```
import random

# Objective function
def objective_function(x):
    return x**2 - 2*x + 10

# Initialize population
def initialize_population(pop_size, x_min, x_max):
    return [random.uniform(x_min, x_max) for _ in range(pop_size)]

# Evaluate fitness
def evaluate_fitness(population):
    return [objective_function(x) for x in population]

# Selection: Tournament Selection
def select_parents(population, fitness, tournament_size=3):
    selected = []
    for _ in range(len(population)):
        competitors = random.sample(list(zip(population, fitness)), tournament_size)
        winner = min(competitors, key=lambda x: x[1])
        selected.append(winner[0])
    return selected

# Crossover: Arithmetic crossover
def crossover(parents, crossover_rate=0.8):
    offspring = []
    for i in range(0, len(parents), 2):
        p1, p2 = parents[i], parents[(i+1) % len(parents)]
        if random.random() < crossover_rate:
            alpha = random.random()
            child1 = alpha * p1 + (1 - alpha) * p2
            child2 = alpha * p2 + (1 - alpha) * p1
        else:
            child1, child2 = p1, p2
        offspring.extend([child1, child2])
    return offspring

# Mutation: Add small random noise
def mutate(offspring, mutation_rate=0.1, mutation_step=0.5):
    for i in range(len(offspring)):
        if random.random() < mutation_rate:
            offspring[i] += random.uniform(-mutation_step, mutation_step)
    return offspring

# Gene Expression Algorithm
def gene_expression_algorithm(pop_size, generations, x_min, x_max):
    population = initialize_population(pop_size, x_min, x_max)
    best_solution = None
    best_fitness = float("inf")

    for gen in range(generations):
        fitness = evaluate_fitness(population)
        if min(fitness) < best_fitness:
            best_fitness = min(fitness)
            best_solution = population[fitness.index(best_fitness)]

        parents = select_parents(population, fitness)
        offspring = crossover(parents)
        population = mutate(offspring)

    return best_solution, best_fitness

pop_size = 20
```

```
generations = 100
x_min, x_max = -10, 10

best_x, best_fitness = gene_expression_algorithm(pop_size, generations, x_min, x_max)
print(f"Best solution: x = {best_x:.5f}, Minimum value: f(x) = {best_fitness:.5f}")
```

```
Best solution: x = 1.00000, Minimum value: f(x) = 9.00000
```