Tarea 4: Gradiente descendente

Alumno: Ricardo De León

Realice código en Python que, recibiendo una función f dada, un valor inicial x_0 y una exactitud (error) dado E, use el método de gradiente descendente (ascendente) para encontrar un mínimo (máximo) local de f. Asegúrese que cuenta el número de iteraciones realizadas.

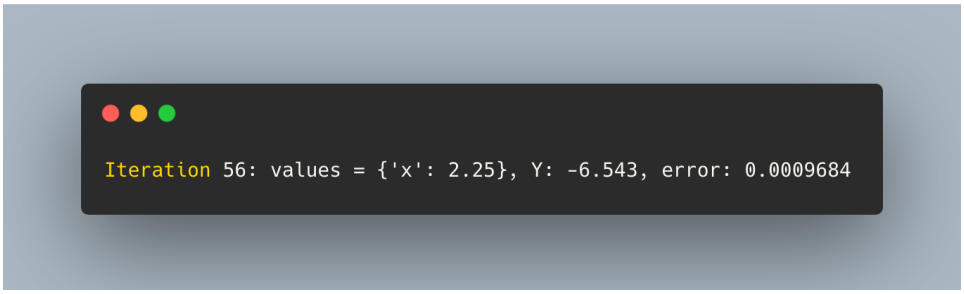
Use su código para encontrar (si existe) los mínimos y máximos locales de cada función con precisión de **4 cifras significativas**. y exactitud de 10^{-3}. En todos los casos indique el(los) valor(es) inicial(es) que utilizó y el número de iteraciones que fueron necesarias para alcanzar la respuesta.

1. $f(x) = x^4 - 3x^3 + 2$
2. $f(x) = 5x^6 + 21x^5 - 180x^4 + 115x^3 + 750x^2 - 1260x + 10$
3. $f(x, y) = x^2 - 24x + y^2 - 10y$
4. $f(x, y) = xy + \frac{1}{x} + \frac{1}{y}$
5. $f(x, y) = \sin x + \sin y + \sin(x + y), 0 \leq x \leq 2\pi, 0 \leq y \leq 2\pi$
6. $f(x, y, z) = x^2 + y^2 + z^2 + 1$
7. $f(x, y, z) = 3x^2 + 4y^2 + z^2 - 9xyz$
8. $f(x, y, z) = x^4 + y^4 + z^4 + xyz$

1.-

Entrada para gradiente descendente

Mínimo local:

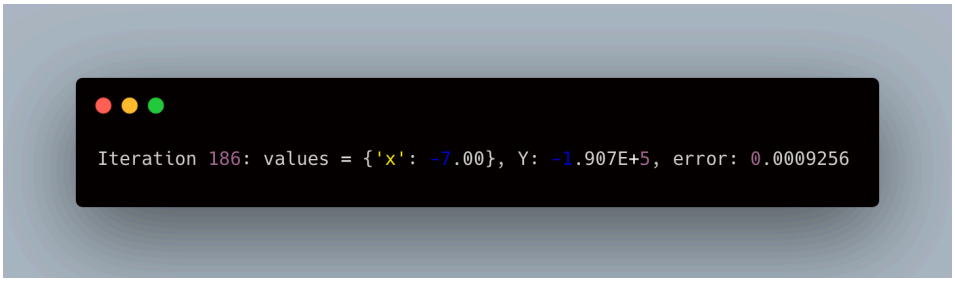


Tarea 4: Gradiente descendente

2.-

Entrada para gradiente descendente con un alpha = .000001

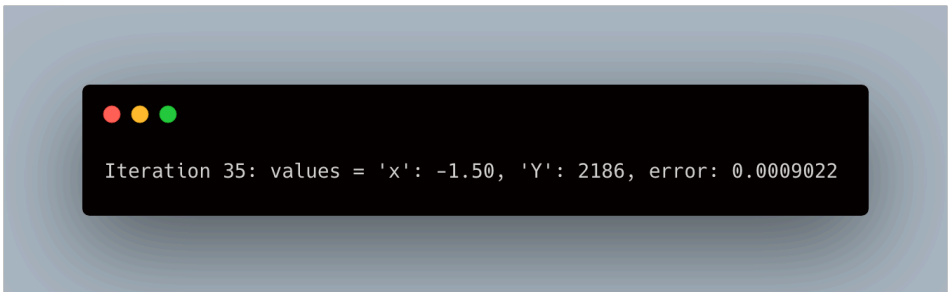
Descendente:



Ascendente 1: Entrada con alpha = .0001

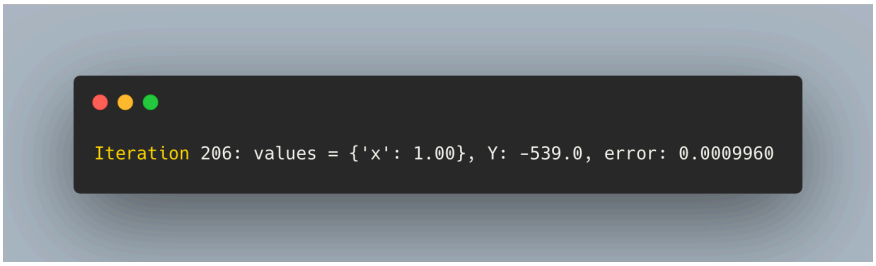
Ascendente 1: Salida

Tarea 4: Gradiente descendente



Ascendente 2: Entrada con alpha = .0001

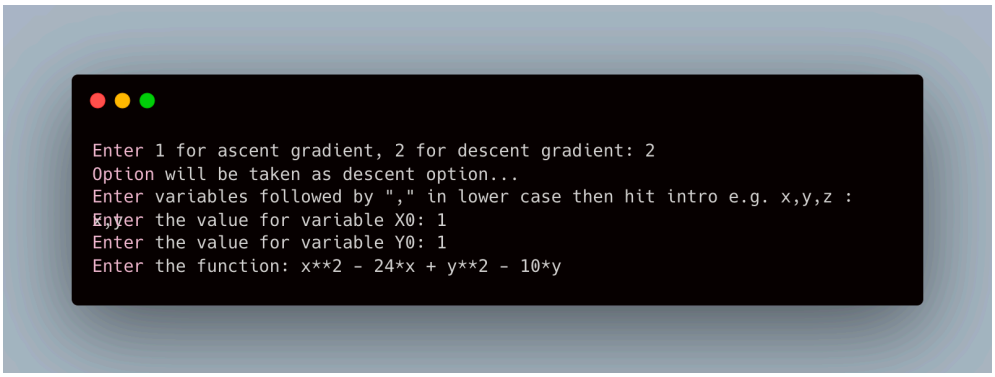
Ascendente Salida:



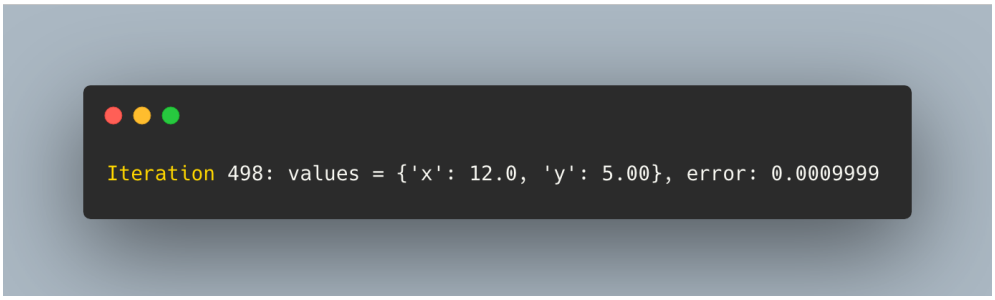
3.-

Entrada para gradiente descendente

Tarea 4: Gradiente descendente



Descendente:



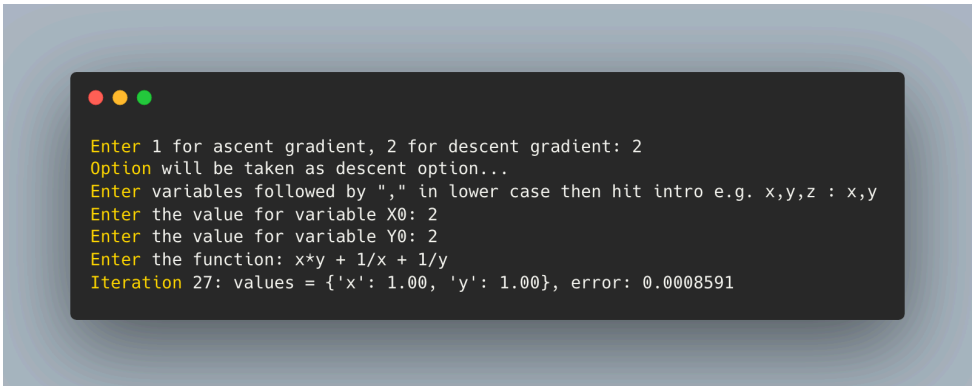
Ascendente:

No existe

Tarea 4: Gradiente descendente

4.-

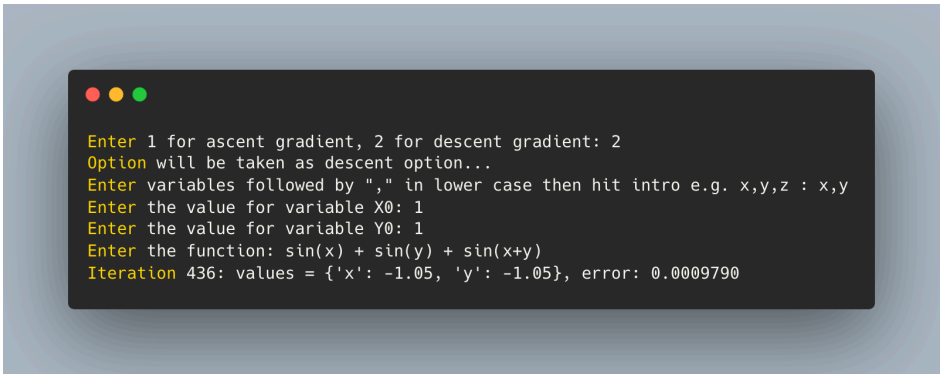
Entrada y salida para gradiente descendente con alpha = .1



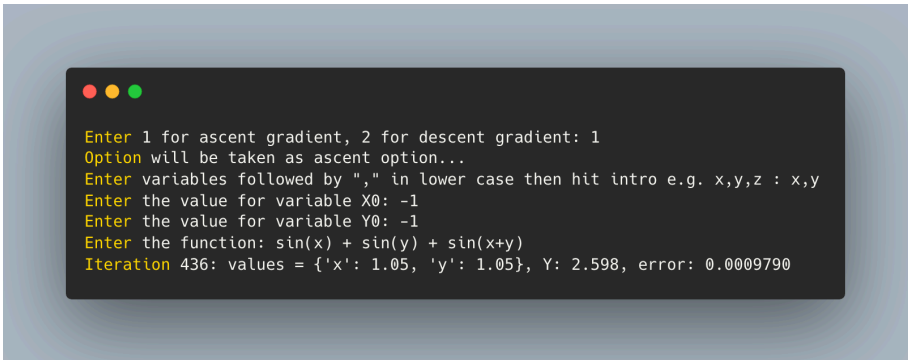
Para ascendente no encontré punto

5.-

Entrada y salida para gradiente descendente



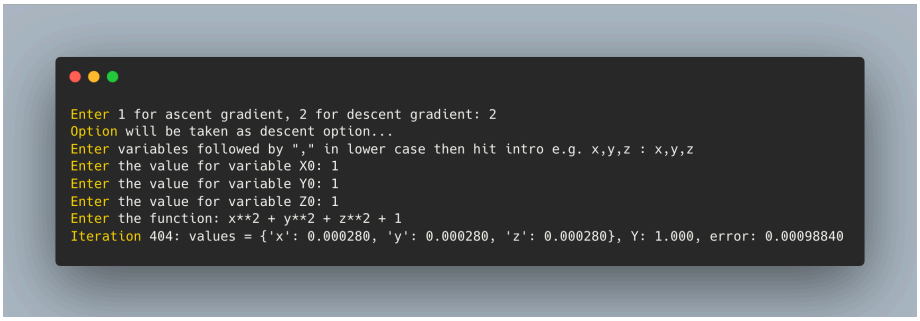
Entrada y salida para gradiente ascendente



Tarea 4: Gradiente descendente

6.-

Entrada y salida para gradiente descendente

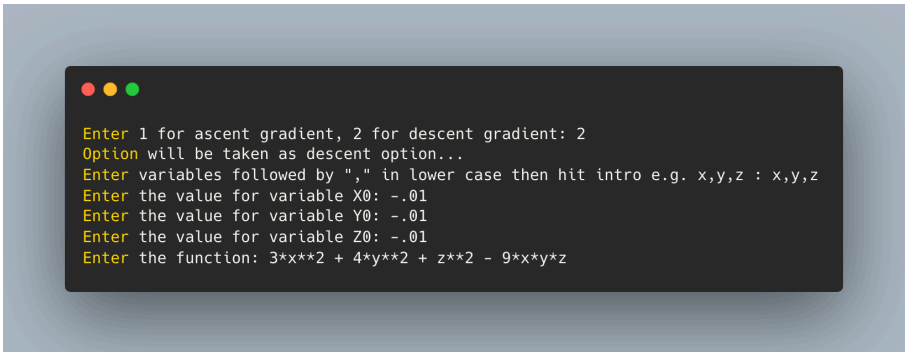


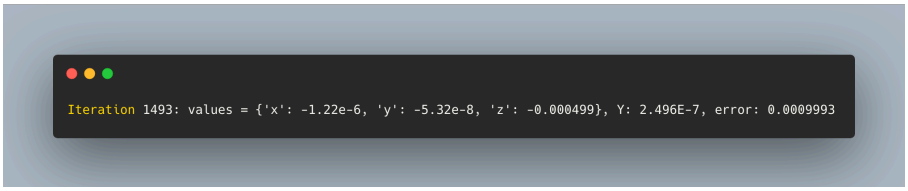
Ascendente:

No encontré

7.-

Entrada y salida para gradiente descendente con alpha = .001



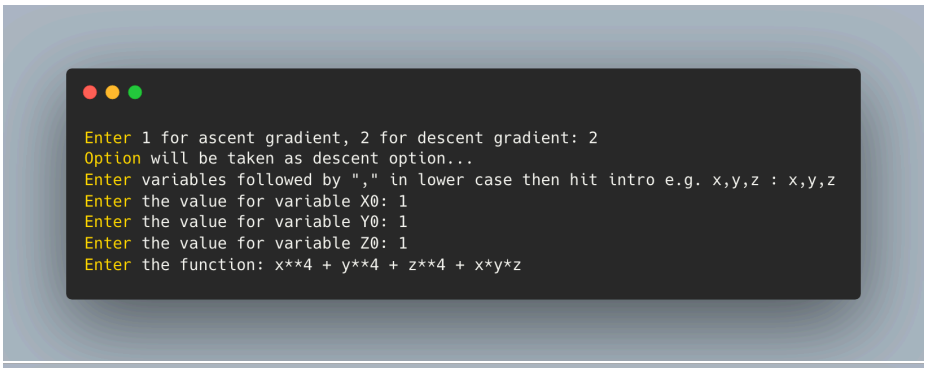


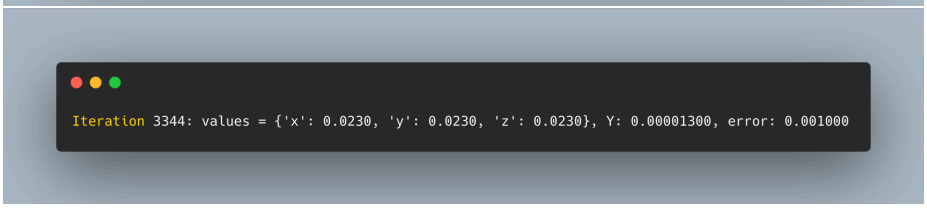
Ascendente: No encontré

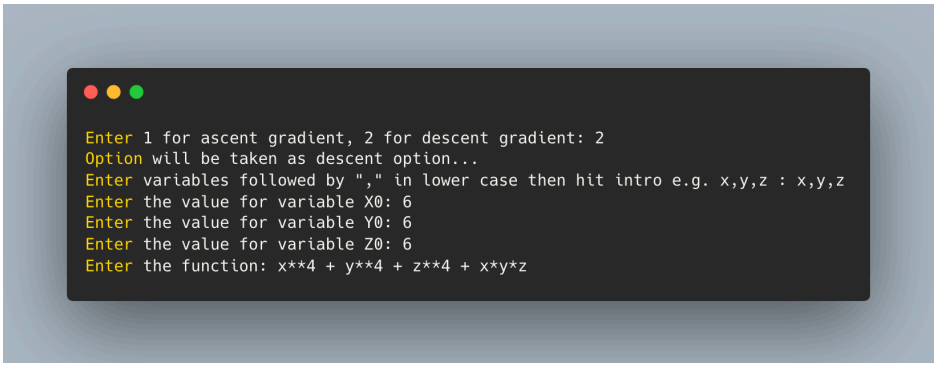
8.

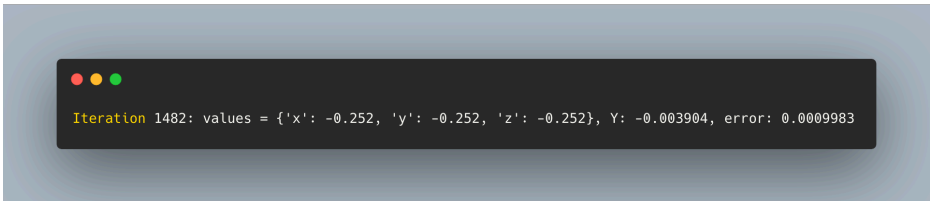
Entrada y salida para gradiente descendiente con alpha = .01

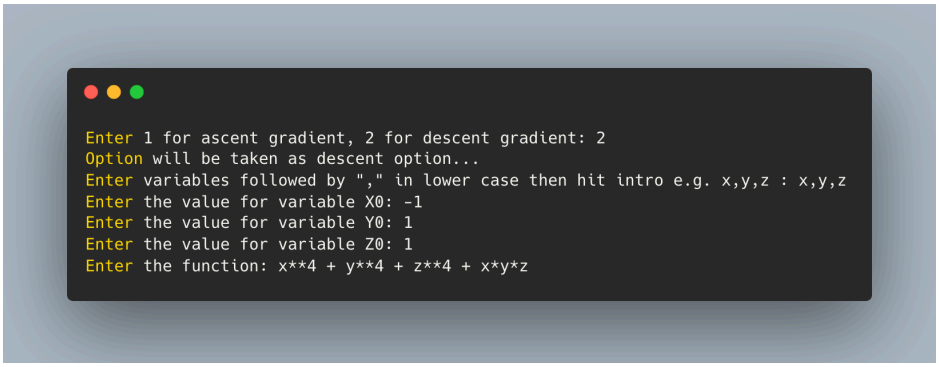
Tarea 4: Gradiente descendente



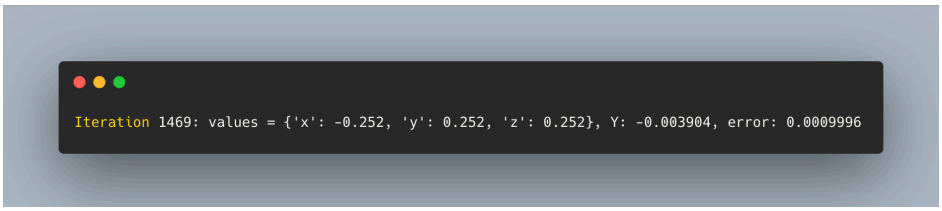


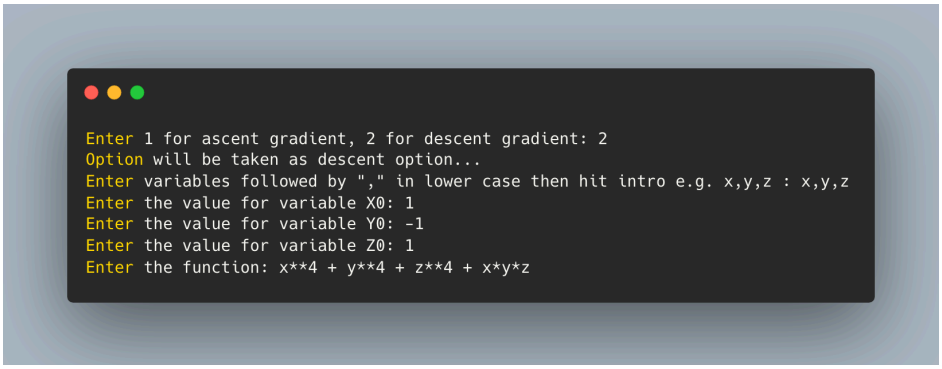


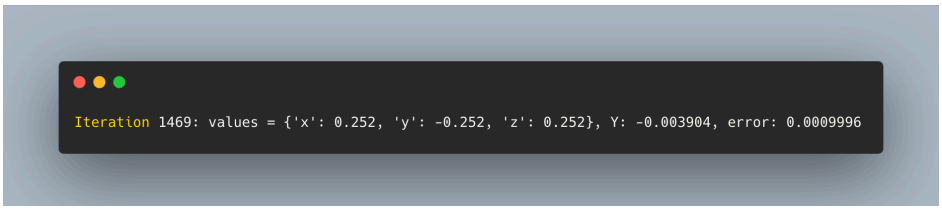


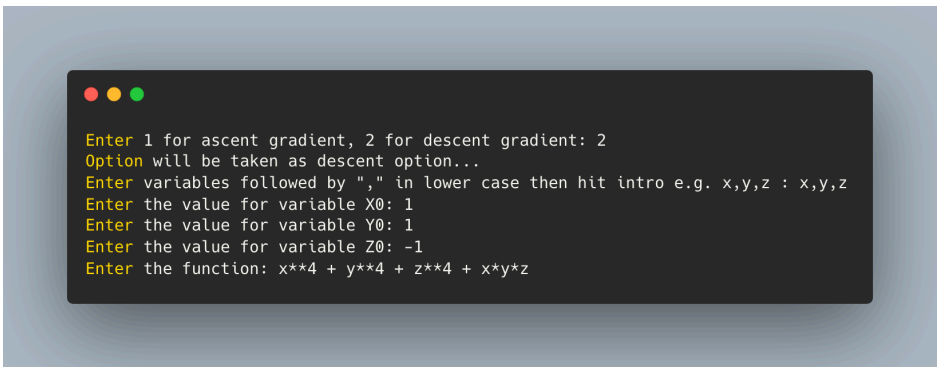


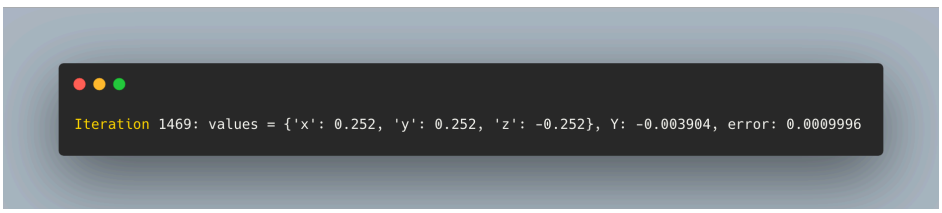
Tarea 4: Gradiente descendente











Ascendente: No encontrado

Tarea 4: Gradiente descendente

Código:

```python
import sympy as sp

# Define the learning rate and number of iterations
alpha = .01
limit = 0.001
num_iterations = 10000
func_variables = []
variable_values = []


# Define the function to optimize
def get_type_asc_desc():
    global alpha
    option = int(input('Enter 1 for ascent gradient, 2 for descent gradient: '))
    if(option == 1):
        print('Option will be taken as ascent option...')
        alpha = alpha * -1
    else:
        print('Option will be taken as descent option...')

def init_func_variables():
    variable = input('Enter variables followed by "," in lower case then hit intro e.g. x,y,z : ')
    global func_variables
    func_variables = variable.split(',')
    for i in func_variables:
        i = sp.symbols(i)
    return func_variables

def create_dictionary_of_variable_and_values():
    global variable_values
    for i  in func_variables:
        value = float(input(f'Enter the value for variable {i.capitalize()}0: '))
        variable_values.append(value)
    return list(zip(func_variables, variable_values))

def get_funct():
    f = input('Enter the function: ')
    return f

def create_gradient(function):
    grad_lst = []
    for i in func_variables:
        grad_lst.append(sp.diff(function, i))
    return sp.Matrix(grad_lst)

get_type_asc_desc()
init_func_variables()
dict_of_variable_and_values = create_dictionary_of_variable_and_values()
f = get_funct()
grad_f = create_gradient(f)

# Perform the gradient descent optimization
curr_variable_values = sp.Matrix(variable_values)
for i in range(num_iterations):

    # Evaluate the gradient at the current point
    grad = sp.Matrix([g.subs(dict_of_variable_and_values) for g in grad_f])

    # Update the current point using the gradient and learning rate
    curr_variable_values -= alpha * grad

    # Update variable values
    dict_of_variable_and_values = list(zip(func_variables, curr_variable_values))

    # Evaluate norm to break the loop
    v = sp.Matrix(grad)
    error = sp.trigsimp(v.norm())
    if(error < limit):
        break

    # Evaluate func in curr point
    derivate = sp.sympify(f).evalf(subs=dict(zip(func_variables, curr_variable_values.evalf(3))))

    print(f"Iteration {i+1}: point={dict(zip(func_variables, curr_variable_values.evalf(3)))}, val in
f(): {derivate:.4}, error: {error:.4}")

print(f"Iteration {i}: values = {dict(zip(func_variables, curr_variable_values.evalf(3)))}, Y:
{derivate:.4}, error: {error:.4}")
```