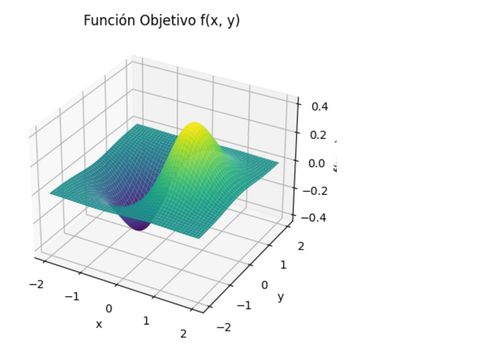
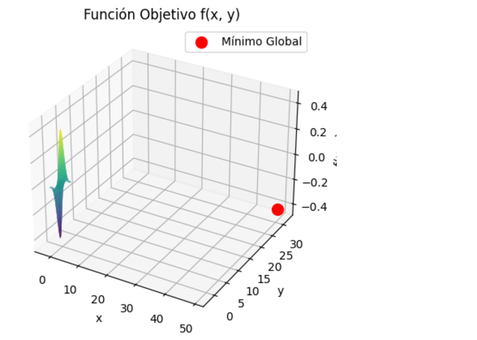
```
import random
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def f(x, y):
    return x * math.exp(-x**2 - y**2)

def hill_climbing_random_mutation():
    # Inicialización: Generamos un punto aleatorio en el dominio [-2, 2]
    x = random.uniform(-2, 2)
    y = random.uniform(-2, 2)
    best_value = f(x, y)

    # Parámetros de mutación
    mutation_step = 0.1
    num_iterations = 1000

    # Lista para almacenar los valores en cada iteración
    convergence_values = []

    for i in range(num_iterations):
        # Generamos una mutación aleatoria en las coordenadas x e y
        new_x = x + random.uniform(-mutation_step, mutation_step)
        new_y = y + random.uniform(-mutation_step, mutation_step)

        # Evaluamos la función en el nuevo punto
        new_value = f(new_x, new_y)

        # Si mejora, actualizamos el punto actual
        if new_value < best_value:
            x, y = new_x, new_y
            best_value = new_value

        # Registramos el valor en esta iteración
        convergence_values.append(best_value)

    return x, y, best_value, convergence_values

# Ejecutamos el algoritmo
best_x, best_y, min_value, convergence_values = hill_climbing_random_mutation()
print(f"El mínimo global se encuentra en (x, y) = ({best_x:.4f}, {best_y:.4f})")
print(f"Valor mínimo global: {min_value:.4f}")

# Graficamos la convergencia
plt.plot(convergence_values)
plt.xlabel('Iteración')
plt.ylabel('Valor de f(x, y)')
plt.title('Convergencia de Hill Climbing')
plt.show()

def f(x, y):
    return x * np.exp(-x**2 - y**2)

# Generamos una malla de puntos en el dominio [-2, 2]
x_vals = np.linspace(-2, 2, 100)
y_vals = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = f(X, Y)

# Creamos una figura 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Graficamos la superficie
ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('Función Objetivo f(x, y)')

# Mostramos la gráfica
plt.show()

# Encontramos el mínimo global
min_x, min_y = np.unravel_index(Z.argmin(), Z.shape)
min_value = Z.min()

# Creamos una figura 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Graficamos la superficie
ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('Función Objetivo f(x, y)')

# Marcamos el mínimo global
ax.scatter(min_x, min_y, min_value, color='red', s=100, label='Mínimo Global')

# Mostramos la gráfica
plt.legend()
plt.show()
```

```
El mínimo global se encuentra en (x, y) = (-0.7061, 0.0012)
Valor mínimo global: -0.4289
```



Primera funcion Busqueda aleatoria.

```
import random
import numpy as np

def f(x, y):
    return x * np.exp(-x**2 - y**2)

def optimize(function, dimensions, lower_boundary, upper_boundary, max_iter, maximize=False):
    best_solution = np.array([float()] * dimensions)
    for i in range(dimensions):
        best_solution[i] = random.uniform(lower_boundary[i], upper_boundary[i])

    for _ in range(max_iter):
        solution1 = function(*best_solution)

        # Generamos una nueva solución aleatoria
        new_solution = [lower_boundary[d] + random.random() * (upper_boundary[d] - lower_boundary[d])
                        for d in range(dimensions)]

        if np.greater_equal(new_solution, lower_boundary).all() and np.less_equal(new_solution, upper_boundary).all():
            solution2 = function(*new_solution)
        elif maximize:
            solution2 = -100000.0
        else:
            solution2 = 100000.0

        # Si la nueva solución es mejor, actualizamos la mejor solución
        if solution2 < solution1:
            best_solution = np.array(new_solution)

    return best_solution

# Ejemplo de uso
dimensions = 2
lower_boundary = [-2, -2]
upper_boundary = [2, 2]
max_iter = 1000

best_solution = optimize(f, dimensions, lower_boundary, upper_boundary, max_iter)
print(f"La mejor solución encontrada es (x, y) = ({best_solution[0]:.4f}, {best_solution[1]:.4f})")
print(f"Valor mínimo global: {f(*best_solution):.4f}")
```

```
La mejor solución encontrada es (x, y) = (-0.7086, -0.0463)
Valor mínimo global: -0.4280
```

Hill Climbing Adaptativo

```
import random
import math

def f(x, y):
    return x * math.exp(-x**2 - y**2)

def hill_climbing_adaptive():
    # Inicialización: Generamos un punto aleatorio en el dominio [-2, 2]
    x = random.uniform(-2, 2)
    y = random.uniform(-2, 2)
    best_value = f(x, y)
```

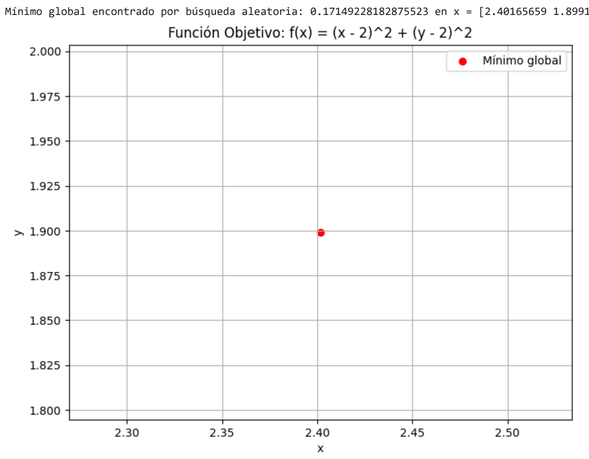
```
    # Parámetros iniciales
    mutation_step = 0.1
    num_iterations = 1000

    for _ in range(num_iterations):
        # Generamos una mutación aleatoria en las coordenadas x e y
        new_x = x + random.uniform(-mutation_step, mutation_step)
        new_y = y + random.uniform(-mutation_step, mutation_step)

        # Evaluamos la función en el nuevo punto
        new_value = f(new_x, new_y)

        # Si mejora, actualizamos el punto actual y aumentamos el tamaño de la mutación
        if new_value < best_value:
            x, y = new_x, new_y
            best_value = new_value
            mutation_step *= 1.1  # Aumentamos el tamaño de la mutación
        else:
            mutation_step *= 0.9  # Reducimos el tamaño de la mutación

    return x, y, best_value

# Ejecutamos el algoritmo
best_x, best_y, min_value = hill_climbing_adaptive()
print(f"El mínimo global se encuentra en (x, y) = ({best_x:.4f}, {best_y:.4f})")
print(f"Valor mínimo global: {min_value:.4f}")
```

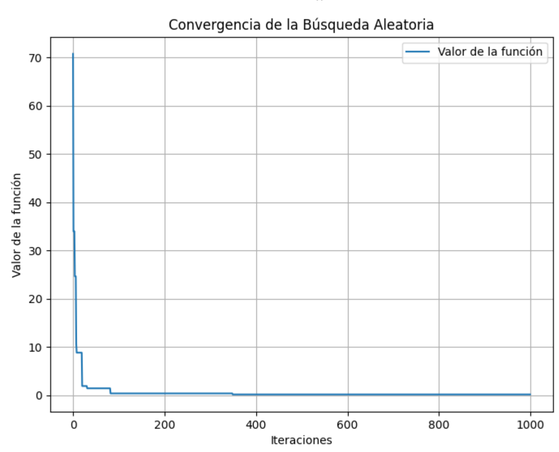
```
El mínimo global se encuentra en (x, y) = (-0.7071, 0.0000)
Valor mínimo global: -0.4289
```

Segunda funcion Busqueda aleatoria.

```
import random
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Búsqueda aleatoria
def busqueda_aleatoria(iteraciones):
    valores_convergencia = []  # Almacenaremos los valores de la función en cada iteración
    min_valor = float('inf')
    min_x = None

    for _ in range(iteraciones):
        x = np.random.uniform(-10, 10, 2)
        valor = f(x)

        if valor < min_valor:
            min_valor = valor
            min_x = x

        valores_convergencia.append(min_valor)  # Guardamos el valor actual en la lista

    return min_x, min_valor, valores_convergencia

# Ejemplo de uso
min_x, min_valor, valores_convergencia = busqueda_aleatoria(1000)
print(f"Mínimo global encontrado por búsqueda aleatoria: {min_valor} en x = {min_x}")

# Rango de valores para x e y
x_vals = np.linspace(-10, 10, 100)
y_vals = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = f(np.array([X, Y]))

# Graficar la función
plt.figure(figsize=(8, 6))
plt.scatter(min_x[0], min_x[1], color='red', marker='o', label='Mínimo global')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Función Objetivo: f(x) = (x - 2)^2 + (y - 2)^2')
plt.legend()
plt.grid(True)
plt.show()

# Graficar la convergencia
plt.figure(figsize=(8, 6))
plt.plot(valores_convergencia, label='Valor de la función')
plt.xlabel('Iteraciones')
plt.ylabel('Valor de la función')
plt.title('Convergencia de la Búsqueda Aleatoria')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mínimo global encontrado por búsqueda aleatoria: 0.17149228182875523 en x = [2.40165659 1.89918202]
```





Segunda funcion Hill climbing mutacion aleatoria

```
import random
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def hill_climbing_mutacion_aleatoria(iteraciones):
    x_actual = np.random.uniform(-10, 10, 2)

    for _ in range(iteraciones):
        mutacion = np.random.uniform(-0.1, 0.1, 2)
        x_nuevo = x_actual + mutacion

        if f(x_nuevo) < f(x_actual):
            x_actual = x_nuevo

    return x_actual, f(x_actual)

# Ejemplo de uso
min_x, min_valor = hill_climbing_mutacion_aleatoria(1000)
print(f"Mínimo global encontrado por Hill Climbing con mutación aleatoria: {min_valor} en x = {min_x}")
```

```
Mínimo global encontrado por Hill Climbing con mutación aleatoria: 1.1122900509023577e-05 en x = [1.99844518 1.9970495 ]
```

Segunda funcion Hill climbing adaptativo

```
import random
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def hill_climbing_adaptativo(iteraciones):
    tam_paso_inicial = 0.1
    x_actual = np.random.uniform(-10, 10, 2)

    for _ in range(iteraciones):
        direccion = np.random.uniform(low=-tam_paso_inicial, high=tam_paso_inicial, size=2)
        x_nuevo = x_actual + direccion

        if f(x_nuevo) < f(x_actual):
            x_actual = x_nuevo
            tam_paso_inicial *= 1.05
        else:
            tam_paso_inicial *= 0.95

    return x_actual, f(x_actual)

# Ejemplo de uso
min_x, min_valor = hill_climbing_adaptativo(1000)
print(f"Mínimo global encontrado por Hill Climbing adaptativo: {min_valor} en x = {min_x}")
```

```
Mínimo global encontrado por Hill Climbing adaptativo: 1.332924959524168e-25 en x = [2. 2.]
```