Universidad de la República - Facultad de Ingeniería

Curso 2023 : Fundamentos de Optimización

Grupo 133

Guido Dinello
5.031.022-5

Mathías Ramilo
5.665.788-5

Obligatorio 2

Índice

1. Ejercicio 1

1.1. Demuestre que si A_1 y A_2 son dos conjuntos convexos, entonces $A_1 \cap A_2$ tambien lo es.

Para demostrar que la intersección de dos conjuntos convexos A_1 y A_2 también es convexa, debemos mostrar que para cualquier par de puntos x, y en $A_1 \cap A_2$ y cualquier $t \in [0, 1]$, el punto $tx + (1-t)y$ también está en $A_1 \cap A_2$.
Sea $x, y \in A_1 \cap A_2$ y sea $t \in [0, 1]$ un escalar arbitrario.
Como $x, y \in A_1$, sabemos que $tx + (1-t)y \in A_1$ por la convexidad de A_1.
De manera similar, como $x, y \in A_2$, sabemos que $tx + (1-t)y \in A_2$ por la convexidad de A_2.
Entonces, como $tx + (1-t)y$ pertenece a ambos conjuntos A_1 y A_2, se sigue que $tx + (1-t)y$ pertenece a su intersección $A_1 \cap A_2$. Por lo tanto, $A_1 \cap A_2$ es convexo, como se quería demostrar. □

1.2. Demuestre que si $\{A_i\}_{i \in \mathbb{N}} \in \mathbb{N}$ es una familia de conjuntos convexos, entonces $\bigcap_{i \in \mathbb{N}}$ tambien lo es.

Como A_i es un conjunto convexo para cada $i \in \mathbf{N}$, tenemos que :
$x \in A_i$, $y \in A_i \Rightarrow tx + (1-t)y \in A_i, t \in [0, 1]$.
Sean dos puntos $a, b \in \bigcap_{i \in \mathbf{N}} A_i$, sabemos entonces que $a \in A_i$ y $b \in A_i$ $\forall i \in \mathbf{N}$.
Lo que nos permite afirmar que $ta + (1-t)b \in A_i$ con $t \in [0, 1], \forall i \in \mathbf{N}$.
O de forma equivalente, $ta + (1-t)b \in \bigcap_{i \in \mathbf{N}} A_i$.
Como a, b son dos puntos cualesquiera en $\bigcap_{i \in \mathbf{N}} A_i$ concluimos que el mismo es convexo. □

2. Ejercicio 2

2.1. Implemente el metodo de descenso por gradiente con direccion de maximo descenso y paso fijo, segun especificaciones en el notebook.

El método de descenso por gradiente con dirección de máximo descenso y paso fijo es un algoritmo utilizado para encontrar el mínimo local de una función convexa. Se utiliza principalmente en el aprendizaje automático y en la optimización de problemas.

Implementacion:

Dado un punto inicial **x_init**, el algoritmo se mueve en la dirección opuesta al gradiente de la función en ese punto, con una longitud de paso fija **alpha**. Este proceso se repite hasta que se alcanza una condición de parada, que en este caso es que la norma del gradiente sea menor que una tolerancia establecida **tol**.

La funcion recibe los siguientes argumentos:

- **grad**: la función que implementa el gradiente de la función a optimizar.
- **x_init**: el punto inicial de la búsqueda.
- **xstar**: el mínimo conocido de la función.
- **alpha**: el tamaño fijo del paso en cada iteración.
- **tol**: la tolerancia para la condición de parada (por defecto, 1e-5).

La función devuelve los siguientes valores:

- **x**: el punto final alcanzado.
- **it**: la cantidad de iteraciones realizadas.
- **xs**: la trayectoria de los puntos x.
- **es**: las distancias de los puntos x al mínimo conocido xstar.

La función realiza las siguientes operaciones:

1. Inicializa el punto actual **x** con el punto inicial **x_init**.
2. Inicializa una lista **xs** con el punto inicial **x_init** y una lista **es** con la distancia entre **x_init** y el mínimo conocido **xstar**.
3. Inicia un bucle while que se ejecutará hasta que se alcance la condición de parada ($\mathbf{np.linalg.norm}(\mathbf{grad}(\mathbf{x})) > \mathbf{tol}$).
4. En cada iteración, actualiza el punto actual **x** mediante la formula $\mathbf{x} = \mathbf{x} - \mathbf{alpha} * \mathbf{grad}(\mathbf{x})$.
5. Agrega el punto actual **x** a la lista **xs** y la distancia entre **x** y el mínimo conocido **xstar** a la lista **es**.
6. Incrementa la cantidad de iteraciones it.
7. Devuelve el punto final **x**, la cantidad de iteraciones it, la trayectoria de los puntos **xs** y las distancias de los puntos es al mínimo conocido **xstar**.

2.2. Implemente un metodo de descenso por gradiente acelerado (Nesterov), segun especificaciones en el notebook.

El método de descenso por gradiente acelerado (Nesterov) es una técnica de optimización que se utiliza para minimizar una función objetivo diferenciable. Este método es una extensión del método de descenso por gradiente clásico, y utiliza la información de la dirección del

gradiente para acelerar la convergencia del algoritmo.

Implementacion:

En esta implementación, inicializamos **y** y **x** en **x_init** y definimos una lista para almacenar la trayectoria de los puntos **xs** y otra lista para almacenar la distancia de cada punto a **xstar (es)**. Luego, comenzamos un ciclo while que se repetirá hasta que la norma del gradiente en **x** sea menor o igual que **tol**.

Dentro del ciclo while, primero calculamos el siguiente punto aproximado **x_next** utilizando el gradiente de y (la aproximación de segundo orden del punto **x**). Luego, actualizamos **y** y **t** utilizando **x_next**, **x** y **t**. Después, actualizamos **x**, **t**, **it**, **xs** y **es** para registrar el nuevo punto **x** en la trayectoria, la nueva distancia a **xstar**, el número de iteraciones y el nuevo valor de **t**.

Finalmente, devolvemos el último punto **x**, el número de iteraciones realizadas, la trayectoria completa de puntos **xs** y la distancia a **xstar** de cada punto **es**.

2.3. Pruebe los dos métodos implementados en las dos funciones incluidas

2.3.1. Experimente con los tamaños de paso

Para la funcion $f(x) = 1/2||Ax - b||^2$
Como $f \in C^2$ es convexa, podemos intentar calcular el α_{optimo}.
Utilizando la fórmula: $\alpha_{opt} = \frac{2}{M+m}$ donde M y m son el máximo y mínimo de los valores propios de la $Hessiana$ de f, se obtuvo:

$$\alpha_{optimo} \approx 0{,}0002093$$

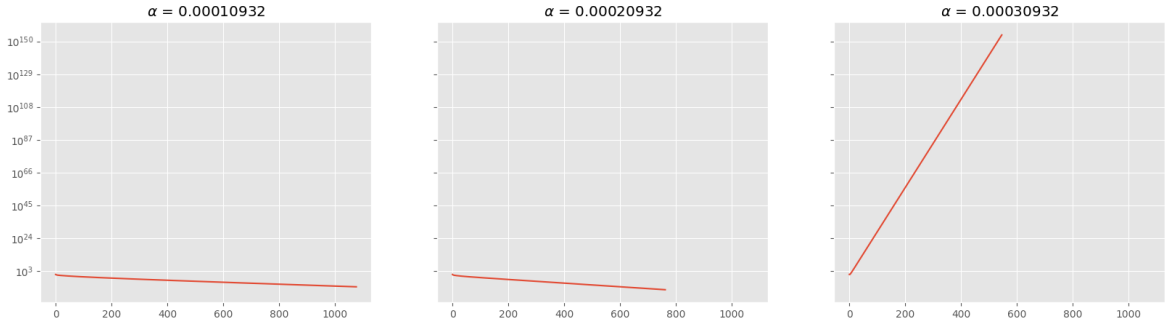

Figura 2: α_{optimo} y límites del entorno $\pm$ ϵ, con ϵ = 0,0001 Para el método de $GradientDescent$ sobre la función $f1$

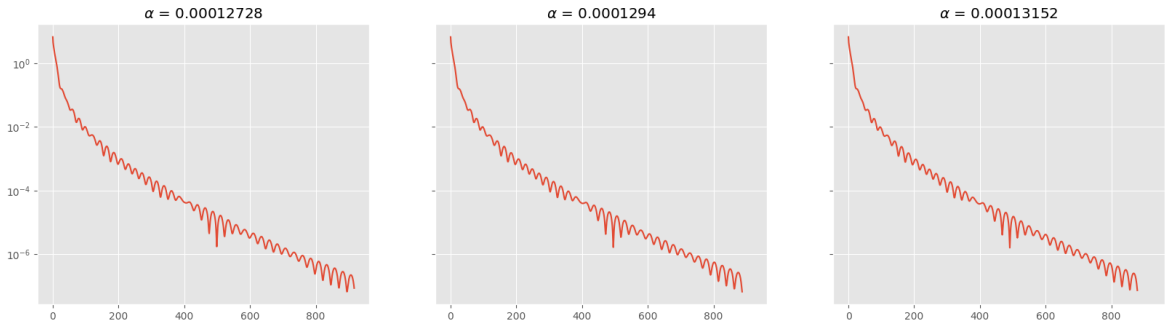

Figura 3: α estimado y límites de un entorno $\pm$ ϵ Para el método de *NesterovGradientDescent* sobre la función $f1$

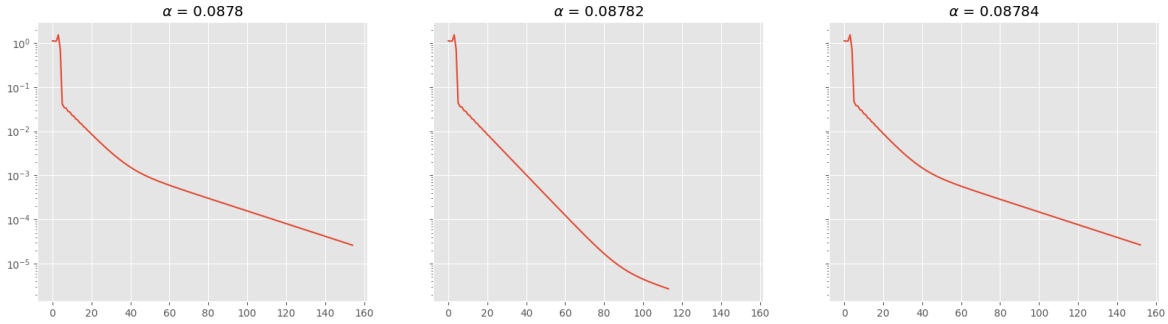

Figura 4: α estimado y límites de un entorno $\pm\epsilon$ Para el método de *GradientDescent* sobre la función *Rosenbrock*

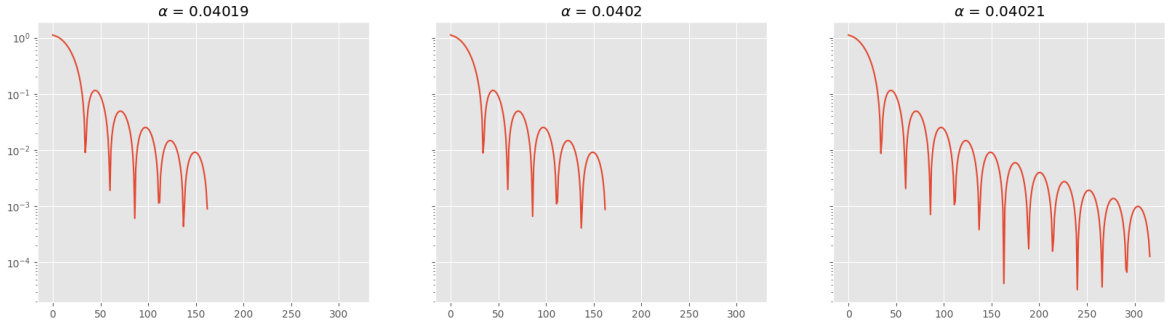

Figura 5: α estimado y límites de un entorno $\pm$ ϵ Para el método de *NesterovGradientDescent* sobre la función *Rosenbrock*

2.3.2. Compare los metodos en terminos de cantidad de iteraciones necesarias y tiempo de ejecucion

Para comparar los metodos en terminos de cantidad de iteraciones necesarias y tiempo de ejecucion es importante utilizar los mismos parametros (por ej.: **alpha**) para ambos metodos.

Primero vamos a comparar los metodos aplicados a la funcion $||Ax - b||^2$ con

$alpha = 0{,}0001$**.**

- Metodo de descenso por gradiente con direccion de maximo descenso y paso fijo:

Iteraciones: 1178
Tiempo de ejecucion: $105ms \pm 35{,}1ms$

- Metodo de descenso por gradiente acelerado (Nesterov):

Iteraciones: 1092
Tiempo de ejecucion: $114ms \pm 47{,}3ms$

Ahora comparemos los metodos aplicados a la funcion de *Rosenbrock* **con** $alpha = 0{,}008$**.**

- Metodo de descenso por gradiente con direccion de maximo descenso y paso fijo:

Iteraciones: 3760
Tiempo de ejecucion: $99{,}6ms \pm 12ms$

- Metodo de descenso por gradiente acelerado (Nesterov):

Iteraciones: 713
Tiempo de ejecucion: $22{,}1ms \pm 4{,}81ms$

Conclusion: Analizando los resultados obtenidos podemos concluir que si bien el metodo de descenso por gradiente acelerado (Nesterov) converge mas rapido (requiere menos iteraciones), debido a la necesidad de realizar mas calculos, cada iteracion es mas costosa en terminos de tiempo de ejecucion que el metodo de descenso por gradiente con direccion de maximo descenso y paso fijo.

2.3.3. Grafique la funcion de error $||x_k - x^*||$ en funcion de las iteraciones. Utilice escala logaritmica en el eje vertical

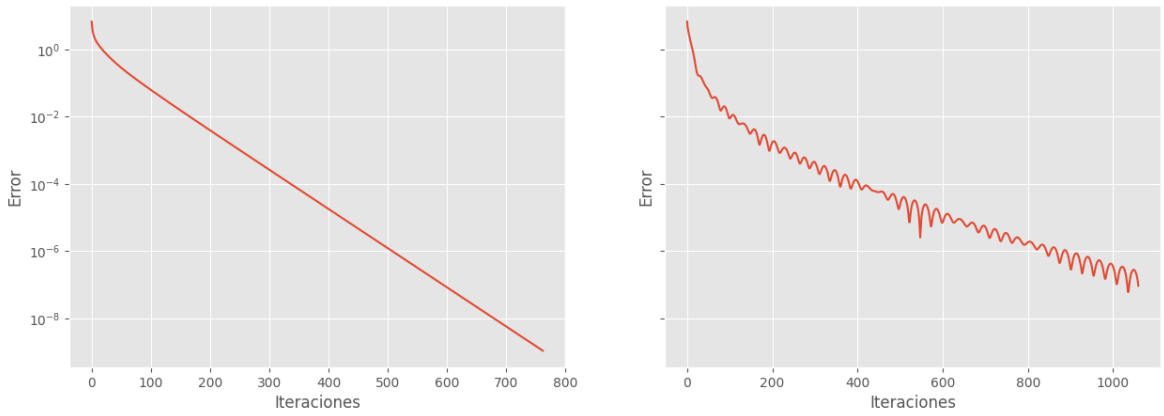

Figura 6: $||Ax - b||^2$

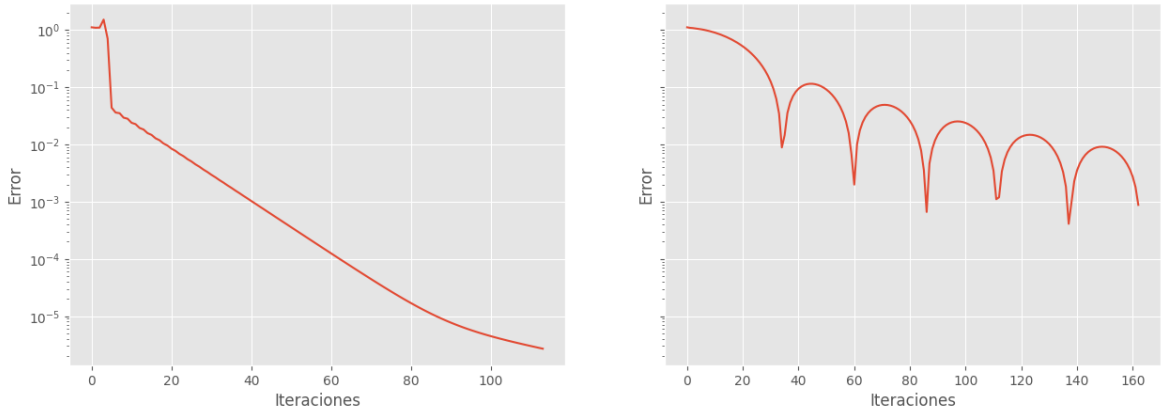

Figura 7: Rosenbrock

2.3.4. Los metodos son de descenso?

Decimos que un método es de descneso si $f(x_{k+1}) \leq f(x_k) \forall k$.

Podemos entonces mirar que sucede al graficar las imágenes de las trayectorias obtenidas por cada método en cada función.

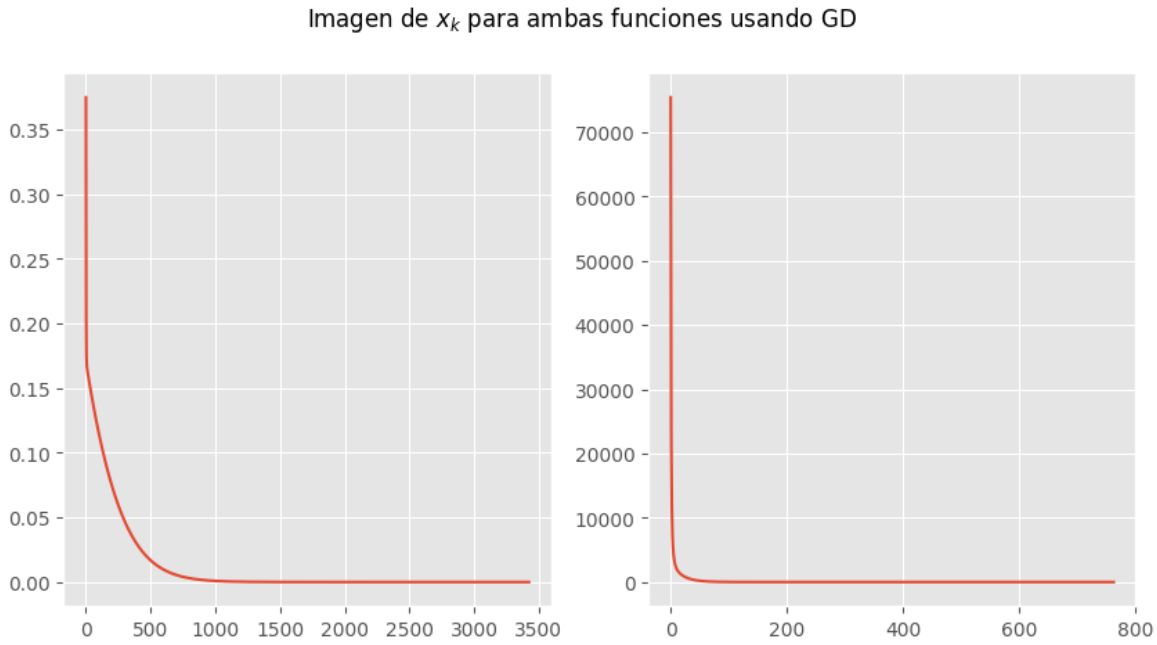

Figura 8: $Img(x_k)$, a la izquierda *Rosenbrock* y a la derecha $f1$

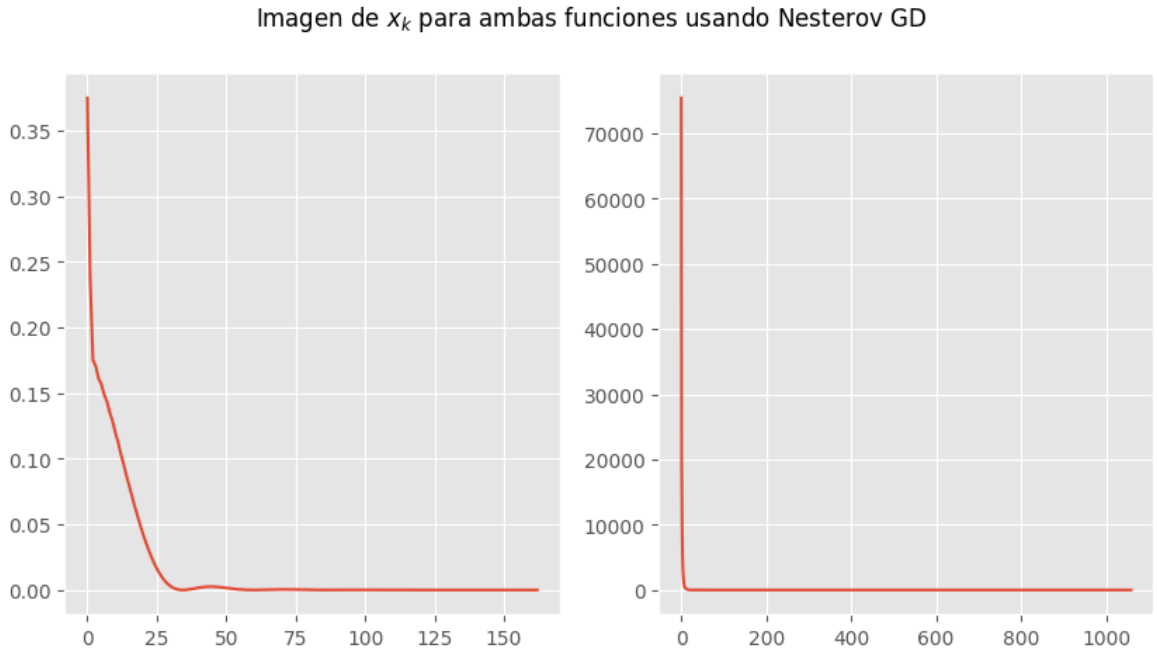

Figura 9: $Img(x_k)$, a la izquierda *Rosenbrock* y a la derecha $f1$

Por un lado vemos que el método de descenso de gradiente, para ambas funciones, devuelve una trayectoria cuya imagen forma un gráfico monótono decreciente por lo que podemos entonces concluir que es un método de descenso.
Por otro lado, para el caso del método Nesterov si bien se puede apreciar resultados similares existe una leve subida entre 25 y 50 para la función de Rosenbrock, por lo que podríamos

decir que no es de descenso.

2.3.5. Superpuesto a la curva de nivel de la funcion Rosenbrock, grafique las trayectorias de ambos metodos

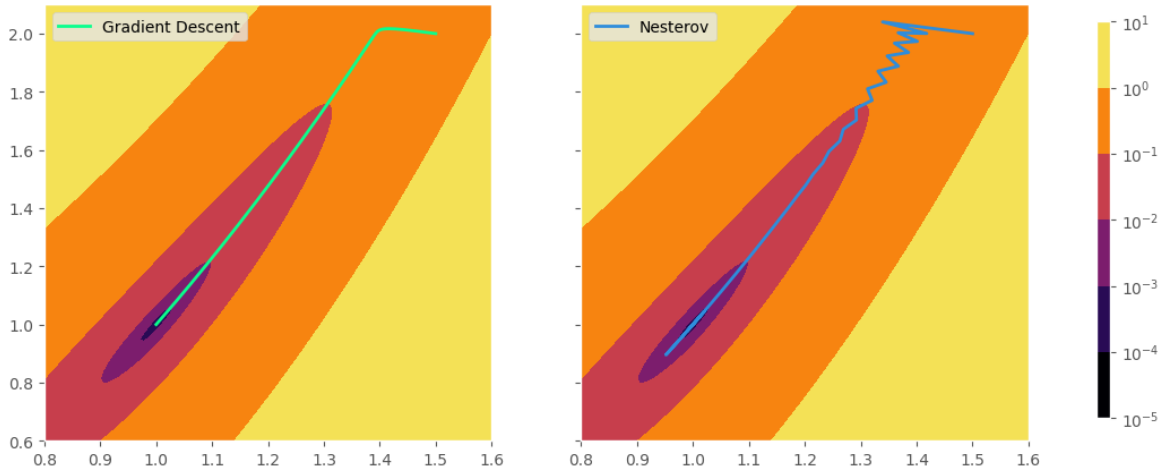

Figura 10: Trayectorias sobre curvas de nivel

Se observa como la trayectoria obtenida por el método de Nesterov sigue un patrón de "zigzagueo.ª diferencia del devuelto por el método clásico, además podemos ver como el primero se "pasa"del óptimo y luego se corrige. Esto es debido al enfoque que toma el método de Nesterov, con el fin de aumentar la velocidad de convergencia este método utiliza una especie de impulso o "momentum", lo que le permite estimar la dirección del próximo punto a lo largo de la trayectoria de descenso.

Esto es de utilidad para evitar que el método quede atrapado en valles estrechos y reducir el tiempo necesario para escapar de mínimos locales, algo de utilidad al trabajar con funciones no convexas.

2.4. Modfique el metodo de descenso por gradiente (en una nueva funcion) implementando el paso decreciente. Pruebe el metodo en alguna de las funciones y comente.

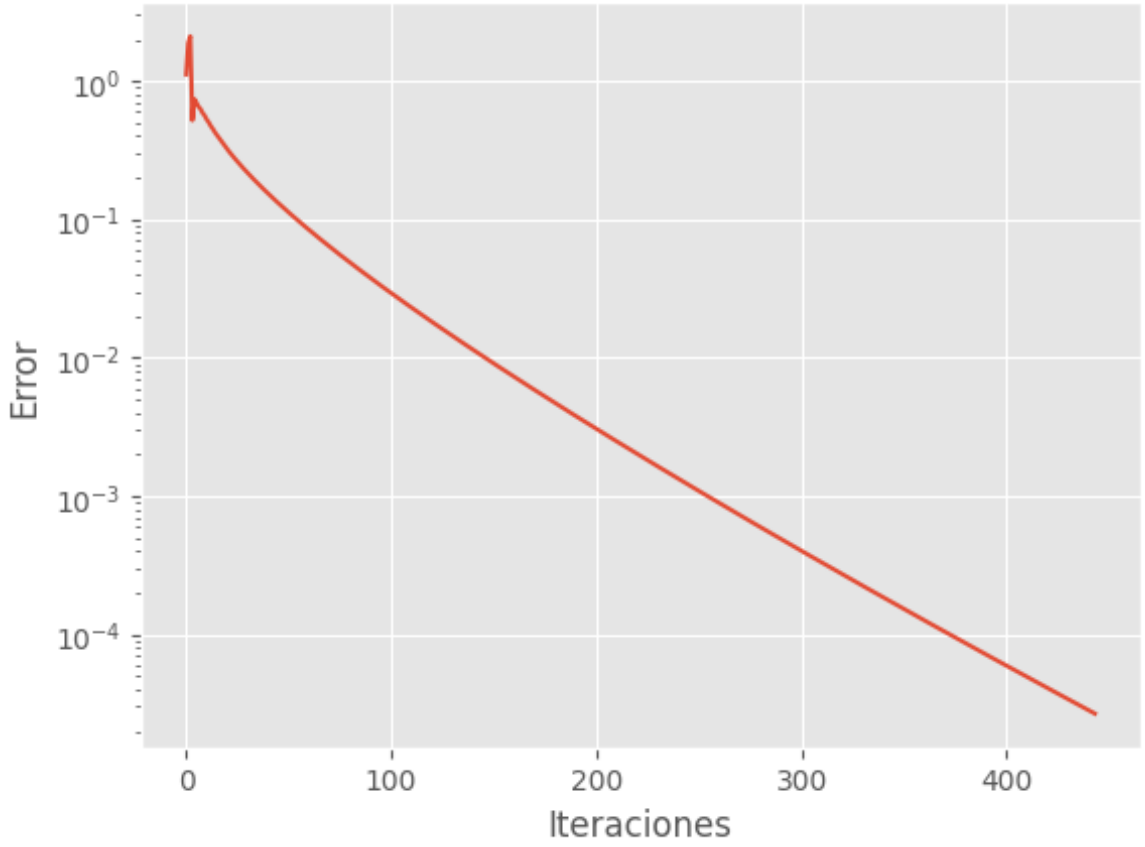

Figura 11: Metodo de descenso por gradiente con paso decreciente probado en la funcion de rosenbrock

En el metodo de descenso por gradiente con paso descendente, la idea es que el tamaño del paso se vaya reduciendo a medida que se realizan más iteraciones, para que el método pueda converger más rápido.

El parámetro **grad** representa la función gradiente que se desea minimizar, **x_init** es el punto inicial desde el cual se inicia la búsqueda, y **xstar** es el punto óptimo que se busca encontrar.

La condición de parada se establece en base a la norma del gradiente, que debe ser menor que una tolerancia establecida. Además, la función devuelve el punto **x** alcanzado, la cantidad de iteraciones realizadas, un array con la trayectoria de los puntos **x_k** y otro array con las distancias de los puntos **x_k** al punto óptimo **xstar**.

Se observa que la función logarítmica funciona bien para este problema, ya que converge rápidamente.

2.5. Modfique el metodo de descenso por gradiente (en una nueva funcion) para corregir la direccion de descenso utilizando el diagonal scaling. Utilizando paso fijo 2, pruebe el metodo con la funcion de Rosenbrock, compare con el metodo implementado en a) y comente.

La función **gradient_descent_diagonal_scaling** implementa el método de **Descenso de Gradiente con escalamiento diagonal**. En este método, se usa una matriz diagonal para escalar el gradiente en cada iteración, lo que puede mejorar la convergencia del método.

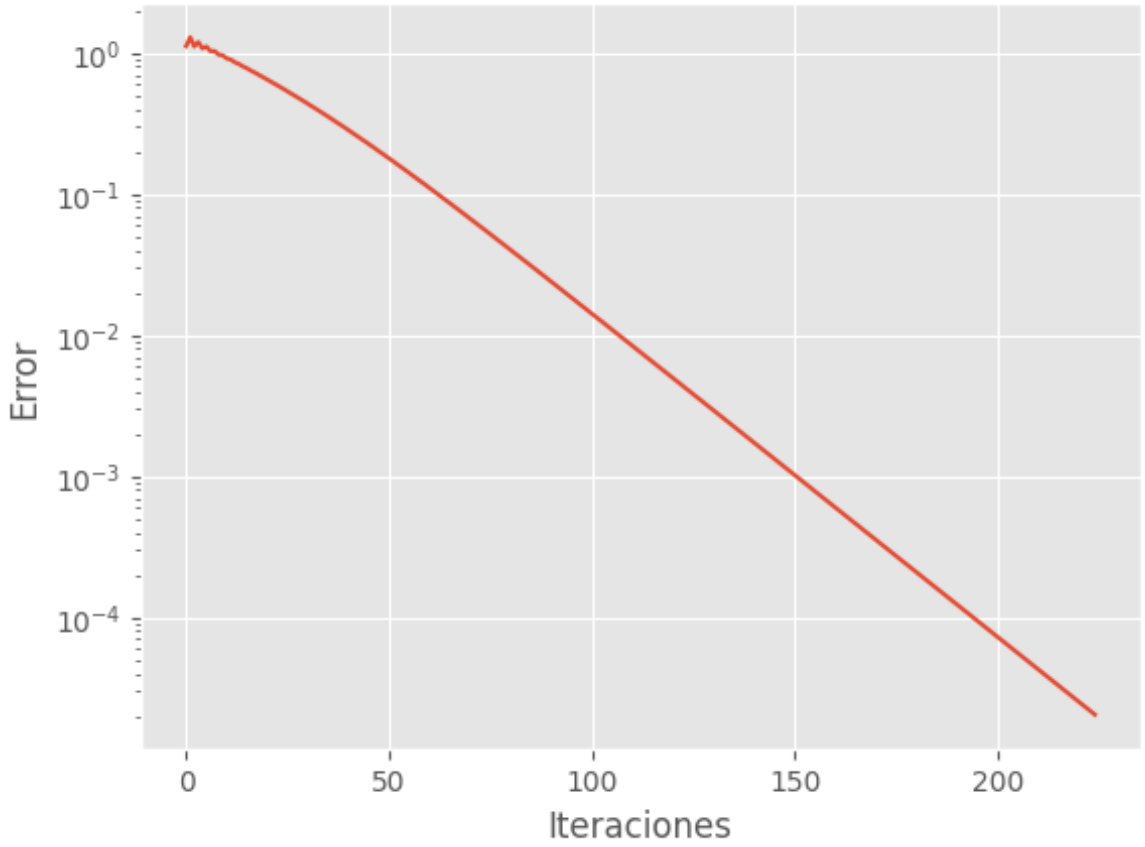

Figura 12: Metodo de descenso por gradiente con diagonal scaling probado en la funcion de rosenbrock

Implementacion:

La función toma como entrada:

1. **grad**: la función gradiente que se quiere minimizar.
2. **diag_scaling**: la función que devuelve una matriz diagonal que se usará para escalar el gradiente.
3. **x_init**: el punto inicial de la búsqueda.

4. **xstar**: el punto óptimo que se quiere encontrar.
5. **alpha**: el tamaño de paso fijo que se usará en cada iteración.
6. **tol**: la tolerancia para la norma del gradiente, que se usará como criterio de parada.

La función comienza inicializando el punto de partida **x** con **x_init**, la cantidad de iteraciones **it** en cero, y una lista **xs** para almacenar los valores de **x** en cada iteración. También se inicializa la lista **es** con la distancia euclidiana entre **x** y **xstar**.

Luego, se inicia un bucle while que se ejecutará mientras la norma del gradiente en el punto **x** sea mayor que la tolerancia **tol**. Dentro del bucle, se incrementa el contador de iteraciones **it**, se calcula la dirección de descenso **dk** multiplicando el gradiente por la matriz diagonal **diag_scaling(x)**, y se actualiza **x** usando el tamaño de paso fijo **alpha**.

Después de actualizar **x**, se agrega su valor a la lista **xs** y se calcula la distancia euclidiana entre **x** y **xstar**, que se agrega a la lista **es**.

Finalmente, la función devuelve el punto final **x**, la cantidad de iteraciones **it**, la lista de todos los valores de x a lo largo de las iteraciones **xs**, y la lista de las distancias euclidianas entre cada x y xstar a lo largo de las iteraciones **es**.