algoritmo genético

Primero creamos las funciones:

objective_funtion: la funcion de entrenamiento

To_binary: convierte a binario

To_decimal: convierte a decimal

Mutate: muta un valor

```python
# Función para calcular el valor de la función objetivo
def objective_function(x):
    return x + abs(math.sin(32 * x))

# Función para convertir un número en su representación binaria con un número fijo de bits
def to_binary(num, num_bits):
    binary = bin(num)[2:]
    padding = '0' * (num_bits - len(binary))
    return padding + binary

# Función para convertir un número binario en su equivalente decimal
def to_decimal(binary):
    return int(binary, 2)

# Función para aplicar la mutación a un individuo
def mutate(individual, mutation_prob):
    mutated = ''
    for bit in individual:
        if random.random() < mutation_prob:
            mutated += '1' if bit == '0' else '0'
        else:
            mutated += str(bit)
    return mutated
```

Creamos nuestra poblacion inicial y algunas variables iniciales

```python
# Población inicial
population = []
for _ in range(10):
    x = random.uniform(0, 100)
    fitness = objective_function(x)
    num_bits = len('{0:.10f}'.format(fitness).replace('.', ''))
    binary = to_binary(int(fitness * 10**10), num_bits)
    population.append((x, fitness, binary))

# Número de bits de los valores de aptitud
num_bits = len(population[0][2])
xG = []
```

Obtenemos los datos de nuestra población y definimos el número de ciclos

```
# Ciclo de evolución (cambiando el range cambia el numero de ciclos)
for generation in range(20):
    # Número de ciclos
    xG.append(generation)
    temp = []
    print("Población")
    for i in range(0,10):
        print("  x = " + str(population[i][0]))
        temp.append(population[i][0])
    # Calculamos los datos a graficar
    max.append(np.max(temp))
    min.append(np.min(temp))
    media.append(np.mean(temp))
    desviacionBut.append(np.mean(temp) - np.std(temp))
    desviacionTop.append(np.mean(temp) + np.std(temp))
    # Selección de parejas para reproducción
    # Parents lo ordenamos de menor a mayor
    parents = sorted(population, reverse=True)
```

Generamos los hijos

```
# Creación de nuevos individuos mediante reproducción
offspring = []
print("hijos")
for i in range(0, 9):
    # Seleciona los padre del menor al mayor
    parent1 = parents[i]
    parent2 = parents[i+1]
    crossover_point = random.randint(1, num_bits-1)
    # Conservamos los 2 primero bits del padre para despues combinar los 3 bits de un padre y los otros 2 del otro padre
    (repetimos el proceso para los bits despues del punto decimal)
    child1 = parent1[2][0:2] + parent1[2][2:5] + parent2[2][5:7] + parent1[2][7:9] + parent1[2][9:12] + parent2[2][12:]
    child2 = parent1[2][0:2] + parent2[2][2:5] + parent1[2][5:7] + parent1[2][7:9] + parent2[2][9:12] + parent1[2][12:]
    child3 = parent2[2][0:2] + parent1[2][2:5] + parent2[2][5:7] + parent1[2][7:9] + parent1[2][9:12] + parent2[2][12:]
    child4 = parent2[2][0:2] + parent2[2][2:5] + parent1[2][5:7] + parent1[2][7:9] + parent2[2][9:12] + parent1[2][12:]
    offspring.append((child1,))
    offspring.append((child2,))
    offspring.append((child3,))
    offspring.append((child4,))
    print("padres : " + str(parent1[0]) + " , " + str(parent2[0]))
    print("hijo 1: " + str(to_decimal(child1) / 10**10) + " -- hijo 2: " + str(to_decimal(child2) / 10**10) + " -- hijo 3: " +
    str(to_decimal(child3) / 10**10) + " -- hijo 4: " + str(to_decimal(child4) / 10**10))
    print("")
```

Creamos la mutación

```
# Mutación de los nuevos individuos
mutation_prob = 1 / num_bits
mutated_offspring = []
numMuta = 0
for individual in offspring:
    if numMuta < 1:
        mutated = mutate(individual[0], mutation_prob)
        mutated_offspring.append((mutated,))
        numMuta = numMuta + 1

# Evaluación de los nuevos individuos
new_population = []
for individual in mutated_offspring:
    fitness = to_decimal(individual[0]) / 10**10
    if fitness > 100:
        fitness = 0
    x = fitness - abs(math.sin(32 * fitness))
    new_population.append((x, fitness, individual[0]))
    temp.append(x)
```

Combinamos y filtramos las poblaciones

```
# Combinación de la población anterior y la nueva
combined_population = population + new_population

# Selección de los 10 mejores individuos para la siguiente generación
sorted_population = sorted(combined_population, key=lambda x: x[0], reverse=True)
population = sorted_population[:10]
```

Imprimimos una grafica para ver los resultados

```
# Resultado final
print("Población Final")
for i in range(0,10):
    print(f"  x = {population[i][1]}")
plt.plot(xG, min, label='minimo')
plt.plot(xG, max, label='maximo')
plt.plot(xG, media, label='media')
plt.plot(xG, desviacionTop, label='desviacion Top')
plt.plot(xG, desviacionBut, label='desviacion But')
plt.xlabel('numero de ciclos')
plt.ylabel('accuracy')
plt.title('Gráfica')
plt.legend()
plt.show()
```

Aquí podemos ver algunos resultados

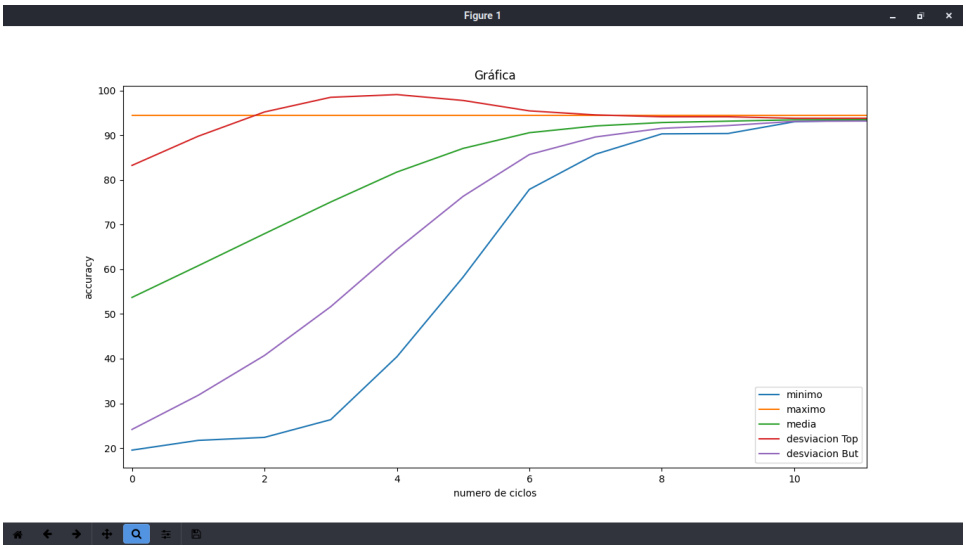
Figure 1
Gráfica
accuracy
numero de ciclos
minimo
maximo
media
desviacion Top
desviacion But

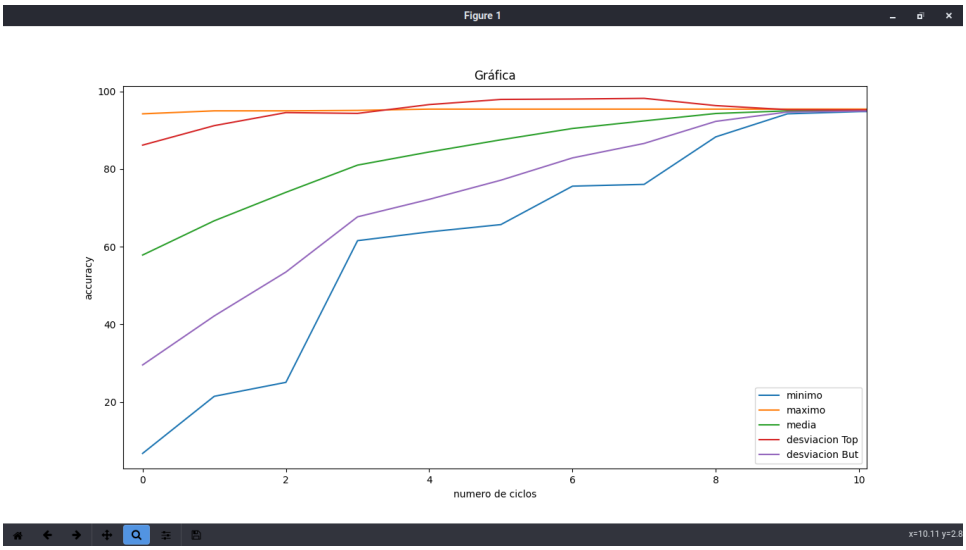
Figure 1
Gráfica
accuracy
numero de ciclos
minimo
maximo
media
desviacion Top
desviacion But
x=10.11 y=2.8

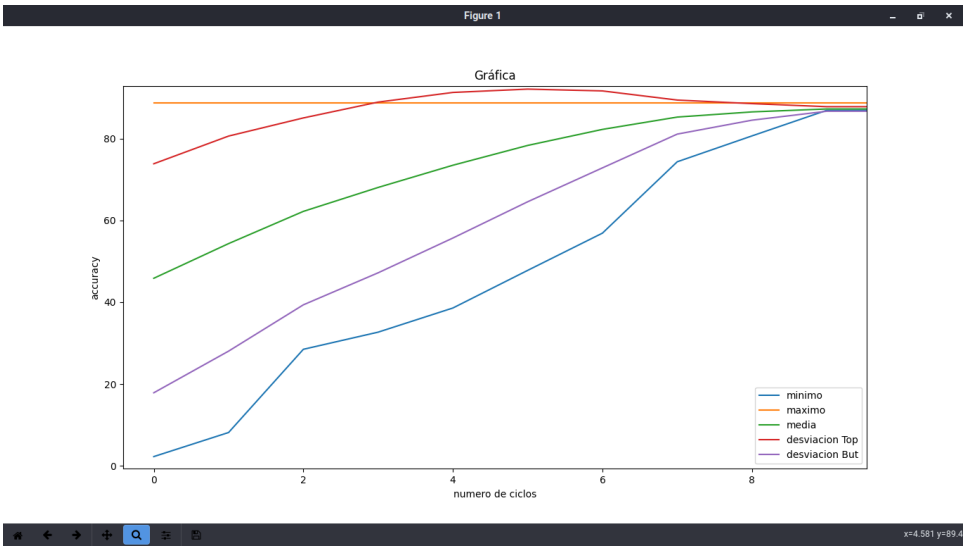
Figure 1
Gráfica
accuracy
numero de ciclos
minimo
maximo
media
desviacion Top
desviacion But
x=4.581 y=89.4