Code:

```
import numpy as np
import random

def sphere_function(solution):
    return np.sum(np.square(solution))

population_size = 20
num_genes = 10
mutation_rate = 0.1
crossover_rate = 0.8
num_generations = 50

def initialize_population(size, num_genes):
  return [np.random.uniform(-10, 10, num_genes) for _ in range(size)]

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

def tournament_selection(population, fitness, k=3):
  selected = []
  for _ in range(len(population)):
    competitors = random.sample(list(zip(population, fitness)), k)
    winner = min(competitors, key=lambda x: x[1])
    selected.append(winner[0])
  return selected

def crossover(parent1, parent2):
  if random.random() < crossover_rate:
    point = random.randint(1, len(parent1) - 1)
    child1 = np.concatenate((parent1[:point], parent2[point:]))
    child2 = np.concatenate((parent2[:point], parent1[point:]))
    return child1, child2
  return parent1.copy(), parent2.copy()

def mutate(individual):
```

```
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] += np.random.normal(0, 1)
    return individual


def gene_expression(individual):
    return individual

population = initialize_population(population_size, num_genes)

for generation in range(num_generations):
    fitness = evaluate_population(population)
    selected_population = tournament_selection(population, fitness)
    next_generation = []

    for i in range(0, len(selected_population), 2):
        parent1 = selected_population[i]
        parent2 = selected_population[i + 1] if i + 1 < len(selected_population) else
selected_population[0]
        child1, child2 = crossover(parent1, parent2)
        next_generation.append(mutate(child1))
        next_generation.append(mutate(child2))

    population = [gene_expression(ind) for ind in next_generation[:population_size]]

    best_fitness = min(fitness)
    best_solution = population[np.argmin(fitness)]
    print(f"Generation {generation + 1}: Best Fitness = {best_fitness}")

fitness = evaluate_population(population)
best_solution = population[np.argmin(fitness)]
best_fitness = min(fitness)
print("Best Solution:", best_solution)
print("Best Fitness:", best_fitness)
```

Output:

```
Generation 1: Best Fitness = 173.45453173462087
Generation 2: Best Fitness = 158.95237984704633
Generation 3: Best Fitness = 73.62165260581389
Generation 4: Best Fitness = 63.28481676041214
Generation 5: Best Fitness = 61.591284739923765
Best Solution: [-0.80333799  1.99833483  0.46494453  2.55857116  0
    .16386048  2.23798819
 -1.80388797  5.14469943 -0.41750817 -2.45611245]
Best Fitness: 52.36534065842369
```