Ejercicios evaluables

EJERCICIO 1

Tal y como ya hemos visto en clase, la variedad de herramientas proporcionadas por el álgebra lineal son cruciales para desarrollar y fundamentar las bases de una variedad de técnicas relacionadas con el aprendizaje automático. Con ella, podemos describir el proceso de propagación hacia adelante en una red neuronal, identificar mínimos locales en funciones multivariables (crucial para el proceso de retropropagación) o la descripción y empleo de métodos de reducción de la dimensionalidad, como el análisis de componentes principales (PCA), entre muchas otras aplicaciones. Cuando trabajamos en la práctica dentro de este ámbito, la cantidad de datos que manejamos puede ser muy grande, por lo que es especialmente importante emplear algoritmos eficientes y optimizados para reducir el coste computacional en la medida de lo posible. Por todo ello, el objetivo de este ejercicio es el de ilustrar las diferentes alternativas que pueden existir para realizar un proceso relacionado con el álgebra lineal y el impacto que puede tener cada variante en términos del coste computacional del mismo. En este caso en particular, y a modo de ilustración, nos centraremos en el cálculo del determinante de una matriz.

In [6]:
```python
#Importacion de librerias

import numpy as np
```

1. Implementa una función, determinante recursivo, que obtenga el determinante de una matriz cuadrada utilizando la definición recursiva de Laplace.

In [7]:
```python
def determinante_recursivo(A):
    A = np.array(A)
    n = A.shape[0]

    # Caso base: matriz 1x1
    if n == 1:
        return A[0, 0]

    # Caso base: matriz 2x2
    if n == 2:
        return A[0,0]*A[1,1] - A[0,1]*A[1,0]

    # Caso general
    det = 0
    for j in range(n):
        # Eliminar primera fila y j-ésima columna
        submatriz = np.delete(np.delete(A, 0, axis=0), j, axis=1)
        cofactor = (-1) ** j * A[0, j] * determinante_recursivo(submatriz)
        det += cofactor

    return det
```

2. Si A es una matriz cuadrada n×n y triangular (superior o inferior, es decir, con entradas nulas por debajo o por encima de la diagonal, respectivamente), ¿existe alguna forma de calcular de forma directa y sencilla su determinante? Justifíquese la respuesta.

Sí, la forma más rápida consiste en calcular el producto de los elementos de la diagonal principal. Esto se debe a que, en una matriz triangular (superior o inferior), todos los determinantes menores involucrados en la expansión de Laplace que no pasan por la diagonal contienen ceros, lo que anula sus contribuciones. Así, el único camino no anulado por ceros es el producto de los elementos diagonales, que coincide con el determinante.

3. Determínese de forma justificada cómo alteran el determinante de una matriz n × n las dos operaciones elementales siguientes:

- Intercambiar una fila (o columna) por otra fila (o columna).
- Sumar a una fila (o columna) otra fila (o columna) multiplicada por un escalar α.

Al intercambiar dos filas o columnas de una matriz, el determinante de la misma cambia de signo. Esto concluye con que:
- Si el número de modificaciones que realizamos es par (n % 2 == 0): El signo original se mantiene
- Si el número de modificaciones que realizamos es impar (n % 2 == 1): El signo cambia.

Al sumar a una fila o columna otra fila o columna multiplicada por un escalar α, no alteraremos el determinante, pues este es lineal respecto a cada fila o columna por separado.

4. Investiga sobre el método de eliminación de Gauss con pivoteo parcial e impleméntalo para escalonar una matriz (es decir, convertirla en una matriz triangular inferior) a partir de las operaciones elementales descritas en el apartado anterior.

La eliminación de Gauss consiste en aplicar operaciones elementales para transformar una matriz en una forma triangular (superior o inferior), lo que facilita resolver sistemas lineales o calcular determinantes.
El pivote es considerado el valor de la diagonal que se va a usar para anular los elementos superiores o inferiores. En la práctica, si el pivote es muy pequeño o cero, pueden aparecer errores numéricos o divisiones por cero. El pivoteo parcial soluciona esto intercambiando la fila actual con la que tiene el valor absoluto mayor en la columna del pivote.

5. ¿Cómo se podría calcular el determinante de una matriz haciendo beneficio de la estrategia anterior y del efecto de aplicar las operaciones elementales pertinentes? Implementa una nueva función, determinante gauss, que calcule el determinante de una matriz utilizando eliminación gaussiana.

In [43]:
```python
def determinante_gauss(A):
    A = A.astype(float).copy() # Copiar para no modificar la original
    n = A.shape[0]
    intercambios_cont = 0

    for i in range(n):
        # Pivoteo parcial: buscar la fila con máximo valor absoluto en columna i
        max_fila = i + np.argmax(np.abs(A[i:,i]))

        #Caso para matriz singular (Det = 0)
        if A[max_fila, i] == 0:
            return 0

        # Intercambiar filas si es necesario
        if max_fila != i:
            A[[i, max_fila]] = A[[max_fila, i]]
            intercambios_cont += 1

        # Eliminar elementos debajo del pivote
        for j in range(i + 1, n):
            factor = A[j, i] / A[i, i]
            A[j, i:] -= factor * A[i, i:]
            A[j, i] = 0  # Para evitar errores numéricos

    # Producto de la diagonal principal
    det = np.prod(np.diag(A))

    # Ajustar signo según número de intercambios
    if intercambios_cont % 2 != 0:
        det = -det


    return det
```

In [42]:
```python
A = np.array([
    [1, 8, 7],
    [4, 10, 9],
    [7, 4, 2]
])

print("Determinante calculado con eliminación gaussiana:", determinante_gauss(A))
print("Determinante con numpy.linalg.det:", np.linalg.det(A))
```

```
Determinante calculado con eliminación gaussiana: 46.00000000000006
Determinante con numpy.linalg.det: 46.00000000000004
```

Como se puede comprobar, se han obtenido resultados similares aplicando la función esencial de Numpy y nuestra función eliminación gaussiana

6. Obtén la complejidad computacional asociada al cálculo del determinante con la definición recursiva y con el método de eliminación de Gauss con pivoteo parcial.

Por un lado, la definición recursiva consiste en descomponer el determinante de una matriz n x n en la suma de determinantes de matrices (n-1) x (n-1). por ello, en cada nivel se realizan 'n' llamadas para calcular determinantes, lo que genera una complejidad exponencial O(n!), la cual no es viable para grandes cantidades de datos.

En cambio, el método de la eliminación gaussiana transforma la matriz en forma triangular mediante operaciones elementales. Por ello, el número total de operaciones aritméticas está en el orden de O(2/3*n^3). Más formalmente, se considera que tiene complejidad cúbica: O(n^3). Por lo tanto, el cálculo del determinante con este método es mucho más eficiente y práctico para matrices grandes.

EJERCICIO 2

En este ejercicio trabajaremos con el m´etodo de descenso de gradiente, el cual constituye otra herramienta crucial, en esta ocasi´on de la rama del c´alculo, para el proceso de retropropagaci´on asociado al entrenamiento de una red neuronal.

1. Prográmese en Python el **método de descenso de gradiente** para funciones de (n) variables. La función debe recibir como parámetros de entrada:

- El gradiente de la función que se desea minimizar el ∇f (puede ser otra función previamente implementada, llamada `grad_f`, que reciba un vector — el punto donde se calcula el gradiente — y devuelva otro vector — el gradiente en dicho punto).
- Un valor inicial x0 ∈ R n (almacenado como un vector de (n) componentes).
- El ratio de aprendizaje γ (constante para cada iteración).
- Un parámetro de tolerancia `tol` para finalizar el proceso cuando ‖∇f(x)‖2 < `tol`).
- Un número máximo de iteraciones `maxit` para evitar ejecuciones indefinidas en caso de divergencia o convergencia lenta.

La salida de la función debe ser la aproximación de x que cumple f(x) ≈ 0, correspondiente a la última iteración realizada en el método.

In [66]:
```
def descenso_gradiente(grad_f, x0, gamma, tol, maxit):
    """
    Método de descenso de gradiente para funciones de n variables.

    Parámetros:
    grad_f : función que calcula el gradiente en un punto (vector).
    x0 : numpy array, punto inicial (vector).
```

```
    gamma : float, ratio de aprendizaje.
    tol : float, tolerancia para la norma del gradiente.
    maxit : int, número máximo de iteraciones.

    Retorna:
    x : numpy array, aproximación al mínimo de la función.
    """
    x = x0.copy()

    for _ in range(maxit):
        grad = grad_f(x)
        if np.linalg.norm(grad) < tol:
            break
        x = x - gamma * grad

    return x
```

2. Sea la función f : R → R dada por

f(x) = 3x^4 + 4x^3 - 12x^2 + 7

---

i) Aplica el método de descenso de gradiente con:

- x_0 = 3
- γ = 0.001
- `tol` = 1 x 10^{-12}
- `maxit` = 10^5

---

ii) Aplica el método con:

- x_0 = 3
- γ = 0.01
- `tol` = 1 x 10^{-12}
- `maxit` = 10^5

Compara e interpreta los resultados de i) y ii) en relación con los mínimos locales analíticos y la influencia del ratio de aprendizaje γ.

---

iii) Aplica el método con:

- x_0 = 3
- γ = 0.1
- `tol` = 1 x 10^{-12}
- `maxit` = 10^5

Interpreta el resultado.

---

iv) Aplica el método con:

- x_0 = 0

- γ = 0.001
- `tol` = 1 \times 10^{-12}
- `maxit` = 10^5

Interpreta el resultado y compáralo con el análisis de f. ¿Es un resultado deseable? ¿Por qué? ¿A qué se debe este fenómeno?

In [63]:
```
# Definimos la función f(x)
def f(x):
    return 3*x**4 + 4*x**3 - 12*x**2 + 7

def grad_f(x):
    return 12*x**3 + 12*x**2 - 24*x

# Implementación del descenso de gradiente para funciones R->R, de una sola vari
def descenso_gradiente_1d(grad_f, x0, gamma, tol, maxit):
    x = x0
    for i in range(int(maxit)):
        grad = grad_f(x)
        if abs(grad) < tol:
            break
        x = x - gamma * grad
    return x, i+1

# Parámetros
tol = 1e-12
maxit = 1e5

# i) gamma = 0.001
x0 = 3
gamma = 0.001
x_min_a, iter_a = descenso_gradiente_1d(grad_f, x0, gamma, tol, maxit)
print(f"i) gamma={gamma} -> mínimo en x = {x_min_a:.10f} tras {iter_a} iteracion

# ii) gamma = 0.01
gamma = 0.01
x_min_b, iter_b = descenso_gradiente_1d(grad_f, x0, gamma, tol, maxit)
print(f"ii) gamma={gamma} -> mínimo en x = {x_min_b:.10f} tras {iter_b} iteracio

# iii) gamma = 0.1
gamma = 0.1
#x_min_d, iter_d = descenso_gradiente_1d(grad_f, x0, gamma, tol, maxit)
#print(f"d) gamma={gamma} -> mínimo en x = {x_min_d:.10f} tras {iter_d} iteracio
print(f"iii) El caso genera un error de Overflow al crecer demasiado el valor de

# iv) x0=0, gamma=0.001
x0 = 0
gamma = 0.001
x_min_e, iter_e = descenso_gradiente_1d(grad_f, x0, gamma, tol, maxit)
print(f"iv) x0={x0}, gamma={gamma} -> mínimo en x = {x_min_e:.10f} tras {iter_e}
```

```
i) gamma=0.001 -> mínimo en x = 1.0000000000 tras 832 iteraciones, f(x) = 2.00000
00000

ii) gamma=0.01 -> mínimo en x = -2.0000000000 tras 32 iteraciones, f(x) = -25.000
0000000

iii) El caso genera un error de Overflow al crecer demasiado el valor de x durant
e las iteraciones y Python no puede manejar el resultado pues se sale del rango

iv) x0=0, gamma=0.001 -> mínimo en x = 0.0000000000 tras 1 iteraciones, f(x) = 7.
0000000000
```

Interpretación

- La elección del parámetro gamma es fundamental, como se observa en los tres primeros casos. En el primero, un valor pequeño de gamma conduce a la convergencia hacia un mínimo local, aunque con un número elevado de iteraciones (832). En el segundo caso, un gamma mayor permite alcanzar un mejor mínimo (con valor de función más bajo, -25 frente a 2) en menos iteraciones (32). Sin embargo, como muestra el tercer caso, un gamma demasiado grande puede provocar que el método diverja, generando errores debido a los saltos excesivamente grandes en el proceso de actualización.
- Por otro lado, en la cuarta opción se modifica el punto inicial $x0$=0. En este caso, el método converge en una única iteración a un punto donde la derivada es cero, pero que no corresponde a un mínimo local. Esto evidencia la importancia de elegir un buen punto de partida para el método de descenso de gradiente.

3. Sea la función f : R^2 → R dada por

g(x,y) = x2 + y3 + 3xy + 1

---

i) Aplica el método de descenso de gradiente sobre la función g(x, y) con los siguientes parámetros:

Punto inicial: x0 = (-1, 1)

Ratio de aprendizaje: gamma = 0.01

Tolerancia: tol = 1e-12

Número máximo de iteraciones: maxit = 1e5

---

ii) ¿Qué ocurre si ahora partimos de x0 = (0, 0)? ¿Se obtiene un resultado deseable?

---

iii) Realícese el estudio analítico de la función y utilícese para explicar y contrastar los resultados obtenidos en los dos apartados anteriores.

In [70]:
```
#volvemos a realizar la funcion descenso de gradiente (se podría usar la anterio
def descenso_gradiente_ej3(grad_f, x0, gamma, tol, maxit):
```

```
    x = np.array(x0, dtype=float)
    for i in range(maxit):
        grad = grad_f(x)
        if np.linalg.norm(grad) < tol:
            break
        x = x - gamma * grad
    return x, i+1

# Función g(x,y)
def g(x):
    return x[0]**2 + x[1]**3 + 3*x[0]*x[1] + 1

# Gradiente de g
def grad_g(x):
    dx = 2*x[0] + 3*x[1]
    dy = 3*x[1]**2 + 3*x[0]
    return np.array([dx, dy])

# Parámetros
gamma = 0.01
tol = 1e-12
maxit = 1e5

# i) Punto inicial (-1,1)
x0_i = [-1, 1]
x_min_i, iter_i = descenso_gradiente_ej3(grad_g, x0_i, gamma, tol, int(maxit))
print(f"i) mínimo en x = {x_min_i} tras {iter_i} iteraciones, g(x) = {g(x_min_i)

# ii) Punto inicial (0,0)
x0_ii = [0, 0]
x_min_ii, iter_ii = descenso_gradiente_ej3(grad_g, x0_ii, gamma, tol, int(maxit)
print(f"ii) mínimo en x = {x_min_ii} tras {iter_ii} iteraciones, g(x) = {g(x_min

# iii) Estudio analítico
print("""
Estudio analítico:
Puntos críticos cumplen:
2x + 3y = 0
3y^2 + 3x = 0

De la primera: x = -3/2 y
Sustituyendo en la segunda:
3 y^2 + 3 (-3/2 y) = 3 y^2 - (9/2) y = 0
3 y (y - 3/2) = 0

Por tanto, y=0 o y=3/2
Si y=0 -> x=0
Si y=3/2 -> x = -9/4 = -2.25

Puntos críticos: (0,0) y (-2.25, 1.5)
""")
```

```
i) mínimo en x = [-2.25  1.5 ] tras 3140 iteraciones, g(x) = -0.6875000000
ii) mínimo en x = [0. 0.] tras 1 iteraciones, g(x) = 1.0000000000

Estudio analítico:
Puntos críticos cumplen:
2x + 3y = 0
3y^2 + 3x = 0

De la primera: x = -3/2 y
Sustituyendo en la segunda:
3 y^2 + 3 (-3/2 y) = 3 y^2 - (9/2) y = 0
3 y (y - 3/2) = 0

Por tanto, y=0 o y=3/2
Si y=0 -> x=0
Si y=3/2 -> x = -9/4 = -2.25

Puntos críticos: (0,0) y (-2.25, 1.5)
```

Interpretación

- En el primer caso, partiendo de ((-1,1)), el método converge a un mínimo local, indicando que el punto inicial está dentro de la zona de atracción de dicho mínimo.
- En el segundo caso, partiendo de ((0,0)), el método puede converger a un punto estacionario que no es mínimo (puede ser un punto de silla o máximo) o no converger correctamente, ya que la derivada es cero pero no se trata de un mínimo.
- Por ello, la elección del punto inicial es fundamental para el éxito del método de descenso de gradiente. Además, el ratio de aprendizaje influye en la velocidad y estabilidad de la convergencia, como se ha observado en ejercicios anteriores.