Minimización por el método de descenso del gradiente

Ana Toxqui, *MMOP, CIMAT,*

***Abstract*—El método de descenso por gradiente es un método general de optimización para minimizar funciones, en este trabajo se muestra una implementación en python de dicho método y se compara con la función de numpy.**

***Index Terms*—Descenso por gradiente, Optimización, Minimización, Métodos iterativos.**

I. Introducción

EL método de descenso por gradiente en el campo de la optimización matemática es una herramienta ampliamente usada para encontrar los mínimos locales de funciones. Este método tiene como principio usar el gradiente de la función a optimizar para calcular de manera iterativa nuevas posiciones dentro del rango de la función de tal forma que en cada paso se aproxima al mínimo local. En este trabajo, se ha implementado el método de descenso por gradiente en Python por medio de una función y se ha comparado su desempeño en comparación con la función de descenso del gradiente de la paquetería de Numpy mediante cuatro funciones distintas.

El método de descenso por gradiente implementado usa el algoritmo Backtraking Line Search. Las funciones de prueba tiene distintos tipos de comportamiento, lo que permite comparar la eficiencia tanto del método como de la implementación computacional.

El trabajo se compone de las secciones: Marco teórico, Implementación del Método de descenso del Gradiente, Resultados y Conclusiones. En la primera sección se desarrollan los fundamentos del método y su importancia, en la sección siguiente se explica los pormenores de la implementación en Python, se explica el algoritmo Backtraking Line Search y como se actualizan las posiciones para alcanzar el mínimo, así como las funciones utilizadas y sus , posteriormente se presentan los resultados del método implementado comparados con la función de Numpy para el método, finalmente se presentan las conclusiones obtenidas.

II. Marco teórico

El método de descenso por gradiente es una técnica clásica usada para la encontrar los mínimos y máximos locales de una función, tiene la ventaja de poder usarse para funciones de dimensión n arbitraria. El problema de encontrar los mínimos y máximos es importante en diferentes áreas del conocimiento como la ciencia de datos e ingeniería así como para la solución de algunos problemas industriales.

El método se basa en la idea de seguir la dirección del gradiente de la función en cada iteración para acercarse al mínimo local, donde la pendiente es aproximadamente cero. En el contexto de la optimización numérica, el gradiente es un vector que apunta en la dirección de máxima tasa de cambio de la función en un punto dado. El negativo del gradiente señala la dirección en la que la función disminuye más rápidamente por lo que con esto se puede aproximar al punto máximo o mínimo dependiendo el signo del gradiente.

El método comienza con un punto de inicio y se mueve en pequeños pasos en la dirección opuesta al gradiente. La magnitud de cada paso está determinada por medio del método de Backtraking Line Search, el cual tiene como salida un coeficiente que multiplica al gradiente de la función en ese punto, este número controla la velocidad de convergencia.

Para aplicar el método de descenso del gradiente es necesario calcular el gradiente de la función en relación con cada variable de entrada. En funciones multivariadas, esto implica calcular las derivadas parciales de la función con respecto a cada variable. El gradiente resultante es un vector que indica la dirección y la magnitud de mayor aumento.°

III. Implementación del Método de descenso por Gradiente

La función implementada en Python se puede explicar mediante los siguientes pasos: inicialización, paso iterativo, backtraking line search, acatualización y convergencia, salida. A continuación se explica cada uno de estos pasos.

Para la inicialización se necesitan como argumentos, un punto de partida, un número α, un número n de iteraciones del algoritmo, dos factores c y p para el bactraking line search y un número pequeño ϵ que será el umbral de convergencia para la norma del gradiente. Durante cada iteración, se evalúa la función objetivo en el punto actual, obteniendo el valor fk y el gradiente $gradientefk$. Luego, se calcula la dirección de búsqueda pk como la negación del gradiente normalizado. α se ajusta mediante el backtraking line search para garantizar una disminución suficiente en el valor de la función.Después de determinar la tasa de aprendizaje adecuada α, se actualiza la posición x utilizando la dirección pk ponderada por alpha. Se verifica si la norma del gradiente al cuadrado es menor que epsilon, lo que indica convergencia, y si es así, el algoritmo se detiene.

La salida de la función será el punto de convergencia y el valor de la función en ese punto. A continuación se muestra el código de la función en Python.

```
def DesGradiente(x0, alpha0, n, funcion,
c, p, epsilon):
    x = x0
    for k in range(n):
        fk, gradientefk = funcion(x)
        pk = -(gradientefk/np.linalg.norm
            (gradientefk)**2)
        alpha = alpha0
        while funcion(x + (alpha*pk))[0]
            > (fk + (c*alpha*
            np.dot(gradientefk,pk))):
                alpha = p*alpha
        alpha0 = alpha
        x = x + (alpha0*pk)
        if np.linalg.norm(gradientefk)**2
            < epsilon:
                break
    return x, funcion(x)[0]
```

Esta implementación proporciona una herramienta valiosa para optimizar funciones en diversas aplicaciones. Al ajustar parámetros de la función, el algoritmo puede adaptarse a diferentes tipos de funciones y problemas de optimización.

A. Funciones de Prueba

A continuación se presentan las cuatro funciones de prueba con las que se pretende dar una idea sobre los alcances de la función para el Método de descenso del Gradiente presentada anteriormente, para este trabajo se seleccionaron tres funciones de dimensión dos y una que puede tomar una dimensión arbitraria n.

Función de Himmelblau
Función objetivo:

$$f(x,y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2$$

Mínimo Global:

$$\text{Min} = \begin{cases} f(3.0, 2.0) & = 0.0 \\ f(-2.805118, 3.131312) & = 0.0 \\ f(-3.779310, -3.283186) & = 0.0 \\ f(3.584428, -1.848126) & = 0.0 \end{cases}$$

Dominio de búsqueda:

$$-5 \leq x, y \leq 5$$

Función McCormick
Función objetivo:

$$f(x,y) = \sin(x+y) + (x-y)^2 - 1.5x + 2.5y + 1$$

Mínimo Global:

$$f(-0.54719, -1.54719) = -1.9133$$

Dominio de búsqueda:

$$-1.5 \leq x \leq 4$$
$$-3 \leq y \leq 4$$

Función Booth
Función objetivo:

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Mínimo Global:

$$f(1,3) = 0$$

Dominio de búsqueda:

$$-10 \leq x, y \leq 10$$

Función Esférica
Función objetivo:

$$f(x) = \sum_{i=1}^{n} x_i^2$$

Mínimo Global:

$$f\left(x_1, \ldots, x_n\right) = f(0, \ldots, 0) = 0$$

Dominio de búsqueda:

$$-\infty \leq x_i \leq \infty$$
$$1 \leq i \leq n$$

IV. Resultados

Los resultados obtenidos para la función de Himmelblau se presentan en la Tabla I. Se muestran los valores de los puntos y las respectivas evaluaciones de la función objetivo para diferentes cantidades de iteraciones utilizando la función creada y el método de optimización con la librería numpy.

Función Himmekblau	Función creada							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-5,5)	[-4.98 4.985]	520.006	[-4.82 4.85]	430.068	[-3.47 3.66]	30.87	[-2.80 3.13]	1.43e-11
(1,4)	[1.002 3.93]	126.04	[1.61 2.76]	36.72	[2.99 1.99]	2.21e-10	[2.99 1.99]	2.21e-10
(-1,-3)	[-1.30 3.02]	39.92	[-2.80 3.13]	1.72e-10	[-2.80 3.13]	1.72e-10	[-2.80 3.13]	1.72e-10
(4,-5)	[3.99 -4.97]	474.007	[3.97 -4.75]	384.08	[3.58 -1.84]	7.70e-10	[3.58 -1.84]	7.70e-10
Función Himmekblau	Función con numpy							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-5,5)	[-2.83 3.14]	0.04545	[-2.80 3.13]	7.88e-31	[-2.80 3.13]	7.88e-31	[-2.80 3.13]	7.88e-31
(1,4)	[2.9 2.17]	0.55	[3, 2]	1.05e-21	[3, 2]	6.31e-30	[3, 2]	6.31e-30
(-1,-3)	[-2.79 3.13]	0.015	[-2.80 3.13]	7.88e-31	[-2.80 3.13]	7.88e-31	[-2.80 3.13]	7.88e-31
(4,-5)	[3.57 -1.70]	0.26	[3.58 -1.84]	1.77	[3.58 -1.84]	0	[3.58 -1.84]	0

TABLE I
Resultados obtenidos para la función de Himmekblau

En la tabla II se dan los resultados para la función de McCormick, mientras que los resultados obtenidos para la función de Beale se presentan en la Tabla III.

Función McCormick	Función creada							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-1.7, 3.4)	[-1.27 2.94]	29.12	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
(0, 1.5)	[0.08 1.38]	7.001	[-0.39 -1.39]	-1.86	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
(2, -2.2)	[1.90 -2.14]	8.94	[0.60 -1.50]	-0.01	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
(3.5, -3)	[3.44 -2.96]	29.98	[2.94 -2.58]	20.98	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
Función McCormick	Función con numpy							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-1.7, 3.4)	[-0.80 2.42]	19.68	[0.44 -0.39]	0.096	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
(0, 1.5)	[0.32 1.06]	4.67	[0.17 -0.75]	-0.83	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
(2, -2.2)	[1.37 -1.83]	4.24	[-0.33 -1.41]	-1.85	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91
(3.5, -3)	[2.52 -2.28]	14.89	[-0.20 -1.34]	-1.75	[-0.54 -1.54]	-1.91	[-0.54 -1.54]	-1.91

TABLE II
Resultados obtenidos para la función de McCormick

Función Boot	Función creada							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-10, -10)	[-9.76 -9.76]	2494.09	[-7.43 -7.38]	1595.21	[1 2.9]	2.06e-08	[1.0 2.9]	2.06e-08
(10, 10)	[9.64 9.65]	1054.22	[4.85 5.06]	158.94	[1.0 2.9]	5.15e-08	[1.0 2.9]	5.15e-08
(-10. 10)	[-7.71 10.35]	137.38	[0.99 3]	8.21e-08	[0.99 3]	8.21e-08	[0.99 3]	8.21e-08
(5, 3.2)	[2.24 1.62]	3.50	[1 2.9]	3.37e-08	[1 2.9]	3.37e-08	[1 2.9]	3.37e-08
Función Boot	Función con numpy							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-10, -10)	[-0.21 0.11]	76.98	[1.16 2.83]	0.052	[1.0 2.99]	2.58e-08	[1.0 2.99]	3.34e-16
(10, 10)	[3.20 3.53]	34.99	[1.16 2.83]	0.052	[1.0 2.99]	2.58e-08	[1.0 2.99]	3.34e-16
(-10. 10)	[-6.84 10.16]	114.75	[-0.46 4.46]	4.28	[0.9 3.0]	2.09e-06	[0.9 3.0]	2.7e-14
(5, 3.2)	[2.94 1.77]	7.33	[1.30 2.69]	0.19	[1.0 2.99]	9.33e-08	[1.0 2.99]	1.2e-15

TABLE III

RESULTADOS OBTENIDOS PARA LA FUNCIÓN DE BOOT

Finalmente la tabla IV presenta los resultados para la función esférica, para esta función se consideraron puntos en dimensión dos, tres, cuatro y cinco.

Función Esférica	Función creada							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-100, 50)	[-95.92 47.96]	11502.07	[-45.074 22.53]	2539.84	[-1.65e-04 -8.26e-05]	1.41e-08	[-1.65e-04 -8.26e-05]	1.41e-08
(2, 5, 22)	[-0.0008 -0.002 -0.009]	4.78e-05	[-8.92e-06 -1.74e-05 -7.21e-05]	6.14e-09	[-8.92e-06 -1.74e-05 -7.21e-05]	6.14e-09	[-8.92e-06 -1.74e-05 -7.21e-05]	6.14e-09
(-10, 50, -2, 0)	[-3.87 -.39 38 -1.57 0]	1615.38	[-2.2e-05 -1.1e-04 -4.4e-06 0]	1.26e-09	[-2.2e-05 -1.1e-04 -4.4e-06 0]	1.26e-09	[-2.2e-05 -1.1e-04 -4.4e-06 0]	1.26e-09
(-20, -20, 20, -20 -20)	[-14.29 -18.24 19.35 -14.28 14.28]	1016.62	[-3.1e-05 2.1e-05 -1.1e-05 2.1e-05 -2.1e-05]	2.26e-09	[-3.1e-05 2.1e-05 -1.1e-05 2.1e-05 -2.1e-05]	2.26e-09	[-2.1e-05 2.1e-05 -1.1e-05 2.1e-05 -2.1e-05]	2.26e-09
Función Esférica	Función con numpy							
Punto/ iteraciones	10		100		500		1000	
	x	f(x)	x	f(x)	x	f(x)	x	f(x)
(-100, 50)	[-83.38 47.49]	9082.481	[-16.26 8.13]	330.52	[-0.01 0.005]	0.00016	[-1.26e-06 6.46e-05]	1.08e-12
(2, 5, 22)	[1.68 4.16 18.34]	356.75	[0.32 3.57]	13.56	[0.0001 0.0005 0]	6.63e-08	[2.58e-08 6.46e-08 2.8e-07]	4.57e-14
(-10, 50, -2, 0)	[-8.33 41.69 -1.66 0]	1810.8	[-1.62 8.11 -0.24 0]	68.85	[-0.001 0.005 -0.0003 0]	1.304e-05	[-1.25e-07 6.46e-05 -2.58e-08 0]	4.53e-13
(-20, -20, 20, -20 -20)	[-16.67 -20.67 18.67 -18.67 16.67]	1096.78	[-2.9 -3.25 3.25 -3.25 2.9]	49.88	[-0.008 -0.002 0.002 -0.002 0.001]	2.94e-05	[-2.5e-07 -2.3e-07 2.5e-07 -2.5e-07 2.5e-07]	4.34e-15

TABLE IV

RESULTADOS OBTENIDOS PARA LA FUNCIÓN DE ESFÉRICA

V. ANÁLISIS Y CONCLUSIÓN

Para la función de Himmelblau, se observa que tanto la función creada como el método implementado de la librería numpy logran encontrar valores cercanos al mínimo global (-3.0, 2.0) después de un número moderado de iteraciones. A medida que aumenta el número de iteraciones, ambos métodos convergen cada vez más hacia el mínimo global, pero a un ritmo ligeramente diferente.

La función de McCormick presenta presenta diferentes comportamientos dependiendo del punto de inicio debido a sus múltiples mínimos locales. En este caso, ambas funciones lograron encontrar valores que se acercan al mínimo global (-0.54, -1.54). Es interesante notar que en las primeras iteraciones, la función creada tiende a explorar más alrededor del punto inicial antes de converger hacia la región del mínimo global. Esto se debe a los parámetros de entrada que se le dan a la función. Lo mismo sucede para la función de Boot

En el caso de la función esférica se muestra que el método es escalable a diferentes dimensiones, ya que en todos los puntos de inicio proporcionados con un ajuste de parámetros adecuado se logra converger al mínimo global en cada punto.

En general la función implementada es buena para el problema de encontrar mínimos de una función sin embargo depende mucho de los parámetros de entrada que se le asignen, en comparación con la función ya implementada en numpy la velocidad de convergencia y eficacia varían, ya que para algunos parámetros de nuestra función la convergencia se alcanza más rápido.