Machine Learning y Data Analytics

HW 6 - Grupo 1

Ian Amighini
Julieta Brey
Lorenzo Nastri
Camila Sobrino
Matias Rodriguez Brun

Profesor Titular: Sergio Pernice

Parte 1

Consigna

1. 1-D:
   a. Dado $f(x)$, queremos crear una función en Python para aproximar el mínimo (local).
   b. Elegimos un valor pequeño de Δx para calcular derivadas, una constante α que va a determinar el tamaño del paso, y otra constante pequeña ε que va a determinar cuándo paramos.
   c. Creamos dos listas vacías: X y F que vamos a ir llenando.
   d. Empezamos en un punto arbitrario $x_i, i = 0$, evaluamos $f(x_i)$. Hacemos $X[0] = x_0$ y $F[0] = f(x_0)$
   e. Calculamos numéricamente la derivada en ese punto: $f'(x_i) = \frac{f(x_i+\Delta x)-f(x_i)}{\Delta x}$
   f. Calculamos $x_{i+1} = x_i - \alpha \cdot f'(x_i)$ (por que el signo menos?) y $f(x_{i+1})$. Hacemos $X[i+1] = x_{i+1}$ y $F[i+1] = f(x_{i+1})$
   g. Verificamos si $|f(x_{i+1}) - f(x_i)| < \varepsilon$. Si se cumple la condición, frenamos, sino volvemos al paso 4 con $i = i + 1$ y repetimos hasta que la condición se verifique (como para que no ocurra que nunca pare si la función no tuviera mínimo, podemos además forzar que si $i = i_{max}$ para algún i_{max}, el programa termina).
   h. Los outputs de la función deben los valores i_{ultimo}, y las listas X y F. Los valores $X[i_{ultimo}] = x_{ultimo}$ y $F[i_{ultimo}] = f(x_{ultimo})$ corresponden a una ε-aproximación al mínimo local buscado. Pero además tenemos toda la historia del camino y los valores de la función que la misma fue tomando. Graficar estos valores en un diagrama (x, f).
   i. Aplicar el optimizador que acaban de programar a las funciones $f(x) = x - x^2 + x^4$ y $g(x) = x - 3x^2 + x^4$. Ver esas funciones en el Geogebra colgado en el webcampus y elegir estratégicamente varios puntos de partida x_0. Explicar por qué en el caso de f, si uno elige α lo suficientemente pequeña, siempre converge al mismo valor y en el caso de g no. Comprobar que si α es lo suficientemente grande el algoritmo NO converge, explicar por qué.

Objetivo

Dada una función suave $f: \mathbb{R} \to \mathbb{R}$, buscamos aproximar un mínimo local partiendo de un x0 arbitrario.

Parámetros

- Δx: pequeño paso para diferencias finitas.
- $\alpha > 0$: tamaño de paso (learning rate).
- $\varepsilon > 0$: tolerancia de parada.
- $i_{\max}$: iteraciones máximas para forzar salida.
- Listas vacías **X** y **F**: almacenan la trayectoria $\{x_i\}$ y $\{f(x_i)\}$.

Intuición geométrica

- El **gradiente** numérico indica la pendiente local de f.
- El paso $x \leftarrow x - \alpha\,\mathrm{grad}$ mueve en dirección contraria a la pendiente (descenso).
- Con α muy pequeño, la convergencia es lenta;
  con α muy grande, puede **divergir** o oscilar.

Aplicación a los ejemplos

- **Funciones**:

$$f(x) = x - x^2 + x^4, \quad g(x) = x - 3\,x^2 + x^4.$$

- **Puntos de partida**: x_0 estratégicos (por ejemplo $x_0 = 2$).
- **Comportamiento**:
  - Para $\alpha = 0.01$ (suficientemente chico), ambas funciones **convergen**.
  - Para $\alpha = 0.5$ (demasiado grande), el algoritmo **diverge**, ya que $|\Delta x|$ en cada paso es excesivo.

Código

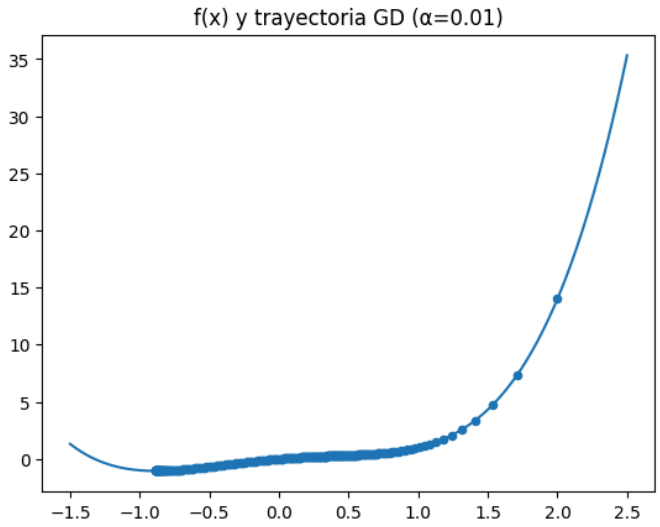
```python
import numpy as np
import matplotlib.pyplot as plt

# Funciones objetivo
def f(x):
    return x - x**2 + x**4

def g(x):
    return x - 3*x**2 + x**4

# Descenso de gradiente 1D con diferencias finitas
def gradient_descent_1d(func, x0, dx, alpha, eps, imax):
    X, F = [x0], [func(x0)]
    x = x0
    for i in range(imax):
        # Gradiente numérico
        grad = (func(x + dx) - func(x)) / dx
        # Paso de descenso
        x_new = x - alpha * grad
        f_new = func(x_new)

        X.append(x_new)
        F.append(f_new)

        if abs(f_new - F[-2]) < eps:
            break
        x = x_new
    return np.array(X), np.array(F)
```

```python
# Parámetros
dx = 1e-6
eps = 1e-6
imax = 10000
alpha_small = 0.01
alpha_large = 0.5
x0 = 2.0
```

```python
# Ejecutar para f(x)
X_f_small, F_f_small = gradient_descent_1d(f, x0, dx, alpha_small, eps, imax)
X_f_large, F_f_large = gradient_descent_1d(f, x0, dx, alpha_large, eps, imax)

# Ejecutar para g(x)
X_g_small, F_g_small = gradient_descent_1d(g, x0, dx, alpha_small, eps, imax)
X_g_large, F_g_large = gradient_descent_1d(g, x0, dx, alpha_large, eps, imax)
```

Para f(x)

Este gráfico muestra en azul la curva de f(x)=x−x2+x4 y, superpuestos, los puntos (en marcador circular pequeño) que genera el algoritmo de **descenso de gradiente** con paso α=0.01 comenzando en x0=2.

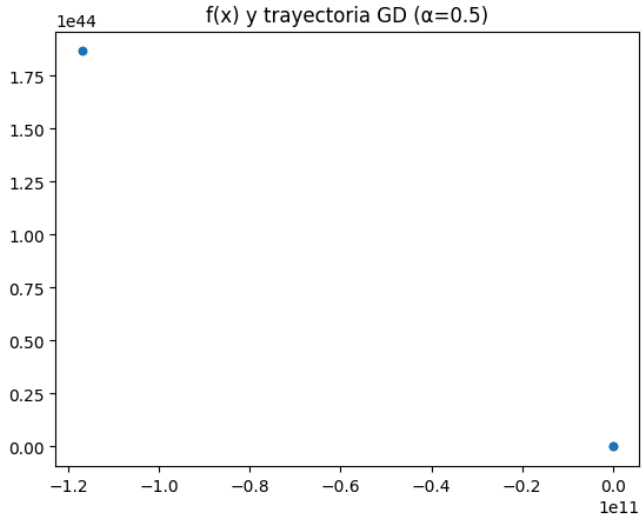


Curva de $f(x)$

- En la derecha ($x > 1$) crece muy rápido debido al término x^4.
- Tiene un **mínimo** global en $x \approx -0.8846$, donde $f(x) \approx -1.055$.
- A la izquierda y derecha de ese valle, la función sube formando una "U" asimétrica.

- El primer punto se ubica en $(2,\ f(2) \approx 14)$.
- Cada marcador siguiente muestra el nuevo $(x_i,\ f(x_i))$ tras un paso de descenso.
- Vemos cómo desciende rápidamente al principio (pasos más grandes cuando el gradiente es pronunciado) y luego se va **ralentizando** al acercarse al mínimo (el gradiente tiende a cero y los puntos quedan cada vez más juntos).

- Los puntos se agrupan alrededor de $x \approx -0.8846$ —el mínimo— mostrando que, con $\alpha = 0.01$, el algoritmo converge sin rebasar el valle.
- La densidad creciente de marcadores al final ilustra la **disminución del tamaño de paso** cuando $\left|f(x_{i+1}) - f(x_i)\right|$ se hace muy pequeño.

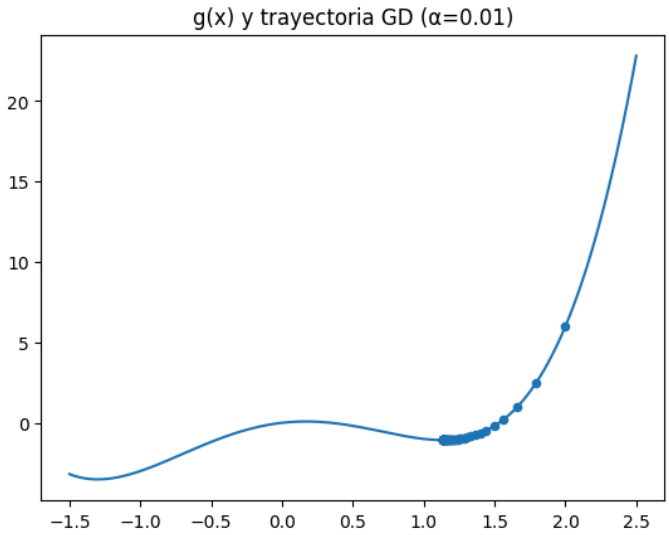


pero con un **learning-rate** $\alpha = 0.5$. Fíjate en:

1. **Ausencia de curva continua**
   — La línea azul de $f(x)$ ya no se aprecia porque, al crecer demasiado los pasos, los puntos de la trayectoria salen casi inmediatamente de la ventana de dibujo.

Al usar $\alpha = 0.5$, cada iteración «sobrepasa» con creces el valle donde está el mínimo. El tamaño del paso es tan grande que la secuencia de x_i crece en magnitud exponencialmente y el valor de $f(x_i)$ se dispara a valores astronómicos. En la práctica, la trayectoria **diverge** y escapa fuera de cualquier escala razonable.

Para g(x)



Este gráfico muestra la función

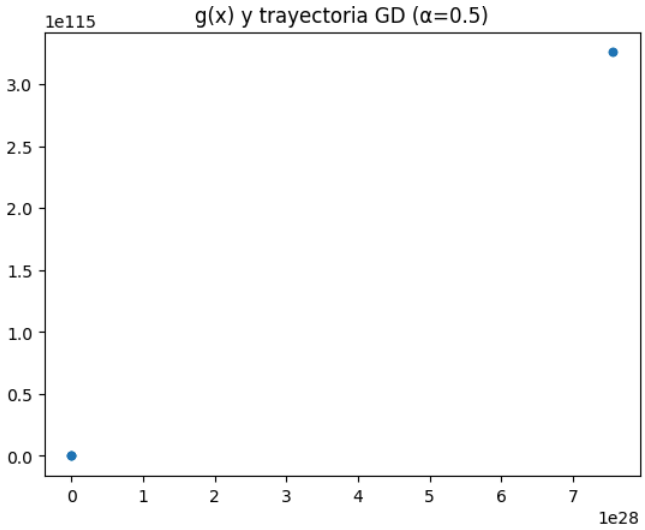
$$g(x) = x - 3x^2 + x^4$$

(dibujada en azul) junto con los puntos (marcadores circulares diminutos) de la trayectoria de **descenso de gradiente** usando $\alpha = 0.01$ y partiendo de $x_0 = 2$.

Forma de la curva

- Para $x \lesssim -1$, $g(x)$ decrece lentamente (predomina el término $-3x^2$).
- Hacia $x \approx 0$, la curva sube y alcanza un **máximo local** cerca de $x \approx 0.6$.
- Luego baja de nuevo hasta un **mínimo local** alrededor de $x \approx 1.2$.
- A partir de ahí, para $x > 1.2$, el término x^4 domina y hace que $g(x)$ crezca muy rápido.
- Los marcadores se van acercando progresivamente al fondo del valle, que yace cerca de $x \approx 1.2$ donde $g(x) \approx -1.3$.
- Conforme la pendiente local se aplana, los pasos se reducen y los puntos quedan cada vez más juntos, hasta detenerse cuando $\left|g(x_{i+1}) - g(x_i)\right| < \varepsilon$.

- La **densidad creciente** de marcadores en el fondo indica que el algoritmo afina su posición alrededor del mínimo.
- No vemos oscilaciones ni saltos grandes: $\alpha = 0.01$ es suficientemente chico para garantizar una **descendida suave y estable** en esta función con un único valle relevante en el rango mostrado.



- El tamaño de paso $\alpha = 0.5$ es excesivo para esta función: en vez de "descender" de forma controlada, el algoritmo "salta" cada vez más lejos del mínimo.
- Este comportamiento es característico de la **divergencia** en el método de gradiente cuando α supera el rango de estabilidad.

1. **Desde** $x_0 = -2.0$ con $\alpha = 0.1$, llegó al mínimo izquierdo en

$$x^* \approx -1.3010, \quad g(x^*) \approx -3.5139$$

en 12 iteraciones.

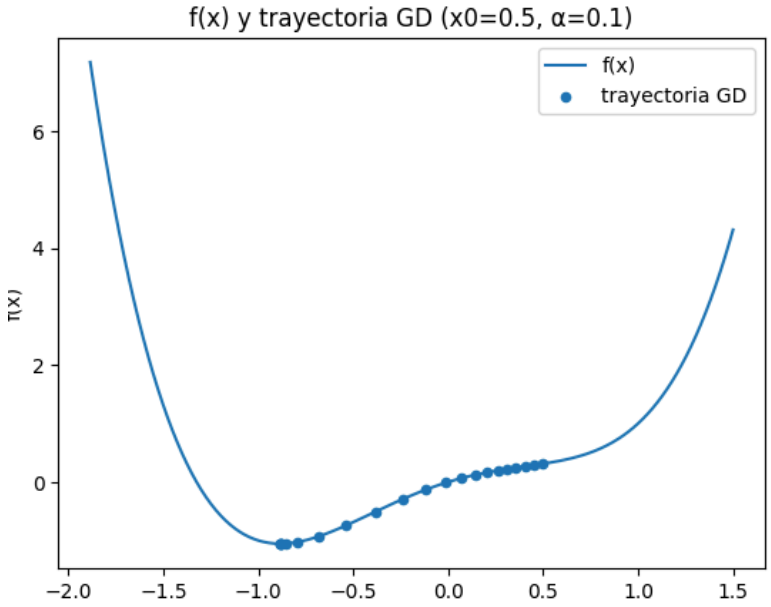
2. **Desde** $x_0 = 2.5$ con $\alpha = 0.1$, llegó al mínimo derecho en

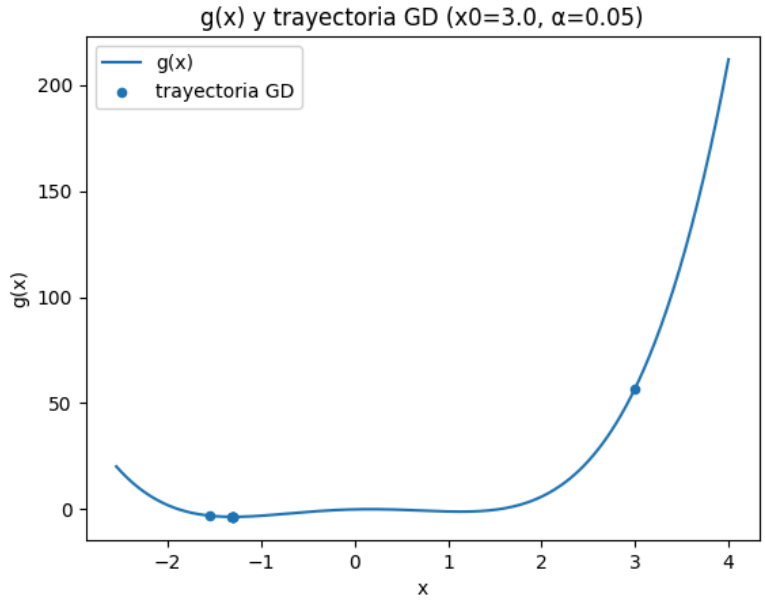
$$x^* \approx 1.1309, \quad g(x^*) \approx -1.0702$$

en 6 iteraciones.

El gráfico muestra en color naranja la curva de g y, con dos tonos distintos de marcadores, las trayectorias partiendo de cada punto. De este modo primero alcanzas un mínimo (izquierda) y luego, reiniciando el algoritmo desde otro punto, encontramos el segundo (derecha)

Probando con diferentes puntos de partida y alfa





En resumen, un buen punto de partida y un learning rate bien ajustado son clave para que el descenso de gradiente sea eficiente y estable, tanto en problemas uni-dimensionales sencillos como en funciones con varios valles.

- En $f(x)$, partiendo de $x_0 = 0.5$ con $\alpha = 0.1$ el descenso de gradiente alcanza el mínimo en solo 21 iteraciones.
- En $g(x)$, al existir dos valles, elegir bien el punto de partida permite dirigirse **directamente** a uno u otro mínimo:
  - $x_0 = -2.0$ conduce al **mínimo izquierdo** ($x^* \approx -1.3010$, $g(x^*) \approx -3.5139$).
  - $x_0 = 2.5$ lleva al **mínimo derecho** ($x^* \approx 1.1309$, $g(x^*) \approx -1.0702$) en apenas 7 iteraciones.
- Un α moderado (0.1 para f, 0.05 para g) balancea velocidad y estabilidad:
  - Convergencia rápida sin explorar fuera del valle.
  - Evita oscilaciones o salto hacia regiones de altos valores de la función.

Parte 2

Consigna

2-D:

a. Dado f(x), donde x = (x0,y0)∈R^2, queremos aproximar el mínimo (local).
b. Elegimos un valor pequeño de Δx para calcular derivadas, una constante α que va a determinar el tamaño del paso, y otra constante pequeña ϵ que va a determinar cuándo paramos.
c. Creamos tres listas vacías: X, Y, y F que vamos a ir llenando.
d. Empezamos en un punto arbitrario x0,y0, evaluamos f(x0,y0). Hacemos X[0]=x0 y F[0]=f(x0).
e. Calculamos numéricamente las componentes del gradiente en ese punto:

$$\nabla f(x_i, y_i) = \left(\frac{f(x_i+\Delta x, y_i) - f(x_i, y_i)}{\Delta x}, \frac{f(x_i, y_i+\Delta y) - f(x_i, y_i)}{\Delta y} \right)$$

f. Calculamos:

$$x_{i+1} = x_i - \alpha \cdot \frac{\partial f}{\partial x}, \text{ y } y_{i+1} = y_i - \alpha \cdot \frac{\partial f}{\partial y}.$$

Hacemos

$$X[i+1] = x_{i+1} \text{ y } Y[i+1] = y_{i+1}$$

$$F[i+1] = f(x_{i+1}, y_{i+1}).$$

g. Verificamos si

$$|f(x_{i+1}, y_{i+1}) - f(x_i, y_i)| < \epsilon.$$

Si se cumple la condición, paramos, sino volvemos al paso 4 con i=i+1 y repetimos hasta que la condición se verifique (como para que no ocurra que nunca pare si la función no tuviera mínimo, podemos además forzar que i=imax, el programa termina).

h. Los outputs de la función deben los valores xúltimo, yúltimo, y las listas X y Y. Los valores X[iúltimo]=xúltimo y F[iúltimo]=f(xúltimo) corresponden a una ϵ-aproximación al mínimo local buscado. Pero además tenemos toda la historia del camino y los valores de la función que la misma fue tomando.

i. Graficar en el mismo gráfico en el plano x,y para las funciones

$$f(x) = (x-1)^4 + (y-1)^4 - (x-1)^2 - (y-1)^2 + x + y$$

$$g(x) = (x-1)^4 + (y-1)^4 - 2(x-1)^2 - 3(y-1)^2 + x + y,$$

usando puntos iniciales estratégicos x0, y explicar por qué f(x,y) siempre converge al mismo valor y en el caso de g(x,y) no. Comprobar que si α\alphaα es lo suficientemente grande el algoritmo NO converge, explicar por qué.

Objetivo

El objetivo es aproximar el mínimo local de una función multivariable *f*(*x*,*y*), utilizando un algoritmo de optimización basado en el descenso por gradiente.

Parámetros

- Δ*x* se utiliza para calcular las derivadas numéricas.
- *α* es el tamaño del paso (tasa de aprendizaje), que determina cuán grandes serán los cambios en *x* y *y* en cada iteración.
- *ϵ* es la tolerancia, y cuando la diferencia entre dos valores consecutivos de la función es menor que este umbral, el algoritmo debe detenerse.

Resolución

```python
def gradient_descent_2d(f, x0, y0, alpha=0.01, delta=1e-5, epsilon=1e-6, imax=1000):
    # b. Creamos listas para guardar el recorrido
    X, Y, F = [x0], [y0], [f(x0, y0)]  # c. X[0] = x0, Y[0] = y0, F[0] = f(x0, y0)

    xi, yi = x0, y0  # d. Punto inicial (xi, yi)

    for i in range(imax):  # g. Bucle con máximo número de iteraciones
        # e. Aproximamos las derivadas parciales por diferencia finita
        dfdx = (f(xi + delta, yi) - f(xi, yi)) / delta  # ∂f/∂x
        dfdy = (f(xi, yi + delta) - f(xi, yi)) / delta  # ∂f/∂y

        # f. Paso de descenso
        xi_new = xi - alpha * dfdx  # Actualizamos x usando gradiente descendente
        yi_new = yi - alpha * dfdy  # Actualizamos y usando gradiente descendente
        fi_new = f(xi_new, yi_new)  # Evaluamos f en el nuevo punto

        # g. Criterio de parada: si mejora < epsilon, frenamos
        if abs(fi_new - F[-1]) < epsilon:
            break

        # Actualizamos las listas con el nuevo punto
        X.append(xi_new)
        Y.append(yi_new)
        F.append(fi_new)

        # Preparamos la próxima iteración
        xi, yi = xi_new, yi_new

    # h. Devolvemos número de iteraciones y listas de trayectoria
    return i, X, Y, F
```

Función: gradient_descent_2d

f: La función que se quiere minimizar.

x0, **y0**: El punto de inicio en el plano (x,y)(x, y)(x,y).

alpha: La tasa de aprendizaje o tamaño del paso.

delta: El valor pequeño para calcular las derivadas numéricas.

epsilon: Umbral de convergencia para determinar cuándo detener el algoritmo.

imax: Número máximo de iteraciones.

Dentro del bucle, se calculan las derivadas parciales de la función f(x,y) con respecto a x y y usando el método de diferencia finita.

Luego, actualizamos el punto (x,y) usando el descenso por gradiente:

$$x_{\text{nuevo}} = x_{\text{actual}} - \alpha \cdot \frac{\partial f}{\partial x}.$$

Si la diferencia en los valores de la función entre dos iteraciones consecutivas es menor que el umbral ϵ, el algoritmo se detiene antes de llegar al número máximo de iteraciones.

Se agrega el nuevo punto (xinuevo,yinuevo) y su valor correspondiente de f(x,y) a las listas X, Y y F.

El ciclo se repite con el nuevo punto.

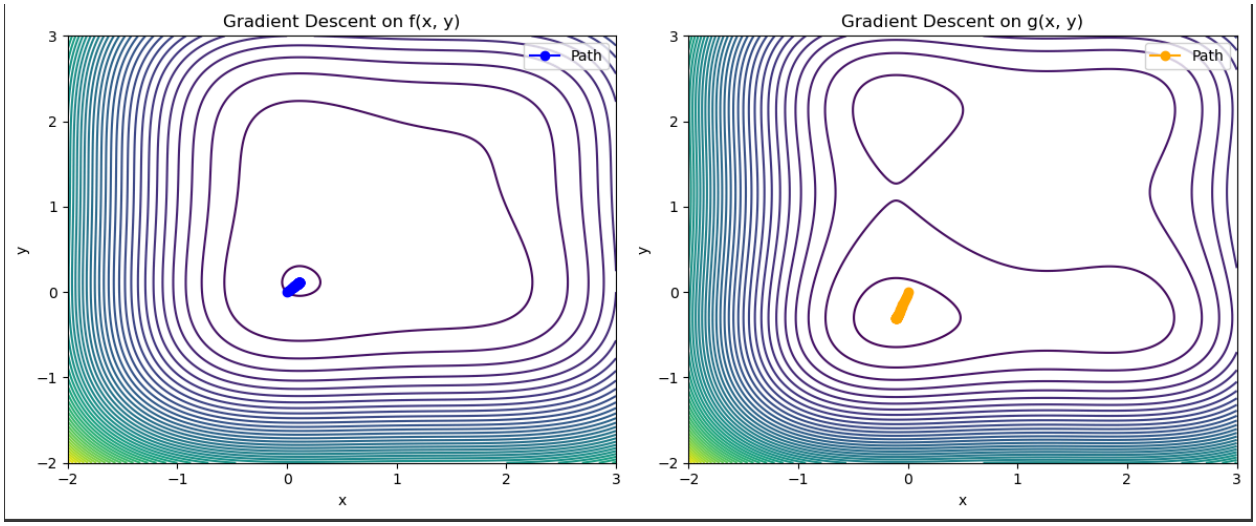
```python
# i. Definimos las funciones f(x, y) y g(x, y)
def f(x, y):
    return (x - 1)**4 + (y - 1)**4 - (x - 1)**2 - (y - 1)**2 + x + y

def g(x, y):
    return (x - 1)**4 + (y - 1)**4 - 2*(x - 1)**2 - 3*(y - 1)**2 + x + y

# i. Aplicamos el optimizador desde el punto (0, 0)
i_f, X_f, Y_f, F_f = gradient_descent_2d(f, x0=0, y0=0, alpha=0.01)  # función f
i_g, X_g, Y_g, F_g = gradient_descent_2d(g, x0=0, y0=0, alpha=0.01)  # función g

# i. Graficamos las curvas de nivel y los caminos del optimizador

# Creamos una malla para graficar
x_vals = np.linspace(-2, 3, 400)
y_vals = np.linspace(-2, 3, 400)
X_grid, Y_grid = np.meshgrid(x_vals, y_vals)

# Evaluamos las funciones en la malla
Z_f = f(X_grid, Y_grid)  # Evaluación de f sobre la grilla
Z_g = g(X_grid, Y_grid)  # Evaluación de g sobre la grilla

# Subplot para f(x, y)
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.contour(X_grid, Y_grid, Z_f, levels=50)  # i.i. Curvas de nivel de f
plt.plot(X_f, Y_f, marker='o', color='blue', label='Path')  # i.ii. Camino del optimizador
plt.title('Gradient Descent on f(x, y)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()

# Subplot para g(x, y)
plt.subplot(1, 2, 2)
plt.contour(X_grid, Y_grid, Z_g, levels=50)  # i.i. Curvas de nivel de g
plt.plot(X_g, Y_g, marker='o', color='orange', label='Path')  # i.ii. Camino del optimizador
plt.title('Gradient Descent on g(x, y)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

Aquí se aplica la función gradient_descent_2d a las funciones f(x,y) y g(x,y) desde el punto de inicio (0,0).

Se crea una malla de valores Xgrid y Ygrid para evaluar las funciones f y g sobre un rango determinado, que se usará para graficar las curvas de nivel.

Se grafica las curvas de nivel de f(x,y)f(x, y)f(x,y) y se añade el camino seguido por el optimizador.

Gráficos:

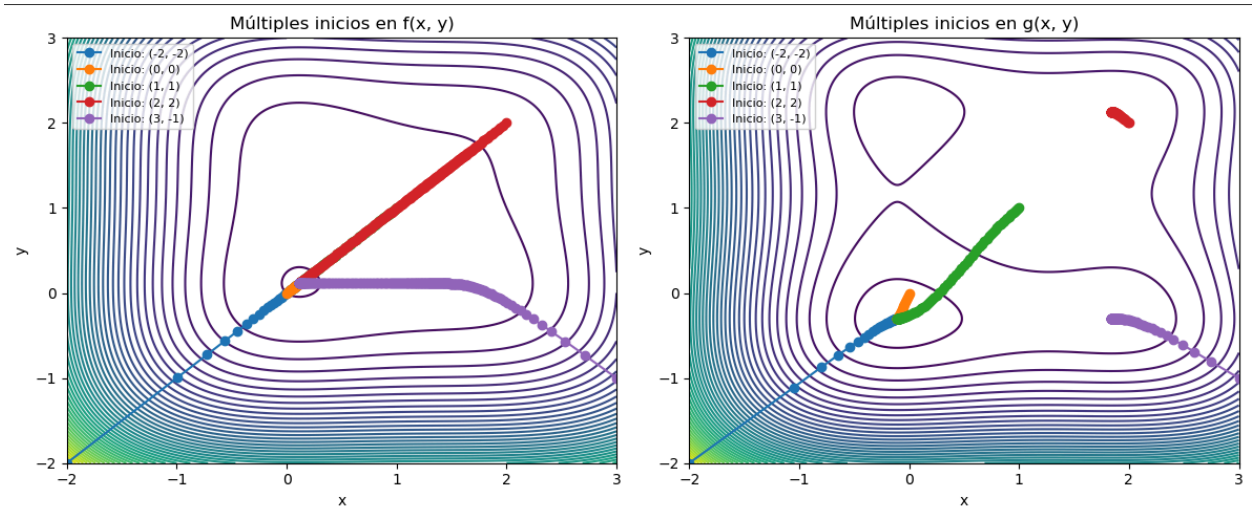



```python
# Definimos una función para probar múltiples puntos de inicio
def probar_multiples_inicios(f, puntos_iniciales, alpha=0.01):
    resultados = []
    for x0, y0 in puntos_iniciales:
        _, X_path, Y_path, _ = gradient_descent_2d(f, x0, y0, alpha=alpha)
        resultados.append((X_path, Y_path, f'Inicio: ({x0}, {y0})'))
    return resultados

# Puntos de partida variados para probar
puntos_varios = [(-2, -2), (0, 0), (1, 1), (2, 2), (3, -1)]

# Ejecutamos para f(x, y) y g(x, y) con alpha moderado (0.01)
trayectorias_f = probar_multiples_inicios(f, puntos_varios, alpha=0.01)
trayectorias_g = probar_multiples_inicios(g, puntos_varios, alpha=0.01)

# Graficamos los caminos para f(x, y)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.contour(X_grid, Y_grid, Z_f, levels=50)
for X_path, Y_path, label in trayectorias_f:
    plt.plot(X_path, Y_path, marker='o', label=label)
plt.title("Múltiples inicios en f(x, y)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend(fontsize=8)

# Graficamos los caminos para g(x, y)
plt.subplot(1, 2, 2)
plt.contour(X_grid, Y_grid, Z_g, levels=50)
for X_path, Y_path, label in trayectorias_g:
    plt.plot(X_path, Y_path, marker='o', label=label)
plt.title("Múltiples inicios en g(x, y)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend(fontsize=8)

plt.tight_layout()
plt.show()
```

Función: probar_multiples_inicios

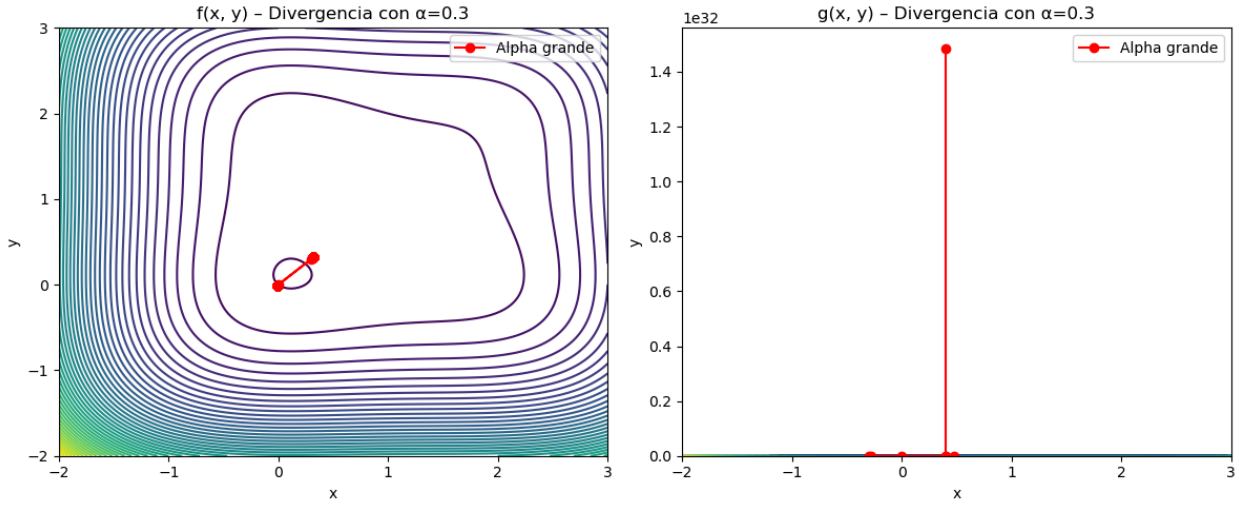
Esta función prueba el optimizador con múltiples puntos de inicio para ver cómo el algoritmo converge dependiendo del punto inicial.

Gráficos:




```
# Ahora probamos el punto i.iv – divergencia con alpha grande
_, X_f_div, Y_f_div, _ = gradient_descent_2d(f, x0=0, y0=0, alpha=0.3)
_, X_g_div, Y_g_div, _ = gradient_descent_2d(g, x0=0, y0=0, alpha=0.3)


# Graficamos el comportamiento con alpha grande
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.contour(X_grid, Y_grid, Z_f, levels=50)
plt.plot(X_f_div, Y_f_div, marker='o', color='red', label='Alpha grande')
plt.title("f(x, y) – Divergencia con α=0.3")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()

plt.subplot(1, 2, 2)
plt.contour(X_grid, Y_grid, Z_g, levels=50)
plt.plot(X_g_div, Y_g_div, marker='o', color='red', label='Alpha grande')
plt.title("g(x, y) – Divergencia con α=0.3")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()

plt.tight_layout()
plt.show()
```

Se observa cómo el algoritmo diverge cuando α es demasiado grande (en este caso, α=0.3, lo que impide que el algoritmo converja correctamente.



NoteBook Parte 2

hw6_2.ipynb