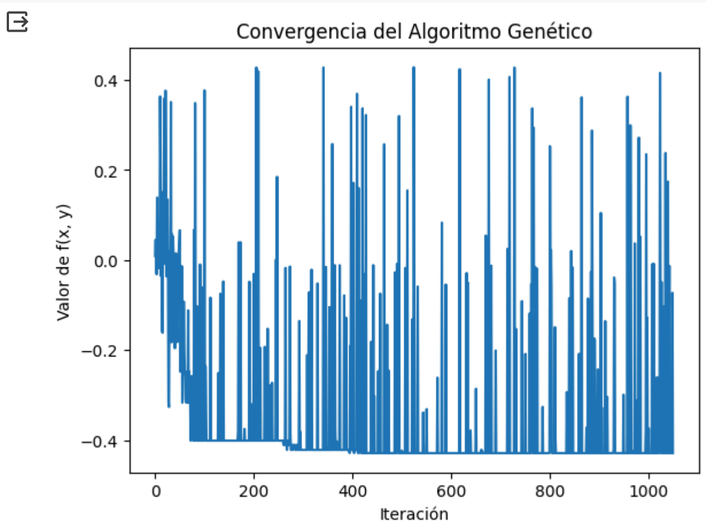
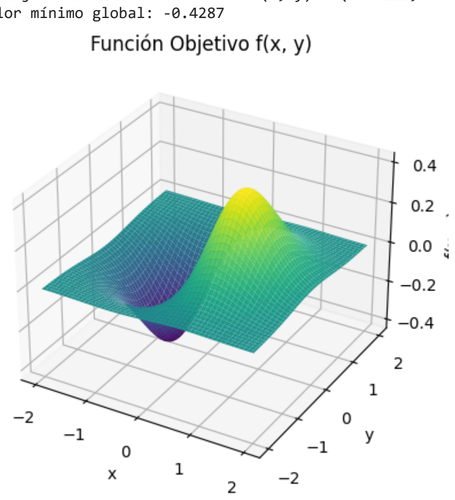
Codigo de Algoritmo Genetico

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

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

def initialize_population(pop_size, dimensions, lower_boundary, upper_boundary):
    population = []
    for _ in range(pop_size):
        individual = [random.uniform(lower_boundary[d], upper_boundary[d]) for d in range(dimensions)]
        population.append(individual)
    return population

def evaluate_population(population):
    return [f(*individual) for individual in population]

def select_parents(population, num_parents):
    sorted_population = sorted(enumerate(population), key=lambda x: f(*x[1]))
    return [sorted_population[i][0] for i in range(num_parents)]

def crossover(parents, dimensions):
    child = [0.0] * dimensions
    for d in range(dimensions):
        child[d] = random.choice(parents)[d]
    return child

def mutate(child, mutation_rate, lower_boundary, upper_boundary):
    for d in range(len(child)):
        if random.random() < mutation_rate:
            child[d] = random.uniform(lower_boundary[d], upper_boundary[d])
    return child

def genetic_algorithm():
    dimensions = 2
    lower_boundary = [-2, -2]
    upper_boundary = [2, 2]
    pop_size = 50
    num_parents = 10
    mutation_rate = 0.1
    num_generations = 1000

    population = initialize_population(pop_size, dimensions, lower_boundary, upper_boundary)

    for _ in range(num_generations):
        parents = select_parents(population, num_parents)
        child = crossover([population[p] for p in parents], dimensions)
        child = mutate(child, mutation_rate, lower_boundary, upper_boundary)
        population.append(child)

    best_solution = min(population, key=lambda x: f(*x))
    # Almacenamos los valores de la función en cada iteración
    convergence_values = []
    for individual in population:
        convergence_values.append(f(*individual))
    # Graficamos la convergencia
    convergence_values = [f(*individual) for individual in population]
    plt.plot(convergence_values)
    plt.xlabel('Iteración')
    plt.ylabel('Valor de f(x, y)')
    plt.title('Convergencia del Algoritmo Genético')
    plt.show()
    return best_solution, f(*best_solution)

# Ejecutamos el algoritmo genético
best_solution, min_value = genetic_algorithm()

print(f"La mejor solución encontrada es (x, y) = ({best_solution[0]:.4f}, {best_solution[1]:.4f})")
print(f"Valor mínimo global: {min_value:.4f}")

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()
```

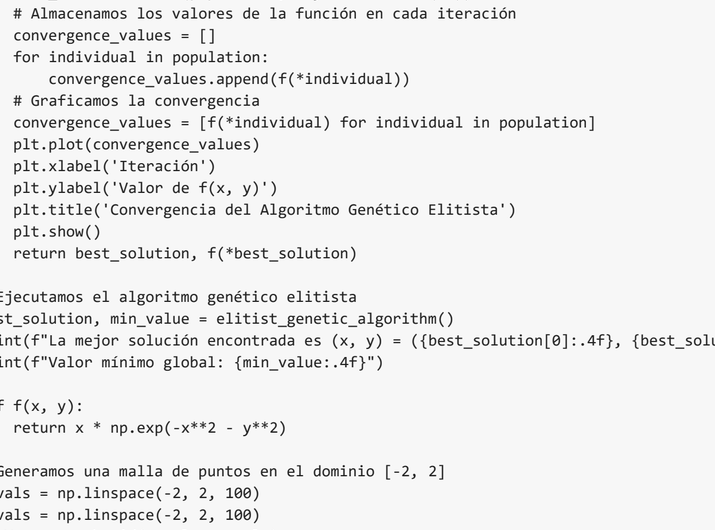



```
La mejor solución encontrada es (x, y) = (-0.7221, -0.0043)
Valor mínimo global: -0.4287
```

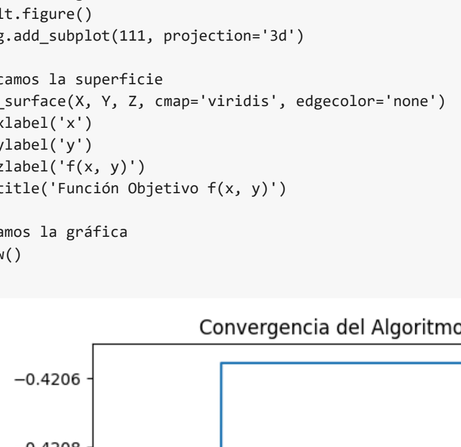


Codigo de algoritmo Genetico Elitista

```
import random
import math

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

def initialize_population(pop_size, dimensions, lower_boundary, upper_boundary):
    population = []
    for _ in range(pop_size):
        individual = [random.uniform(lower_boundary[d], upper_boundary[d]) for d in range(dimensions)]
        population.append(individual)
    return population

def evaluate_population(population):
    return [f(*individual) for individual in population]

def select_parents(population, num_parents):
    sorted_population = sorted(enumerate(population), key=lambda x: f(*x[1]))
    return [sorted_population[i][0] for i in range(num_parents)]

def crossover(parents, dimensions):
    child = [0.0] * dimensions
    for d in range(dimensions):
        child[d] = random.choice(parents)[d]
    return child

def mutate(child, mutation_rate, lower_boundary, upper_boundary):
    for d in range(len(child)):
        if random.random() < mutation_rate:
            child[d] = random.uniform(lower_boundary[d], upper_boundary[d])
    return child

def elitist_genetic_algorithm():
    dimensions = 2
    lower_boundary = [-2, -2]
    upper_boundary = [2, 2]
    pop_size = 50
    num_parents = 10
    mutation_rate = 0.1
    num_generations = 1000

    population = initialize_population(pop_size, dimensions, lower_boundary, upper_boundary)

    for _ in range(num_generations):
        parents = select_parents(population, num_parents)
        child = crossover([population[p] for p in parents], dimensions)
        child = mutate(child, mutation_rate, lower_boundary, upper_boundary)
        population.append(child)

        # Elitismo: Reemplazamos el peor individuo con el mejor individuo de la generación anterior
        worst_index = population.index(max(population, key=lambda x: f(*x)))
        population[worst_index] = min(population, key=lambda x: f(*x))

    best_solution = min(population, key=lambda x: f(*x))
    # Almacenamos los valores de la función en cada iteración
    convergence_values = []
    for individual in population:
        convergence_values.append(f(*individual))
    # Graficamos la convergencia
    convergence_values = [f(*individual) for individual in population]
    plt.plot(convergence_values)
    plt.xlabel('Iteración')
    plt.ylabel('Valor de f(x, y)')
    plt.title('Convergencia del Algoritmo Genético Elitista')
    plt.show()
    return best_solution, f(*best_solution)

# Ejecutamos el algoritmo genético elitista
best_solution, min_value = elitist_genetic_algorithm()
print(f"La mejor solución encontrada es (x, y) = ({best_solution[0]:.4f}, {best_solution[1]:.4f})")
print(f"Valor mínimo global: {min_value:.4f}")

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()
```

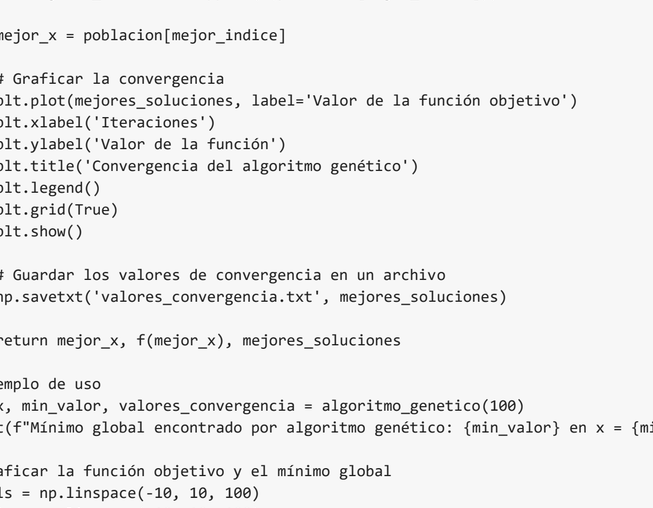



```
La mejor solución encontrada es (x, y) = (-0.7212, -0.1285)
Valor mínimo global: -0.4217
```

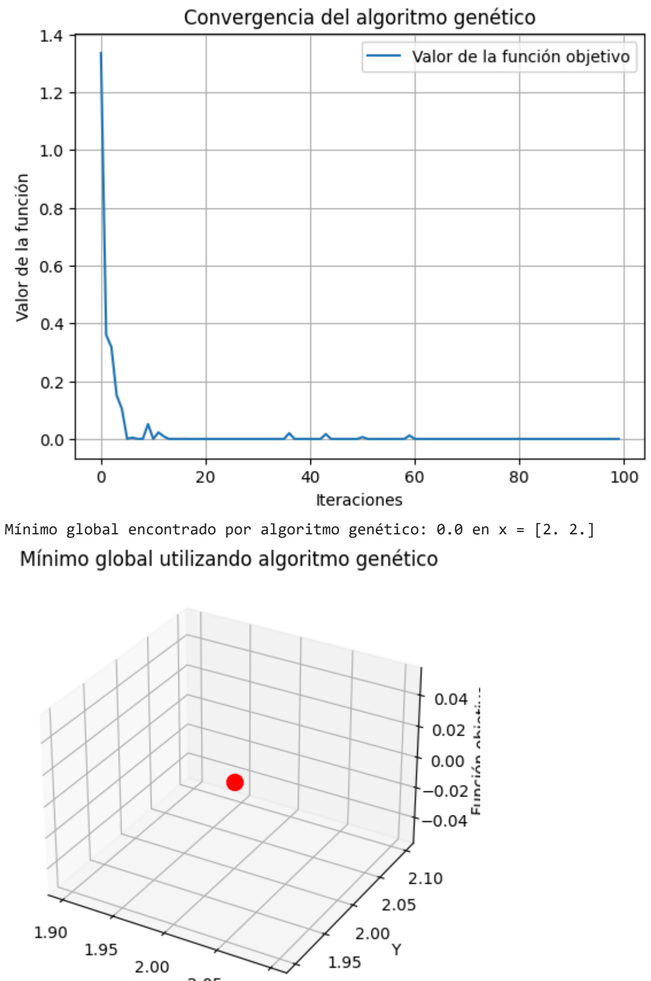


Codigo Algoritmo genetico, de la segunda funcion.

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

# Función objetivo
def f(x):
    return np.sum((x - 2)**2)

# Algoritmo genético
def algoritmo_genetico(iteraciones):
    poblacion_tamano = 100
    poblacion = np.random.uniform(-10, 10, size=(poblacion_tamano, 2))
    mejores_soluciones = []

    for _ in range(iteraciones):
        # Evaluación de la población
        fitness = np.array([f(individuo) for individuo in poblacion])

        # Selección de padres
        padres_indices = np.argsort(fitness)[:poblacion_tamano // 2]
        padres = poblacion[padres_indices]

        # Cruza
        hijos = []
        for _ in range(poblacion_tamano // 2):
            padre1, padre2 = random.choice(padres), random.choice(padres)
            hijo = (padre1 + padre2) / 2
            hijos.append(hijo)

        # Mutación
        for i in range(poblacion_tamano // 2):
            if random.random() < 0.1:
                hijos[i] += np.random.normal(scale=0.1, size=2)

        # Reemplazo de la población
        poblacion = np.vstack((padres, hijos))

        # Mejor solución actual
        mejor_indice = np.argmin(fitness)
        mejores_soluciones.append(f(poblacion[mejor_indice]))

    mejor_x = poblacion[mejor_indice]

    # Graficar la convergencia
    plt.plot(mejores_soluciones, label='Valor de la función objetivo')
    plt.xlabel('Iteraciones')
    plt.ylabel('Valor de la función')
    plt.title('Convergencia del algoritmo genético')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Guardar los valores de convergencia en un archivo
    np.savetxt('valores_convergencia.txt', mejores_soluciones)

    return mejor_x, f(mejor_x), mejores_soluciones

# Ejemplo de uso
min_x, min_valor, valores_convergencia = algoritmo_genetico(100)
print(f"Mínimo global encontrado por algoritmo genético: {min_valor} en x = {min_x}")

# Graficar la función objetivo y el mínimo global
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]))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7)
ax.scatter(min_x[0], min_x[1], min_valor, color='red', s=100, label='Mínimo global')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Función objetivo')
ax.set_title('Mínimo global utilizando algoritmo genético')

plt.show()
```




```
Mínimo global encontrado por algoritmo genético: 0.0 en x = [2. 2.]
```



Codigo Algoritmo genetico elitista, de la segunda funcion.

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

# Función dada
def f(x):
    return np.sum((x - 2)**2)

# Inicialización de la población
def inicializar_poblacion(tam_poblacion, dim):
    return [np.random.uniform(-10, 10, dim) for _ in range(tam_poblacion)]

# Evaluación de la población
def evaluar_poblacion(poblacion):
    return [f(x) for x in poblacion]

# Selección de los mejores individuos (elitismo)
def seleccion_elitista(poblacion, valores):
    num_elitismo = int(0.1 * len(poblacion))  # Mantenemos el 10% de los mejores individuos
    indices_mejores = np.argsort(valores)[:num_elitismo]
    return [poblacion[i] for i in indices_mejores]

# Cruza de dos individuos
def cruzar(padre1, padre2):
    punto_cruza = np.random.randint(len(padre1))
    hijo = np.concatenate((padre1[:punto_cruza], padre2[punto_cruza:]))
    return hijo

# Mutación de un individuo
def mutar(individuo, prob_mutacion):
    for i in range(len(individuo)):
```

```
        if random.random() < prob_mutacion:
            individuo[i] += np.random.normal(0, 0.1)  # Pequeña perturbación
    return individuo

# Algoritmo genético elitista
def algoritmo_genetico_elitista(tam_poblacion, dim, num_generaciones):
    poblacion = inicializar_poblacion(tam_poblacion, dim)
    for _ in range(num_generaciones):
        valores = evaluar_poblacion(poblacion)
        poblacion = seleccion_elitista(poblacion, valores)
        nueva_poblacion = []
        while len(nueva_poblacion) < tam_poblacion:
            padre1, padre2 = random.choices(poblacion, k=2)
            hijo = cruzar(padre1, padre2)
            hijo = mutar(hijo, prob_mutacion=0.1)
            nueva_poblacion.append(hijo)
        poblacion = nueva_poblacion

    mejor_x = poblacion[0]
    mejor_valor = f(mejor_x)

    # Guardar los valores de convergencia en un archivo
    np.savetxt('valores_convergencia.txt', mejor_x)
    return mejor_x, mejor_valor

# Ejemplo de uso
min_x, min_valor = algoritmo_genetico_elitista(tam_poblacion=100, dim=2, num_generaciones=1000)
print(f"Mínimo global encontrado por algoritmo genético elitista: {min_valor} en x = {min_x}")
 # Graficar la convergencia
plt.plot(min_x, label='Valor de la función objetivo')
plt.xlabel('Iteraciones')
plt.ylabel('Valor de la función')
plt.title('Convergencia del algoritmo genético')
plt.legend()
plt.grid(True)
plt.show()
# Graficar la función objetivo y el mínimo global
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]))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7)
ax.scatter(min_x[0], min_x[1], min_valor, color='red', s=100, label='Mínimo global')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Función objetivo')
ax.set_title('Mínimo global utilizando algoritmo genético elitista')

plt.show()
```

```
Mínimo global encontrado por algoritmo genético elitista: 0.003256361646680925 en x = [1.94293546 2.0000051 ]
```