UNIVERSIDAD NACIONAL DEL ALTIPLANO PUNO
FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA
Curso: Métodos de Optimización
Alumno: Eddy Kennedy Mamani Hallasi
Docente: Fred Torres Cruz

ACTIVIDAD N° 2 - EJERCICIOS DE FUNCIONES

REPOSITORIO EN GITHUB

https://github.com/MetodosDeOptimizacion/2.-Ejercicios-de-Funciones.git

Ejercicio 1

Concepto: El precio de una vivienda (P) depende linealmente del área construida (A) y puede expresarse como:

$$P = mA + b \tag{1}$$

Donde:

- m: Costo por metro cuadrado.
- A: Área construida.
- b: Costos fijos.

Gráfica y Código: La figura muestra la relación entre el área construida y el precio de la vivienda.

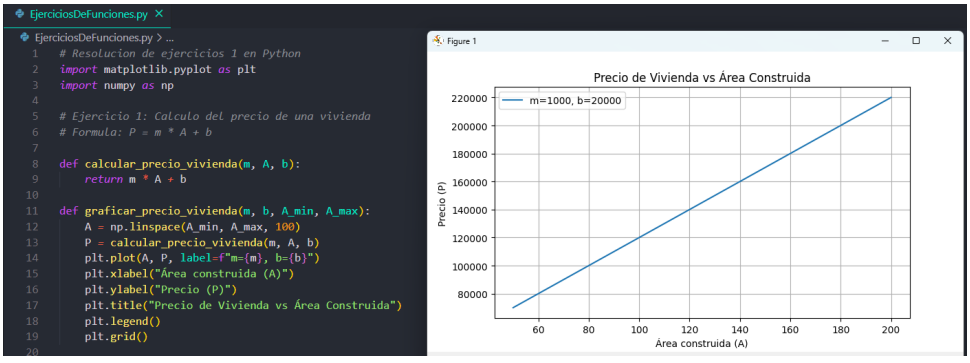

Figura 1: Gráfica del Precio de Vivienda vs Área Construida.

Código en Python:

```
# Ejercicio 1: Calculo del precio de una vivienda
# Formula: P = m * A + b

def calcular_precio_vivienda(m, A, b):
    return m * A + b
```

```
def graficar_precio_vivienda(m, b, A_min, A_max):
    A = np.linspace(A_min, A_max, 100)
    P = calcular_precio_vivienda(m, A, b)
    plt.plot(A, P, label=f"m={m}, b={b}")
    plt.xlabel("Área construida (A)")
    plt.ylabel("Precio (P)")
    plt.title("Precio de Vivienda vs Área Construida")
    plt.legend()
    plt.grid()
```

Ejercicio 2

Concepto: La ganancia mensual (G) de un modelo depende linealmente del número de predicciones realizadas (N) y se expresa como:

$$G = cN + b \tag{2}$$

Donde:

- c: Ganancia por predicción.
- N: Número de predicciones realizadas.
- b: Ingresos fijos.

Gráfica y Código:

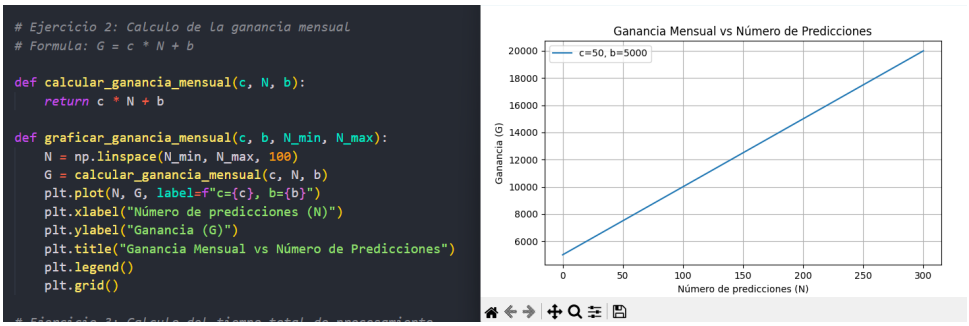

Figura 2: Gráfica de Ganancia Mensual vs Número de Predicciones.

Código en Python:

```
# Ejercicio 2: Calculo de la ganancia mensual
# Formula: G = c * N + b

def calcular_ganancia_mensual(c, N, b):
    return c * N + b

def graficar_ganancia_mensual(c, b, N_min, N_max):
    N = np.linspace(N_min, N_max, 100)
    G = calcular_ganancia_mensual(c, N, b)
    plt.plot(N, G, label=f"c={c}, b={b}")
```

```
    plt.xlabel("Número de predicciones (N)")
    plt.ylabel("Ganancia (G)")
    plt.title("Ganancia Mensual vs Número de Predicciones")
    plt.legend()
    plt.grid()
```

Ejercicio 3

Concepto: El tiempo total de procesamiento (T) depende linealmente del tamaño de los datos (D) y se expresa como:

$$T = kD + c \tag{3}$$

Donde:

- k: Tiempo por unidad de dato.
- D: Tamaño de los datos.
- c: Tiempo de procesamiento fijo.

Gráfica y Código:

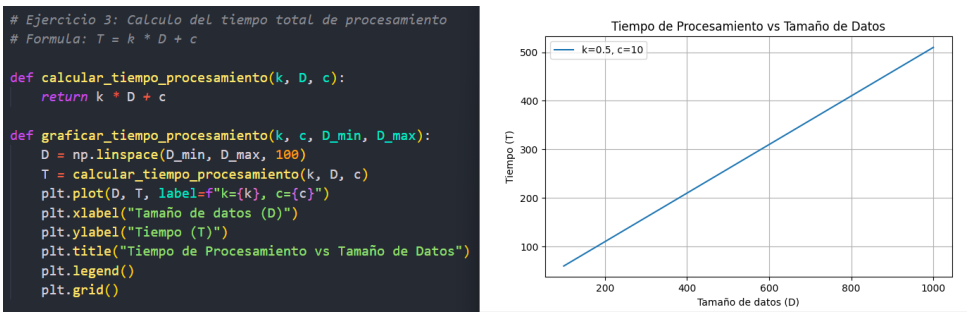

Figura 3: Gráfica de Tiempo de Procesamiento vs Tamaño de Datos.

Código en Python:

```
# Ejercicio 3: Calculo del tiempo total de procesamiento
# Formula: T = k * D + c

def calcular_tiempo_procesamiento(k, D, c):
    return k * D + c

def graficar_tiempo_procesamiento(k, c, D_min, D_max):
    D = np.linspace(D_min, D_max, 100)
    T = calcular_tiempo_procesamiento(k, D, c)
    plt.plot(D, T, label=f"k={k}, c={c}")
    plt.xlabel("Tamaño de datos (D)")
    plt.ylabel("Tiempo (T)")
    plt.title("Tiempo de Procesamiento vs Tamaño de Datos")
    plt.legend()
    plt.grid()
```

Ejercicio 4

Concepto: El costo total (C) depende linealmente de la cantidad de datos almacenados (D):

$$C = pD + f \tag{4}$$

Donde:

- p: Costo por unidad de datos.
- D: Cantidad de datos almacenados.
- f: Tarifas fijas.

Código en Python:

```
# Ejercicio 4: Calculo del costo total
def calcular_costo_total(p, D, f):
    return p * D + f

def graficar_costo_total(p, f, D_min, D_max):
    D = np.linspace(D_min, D_max, 100)
    C = calcular_costo_total(p, D, f)
    plt.plot(D, C, label=f"p={p}, f={f}")
    plt.xlabel("Cantidad de datos (D)")
    plt.ylabel("Costo Total (C)")
    plt.title("Costo Total vs Cantidad de Datos")
    plt.legend()
    plt.grid()
```

Gráfica:

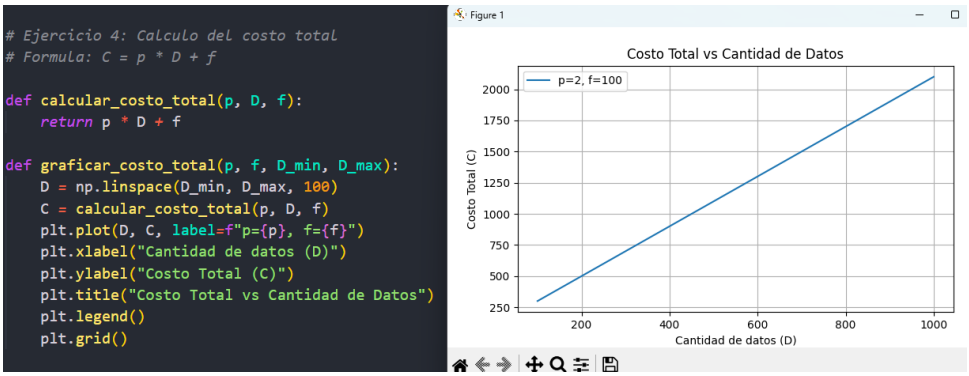

Figura 4: Gráfica del Costo Total vs Cantidad de Datos.

Ejercicio 5

Concepto: La medición calibrada (M) de un sensor depende linealmente de la medición en crudo (R):

$$M = aR + b \tag{5}$$

Donde:

- a: Factor de ajuste.
- R: Medición en crudo.
- b: Desplazamiento constante.

Código en Python:

```
# Ejercicio 5: Medición calibrada
def calcular_medicion_calibrada(a, R, b):
    return a * R + b

def graficar_medicion_calibrada(a, b, R_min, R_max):
    R = np.linspace(R_min, R_max, 100)
    M = calcular_medicion_calibrada(a, R, b)
    plt.plot(R, M, label=f"a={a}, b={b}")
    plt.xlabel("Medición en crudo (R)")
    plt.ylabel("Medición Calibrada (M)")
    plt.title("Medición Calibrada vs Medición en Crudo")
    plt.legend()
    plt.grid()
```

Gráfica:

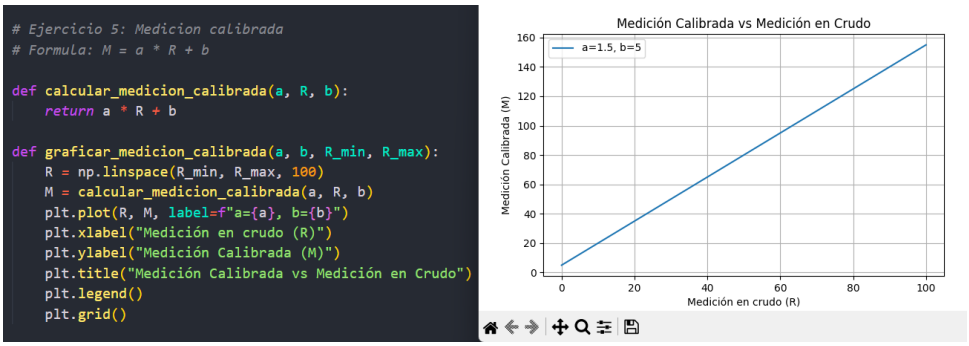

Figura 5: Gráfica de Medición Calibrada vs Medición en Crudo.

Ejercicio 6

Concepto: El tiempo de respuesta promedio (T) de un servidor depende linealmente del número de solicitudes simultáneas (S):

$$T = mS + b \tag{6}$$

Donde:

- m: Incremento por solicitud.
- S: Solicitudes simultáneas.
- b: Tiempo base.

Código en Python:

```
# Ejercicio 6: Tiempo de respuesta promedio
def calcular_tiempo_respuesta(m, S, b):
    return m * S + b

def graficar_tiempo_respuesta(m, b, S_min, S_max):
    S = np.linspace(S_min, S_max, 100)
    T = calcular_tiempo_respuesta(m, S, b)
    plt.plot(S, T, label=f"m={m}, b={b}")
    plt.xlabel("Solicitudes simultáneas (S)")
    plt.ylabel("Tiempo de Respuesta (T)")
    plt.title("Tiempo de Respuesta vs Solicitudes Simultáneas")
    plt.legend()
    plt.grid()
```

Gráfica:

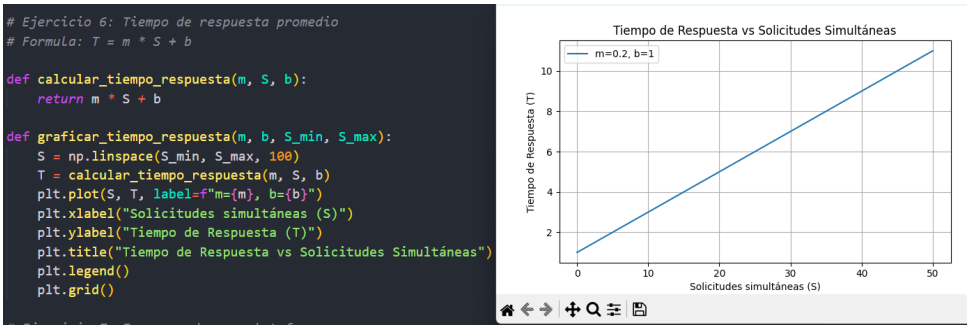

Figura 6: Gráfica de Tiempo de Respuesta vs Solicitudes Simultáneas.

Ejercicio 7

Concepto: Los ingresos (I) de una plataforma dependen del número de suscriptores (S):

$$I = pS + b \tag{7}$$

Donde:

- p: Ingreso promedio por suscriptor.
- S: Número de suscriptores.
- b: Ingresos adicionales.

Código en Python:

```
# Ejercicio 7: Ingresos de una plataforma
def calcular_ingresos(p, S, b):
    return p * S + b

def graficar_ingresos(p, b, S_min, S_max):
```

```
    S = np.linspace(S_min, S_max, 100)
    I = calcular_ingresos(p, S, b)
    plt.plot(S, I, label=f"p={p}, b={b}")
    plt.xlabel("Número de suscriptores (S)")
    plt.ylabel("Ingresos (I)")
    plt.title("Ingresos vs Número de Suscriptores")
    plt.legend()
    plt.grid()
```

Gráfica:

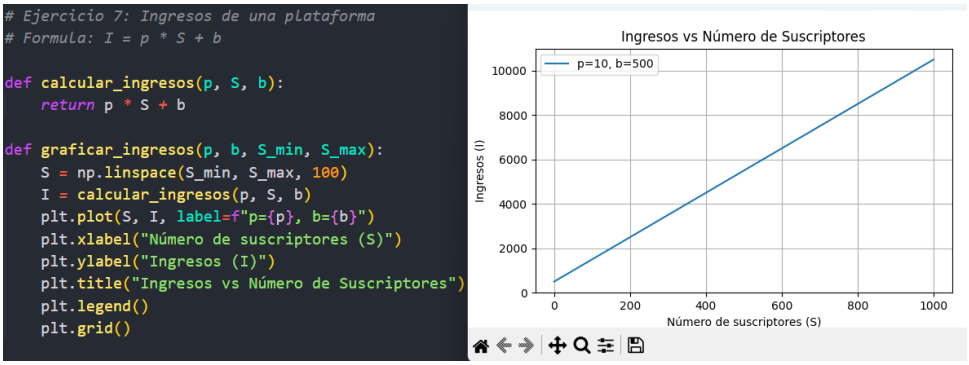

Figura 7: Gráfica de Ingresos vs Número de Suscriptores.

Ejercicio 8

Concepto: La energía consumida (E) depende linealmente del número de operaciones realizadas (O):

$$E = kO + b \tag{8}$$

Donde:

- k: Energía consumida por operación.
- O: Operaciones realizadas.
- b: Energía base.

Código en Python:

```
# Ejercicio 8: Energía consumida
def calcular_energia_consumida(k, O, b):
    return k * O + b

def graficar_energia_consumida(k, b, O_min, O_max):
    O = np.linspace(O_min, O_max, 100)
    E = calcular_energia_consumida(k, O, b)
    plt.plot(O, E, label=f"k={k}, b={b}")
    plt.xlabel("Operaciones realizadas (O)")
    plt.ylabel("Energía Consumida (E)")
```

```
    plt.title("Energía Consumida vs Operaciones Realizadas")
    plt.legend()
    plt.grid()
```

Gráfica:

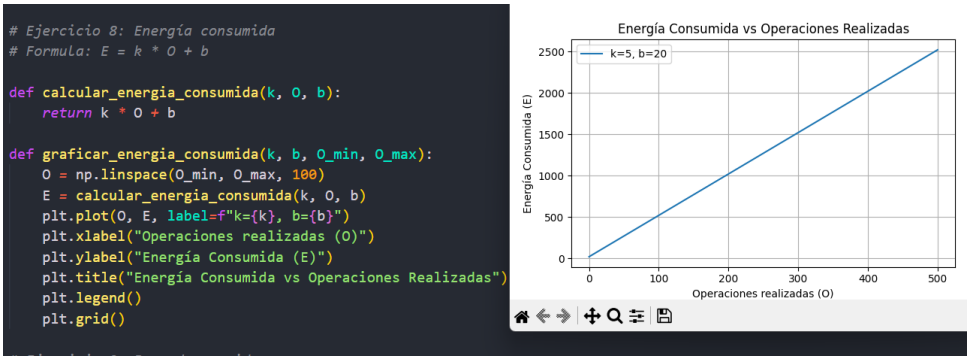

Figura 8: Gráfica de Energía Consumida vs Operaciones Realizadas.

Ejercicio 9

Concepto: El número de likes (L) en una publicación depende linealmente del número de seguidores (F):

$$L = mF + b \tag{9}$$

Donde:

- m: Proporción promedio de interacción.
- F: Número de seguidores.
- b: Nivel base de likes.

Código en Python:

```
# Ejercicio 9: Base de seguidores
def calcular_base_seguidores(m, F, b):
    return m * F + b

def graficar_base_seguidores(m, b, F_min, F_max):
    F = np.linspace(F_min, F_max, 100)
    L = calcular_base_seguidores(m, F, b)
    plt.plot(F, L, label=f"m={m}, b={b}")
    plt.xlabel("Número de Seguidores (F)")
    plt.ylabel("Base de Likes (L)")
    plt.title("Base de Likes vs Número de Seguidores")
    plt.legend()
    plt.grid()
```

Gráfica:

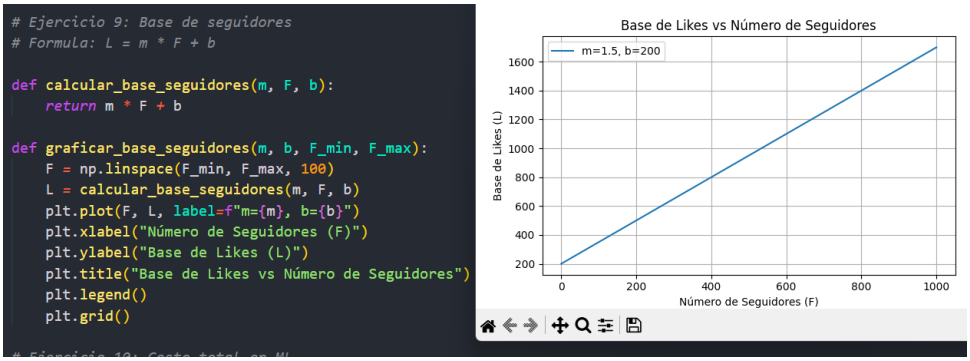

Figura 9: Gráfica de Likes vs Número de Seguidores.

Ejercicio 10

Concepto: El costo total (C) para entrenar un modelo depende del número de iteraciones (t):

$$C = pt + c \tag{10}$$

Donde:

- p: Costo por iteración.
- t: Número de iteraciones.
- c: Costos iniciales.

Código en Python:

```
# Ejercicio 10: Costo total en ML
def calcular_costo_ml(p, t, c):
    return p * t + c

def graficar_costo_ml(p, c, t_min, t_max):
    t = np.linspace(t_min, t_max, 100)
    C = calcular_costo_ml(p, t, c)
    plt.plot(t, C, label=f"p={p}, c={c}")
    plt.xlabel("Número de Iteraciones (t)")
    plt.ylabel("Costo Total (C)")
    plt.title("Costo Total vs Número de Iteraciones")
    plt.legend()
    plt.grid()
```

Gráfica:


```python
# Ejercicio 10: Costo total en ML
# Formula: C = p * t + c

def calcular_costo_ml(p, t, c):
    return p * t + c

def graficar_costo_ml(p, c, t_min, t_max):
    t = np.linspace(t_min, t_max, 100)
    C = calcular_costo_ml(p, t, c)
    plt.plot(t, C, label=f"p={p}, c={c}")
    plt.xlabel("Número de Iteraciones (t)")
    plt.ylabel("Costo Total (C)")
    plt.title("Costo Total vs Número de Iteraciones")
    plt.legend()
    plt.grid()

# Ejemplo de uso y gráficas
```

Figura 10: Gráfica del Costo Total vs Número de Iteraciones.