## Code:

```
import numpy as np
import random

class Chromosome:
    def __init__(self, genes):
        self.genes = genes
        self.fitness = 0
        self.calculate_fitness()

    def calculate_fitness(self):
        self.fitness = 0
        for i,gene in enumerate(self.genes):
            self.fitness += (i+1)*(gene**2)

class GeneticAlgorithm:
    def __init__(self):
        self.n_chromosomes = 0
        self.n_genes = 0
        self.population = []
        self.max_fitness = 0
        # self.fitness_graph = []
        self.generate_population()
        self.calculate_max_fitness()

    def generate_population(self):
        self.n_chromosomes = int(input("Enter number of chromosomes: "))
        self.n_genes = int(input("Enter number of genes: "))
        print("Chromosomes: {}\nGenes: {}\n".format(self.n_chromosomes,self.n_genes))
        self.population = []
        for i in range(self.n_chromosomes):
            genes = np.random.standard_normal(size=self.n_genes).tolist()
            self.population.append(Chromosome(genes))

    def calculate_max_fitness(self):
        self.max_fitness = 0
        for i in range(self.n_genes):
            self.max_fitness += (i+1)*100

    def selection(self):
        population_size = (self.n_chromosomes*2)//2
        selected_population = []
        while len(selected_population) != self.n_chromosomes:
            # Creating tournament
```

```
            tournament_size = population_size//2 if population_size//10 < 5 else size//10
            tournament_population = random.sample(self.population, tournament_size)
            # Appending winner in selected population to generate offsprings
            winner = max(tournament_population, key = lambda chromosome :
chromosome.fitness)
            #print(winner.genes)
            selected_population.append(winner)
        return selected_population

    def cross_over(self, selected_population):
        offsprings = []
        while len(offsprings) != self.n_chromosomes:
            # Randomly generating crossover point
            crossover_point = random.randint(0,self.n_genes-2)
            # Randomly selecting two parents
            parent_a, parent_b = random.sample(selected_population, 2)
            # Generating offsprings by swapping genes
            parent_a.genes[crossover_point:self.n_genes],
parent_b.genes[crossover_point:self.n_genes] = parent_b.genes[crossover_point:self.n_genes],
parent_a.genes[crossover_point:self.n_genes]
            offsprings += [parent_a, parent_b]
        return offsprings

    def mutation(self, offsprings):
        mutation_range = random.randint(1,len(offsprings)//3)
        random_offsprings = random.sample(range(0, len(offsprings)-1), mutation_range)
        for i in random_offsprings:
            index = random.randint(0,self.n_genes-1)
            offsprings[i].genes[index] = random.uniform(0, 10)
        return offsprings

    def replacement(self, offsprings):
        self.population = offsprings
        for i in range(self.n_chromosomes):
            self.population[i].calculate_fitness()

    def check_termination(self, i):
        best_chromosome = max(self.population, key = lambda chromosome :
chromosome.fitness)
        if i%10==0:
          print("Best chromosome after iteration {}: {} \nFitness: {}\
n".format(i,best_chromosome.genes,best_chromosome.fitness))
        # self.fitness_graph.append(best_chromosome.fitness)
        if best_chromosome.fitness >= self.max_fitness:
            print("Termination criteria reached")
```

```
            return True
        return False


genetic_algorithm = GeneticAlgorithm()
for i in range(50):
    selected_population = genetic_algorithm.selection()
    offsprings = genetic_algorithm.cross_over(selected_population)
    offsprings = genetic_algorithm.mutation(offsprings)
    genetic_algorithm.replacement(offsprings)
    if genetic_algorithm.check_termination(i+1):
        break
```

Output: