Sanjana Shetty
5E

Genetic Algorithm for Optimization Problems

```
import numpy as np

# Objective function: f(x) = x^2
def objective(x):
    return x[0] ** 2 + 2*x[0] + 1  # x is expected to be a list

# Initialization: generate initial population
def initialize_population(bounds, n_pop):
    return [np.random.uniform(bounds[0], bounds[1], 1).tolist() for _ in
range(n_pop)]

# Fitness evaluation
def evaluate_fitness(pop):
    return [objective(ind) for ind in pop]

# Roulette wheel selection
def roulette_wheel_selection(pop, scores):
    total_fitness = sum(scores)
    probabilities = [1 - (score / total_fitness) for score in scores]  #
Inverting for minimization
    selection_ix = np.random.choice(len(pop), p=np.array(probabilities) /
sum(probabilities))
    return pop[selection_ix]

# Crossover: linear combination of parents
def crossover(p1, p2, alpha=0.5):
    offspring = alpha * p1[0] + (1 - alpha) * p2[0]
    return [offspring]  # Ensure offspring is a list

# Mutation: random value within bounds
def mutation(individual, bounds, r_mut):
    if np.random.rand() < r_mut:
```

```
        return [np.random.uniform(bounds[0], bounds[1])]  # Return as a
list
    return individual  # Ensure we return the individual as a list

# Genetic algorithm
def genetic_algorithm(bounds, n_iter, n_pop, r_mut):
    # Initialize population
    pop = initialize_population(bounds, n_pop)
    best, best_eval = pop[0], objective(pop[0])

    for gen in range(n_iter):
        # Evaluate fitness
        scores = evaluate_fitness(pop)

        # Check for new best solution
        for i in range(n_pop):
            if scores[i] < best_eval:
                best, best_eval = pop[i], scores[i]
                print(f">{gen}, new best f({pop[i]}) = {scores[i]:.6f}")

        # Select parents and create offspring
        children = []
        for _ in range(n_pop):
            p1 = roulette_wheel_selection(pop, scores)
            p2 = roulette_wheel_selection(pop, scores)
            offspring = crossover(p1, p2)
            offspring = mutation(offspring, bounds, r_mut)  # Pass as list
            children.append(offspring)

        # Replace population with new offspring
        pop = children

    return [best, best_eval]

# Define range for input
bounds = [-10.0, 10.0]
# Define the total iterations
n_iter = 50
# Define the population size
n_pop = 100
```

```
# Mutation rate
r_mut = 0.1  # 10% mutation probability

# Perform the genetic algorithm search
best, score = genetic_algorithm(bounds, n_iter, n_pop, r_mut)
print('Done!')
print(f'f({best}) = {score:.6f}')
```

OUTPUT:

```
>0, new best f([0.9577382801046337]) = 3.832739
>0, new best f([0.8298284888593841]) = 3.348272
>0, new best f([-0.46579945537471623]) = 0.285370
>0, new best f([-0.9720350186912707]) = 0.000782
>7, new best f([-1.0096817878612878]) = 0.000094
>7, new best f([-0.9968410441255329]) = 0.000010
>9, new best f([-0.9994840936041086]) = 0.000000
>26, new best f([-0.9998181775606392]) = 0.000000
Done!
f([-0.9998181775606392]) = 0.000000
```