.

Resolución del problema mediante los métodos seleccionados:

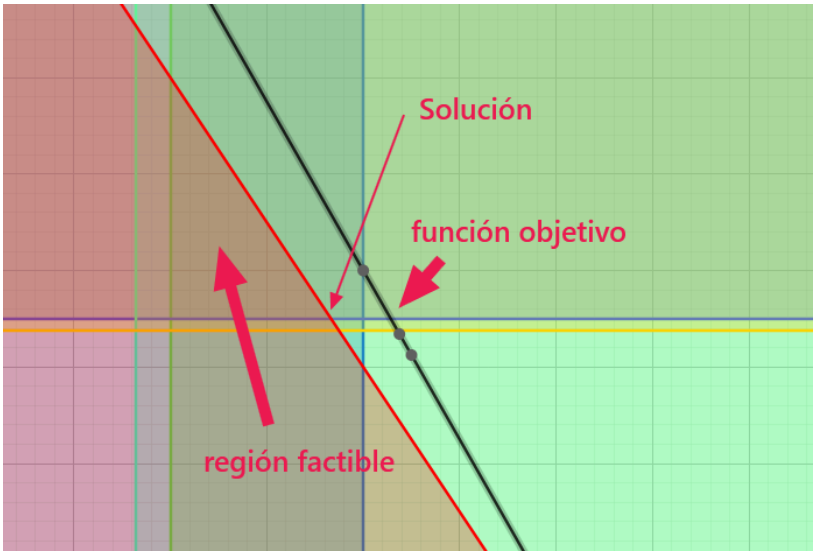
- **Método Gráfico**:
  - Restricciones



  - Función objetivo



- **Solución**



$$3x_1 + 2x_2 = 60$$

$$x_2 = 13$$

$$3x_1 + 2(13) = 60$$

$$3x_1 = 34$$

$$x_1 = 34/3$$

Reemplazando:

$$Z = 7(34/3) + 4(13) = 238/3 + 52 = 394/3 = 131.3333333$$

Respuesta: El número de horas que debe dedicar el estudiante a Matemáticas es de 11 horas y 20 minutos, y el número de horas a Física es de 13 horas, alcanzando 79 puntos en Matemáticas y 52 en Física, haciendo un total de 131 puntos en las dos materias.

- **Método gráfico en Python:**

Código:

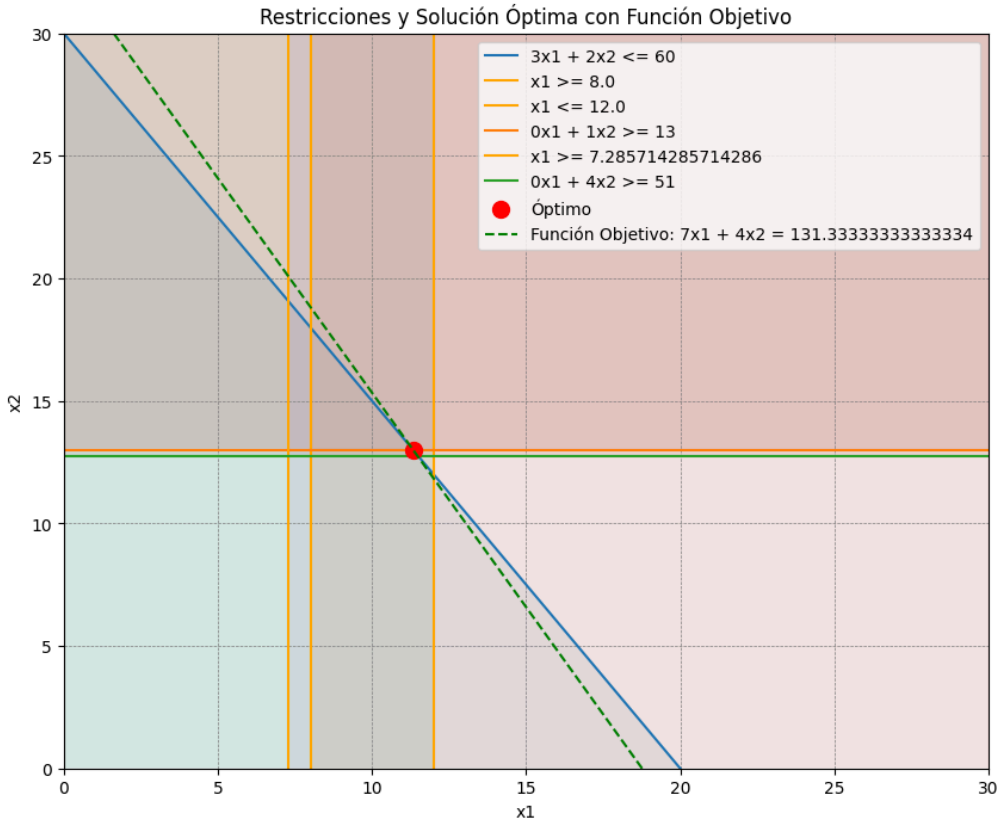
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog
def resolver_y_graficar_con_funcion_objetivo(A, b, c, restricciones, x1_bounds=(0,
None), x2_bounds=(0, None)):
    """
    Resuelve y grafica problemas de programación lineal con restricciones de tipo <=,
>=, =,
    y grafica la función objetivo basada en los valores óptimos encontrados.
    :param A: Matriz de coeficientes de las restricciones.
    :param b: Vector de constantes del lado derecho de las restricciones.
    :param c: Vector de coeficientes de la función objetivo.
    :param restricciones: Lista de tipos de restricciones ('<=', '>=', '=').
    :param x1_bounds: Límite inferior y superior para x1.
    :param x2_bounds: Límite inferior y superior para x2.
    """
    x1 = np.linspace(0, 100, 1000)  # Rango para graficar x1
    x2 = np.linspace(0, 100, 1000)  # Rango para graficar x2
    plt.figure(figsize=(10, 8))
    # Variables auxiliares para restricciones
    A_ub, b_ub, A_eq, b_eq = [], [], [], []
    # Procesar las restricciones
    for i in range(len(restricciones)):
        if restricciones[i] == '<=':
            A_ub.append(A[i])
            b_ub.append(b[i])
            if A[i][1] != 0:  # Restricción no horizontal
                y = (b[i] - A[i][0] * x1) / A[i][1]
                plt.plot(x1, y, label=f'{A[i][0]}x1 + {A[i][1]}x2 <= {b[i]}',
linestyle='-')
                plt.fill_between(x1, 0, y, where=(y >= 0), alpha=0.1)
            else:  # Restricción vertical (x1 = constante)
                x_vert = np.full_like(x2, b[i] / A[i][0])
```

```
                plt.plot(x_vert, x2, color='orange', linestyle='-', label=f'x1 <=
{b[i] / A[i][0]}')
                plt.fill_betweenx(x2, 0, x_vert, where=(x_vert >= 0), alpha=0.1)
        elif restricciones[i] == '>=':
            A_ub.append(-np.array(A[i]))
            b_ub.append(-b[i])
            if A[i][1] != 0:  # Restricción no horizontal
                y = (b[i] - A[i][0] * x1) / A[i][1]
                plt.plot(x1, y, label=f'{A[i][0]}x1 + {A[i][1]}x2 >= {b[i]}',
linestyle='-')
                plt.fill_between(x1, y, 100, where=(y >= 0), alpha=0.1)
            else:  # Restricción vertical (x1 = constante)
                x_vert = np.full_like(x2, b[i] / A[i][0])
                plt.plot(x_vert, x2, color='orange', linestyle='-', label=f'x1 >=
{b[i] / A[i][0]}')
                plt.fill_betweenx(x2, x_vert, 100, where=(x_vert >= 0), alpha=0.1)
        elif restricciones[i] == '=':
            A_eq.append(A[i])
            b_eq.append(b[i])
            if A[i][1] != 0:  # Restricción no horizontal
                y = (b[i] - A[i][0] * x1) / A[i][1]
                plt.plot(x1, y, '--', label=f'{A[i][0]}x1 + {A[i][1]}x2 = {b[i]}')
            else:  # Restricción vertical (x1 = constante)
                x_vert = np.full_like(x2, b[i] / A[i][0])
                plt.plot(x_vert, x2, '--', label=f'x1 = {b[i] / A[i][0]}')
    # Convertir las restricciones en arrays
    A_ub = np.array(A_ub) if A_ub else None
    b_ub = np.array(b_ub) if b_ub else None
    A_eq = np.array(A_eq) if A_eq else None
    b_eq = np.array(b_eq) if b_eq else None
    # Resolver el problema
    res = linprog(-c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=[x1_bounds,
x2_bounds], method='highs')
    # Mostrar solución óptima
    if res.success:
        vertice_optimo = res.x
        valor_optimo = -res.fun
        print(f'Solución óptima: x1 = {vertice_optimo[0]:.2f}, x2 =
{vertice_optimo[1]:.2f}')
        print(f'Valor máximo de la función objetivo: Z = {valor_optimo:.2f}')
        # Agregar el vértice óptimo al gráfico
        plt.plot(vertice_optimo[0], vertice_optimo[1], 'ro', markersize=10,
label='Óptimo')
        # Graficar la función objetivo pasando por el punto óptimo
        if c[1] != 0:  # Función objetivo no vertical
            z_opt = valor_optimo
            y_obj = (z_opt - c[0] * x1) / c[1]
```

```
            plt.plot(x1, y_obj, '--', color='green', label=f'Función Objetivo:
{c[0]}x1 + {c[1]}x2 = {z_opt}')
        else:  # Función objetivo vertical
            x_obj = z_opt / c[0]
            plt.axvline(x=x_obj, linestyle='--', color='green', label=f'Función
Objetivo: x1 = {x_obj}')
    else:
        print("No se encontró una solución factible.")
    # Configuración del gráfico
    plt.xlim(0, 30)
    plt.ylim(0, 30)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Restricciones y Solución Óptima con Función Objetivo')
    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.legend()
    plt.show()
# Ejemplo de uso
c = np.array([7, 4]) # Vector c de coeficientes de la funcion objetivo
# Matriz A de coeficientes de las restricciones
A = np.array([[3, 2],
              [1, 0],
              [1, 0],
              [0, 1],
              [7, 0],
              [0, 4]])
restricciones = ['<=', '>=', '<=','>=', '>=', '>='] # tipo de restricciones
#vector b de las restricciones
b = np.array([60,
              8,
              12,
              13,
              51,
              51])
resolver_y_graficar_con_funcion_objetivo(A, b, c, restricciones)
```

Resultados y gráfica:

```
Solución óptima: x1 = 11.33, x2 = 13.00
Valor máximo de la función objetivo: Z = 131.33
```



- **Método de Penalizaciones (Técnica en M):**

$Max\ Z = 7x_1 + 4x_2$

s.a.

$3x_1 + 2x_2 \leq 60$

$x_1 \geq 8$

$x_1 \leq 12$

$x_2 \geq 13$

$7x_1 \geq 51$

$4x_2 \geq 51$

$x \geq 0$

Volvemos a su forma estándar.

$Max\ Z = 7x_1 + 4x_2 + 0s_1 + 0s_2 + 0s_3 + 0s_4 + 0s_5 + 0s_6 - MR_2 - MR_4 - MR_5 - MR_6$

S.A.

$3x_1 + 2x_2 + s_1 = 60$

$x_1 - s_2 + R_2 = 8$

$$x_1 + s_3 = 12$$
$$x_2 - s_4 + R_4 = 13$$
$$7x_1 - s_5 + R_5 = 51$$
$$4x_2 - s_6 + R_6 = 51$$
$$x \geq 0,\ s \geq 0,\ R \geq 0$$

	Vb	z	x_1	x_2	s_2	s_4	s_5	s_6	s_1	s_3	R_2	R_4	R_5	R_6	Solución		
	z	1	-7	-4	0	0	0	0	0	0	M	M	M	M	0		
	s_1	0	3	2	0	0	0	0	1	0	0	0	0	0	60		
	R_2	0	1	0	-1	0	0	0	0	0	1	0	0	0	8		
	s_3	0	1	0	0	0	0	0	0	1	0	0	0	0	12		
	R_4	0	0	1	0	-1	0	0	0	0	0	1	0	0	13		
	R_5	0	7	0	0	0	-1	0	0	0	0	0	1	0	51		
	R_6	0	0	4	0	0	0	-1	0	0	0	0	0	1	51		
	z	1	-7-8M	-4-5M	M	M	M	M	0	0	0	0	0	0	-123M		
	s_1	0	3	2	0	0	0	0	1	0	0	0	0	0	60	60/3=20	
	R_2	0	1	0	-1	0	0	0	0	0	1	0	0	0	8	8/1=8	
	s_3	0	1	0	0	0	0	0	0	1	0	0	0	0	12	12/1=12	
	R_4	0	0	1	0	-1	0	0	0	0	0	1	0	0	13	13/0=ind	
(1/7)	R_5	0	7	0	0	0	-1	0	0	0	0	0	1	0	51 →	51/7=7.2	
	R_6	0	0	4	0	0	0	-1	0	0	0	0	0	1	51	51/0=ind	
	z	1	0	-4-5M	M	M	-1-M/7	M	0	0	0	0	1+8M/7	0	51-453M/7		R1=R1+(8M+7)R6
	s_1	0	0	2	0	0	3/7	0	1	0	0	0	-3/7	0	267/7	(267/7)/2=267/14	R2=R2-3R6
	R_2	0	0	0	-1	0	1/7	0	0	0	1	0	-1/7	0	5/7	(5/7)/0=ind	R3=R3-R6
	s_3	0	0	0	0	0	1/7	0	0	1	0	0	-1/7	0	33/7	33/7/0=ind	R4=R4-R6
	R_4	0	0	1	0	-1	0	0	0	0	0	1	0	0	13	13/1=13	
	x_1	0	1	0	0	0	-1/7	0	0	0	0	0	1/7	0	51/7	51/7/0=ind	
(1/4)	R_6	0	0	4	0	0	0	-1	0	0	0	0	0	1/4	51 →	51/4	
	z	1	0	0	M	M	-1-M/7	-1-M/4	0	0	0	0	1+8M/7	1+5M/4	102-(27/28)M		R1=R1+(5M+4)R7
	s_1	0	0	0	0	0	3/7	1/2	1	0	0	0	-3/7	-1/2	177/14	177/14/1/2=177/7	R2=R2-2R7
	R_2	0	0	0	-1	0	1/7	0	0	0	1	0	-1/7	0	5/7	5/7/0=ind	
	s_3	0	0	0	0	0	1/7	0	0	1	0	0	-1/7	0	33/7	33/7/0=ind	
1/(1/4)	R_4	0	0	0	0	-1	0	1/4	0	0	0	1	0	-1/4	1/4 →	1/4/1/4=1	R5=R5-R7
	x_1	0	1	0	0	0	-1/7	0	0	0	0	0	1/7	0	51/7	51/7/0=ind	
	x_2	0	0	1	0	0	0	-1/4	0	0	0	0	0	1/4	51/4		
	z	1	0	0	M	-4	-1-M/7	0	0	0	0	4+M	1+8M/7	M	103-(5/7)M		R1=R1+(M/4+1)R5
	s_1	0	0	0	0	2	3/7	0	1	0	0	-2	-3/7	0	85/7	85/7/1/7=85/3	R2=R2-R5/2
1/(1/7)	R_2	0	0	0	-1	0	1/7	0	0	0	1	0	-1/7	0	5/7 →	5/7/1/7=5	
	s_3	0	0	0	0	0	1/7	0	0	1	0	0	-1/7	0	33/7	33/7/1/7=33	
	s_6	0	0	0	0	-4	0	1	0	0	0	4	0	-1	1	1/0=ind	
	x_1	0	1	0	0	0	-1/7	0	0	0	0	0	1/7	0	51/7		
	x_2	0	0	1	0	-1	0	0	0	0	0	1	0	0	13	13/0=ind	R7=R7+R5/4
	z	1	0	0	-7	-4	0	0	0	0	7+M	4+M	M	M	108		R1=R1+(M/7+1)R5
(1/3)	s_1	0	0	0	3	2	0	0	1	0	-3	-2	0	0	10 →	(10)/3	R2=R2-(3/7)R3
	s_5	0	0	0	-7	0	1	0	0	0	7	0	-1	0	5		
	s_3	0	0	0	1	0	0	0	0	1	-1	0	0	0	4	4/1=4	R4=R4-R3/7
	s_6	0	0	0	0	-4	0	1	0	0	0	4	0	-1	1	1/0=ind	
	x_1	0	1	0	-1	0	0	0	0	0	1	0	0	0	8		R6=R6+R3/7
	x_2	0	0	1	0	-1	0	0	0	0	0	1	0	0	13	13/0=ind	
	z	1	0	0	0	2/3	0	0	7/3	0	M	M-2/3	M	M	394/3		R1=R1+7R2
	s_1	0	0	0	1	2/3	0	0	1/3	0	-1	-2/3	0	0	10/3		
	s_5	0	0	0	0	14/3	1	0	7/3	0	0	-14/3	-1	0	85/3		R3=R3+7R2
	s_3	0	0	0	0	-2/3	0	0	-1/3	1	0	2/3	0	0	2/3		R4=R4-R2
	s_6	0	0	0	0	-4	0	1	0	0	0	4	0	-1	1		
	x_1	0	1	0	0	2/3	0	0	1/3	0	0	-2/3	0	0	34/3		R6=R6+R2
	x_2	0	0	1	0	-1	0	0	0	0	0	1	0	0	13		

Solución:

$x_1 = \frac{34}{3},\ x_2 = 13$

$s_1 = 0,\ s_2 = \frac{10}{3},\ s_3 = \frac{2}{3},\ s_4 = 0,\ s_5 = \frac{85}{3},\ s_6 = 1$

$R_2 = 0,\ R_4 = 0,\ R_5 = 0,\ R_6 = 0$

$Max\ Z = 7x_1 + 4x_2$

$Max\ Z = 7 * \frac{34}{3} + 4 * 13$

$Max\ Z = \frac{394}{3}$

- **Método de Penalizaciones (Librería ligprog de scipy) en Python:**

Código:

```
from scipy.optimize import linprog
import numpy as np
def resolver_programacion_lineal(c, A, b, tipos, objetivo="max"):
    """
    Resuelve problemas de programación lineal con entrada simplificada e incluye
variables S.
    :param c: Coeficientes de la función objetivo.
    :param A: Matriz de coeficientes de las restricciones.
    :param b: Lado derecho de las restricciones.
    :param tipos: Tipos de restricciones ('<=', '>=', '=').
    :param objetivo: "max" para maximización, "min" para minimización.
    """
    A_ub, b_ub = [], []
    A_eq, b_eq = [], []
    variables_s = []  # Guardar las variables S
    # Procesar las restricciones según su tipo
    for i, tipo in enumerate(tipos):
        if tipo == '<=':
            A_ub.append(A[i])
            b_ub.append(b[i])
            variables_s.append(f"S{i+1}")  # Variable S para restricciones <=
        elif tipo == '>=':
            A_ub.append([-coef for coef in A[i]])  # Convertir >= a <=
            b_ub.append(-b[i])
            variables_s.append(f"S{i+1}")  # Variable S para restricciones >=
        elif tipo == '=':
            A_eq.append(A[i])
            b_eq.append(b[i])
        else:
```

```
            raise ValueError(f"Tipo de restricción no válido: {tipo}")
    # Convertir a numpy arrays si es necesario
    A_ub = np.array(A_ub) if A_ub else None
    b_ub = np.array(b_ub) if b_ub else None
    A_eq = np.array(A_eq) if A_eq else None
    b_eq = np.array(b_eq) if b_eq else None
    # Cambiar el signo de c si es maximización
    if objetivo == "max":
        c = -np.array(c)
    else:
        c = np.array(c)
    # Resolver el problema
    result = linprog(
        c,
        A_ub=A_ub,
        b_ub=b_ub,
        A_eq=A_eq,
        b_eq=b_eq,
        method='highs'
    )
    # Mostrar resultados
    if result.success:
        print("\n** Resultados **")
        # Interpretar solución
        print("Variables originales:")
        for i, valor in enumerate(result.x[:len(c)]):
            print(f"x{i+1}: {valor}")

        if variables_s:
            print("\nVariables S (holgura):")
            # Calcular las variables S manualmente
            S = np.zeros(len(variables_s))
            for i, (fila, lado_derecho) in enumerate(zip(A_ub, b_ub)):
                S[i] = lado_derecho - np.dot(fila, result.x[:len(c)])
                print(f"{variables_s[i]}: {S[i]}")
        # Valor óptimo
        valor_optimo = -result.fun if objetivo == "max" else result.fun
        print("\nValor óptimo de la función objetivo:", valor_optimo)
    else:
        print("\nNo se encontró una solución factible. Detalles:", result.message)
# Ejemplo de uso
c = np.array([7, 4])  # Coeficientes de la función objetivo
A = np.array([[3, 2],
              [1, 0],
              [1, 0],
              [0, 1],
              [7, 0],
```

```
              [0, 4]])  # Restricciones
b = np.array([60, 8, 12, 13, 51, 51])  # Lado derecho de las restricciones
tipos = ['<=', '>=', '<=', '>=', '>=', '>=']  # Tipos de restricciones
# Cambiar entre "max" y "min" según el objetivo
resolver_programacion_lineal(c, A, b, tipos, objetivo="max")
```

Resultados:

```
Variables originales:
x1: 11.333333333333334
x2: 13.0
Variables S (holgura):
S1: 0.0
S2: 3.333333333333334
S3: 0.6666666666666661
S4: 0.0
S5: 28.333333333333343
S6: 1.0
Valor óptimo de la función objetivo: 131.33333333333334
```

4. CONCLUSIÓN

El trabajo evidencia cómo las técnicas de programación lineal, como el Método Simplex y sus variantes, junto con herramientas computacionales como Python, pueden ser aplicadas eficazmente a problemas de distribución óptima de recursos en contextos académicos. Los resultados obtenidos demuestran la capacidad de estas metodologías para resolver problemas complejos, maximizando resultados bajo restricciones definidas. Esto resalta la importancia de comprender y dominar estas herramientas en disciplinas como la investigación operativa, donde se busca optimizar decisiones y recursos de manera sistemática.

En el caso de los ejemplos en el ámbito académico, se observó cómo el estudiante logró distribuir su tiempo de manera óptima, según los tiempos que manejaba, así teniendo las horas de suficiente descanso y estudio, alcanzando los puntajes necesarios para aprobar sus materias principales. Este análisis práctico enfatiza el valor de combinar técnicas matemáticas con implementaciones computacionales para garantizar soluciones precisas y escalables.

5. RECOMENDACIONES

- Ampliar el Uso de Herramientas Computacionales: Se recomienda seguir explorando herramientas como Python y sus librerías especializadas en optimización (por ejemplo, `scipy.optimize`) para abordar problemas más complejos o en distintos contextos, como logísticos o empresariales.

- Desarrollar Intuición Matemática: Aunque las herramientas computacionales son poderosas, es fundamental entender las bases matemáticas detrás de los métodos empleados. Esto permitirá diagnosticar y ajustar modelos en caso de problemas.
- Practicar con Escenarios Reales: Aplicar los métodos a situaciones cotidianas o casos reales no solo ayuda a reforzar el aprendizaje, sino que también puede abrir puertas a soluciones innovadoras en diferentes áreas profesionales.
- Documentación y Análisis: Asegúrese de documentar adecuadamente las limitaciones encontradas y las posibles mejoras al código ya modelado.
- Explorar Técnicas Avanzadas: Una vez dominados los métodos básicos, se puede investigar sobre enfoques más avanzados como la programación no lineal o algoritmos de optimización heurísticos, dependiendo de las necesidades del problema.