```python
import numpy as np
import random

# Define the objective function
def objective_function(x):
    return x ** 2

# Genetic Algorithm parameters
population_size = 100
num_generations = 50
mutation_rate = 0.1
crossover_rate = 0.8
value_range = (-10, 10)

# Initialize the population
def initialize_population(size, value_range):
    return np.random.uniform(value_range[0], value_range[1], size)

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

# Selection using roulette-wheel selection
def selection(population, fitness):
    total_fitness = np.sum(fitness)
    probabilities = fitness / total_fitness
    return np.random.choice(population, size=2, replace=False, p=probabilities)

# Crossover between two parents
def crossover(parent1, parent2):
    if random.random() < crossover_rate:
        return (parent1 + parent2) / 2  # Simple average for linear crossover
    return parent1 if random.random() < 0.5 else parent2

# Mutation of an individual
def mutate(individual):
    if random.random() < mutation_rate:
        return np.random.uniform(value_range[0], value_range[1])
    return individual

# Genetic Algorithm process
def genetic_algorithm():
    # Step 1: Initialize population
    population = initialize_population(population_size, value_range)

    best_solution = None
    best_fitness = -1
```

```
    for generation in range(num_generations):
        # Step 2: Evaluate fitness
        fitness = evaluate_fitness(population)

        # Track the best solution
        current_best_index = np.argmax(fitness)
        current_best = population[current_best_index]
        current_best_fitness = fitness[current_best_index]

        if current_best_fitness > best_fitness:
            best_fitness = current_best_fitness
            best_solution = current_best

        # Step 3: Create new population
        new_population = []
        for _ in range(population_size):
            parent1, parent2 = selection(population, fitness)
            offspring = crossover(parent1, parent2)
            offspring = mutate(offspring)
            new_population.append(offspring)

        population = np.array(new_population)

    return best_solution, best_fitness

# Run the Genetic Algorithm
best_x, best_value = genetic_algorithm()
print(f"Best x: {best_x}, Maximum value of f(x): {best_value}")
```

```
Best x: 9.989911520510962, Maximum value of f(x): 99.79833218763764
```