CODE:

```
import random

# Define the problem: f(x) = x^2
def fitness(x):
    return x**2

# Initialize the population (create random individuals)
def create_population(pop_size, lower_bound, upper_bound):
    return [random.uniform(lower_bound, upper_bound) for _ in range(pop_size)]

# Select parents based on fitness (tournament selection)
def select_parents(population, fitness_values):
    total_fitness = sum(fitness_values)
    selection_probs = [f / total_fitness for f in fitness_values]
    return random.choices(population, weights=selection_probs, k=2)

# Crossover between two parents to create two offspring
def crossover(parent1, parent2):
    crossover_point = random.uniform(0, 1)
    offspring1 = crossover_point * parent1 + (1 - crossover_point) * parent2
    offspring2 = crossover_point * parent2 + (1 - crossover_point) * parent1
    return offspring1, offspring2

# Mutation (randomly change a value)
def mutate(offspring, mutation_rate, lower_bound, upper_bound):
    if random.random() < mutation_rate:
        offspring += random.uniform(-0.1, 0.1)  # Smaller mutation step for finer adjustment
        offspring = max(min(offspring, upper_bound), lower_bound)  # Keep within bounds
    return offspring
```

```
# Main Genetic Algorithm function
def genetic_algorithm(pop_size, generations, mutation_rate, lower_bound,
upper_bound):
    population = create_population(pop_size, lower_bound, upper_bound)
    best_solution = None
    best_fitness = -float('inf')

    for generation in range(generations):
        # Evaluate fitness
        fitness_values = [fitness(individual) for individual in population]

        # Track the best solution
        max_fitness = max(fitness_values)
        if max_fitness > best_fitness:
            best_fitness = max_fitness
            best_solution = population[fitness_values.index(max_fitness)]

        # Create new population using selection, crossover, and mutation
        new_population = []
        while len(new_population) < pop_size:
            parent1, parent2 = select_parents(population, fitness_values)
            offspring1, offspring2 = crossover(parent1, parent2)
            new_population.append(mutate(offspring1, mutation_rate, lower_bound,
upper_bound))
            new_population.append(mutate(offspring2, mutation_rate, lower_bound,
upper_bound))

        # Replace old population with new population
        population = new_population

    return best_solution, best_fitness

# Parameters
```

```
pop_size = 100
generations = 200  # Increase generations for better refinement
mutation_rate = 0.05  # Lower mutation rate for finer control
lower_bound = -10
upper_bound = 10

# Run the algorithm
best_solution, best_fitness = genetic_algorithm(pop_size, generations,
mutation_rate, lower_bound, upper_bound)
print(f"Best Solution: x = {best_solution}, Fitness = {best_fitness}")
```

OUTPUT:

```
Best Solution: x = 9.998198198825868, Fitness = 99.96396722300483
```