Universidad Nacional del Altiplano Puno

Facultad de Ingeniería Estadística e Informática

Escuela profesional de Ingeniería Estadística e Informática

Optimization Metods

Resolución de Ejercicios: Variables, funciones y Restricciones

Estudiante: Eder Elviz Luna Ochochoque

Link GitHub:

https://github.com/Ederluna-3

Docente: Dr.Fred Torres Cruz

14 de Enero 2025

Ejercicio 01:

- El precio de una vivienda (P) depende linealmente del área construida (A) y puede expresarse como P = mA + b, donde m es el costo por metro cuadrado y b representa costos fijos.

Concepto: El precio de una vivienda depende linealmente del área construida:

$$P = mA + b$$

donde:

- A: Área construida (en m^2).
- m: Costo por metro cuadrado ($300).
- b: Costos fijos ($50,000).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula el precio de la vivienda
def precio_vivienda(area, costo_m2=300, costos_fijos=5000):
    return costo_m2 * area + costos_fijos

# Simulaci n para diferentes  reas  construidas
areas = range(50, 201, 10)  #  reas  de 50 a 200  m  con
    incrementos de 10
precios = [precio_vivienda(area) for area in areas]

# Imprimir resultados
for area, precio in zip(areas, precios):
    print(f"Area construida: {area} m  -> Precio: S/.{precio}")

# Graficar
plt.plot(areas, precios, marker='o', color='b')
plt.title('precio de la vivienda y area construida')
plt.xlabel('area construida ( m  )')
plt.ylabel('Precio')
plt.grid()
plt.show()
```

Explicación: El precio de la vivienda (P) aumenta linealmente con el área construida (A). Los costos fijos (b) representan el precio mínimo incluso si el área es cero.

Resultados:

Área construida (m^2)	Precio (S/.)
50	20000
60	23000
70	26000
80	29000
90	32000
100	35000
110	38000
120	41000
130	44000
140	47000
150	50000
160	53000
170	56000
180	59000
190	62000
200	65000

article graphicx listings

Gráfico:

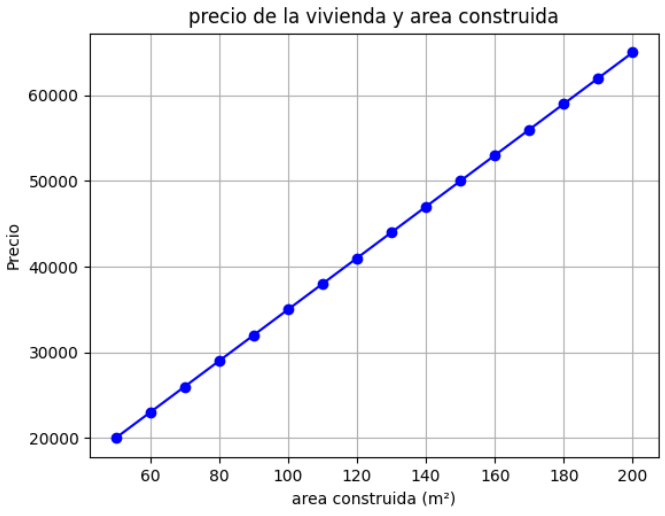

Figure 1: Descripción del gráfico.

Ejercicio 02:

La ganancia mensual G de un modelo depende linealmente del número de predicciones realizadas N:

$$G = cN + b$$

donde:

- N: Número de predicciones realizadas.
- c: Ganancia por predicción ($2).
- b: Ingresos fijos ($100).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula la ganancia mensual
def ganancia_mensual(ganancia_por_prediccion, num_predicciones
    =1000, ingresos_fijos=5000):
    return ganancia_por_prediccion * num_predicciones +
        ingresos_fijos

# Simulaci n para diferentes ganancias por predicci n
ganancias_por_prediccion = range(1, 11)  # Ganancias por
    predicci n de 1 a 10
ganancias_totales = [ganancia_mensual(c) for c in
    ganancias_por_prediccion]

# Imprimir resultados
for c, ganancia in zip(ganancias_por_prediccion, ganancias_totales)
    :
    print(f"Ganancia por predicci n: S/.{c} -> Ganancia mensual: S
        /.{ganancia}")

# Graficar
plt.plot(ganancias_por_prediccion, ganancias_totales, marker='o',
    color='g')
plt.title('Ganancia mensual y Ganancia por predicci n')
plt.xlabel('Ganancia por predicci n')
plt.ylabel('Ganancia mensual')
plt.grid()
plt.show()
```

Explicación: La ganancia mensual (G) aumenta proporcionalmente al número de predicciones realizadas (N). Los ingresos fijos (b) representan ganancias mínimas aseguradas.

Resultados:

Ganancia por Predicción (S/.)	Ganancia Mensual (S/.)
1	6000
2	7000
3	8000
4	9000
5	10000
6	11000
7	12000
8	13000
9	14000
10	15000

Ejercicio 03:

- El tiempo total de procesamiento (T) en un algoritmo depende linealmente del tamaño de los datos (D), expresado como T = kD + c, donde k es el tiempo por unidad de datos y c es un tiempo constante de configuración.

Concepto: El tiempo total de procesamiento depende del tamaño de los datos:

$$T = kD + c$$

donde:

- D: Tamaño de los datos (en MB).
- k: Tiempo por unidad de datos (0.5 segundos).
- c: Tiempo constante de configuración (10 segundos).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula el tiempo total de procesamiento
def tiempo_procesamiento(tamano_datos, tiempo_por_unidad=0.5,
    tiempo_constante=10):
    return tiempo_por_unidad * tamano_datos + tiempo_constante

# Simulaci n para diferentes tama os de datos
tamanos_datos = range(100, 1100, 100)  # Tama os de datos de 100
    MB a 1000 MB con incrementos de 100
tiempos_totales = [tiempo_procesamiento(d) for d in tamanos_datos]

# Imprimir resultados
for datos, tiempo in zip(tamanos_datos, tiempos_totales):
    print(f"Tama o de datos: {datos} MB -> Tiempo total: {tiempo}
        segundos")

# Graficar
plt.plot(tamanos_datos, tiempos_totales, marker='o', color='r')
```

```
plt.title('Tiempo total de procesamiento vs Tama o de datos')
plt.xlabel('Tama o de datos (MB)')
plt.ylabel('Tiempo total (Sg)')
plt.grid()
plt.show()
```

Explicación: El tiempo de procesamiento total (T) aumenta proporcionalmente al tamaño de los datos (D). El tiempo constante de configuración (c) representa un tiempo mínimo necesario.

Ejercicio 04:

- El costo total (C) para almacenar datos depende linealmente de la cantidad de datos almacenados (D), según C = pD + f, donde p es el costo por gigabyte y f son tarifas fijas.

Concepto: El costo total de almacenamiento depende del tamaño de los datos almacenados:

$$C = pD + f$$

donde:

- D: Cantidad de datos almacenados (en GB).
- p: Costo por GB ($5).
- f: Tarifas fijas ($20).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula el costo total de almacenamiento
def costo_almacenamiento(cantidad_datos, costo_por_gb=1,
    tarifas_fijas=5):
    return costo_por_gb * cantidad_datos + tarifas_fijas

# Simulaci n para diferentes cantidades de datos
cantidades_datos = range(50, 501, 50)  # Cantidad de datos de 50 GB
     a 500 GB con incrementos de 50
costos_totales = [costo_almacenamiento(d) for d in cantidades_datos
    ]

# Imprimir resultados
for datos, costo in zip(cantidades_datos, costos_totales):
    print(f"Cantidad de datos: {datos} GB -> Costo total: S/.{costo
        :.2f}")

# Graficar
import matplotlib.pyplot as plt
plt.plot(cantidades_datos, costos_totales, marker='o', color='
    purple')
plt.title('Costo de almacenamiento vs Cantidad de datos')
plt.xlabel('Cantidad de datos (GB)')
plt.ylabel('Costo total')
```

```
plt.grid()
plt.show()
```

Explicación: El costo total (C) aumenta proporcionalmente con el tamaño de los datos almacenados (D). Las tarifas fijas (f) aseguran un costo base incluso con cero datos.

—

Ejercicio 05:

- La medición calibrada (M) de un sensor depende linealmente de la medición en crudo (R) como M = aR + b, donde a es el factor de ajuste y b es un desplazamiento constante.

Concepto: La medición calibrada depende de la medición en crudo:

$$M = aR + b$$

donde:

- R: Medición en crudo.
- a: Factor de ajuste (1.5).
- b: Desplazamiento constante (5).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula la medici n calibrada
def medicion_calibrada(medicion_crudo, factor_ajuste=1.2,
    desplazamiento=5):
    return factor_ajuste * medicion_crudo + desplazamiento

# Simulaci n para diferentes mediciones en crudo
mediciones_crudo = range(10, 101, 10)  # Mediciones de 10 a 100 con
     incrementos de 10
mediciones_calibradas = [medicion_calibrada(r) for r in
    mediciones_crudo]

# Imprimir resultados
for crudo, calibrada in zip(mediciones_crudo, mediciones_calibradas
    ):
    print(f"Medici n en crudo: {crudo} -> Medici n calibrada: {
        calibrada:.2f}")

# Graficar
plt.plot(mediciones_crudo, mediciones_calibradas, marker='o', color
    ='orange')
plt.title('Medici n calibrada vs Medici n en crudo')
plt.xlabel('Medici n en crudo')
plt.ylabel('Medici n calibrada')
plt.grid()
plt.show()
```

Explicación: La medición calibrada (M) depende linealmente de la medición en crudo (R), ajustada por el factor a y desplazada por b.

Ejercicio 06:

- El tiempo de respuesta promedio (T) de un servidor depende linealmente del número de solicitudes simultáneas (S) como T = mS +b, donde m es el tiempo incremental por solicitud y b es el tiempo base.

Concepto: El tiempo de respuesta promedio de un servidor depende del número de solicitudes simultáneas:

$$T = mS + b$$

donde:

- S: Número de solicitudes simultáneas.
- m: Tiempo incremental por solicitud (0.3 segundos).
- b: Tiempo base (5 segundos).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula el tiempo de respuesta promedio
def tiempo_respuesta(solicitudes, tiempo_incremental=0.1,
    tiempo_base=2):
    return tiempo_incremental * solicitudes + tiempo_base

# Simulaci n para diferentes n meros de solicitudes
solicitudes = range(10, 101, 10)  # Solicitudes de 10 a 100 con
    incrementos de 10
tiempos_respuesta = [tiempo_respuesta(s) for s in solicitudes]

# Imprimir resultados
for solicitud, tiempo in zip(solicitudes, tiempos_respuesta):
    print(f"Solicitudes: {solicitud} -> Tiempo de respuesta: {
        tiempo:.2f} segundos")

# Graficar
plt.plot(solicitudes, tiempos_respuesta, marker='o', color='red')
plt.title('Tiempo de respuesta y Solicitudes simult neas')
plt.xlabel('Solicitudes simult neas')
plt.ylabel('Tiempo de respuesta (Sg)')
plt.grid()
plt.show()
```

Interpretación: El tiempo promedio de respuesta (T) aumenta linealmente con el número de solicitudes simultáneas (S).

Ejercicio 07:

- Los ingresos (I) de una plataforma dependen linealmente del número de suscriptores (S) como I = pS + b, donde p es el ingreso promedio por suscriptor y b son ingresos adicionales.

Concepto: Los ingresos de una plataforma dependen del número de suscriptores:

$$I = pS + b$$

donde:

- S: Número de suscriptores.
- p: Ingreso por suscriptor ($10).
- b: Ingresos adicionales ($500).

Código:

```
import matplotlib.pyplot as plt

# Funci n que calcula los ingresos de la plataforma
def ingresos_plataforma(suscriptores, ingreso_por_suscriptor=10,
    ingresos_adicionales=500):
    return ingreso_por_suscriptor * suscriptores +
        ingresos_adicionales

# Simulaci n para diferentes n meros de suscriptores
suscriptores = range(100, 1101, 100)  # Suscriptores de 100 a 1000
    con incrementos de 100
ingresos = [ingresos_plataforma(s) for s in suscriptores]

# Imprimir resultados
for s, ingreso in zip(suscriptores, ingresos):
    print(f"Suscriptores: {s} -> Ingresos: S/.{ingreso}")

# Graficar
import matplotlib.pyplot as plt
plt.plot(suscriptores, ingresos, marker='o', color='blue')
plt.title('Ingresos y Suscriptores')
plt.xlabel('N mero de suscriptores')
plt.ylabel('Ingresos')
plt.grid()
plt.show()
```

Explicación: A medida que aumentan los suscriptores (S), los ingresos (I) aumentan proporcionalmente.

article amsmath listings xcolor

Ejercicio 08:

- La energía consumida (E) depende linealmente del número de operaciones realizadas (O) como E = kO + b, donde k es la energía consumida por operación y b es la energía base para encender el sistema.

Concepto

La energía consumida depende del número de operaciones realizadas:

$$E = kO + b$$

donde:

- O: Número de operaciones realizadas.
- k: Energía consumida por operación (0.05 kWh).
- b: Energía base (1 kWh).

Datos utilizados

Simulamos operaciones realizadas entre 1000 y 10000, con incrementos de 1000.

Código

```
import matplotlib.pyplot as plt

# Funci n que calcula la energ a consumida
def energia_consumida(operaciones, energia_por_operacion=0.05,
    energia_base=1):
    return energia_por_operacion * operaciones + energia_base

# Simulaci n para diferentes n meros de operaciones
operaciones = range(1000, 11000, 1000)  # Operaciones de 1000 a
    10000 con incrementos de 1000
energias = [energia_consumida(o) for o in operaciones]

# Imprimir resultados
for o, energia in zip(operaciones, energias):
    print(f"Operaciones: {o} -> Energ a consumida: {energia:.2f}
        kWh")

# Graficar
plt.plot(operaciones, energias, marker='o', color='green')
plt.title('Energ a consumida vs Operaciones')
plt.xlabel('N mero de operaciones')
plt.ylabel('Energ a consumida (kWh)')
plt.grid()
plt.show()
```

Explicación:

- A mayor número de operaciones (O), la energía consumida (E) aumenta proporcionalmente.
- La energía base ($b = 1$) asegura un consumo mínimo incluso si no hay operaciones realizadas.

Ejercicio 09:

- El número de likes (L) en una publicación depende linealmente del número de seguidores (F) como L = mF + b, donde m es la proporción promedio de interacción y b es un nivel base de likes.

Concepto

El número de likes en una publicación depende del número de seguidores:

$$L = mF + b$$

donde:

- F: Número de seguidores.
- m: Proporción promedio de interacción (0.02).
- b: Nivel base de likes (10).

Datos utilizados

Simulamos seguidores entre 500 y 5000, con incrementos de 500.

Código

```
import matplotlib.pyplot as plt

# Funci n que calcula el n mero de likes
def numero_likes(seguidores, proporcion_interaccion=0.02,
    likes_base=10):
    return proporcion_interaccion * seguidores + likes_base

# Simulaci n para diferentes n meros de seguidores
seguidores = range(500, 5501, 500)  # Seguidores de 500 a 5000 con
    incrementos de 500
likes = [numero_likes(f) for f in seguidores]

# Imprimir resultados
for f, l in zip(seguidores, likes):
    print(f"Seguidores: {f} -> Likes: {l:.2f}")

# Graficar
plt.plot(seguidores, likes, marker='o', color='purple')
plt.title('Numero de likes y Seguidores')
plt.xlabel('Numero de seguidores')
plt.ylabel('Numero de likes')
plt.grid()
plt.show()
```

Explicación:

- El número de likes (L) crece proporcionalmente al número de seguidores (F).
- El nivel base de likes ($b = 10$) representa una interacción mínima asegurada.

Ejercicio 10:

- El costo total (C) para entrenar un modelo de machine learning depende linealmente del número de iteraciones (I) como C = pI + c, donde p es el costo por iteración y c son costos iniciales.

Concepto

El costo total para entrenar un modelo depende del número de iteraciones realizadas:

$$C = pI + c$$

donde:

- I: Número de iteraciones realizadas.
- p: Costo por iteración (0.1 dólares).
- c: Costos iniciales (50 dólares).

Datos utilizados

Simulamos iteraciones entre 100 y 1000, con incrementos de 100.

Código

```
import matplotlib.pyplot as plt

# Funci n que calcula el costo total de entrenamiento
def costo_entrenamiento(iteraciones, costo_por_iteracion=0.1,
    costo_inicial=50):
    return costo_por_iteracion * iteraciones + costo_inicial

# Simulaci n para diferentes n meros de iteraciones
iteraciones = range(100, 1100, 100)  # Iteraciones de 100 a 1000
    con incrementos de 100
costos = [costo_entrenamiento(i) for i in iteraciones]

# Imprimir resultados
for i, c in zip(iteraciones, costos):
    print(f"Iteraciones: {i} -> Costo total: S/.{c:.2f}")
```

```
# Graficar
plt.plot(iteraciones, costos, marker='o', color='red')
plt.title('Costo de entrenamiento vs Iteraciones')
plt.xlabel('N mero de iteraciones')
plt.ylabel('Costo total')
plt.grid()
plt.show()
```

Explicación:

- A mayor número de iteraciones (I), el costo total (C) aumenta linealmente.
- Los costos iniciales ($c = 50$) representan un costo fijo incluso si no se realizan iteraciones.