UNIVERSIDAD NACIONAL DEL ALTIPLANO PUNO

FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

10 EJERCICIOS VARIBLE ,FUNCIONES Y RESTRICCIONES

CURSO: MÉTODOS OPTIMIZACION

DOCENTE:

ING. Fred Torres Cruz

PRESENTADO POR:

Edilfonso Muñoz Anccori

SEMESTRE: V NIV

PUNO-PERÚ
2025

Implementación de Ejercicios Lineales

15 de enero de 2025

Introducción

En este documento, se presentan una serie de problemas lineales que modelan diversas situaciones prácticas, implementados en Python utilizando bucles `for`. Cada problema se describe matemáticamente y se acompaña de su respectivo código Python.

Ejercicios

Ejercicio 1: Precio de una Vivienda

El precio de una vivienda (P) depende linealmente del área construida (A):

$$P = mA + b,$$

donde m es el costo por metro cuadrado y b representa los costos fijos.

Código en Python

```python
def precio_vivienda(areas, costo_por_m2, costos_fijos):
    precios = []
    for area in areas:
        precios.append(costo_por_m2 * area + costos_fijos)
    return precios

# Ejemplo 1
areas = [120, 150, 180]
costo_por_m2 = 1000
costos_fijos = 50000
precios = precio_vivienda(areas, costo_por_m2, costos_fijos)
print(f"Los precios de las viviendas son: {precios}")
```

Ejercicio 2: Ganancia Mensual

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

$$G = cN + b,$$

donde c es la ganancia por predicción y b son ingresos fijos.

Código en Python

```python
def ganancia_mensual(predicciones, ganancia_por_prediccion, ingresos_fijos):
    ganancias = []
    for prediccion in predicciones:
        ganancias.append(ganancia_por_prediccion * prediccion + ingresos_fijos)
    return ganancias

# Ejemplo 2
predicciones = [200, 300, 400]
ganancia_por_prediccion = 50
ingresos_fijos = 1000
ganancias = ganancia_mensual(predicciones, ganancia_por_prediccion, ingresos_fijos)
print(f"Las ganancias mensuales son: {ganancias}")
```

Ejercicio 3: Tiempo Total de Procesamiento

El tiempo total de procesamiento (T) en un algoritmo depende linealmente del tamaño de los datos (D):

$$T = kD + c,$$

donde k es el tiempo por unidad de datos y c es un tiempo constante de configuración.

Código en Python

```python
def tiempo_procesamiento(datos, tiempo_por_dato, tiempo_configuracion):
    tiempos = []
    for dato in datos:
        tiempos.append(tiempo_por_dato * dato + tiempo_configuracion)
    return tiempos

# Ejemplo 3
datos = [500, 600, 700]
tiempo_por_dato = 0.1
tiempo_configuracion = 5
tiempos = tiempo_procesamiento(datos, tiempo_por_dato, tiempo_configuracion)
print(f"Los tiempos totales de procesamiento son: {tiempos}")
```

Ejercicio 4: Costo de Almacenamiento de Datos

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

$$C = pD + f,$$

donde p es el costo por gigabyte y f son tarifas fijas.

Código en Python

```python
def costo_almacenamiento(datos, costo_por_gb, tarifa_fija):
    costos = []
    for dato in datos:
        costos.append(costo_por_gb * dato + tarifa_fija)
    return costos

# Ejemplo 4
datos = [100, 150, 200]
costo_por_gb = 0.2
tarifa_fija = 10
costos = costo_almacenamiento(datos, costo_por_gb, tarifa_fija)
print(f"Los costos totales de almacenamiento son: {costos}")
```

Ejercicio 5: Medición Calibrada de un Sensor

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

$$M = aR + b,$$

donde a es el factor de ajuste y b es un desplazamiento constante.

Código en Python

```python
def medicion_calibrada(mediciones, factor_ajuste,
   desplazamiento):
    calibradas = []
    for medicion in mediciones:
        calibradas.append(factor_ajuste * medicion +
           desplazamiento)
    return calibradas

# Ejemplo 5
mediciones = [25, 30, 35]
factor_ajuste = 1.1
desplazamiento = 2
calibradas = medicion_calibrada(mediciones, factor_ajuste,
   desplazamiento)
print(f"Las mediciones calibradas son: {calibradas}")
```

Ejercicio 6: Tiempo de Respuesta Promedio

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

$$T = mS + b,$$

donde m es el tiempo incremental por solicitud y b es el tiempo base.

Código en Python

```python
def tiempo_respuesta(solicitudes, tiempo_por_solicitud, tiempo_base):
    tiempos = []
    for solicitud in solicitudes:
        tiempos.append(tiempo_por_solicitud * solicitud + tiempo_base)
    return tiempos

# Ejemplo 6
solicitudes = [50, 100, 150]
tiempo_por_solicitud = 0.05
tiempo_base = 1
tiempos = tiempo_respuesta(solicitudes, tiempo_por_solicitud, tiempo_base)
print(f"Los tiempos de respuesta promedio son: {tiempos}")
```

Ejercicio 7: Ingresos de una Plataforma

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

$$I = pS + b,$$

donde p es el ingreso promedio por suscriptor y b son ingresos adicionales.

Código en Python

```python
def ingresos_plataforma(suscriptores, ingreso_por_suscriptor, ingreso_adicional):
    ingresos = []
    for suscriptor in suscriptores:
        ingresos.append(ingreso_por_suscriptor * suscriptor + ingreso_adicional)
    return ingresos

# Ejemplo 7
suscriptores = [1000, 2000, 3000]
ingreso_por_suscriptor = 10
ingreso_adicional = 500
ingresos = ingresos_plataforma(suscriptores, ingreso_por_suscriptor, ingreso_adicional)
print(f"Los ingresos totales son: {ingresos}")
```

Ejercicio 8: Energía Consumida

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

$$E = kO + b,$$

donde k es la energía consumida por operación y b es la energía base para encender el sistema.

Código en Python

```python
def energia_consumida(operaciones, energia_por_operacion, energia_base):
    energias = []
    for operacion in operaciones:
        energias.append(energia_por_operacion * operacion + energia_base)
    return energias

# Ejemplo 8
operaciones = [100, 200, 300]
energia_por_operacion = 0.5
energia_base = 10
energias = energia_consumida(operaciones, energia_por_operacion, energia_base)
print(f"Las energ as consumidas totales son: {energias}")
```

Ejercicio 9: Número de Likes

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

$$L = mF + b,$$

donde m es la proporción promedio de interacción y b es un nivel base de likes.

Código en Python

```python
def numero_likes(seguidores, proporcion_interaccion, nivel_base):
    likes = []
    for seguidor in seguidores:
        likes.append(proporcion_interaccion * seguidor + nivel_base)
    return likes

# Ejemplo 9
seguidores = [10000, 15000, 20000]
proporcion_interaccion = 0.02
nivel_base = 50
likes = numero_likes(seguidores, proporcion_interaccion, nivel_base)
print(f"El n mero total de likes es: {likes}")
```

Ejercicio 10: Costo de Entrenar un Modelo

El costo total (C) para entrenar un modelo de machine learning depende linealmente del número de iteraciones (I):

$$C = pI + c,$$

donde p es el costo por iteración y c son costos iniciales.

Código en Python

```python
def costo_entrenamiento(iteraciones, costo_por_iteracion, costo_inicial):
    costos = []
    for iteracion in iteraciones:
        costos.append(costo_por_iteracion * iteracion + costo_inicial)
    return costos

# Ejemplo 10
iteraciones = [500, 1000, 1500]
costo_por_iteracion = 0.1
costo_inicial = 50
costos = costo_entrenamiento(iteraciones, costo_por_iteracion, costo_inicial)
print(f"Los costos totales de entrenamiento son: {costos}")
```

Figura 1: Codigo QR github

El código está disponible en GitHub. Escanea el código QR a continuación para acceder al repositorio:

Conclusiones

En este trabajo se presentaron 10 modelos lineales que ilustran cómo las matemáticas pueden ser aplicadas para resolver problemas prácticos en diversos contextos. Estos modelos cubrieron áreas como la predicción de costos, la optimización de recursos, y la simulación de procesos, mostrando la versatilidad y utilidad de las ecuaciones lineales en la vida real.

Además, se implementaron estos modelos en Python, demostrando cómo la programación es una herramienta clave para automatizar cálculos y analizar grandes volúmenes de datos de manera eficiente. El uso de bucles como for permitió procesar múltiples entradas simultáneamente, lo cual refleja el potencial de la programación para manejar tareas repetitivas y obtener resultados consistentes.