Curso de Optimización

Tarea 8

Descripción:	Fechas
Fecha de publicación del documento:	**Abril 29, 2025**
Fecha límite de entrega de la tarea:	**Mayo 8, 2025**

Indicaciones

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.
- Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimirlo y puede anexar sólo el notebook en la tarea
- Exportar el notebook a un archivo PDF y anexarlo en la tarea como un archivo independiente. **No lo incluya dentro del ZIP**, porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.

Ejercicio 1. Mínimos cuadrados lineales (2 puntos)

Encontrar los coeficientes $c_0, c_1, \ldots$ de un polinomio de dos variables $p_n(x_1, x_2)$ de grado n que resuelve el problema de mínimos cuadrados lineales:

$$\min_{c_i} \sum_{i=1}^{m} [p_n(x_{1i}, x_{2i}) - y_i]^2.$$

El conjunto de datos $\{(x_{11}, x_{21}, y_1), (x_{12}, x_{22}, y_2), \ldots, (x_{1m}, x_{2m}, y_m)\}$ está generado como se muestra en el siguiente código. Las coordenadas de las variables x_1 y x_2 están almacenadas como columnas en el arreglo $\mathbf{X}$ y los valores y_i se almacenan en el arreglo $\mathbf{y}$:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \vdots \\ x_{1m} & x_{2m} \end{bmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}.$$

In [1]:

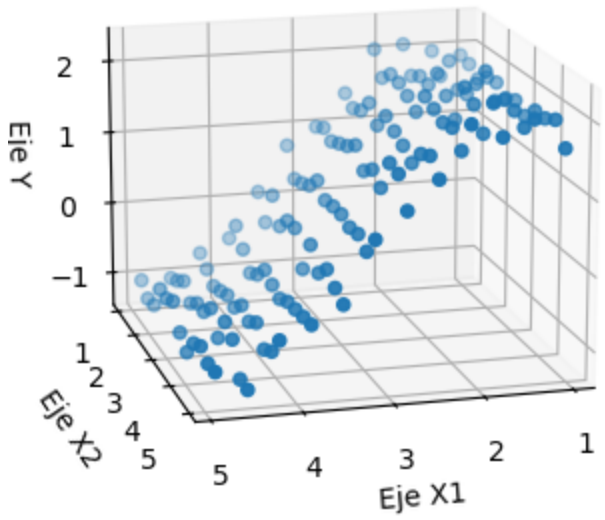
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

# Generate the true function and the data set
np.random.seed(42)
true_function = lambda x: 1.5*np.sin(x[0]) + 0.5

x1 = np.array([i*np.pi/180 for i in range(60,300,20)])
x2 = np.linspace(1,5,len(x1))
x1_grid,x2_grid = np.meshgrid(x1,x2)
m  = len(x1)*len(x2)
X  = np.zeros((m,2))
X[:,0]  = x1_grid.flatten()
X[:,1]  = x2_grid.flatten()

y = np.zeros(m)
for i,x1x2 in enumerate(X):
    y[i] = true_function(x1x2)
y += np.random.normal(0, 0.15, m)

# Grafica del conjunto de datos
fig = plt.figure(figsize=(6,4))
ax = plt.axes(projection='3d')
ax.scatter3D(X[:,0], X[:,1], y)
ax.set_xlabel('Eje X1')
ax.set_ylabel('Eje X2')
ax.set_zlabel('Eje Y')
ax.view_init(15, 75)
```



El polinomio $p_n(x_1, x_2)$ de grado n es de la forma

$$p_n(x_1, x_2) = \sum_{i_1=0}^{n} \sum_{i_2=0}^{n-i_1} c_i x_1^{i_1} x_2^{i_2},$$

es decir, el polinomio está formado por los términos $c_i x_1^{i_1} x_2^{i_2}$, donde i indexa cada término.

Ejemplos:

$$p_1(x_1, x_2) = c_0 + c_1 x_1 + c_2 x_2$$

$$p_2(x_1, x_2) = c_0 + c_1 x_1 + c_2 x_2 + c_3 x_1 x_2 + c_4 x_1^2 + c_5 x_2^2$$

Entonces

$$\sum_{i=1}^{m} [p_n(x_{1i}, x_{2i}) - y_i]^2 = (\mathbf{X}_n \mathbf{c} - \mathbf{y})^\top (\mathbf{X}_n \mathbf{c} - \mathbf{y})$$

donde $\mathbf{c} = (c_0, c_1, \ldots, c_r)^\top$ es el vector de coeficientes del polinomio, $r + 1$ es la cantidad de coeficientes y $\mathbf{X}_n$ es la matriz que tiene en cada columna los valores

$$\begin{matrix} x_{11}^{i_1} x_{21}^{i_2} \\ x_{12}^{i_1} x_{22}^{i_2} \\ \vdots \\ x_{1m}^{i_1} x_{2m}^{i_2} \end{matrix}$$

Para generar generar la matriz $\mathbf{X}_n$ se puede usar la clase `PolynomialFeatures`. Por ejemplo, crear la matriz $\mathbf{X}_3$ asociada al polinomio de grado 3:

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures

n    = 3
poly = PolynomialFeatures(n, include_bias=True)
Xn   = poly.fit_transform(X)
print('Tamaño de la matriz Xn:', Xn.shape)
```

```
Tamaño de la matriz Xn: (144, 10)
```

De acuerdo con la teoría, la solución $\mathbf{c}$ del problema de mínimos cuadrados se obtiene resolviendo el sistema de ecuaciones

$$\mathbf{X}_n^\top \mathbf{X}_n \mathbf{c} = \mathbf{X}_n^\top \mathbf{y}$$

Dado el vector $\mathbf{c}$, los valores del polinomio en los puntos (x_{1i}, x_{2i}) están dados por

$$\mathbf{y}_{pred} = \mathbf{X}_n \mathbf{c},$$

y la raíz del error cuadrático medio entre los valores verdaderos y los que predice el modelo está dado por

$$RMSE = \sqrt{\frac{(\mathbf{y} - \mathbf{y}_{pred})^\top (\mathbf{y} - \mathbf{y}_{pred})}{m}}.$$

1. Escriba una función que reciba como argumentos la matriz $\mathbf{X}$, el vector $\mathbf{y}$ y el grado n del polinomio. La función devuelve el vector de coeficientes $\mathbf{c}$ del polinomio $p_n(x_1, x_2)$ y el número de condición de la matriz $\mathbf{X}_n$.
2. Pruebe la función para $n = 1, 3, 5$ y 7. En cada caso calcule el vector $\mathbf{y}_{pred}$, el RMSE e imprima los valores:

- El grado n del polinomio
- El número de condición de la matriz $\mathbf{X}_n$
- El RMSE

Puede usar el siguiente código para graficar la superficie del polinomio usando los valores que toma en los nodos de la retícula:

```
Y_grid  = y_pred.reshape( len(x1), len(x2) )
fig = plt.figure(figsize=(8,6))
ax = plt.axes(projection='3d')
ax.plot_surface(x1_grid, x2_grid, Y_grid, rstride=1,
cstride=1, cmap='coolwarm', edgecolor='none')
ax.scatter3D(X[:,0], X[:,1], y)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Y')
ax.view_init(15, 75)
```

Solución:

In [ ]:

Ejercicio 2. Mínimos cuadrados no lineales (3 puntos)

Usando los datos del Ejercicio 1, calcular los coeficientes $\mathbf{c} = (c_0, c_1, c_2, c_3)^\top$ del modelo

$$f(\mathbf{x}; \mathbf{c}) = f(x_1, x_2; \mathbf{c}) = c_0 + c1 \sin(c_2 x_1) \cos(c_3 x_2)$$

resolviendo el problema de mínimos cuadrados no lineales

$$\min_{\mathbf{c}} \sum_{i=1}^{m} r_i(\mathbf{c})^2$$

donde $r_i(\mathbf{c}) = f(x_{1i}, x_{2i}, \mathbf{c}) - y_i$ son las componentes del vector de residuales

$$r(\mathbf{c}) = \begin{pmatrix} r_1(\mathbf{c}) \\ r_2(\mathbf{c}) \\ \vdots \\ r_m(\mathbf{c}) \end{pmatrix}.$$

1. Escriba el código de la función que evalúa el modelo $f(\mathbf{x}; \mathbf{c})$.
2. Escriba el código de la función que evalúa el residual $r(\mathbf{c})$ para el conjunto de datos del Ejercicio 1.
3. Escriba el código de la función que calcula el valor de la matriz Jacobiana $\mathbf{J}$ del residual $r(\mathbf{c})$, calculando las derivadas parciales de los residuales de manera analítica.
4. Escriba la función que implementa el algoritmo de Levenberg-Marquart para resolver el problema de mínimos cuadrados no lineales (Algoritmo 1 de la Clase 18).
5. Pruebe el algoritmo usando:

- El punto inicial $\mathbf{c}_0 = (1.0, 1.0, 0.75, 0.5)^\top$.
- $\mu_{ref} = 0.001$.
- La tolerancia $\tau = \sqrt{m}\epsilon_m^{1/3}$.
- El máximo número de iteraciones $N = 200$.

Imprima el vector $\mathbf{c}_k$, el valor f_k, el número de iteraciones k y la variable res que indica si se cumplió la tolerancia antes de terminar las iteraciones.

Además calcule el vector $\mathbf{y}_{pred}$, que tiene los valores que predice el modelo en los nodos (x_{1i}, x_{2i}) de la retícula, calcule el RMSE e imprima su valor.

También grafique la superficie del modelo y los datos para observar el ajuste que hace el modelo a los datos.

6. Repita la prueba usado como punto inicial $\mathbf{c}_0 = (1.0, 0.5, 0.75, 0.5)^\top$.
7. Agregue un comentario sobre los resultados obtenidos con respecto a los resultados en el Ejercicio 1.

Solución:

In [ ]:

Ejercicio 3. Aproximación de derivadas (2 puntos)

Repita el Ejercicio 2 aproximando las derivadas parciales que se utilizan para calcular la matriz Jacobiana del vector de residuales por medio del esquema forward finite differences de orden 1.

1. Programe la función que calcula la matriz Jacobiana del vector residuales. Se requiere usar el incremento h del esquema de diferencias finitas para aproximar las derivadas parciales:

$$\frac{\partial f}{\partial c_i}(\mathbf{c}) \approx \frac{f(\mathbf{c} + h\mathbf{e}_i) - f(\mathbf{c})}{h}.$$

2. Repita las pruebas del Ejercicio 2 con $h = 0.00001$ y $h = 0.001$ para ver como influye la elección del parámetro h en el resultado.

Solución:

In [ ]:

Ejercicio 4. Algoritmo BFGS (3 puntos)

Programe el algoritmo BFGS modificado y realice las pruebas para ver su desempeño.

Algoritmo 1: Método BFGS modificado

Entrada: La función $f(\boldsymbol{x})$, su gradiente $\nabla f(\boldsymbol{x})$, un punto inicial $\boldsymbol{x}_0$, una tolerancia $\tau > 0$, una aproximación $\boldsymbol{H}_0$ de la inversa de la Hessiana de f en el punto $\boldsymbol{x}_0$, el número máximo de iteraciones N y los parámetros para el algoritmo de backtracking.

1 Calcular $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$;
2 Definir $\boldsymbol{I}$ como la matriz identidad y $res = 0$;
3 **for** $k = 0, 1, 2, \ldots, N-1$ **do**
4 **if** $\|\boldsymbol{g}_k\| < \tau$ **then**
5 Hacer $res = 1$ y terminar el ciclo;
6 **end**
7 Calcular la dirección $\boldsymbol{p}_k = -\boldsymbol{H}_k \boldsymbol{g}_k$;
8 **if** $\boldsymbol{p}_k^\top \boldsymbol{g}_k > 0$ **then**
9 Calcular $\lambda_1 = 10^{-5} + \frac{\boldsymbol{p}_k^\top \boldsymbol{g}_k}{\boldsymbol{g}_k^\top \boldsymbol{g}_k}$;
10 Reemplazar $\boldsymbol{H}_k$ por $\boldsymbol{H}_k + \lambda_1 \boldsymbol{I}$;
11 Redefinir $\boldsymbol{p}_k$ como $\boldsymbol{p}_k - \lambda_1 \boldsymbol{g}_k$;
12 **end**
13 Calcular $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$, donde α_k se obtiene con el algoritmo de backtracking;
14 Calcular $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$;
15 Calcular $\boldsymbol{y}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$;
16 **if** $\boldsymbol{y}_k^\top \boldsymbol{s}_k \leq 0$ **then**
17 Calcular $\lambda_2 = 10^{-5} - \frac{\boldsymbol{y}_k^\top \boldsymbol{s}_k}{\boldsymbol{y}_k^\top \boldsymbol{y}_k}$;
18 Definir $\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \lambda_2 \boldsymbol{I}$;
19 **else**
20 Calcular $\rho_k = \frac{1}{\boldsymbol{y}_k^\top \boldsymbol{s}_k}$;
21 Definir $\boldsymbol{H}_{k+1} = (\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^\top) \boldsymbol{H}_k (\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^\top) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^\top$;
22 **end**
23 **end**
24 Devolver $\boldsymbol{x}_k, \boldsymbol{g}_k, k, res$;

Las modificaciones que se hacen al algoritmo BFGS sirven cuando la matriz $\mathbf{H}_k$ no es definida positiva, por lo que no se puede garantizar que $\mathbf{p}_k$ sea una dirección de descenso. En ese caso se puede perturbar la matriz para reemplazarla por $\mathbf{H}_k + \lambda_1 \mathbf{I}$, pero la perturbación sólo garantiza que la dirección $\mathbf{p}_k$ resultante cumpla con que $\mathbf{p}_k^\top \mathbf{g}_k < 0$, que es indespensable para el algoritmo de backtracking, pero no garantiza que la matriz sea definida positiva. Por eso es que solo se aplica la fórmula para

actualizar la matriz $\mathbf{H}_k$ sólo si $\mathbf{y}_k^\top \mathbf{s}_k > 0$, y en caso contrario, se prefiere volver a perturbar la matriz.

1. Programe la función que implementa el algoritmo BFGS modificado. Si la dimensión de las variables es 2, almacene los puntos $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$ que genera el algoritmo y haga que la función devuelva esta lista para usarlos para graficar la trayectoria.
2. Pruebe el algoritmo con las funciones siguientes usando

- los puntos iniciales $\mathbf{x_0}$ indicados,
- definir $\mathbf{H}_0$ como la matriz identidad,
- el número de iteraciones máximas $N = 10000$ y la tolerancia
- $\tau = \sqrt{n\epsilon_m}$, donde ϵ_m es el épsilon de máquina.
- Para el algoritmo del backtraking use $\alpha_{ini} = 1$, $c_1 = 0.1$, $\rho = 0.6$ y el máximo de iteraciones $N_b = 100$.

  Una vez que se haya ejecutado el algoritmo imprima los valores siguientes:

- $f(\mathbf{x}_0)$,
- el número de iteraciones k,
- la norma $\|\mathbf{g}_k\|$.
- Sea n la dimensión de la variable $\mathbf{x}$. Si $n <= 6$, imprimir el vector $\mathbf{x}_k$. En caso contrario, imprimir las primeras y las últimas 3 componentes de $\mathbf{x}_k$.
- $f(\mathbf{x}_k)$.

  Además, si $n = 2$, grafique los contornos de nivel de la función y la trayectoria definida por los puntos $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$.

3. Escriba un comentario sobre el desempeño del algoritmo BFGS comparado con el método de Newton implementado en el Ejercicio 3 de la Tarea 6.

Funciones de prueba

Función de Beale : Para $\mathbf{x} = (x_1, x_2)$

$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2.$$

- $\mathbf{x}_0 = (2, 3)$

Función de Himmelblau: Para $\mathbf{x} = (x_1, x_2)$

$$f(\mathbf{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

- $\mathbf{x}_0 = (2, 4)$

Función de Hartmann de dimensión 6 (Referencia): Para $\mathbf{x} = (x_1, x_2, \ldots, x_6)$

$$f(\mathbf{x}) = -\frac{1}{1.94}\left[2.58 + \sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{i=1}^{6} a_{ij}(x_j - p_{ij})^2\right)\right],$$

donde

$$\alpha = (1.0, 1.2, 3.0, 3.2)$$

$$\mathbf{A} = [a_{ij}] = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix},$$

$$\mathbf{P} = [p_{ij}] = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}.$$

Esta función tiene 6 óptimos locales. El óptimo global es
$\mathbf{x}_* = (0.20169, 0.15001, 0.476874, 0.275332, 0.311652, 0.6573)$, y $f(\mathbf{x}_*) = -3.0424$.

Función de Rosenbrock: Para $\mathbf{x} = (x_1, x_2, \ldots, x_n)$

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] \quad n \geq 2.$$

- $\mathbf{x}_0 = (-1.2, 1.0) \in \mathbb{R}^2$
- $\mathbf{x}_0 = (-1.2, 1.0, \ldots, -1.2, 1.0) \in \mathbb{R}^{200}$
- $\mathbf{x}_0 = (-1.2, 1.0, \ldots, -1.2, 1.0) \in \mathbb{R}^{600}$

Solución:

In [ ]: