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

# Define the problem (fitness function)
def fitness_function(x):
    return x * np.sin(10 * np.pi * x) + 1.0

# Parameters
population_size = 20
mutation_rate = 0.1
crossover_rate = 0.7
generations = 50
bounds = [0, 1]  # Range of x values

# Create the initial population
def create_population(size, bounds):
    return np.random.uniform(bounds[0], bounds[1], size)

# Evaluate fitness of the population
def evaluate_fitness(population):
    return np.array([fitness_function(ind) for ind in population])

# Select parents using roulette wheel selection
def select_parents(population, fitness):
    total_fitness = np.sum(fitness)
    probabilities = fitness / total_fitness
    indices = np.random.choice(len(population), size=2, p=probabilities)
    return population[indices]

# Perform crossover
def crossover(parent1, parent2):
    if np.random.rand() < crossover_rate:
        point = np.random.uniform()
        child1 = point * parent1 + (1 - point) * parent2
        child2 = (1 - point) * parent1 + point * parent2
        return child1, child2
    return parent1, parent2

# Perform mutation
def mutate(individual, bounds):
    if np.random.rand() < mutation_rate:
        mutation = np.random.uniform(bounds[0], bounds[1])
        return mutation
    return individual

# Genetic Algorithm Implementation
def genetic_algorithm():
    population = create_population(population_size, bounds)
    best_fitness_values = []

    for generation in range(generations):
        fitness = evaluate_fitness(population)
        new_population = []

        # Save the best fitness of the current generation
        best_fitness_values.append(np.max(fitness))

        for _ in range(population_size // 2):  # Produce new generation
            parent1, parent2 = select_parents(population, fitness)
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1, bounds)
            child2 = mutate(child2, bounds)
            new_population.extend([child1, child2])
```

```
        population = np.clip(new_population, bounds[0], bounds[1])  # Ensure
within bounds

    # Return the best solution found and fitness over generations
    best_individual = population[np.argmax(fitness)]
    return best_individual, best_fitness_values

# Run the Genetic Algorithm
best_solution, fitness_history = genetic_algorithm()

# Print the results
print("Best solution found:", best_solution)
print("Fitness of the best solution:", fitness_function(best_solution))

# Plot the fitness over generations
plt.plot(fitness_history)
plt.title("Fitness Over Generations")
plt.xlabel("Generation")
plt.ylabel("Fitness")
plt.show()
```