GENETIC ALGORITHM

CODE:

```
import random

import numpy as np


# ---------------------------

# Objective function

# ---------------------------

def fitness_function(x):

    # Minimize x^2

    return x ** 2




# ---------------------------

# GA Parameters

# ---------------------------

POP_SIZE = 50

GENS = 100

MUTATION_RATE = 0.1

CROSSOVER_RATE = 0.8

BOUNDS = (-10, 10)




# ---------------------------

# Initialization
```

```
# ---------------------------
def initialize_population():
    return np.random.uniform(BOUNDS[0], BOUNDS[1], POP_SIZE)


# ---------------------------
# Selection (Tournament)
# ---------------------------
def tournament_selection(population, fitness, k=3):
    selected = random.sample(list(zip(population, fitness)), k)
    selected.sort(key=lambda x: x[1])
    return selected[0][0]


# ---------------------------
# Crossover (Arithmetic)
# ---------------------------
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_RATE:
        alpha = random.random()
        child1 = alpha * parent1 + (1 - alpha) * parent2
        child2 = alpha * parent2 + (1 - alpha) * parent1
        return child1, child2
    return parent1, parent2
```

```
# ---------------------------
# Mutation
# ---------------------------
def mutate(x):
    if random.random() < MUTATION_RATE:
        x += np.random.normal(0, 1)
    return np.clip(x, BOUNDS[0], BOUNDS[1])


# ---------------------------
# Main GA Loop
# ---------------------------
def genetic_algorithm():
    population = initialize_population()

    for gen in range(GENS):
        fitness = np.array([fitness_function(ind) for ind in population])
        new_population = []

        while len(new_population) < POP_SIZE:
            p1 = tournament_selection(population, fitness)
            p2 = tournament_selection(population, fitness)

            c1, c2 = crossover(p1, p2)
```

```
            new_population.append(mutate(c1))

            new_population.append(mutate(c2))


        population = np.array(new_population[:POP_SIZE])


        best_idx = np.argmin(fitness)

        print(f"Gen {gen:3d} | Best x = {population[best_idx]:.5f} | Fitness =
{fitness[best_idx]:.5f}")


    best_idx = np.argmin(fitness)

    return population[best_idx], fitness[best_idx]




# ---------------------------

# Run GA

# ---------------------------

best_x, best_fitness = genetic_algorithm()

print("\nBest solution found:")

print("x =", best_x)

print("f(x) =", best_fitness)
```

----

output:

```
Gen   0 | Best x = 2.70314 | Fitness = 0.25917
Gen   1 | Best x = 0.74654 | Fitness = 0.01681
Gen   2 | Best x = -0.71944 | Fitness = 0.00314
Gen   3 | Best x = -0.14067 | Fitness = 0.00001
Gen   4 | Best x = -0.11921 | Fitness = 0.00001
```

```
Gen   5 | Best x = 0.01997 | Fitness = 0.00000
Gen   6 | Best x = 0.00020 | Fitness = 0.00000
Gen   7 | Best x = 0.00150 | Fitness = 0.00000
Gen   8 | Best x = 0.00133 | Fitness = 0.00000
Gen   9 | Best x = -0.00026 | Fitness = 0.00000
Gen  10 | Best x = 0.00010 | Fitness = 0.00000
Gen  11 | Best x = 0.00010 | Fitness = 0.00000
Gen  12 | Best x = -0.00002 | Fitness = 0.00000
Gen  13 | Best x = -0.00000 | Fitness = 0.00000
Gen  14 | Best x = -0.00000 | Fitness = 0.00000
Gen  15 | Best x = 0.00000 | Fitness = 0.00000
Gen  16 | Best x = -0.00000 | Fitness = 0.00000
Gen  17 | Best x = -0.00000 | Fitness = 0.00000
Gen  18 | Best x = -0.00000 | Fitness = 0.00000
Gen  19 | Best x = -0.00000 | Fitness = 0.00000
Gen  20 | Best x = -0.00000 | Fitness = 0.00000
Gen  21 | Best x = 0.00000 | Fitness = 0.00000
Gen  22 | Best x = 0.00000 | Fitness = 0.00000
Gen  23 | Best x = -0.00000 | Fitness = 0.00000
Gen  24 | Best x = -0.00000 | Fitness = 0.00000
Gen  25 | Best x = -0.00000 | Fitness = 0.00000
Gen  26 | Best x = -0.00000 | Fitness = 0.00000
Gen  27 | Best x = -0.00000 | Fitness = 0.00000
Gen  28 | Best x = 0.00000 | Fitness = 0.00000
Gen  29 | Best x = -0.00000 | Fitness = 0.00000
Gen  30 | Best x = -0.00000 | Fitness = 0.00000
Gen  31 | Best x = 0.00000 | Fitness = 0.00000
Gen  32 | Best x = 0.00000 | Fitness = 0.00000
Gen  33 | Best x = -0.33052 | Fitness = 0.00000
Gen  34 | Best x = 0.00000 | Fitness = 0.00000
Gen  35 | Best x = 0.75369 | Fitness = 0.00000
Gen  36 | Best x = 0.00000 | Fitness = 0.00000
Gen  37 | Best x = -1.51123 | Fitness = 0.00000
Gen  38 | Best x = 0.00000 | Fitness = 0.00000
Gen  39 | Best x = 0.00000 | Fitness = 0.00000
Gen  40 | Best x = 0.00000 | Fitness = 0.00000
Gen  41 | Best x = 0.00000 | Fitness = 0.00000
Gen  42 | Best x = 0.00000 | Fitness = 0.00000
Gen  43 | Best x = 0.00000 | Fitness = 0.00000
Gen  44 | Best x = 0.00000 | Fitness = 0.00000
Gen  45 | Best x = 0.00000 | Fitness = 0.00000
Gen  46 | Best x = 0.00000 | Fitness = 0.00000
Gen  47 | Best x = 0.00000 | Fitness = 0.00000
Gen  48 | Best x = -1.22670 | Fitness = 0.00000
Gen  49 | Best x = 0.00000 | Fitness = 0.00000
Gen  50 | Best x = 0.00000 | Fitness = 0.00000
Gen  51 | Best x = 0.00000 | Fitness = 0.00000
Gen  52 | Best x = 0.00000 | Fitness = 0.00000
Gen  53 | Best x = 0.00000 | Fitness = 0.00000
Gen  54 | Best x = 0.00000 | Fitness = 0.00000
Gen  55 | Best x = 0.00000 | Fitness = 0.00000
Gen  56 | Best x = 0.00000 | Fitness = 0.00000
Gen  57 | Best x = 0.00000 | Fitness = 0.00000
Gen  58 | Best x = -0.19375 | Fitness = 0.00000
Gen  59 | Best x = 0.00000 | Fitness = 0.00000
Gen  60 | Best x = 0.00000 | Fitness = 0.00000
Gen  61 | Best x = 0.00000 | Fitness = 0.00000
Gen  62 | Best x = 0.00000 | Fitness = 0.00000
Gen  63 | Best x = 0.00000 | Fitness = 0.00000
Gen  64 | Best x = 0.00000 | Fitness = 0.00000
Gen  65 | Best x = 0.00000 | Fitness = 0.00000
```

```
Gen  66 | Best x = 0.00000 | Fitness = 0.00000
Gen  67 | Best x = 0.00000 | Fitness = 0.00000
Gen  68 | Best x = 0.00000 | Fitness = 0.00000
Gen  69 | Best x = -0.08670 | Fitness = 0.00000
Gen  70 | Best x = 0.00000 | Fitness = 0.00000
Gen  71 | Best x = 0.00000 | Fitness = 0.00000
Gen  72 | Best x = 0.00000 | Fitness = 0.00000
Gen  73 | Best x = 0.00000 | Fitness = 0.00000
Gen  74 | Best x = 0.00000 | Fitness = 0.00000
Gen  75 | Best x = 1.57480 | Fitness = 0.00000
Gen  76 | Best x = 0.00000 | Fitness = 0.00000
Gen  77 | Best x = 0.00000 | Fitness = 0.00000
Gen  78 | Best x = 0.00000 | Fitness = 0.00000
Gen  79 | Best x = 0.00000 | Fitness = 0.00000
Gen  80 | Best x = 0.00000 | Fitness = 0.00000
Gen  81 | Best x = 0.00000 | Fitness = 0.00000
Gen  82 | Best x = 0.00000 | Fitness = 0.00000
Gen  83 | Best x = 0.00000 | Fitness = 0.00000
Gen  84 | Best x = 0.00000 | Fitness = 0.00000
Gen  85 | Best x = 0.00000 | Fitness = 0.00000
Gen  86 | Best x = 0.00000 | Fitness = 0.00000
Gen  87 | Best x = 0.00000 | Fitness = 0.00000
Gen  88 | Best x = 0.00000 | Fitness = 0.00000
Gen  89 | Best x = 0.00000 | Fitness = 0.00000
Gen  90 | Best x = 0.00000 | Fitness = 0.00000
Gen  91 | Best x = 0.00000 | Fitness = 0.00000
Gen  92 | Best x = 0.00000 | Fitness = 0.00000
Gen  93 | Best x = 0.00000 | Fitness = 0.00000
Gen  94 | Best x = 0.00000 | Fitness = 0.00000
Gen  95 | Best x = 0.27095 | Fitness = 0.00000
Gen  96 | Best x = 2.21221 | Fitness = 0.00000
Gen  97 | Best x = 0.00000 | Fitness = 0.00000
Gen  98 | Best x = 0.00000 | Fitness = 0.00000
Gen  99 | Best x = 0.00000 | Fitness = 0.00000

Best solution found:
x = 4.956510887211463e-16
f(x) = 2.4567000175045765e-31
```

Paper Chosen: Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning

Abstract:

Abstract In this study, an improved crossover operator is suggested, for solving path planning problems using genetic algorithms (GA) in static environment. GA has been widely applied in path optimization problem which consists in finding a valid and feasible path between two positions while avoiding obstacles and optimizing some criteria such as distance (length of the path), safety (the path must be as far as possible from the obstacles) ...etc. Several researches have provided new approaches used GA to produce an optimal path. Crossover operators existing in the literature can generate infeasible paths, most of these methods dont take into account the variable length chromosomes. The proposed crossover operator avoids premature convergence and offers feasible paths with better fitness value than its parents, thus the algorithm converges more rapidly. A new fitness function which takes into account the distance, the safety and the energy, is also suggested. In order to prove the validity of the proposed method, it is applied to many different environments and compared with three studies in the literature. The simulation results show that using GA with the improved crossover operators and the fitness function helps to find optimal solutions compared to other methods

Comparison with existing methods:

- Existing GA approaches often generate infeasible paths, converge slowly, or produce paths with many turns.
- The proposed method reduces premature convergence and maintains population diversity.
- It consistently finds paths with **fewer turns** and **better safety**, while converging in **fewer iterations**.

Results:

- In all tested environments, the proposed method achieved the **lowest number of turns** compared to other methods.
- For example, in Env03, the proposed approach reduced the average number of turns to **7.6**, compared to **23.6**, **11.9**, and **23.8** for the other methods.
- The algorithm also converged faster or comparably in terms of iterations.

Conclusion:
The proposed GA with the improved crossover operator and fitness function outperforms existing GA-based path planning methods by producing **smoother, safer, and more energy-efficient paths** with **faster convergence**.