Ejercicios Intermedios: Python con NumPy y Matemáticas

Maestría en Investigación en Ciencia de Datos — FCFM, BUAP

Ejercicio 1: Derivada Numérica con Diferencias Finitas

La derivada de una función puede aproximarse numéricamente usando la fórmula de **diferencias finitas centradas**:

$$f'(x) \approx [\, f(x + h) - f(x - h) \,] \,/\, (2h)$$

Escribe una función **derivada_numerica(f, x, h=1e-5)** que reciba una función f y un arreglo de puntos x, y devuelva la derivada aproximada en cada punto. Luego:

a) Calcula la derivada numérica de $f(x) = x^3 - 3x^2 + 2$ en el intervalo $[-2, 4]$.

b) Calcula la derivada exacta analítica: $f'(x) = 3x^2 - 6x$.

c) Grafica ambas derivadas en la misma figura y calcula el error máximo absoluto entre ellas.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 4, 200)
f = lambda x: x**3 - 3*x**2 + 2
```

Funciones sugeridas: np.linspace(), np.abs(), np.max(), plt.plot()

Ejercicio 2: Regresión Lineal con Mínimos Cuadrados

Dado un conjunto de puntos (x, y), el modelo de regresión lineal $y = mx + b$ puede ajustarse resolviendo el sistema de ecuaciones normales:

$$(X^T X)\, \beta = X^T y$$

donde X es la matriz de diseño (con una columna de unos y una columna con los valores x), y $\beta = [b, m]^T$. Escribe una función que:

a) Construya la matriz de diseño X a partir de un arreglo x.

b) Resuelva el sistema normal usando np.linalg.lstsq() para encontrar m y b.

c) Grafique los puntos originales y la recta de regresión.

d) Calcule el coeficiente de determinación R^2:

$$R^2 = 1 - SS_{res} / SS_{tot}$$

```
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=float)
y = np.array([2.1, 3.9, 6.2, 7.8, 10.1, 12.0, 14.1, 15.9, 18.2, 20.0])
```

Funciones sugeridas: np.column_stack(), np.ones(), np.linalg.lstsq(), np.corrcoef()

Ejercicio 3: Transformación Lineal y Rotación de Puntos

En geometría, una rotación de ángulo θ en el plano se puede aplicar a un conjunto de puntos mediante la matriz de rotación:

$$R(\theta) = [[\cos(\theta), -\sin(\theta)], [\sin(\theta), \cos(\theta)]]$$

Escribe una función **rotar_puntos(puntos, angulo_grados)** que:

a) Construya la matriz de rotación R para el ángulo dado (en grados).

b) Aplique la rotación a todos los puntos usando multiplicación matricial.

c) Grafique los puntos originales y los rotados en la misma figura con ejes iguales.

d) Verifica que la distancia al origen de cada punto se conserva antes y después de la rotación.

```
# Puntos: vértices de un cuadrado unitario
puntos = np.array([[1,0],[1,1],[0,1],[0,0],[1,0]], dtype=float)
angulo = 45  # grados
```

Funciones sugeridas: np.cos(), np.sin(), np.radians(), np.dot(), np.linalg.norm()

Ejercicio 4: Simulación de Monte Carlo para Estimar π

El método de **Monte Carlo** permite estimar el valor de π generando puntos aleatorios dentro del cuadrado [–1, 1] × [–1, 1] y verificando cuántos caen dentro del círculo unitario ($x^2 + y^2 \leq 1$). La estimación se obtiene con:

$$\pi \approx 4 \times (\text{puntos dentro del círculo}) / (\text{total de puntos})$$

Escribe una función **estimar_pi(n)** que:

a) Genere n puntos aleatorios (x, y) en el cuadrado usando NumPy.

b) Determine cuáles caen dentro del círculo unitario.

c) Calcule y devuelva la estimación de π.

d) Ejecuta la función para n = 100, 1,000, 10,000 y 100,000 puntos. Muestra en una tabla cómo mejora la estimación y el error respecto al valor real.

```
np.random.seed(42)

# Pista para el inciso b):
# dentro = x**2 + y**2 <= 1
```

Funciones sugeridas: np.random.uniform(), np.sum(), np.pi

Ejercicio 5: Serie de Taylor y Aproximación de Funciones

La función exponencial e^x puede aproximarse mediante su serie de Taylor truncada en N términos:

$$e^x \approx \Sigma_{k=0 \to N} \, x^k / k!$$

Escribe una función **taylor_exp(x, N)** que:

a) Reciba un arreglo x y el número de términos N.

b) Calcule la aproximación de e^x sumando los N términos de la serie usando operaciones vectorizadas de NumPy (sin ciclos for sobre x).

c) Compare la aproximación con np.exp(x) y grafique ambas.

d) Grafica el error absoluto $|e^x$ – aproximación| para N = 3, 6, 10 en el intervalo [–3, 3] y comenta cómo varía la precisión.

```
from scipy.special import factorial
# o bien: usar np.math.factorial dentro de un loop sobre k

x = np.linspace(-3, 3, 300)
# Prueba con N = 3, 6 y 10 términos
```

Funciones sugeridas: np.exp(), np.abs(), np.sum() con axis, np.arange(), plt.semilogy()

Nota: La funcionalidad del código vale 70% y la calidad del código 30%.