EXPERIMENT 3.3

Student Name: Samuel **UID: 24MAI10018**
Branch: AIML **Section/Group: 24MAI-1**
Semester: 2nd **Date of Performance: / /2025**
Subject Name: SC Lab **Subject Code: 24CSH–668**

AIM: Implement a complex optimization problem such as banana function etc. using particle swarm optimization (PSO).

SOFTWARE REQUIED: Jupyter notebook

THEORY:

Optimization problems defined by functions for which derivatives are unavailable or available at a prohibitive cost are appearing more and more frequently in computational science and engineering. Increasing complexity in mathematical modelling, higher sophistication of scientific computing, and abundance of legacy codes are some of the reasons why derivative-free optimization is currently an area of great demand. Difficulties and challenges arise from multiple sources: expensive function evaluation, black-box/legacy codes, noise and uncertainty, unknown a priori function domains, and hiddenconstraints.

Derivative-Free Optimization Method:

Derivative-free optimization involves problems where derivatives are impractical, using methods that don't require them. One such method is Particle Swarm Optimization (PSO).

Particle Swarm Optimization (PSO):

Proposed by J. Kennedy and R. Eberhart in 1995, PSO is popular for continuous optimization and handling multiple targets. It moves particles (positions in space) with velocity calculated each iteration, influenced by their best-known position and the best-known in the search space. PSO converges toward the optimal solution without needing differentiability, making it suitable for nonlinear problems.

The process starts with a random population, evaluates fitness, selects fitter particles, and forms a new generation. This repeats until the algorithm reaches a maximum number of generations or a satisfactory fitness level.

.

ALGORITHM:

1. Initialize Parameters:
    - Define the number of particles (swarm size).
    - Set the maximum number of iterations (generations).
    - Initialize the position and velocity of each particle randomly in the search space.
2. Define the Objective Function:
    - Use a complex function like the Rosenbrock function (Banana function) as the objective to minimize.
3. Initialize Fitness and Best Positions:
    - For each particle, evaluate the fitness using the objective function.
    - Set each particle's initial position and fitness as its personal best (pBest).
4. Initialize Global Best (gBest):
    - Find the best fitness value from all particles and set it as the global best (gBest).
5. Update Particle Velocity:
    - For each particle, update its velocity using the formula:

$vi=w\cdot vi+c1\cdot rand(0,1)\cdot(pBesti-xi)+c2\cdot rand(0,1)\cdot(gBest-xi)v_i = w \cdot v_i + c_1 \cdot \text{rand}(0,1) \cdot (pBest_i - x_i) + c_2 \cdot \text{rand}(0,1) \cdot (gBest - x_i)$

where:

    - viv_i is the velocity,
    - ww is the inertia weight,
    - $c1c_1$ and $c2c_2$ are learning factors,
    - $rand(0,1)\text{rand}(0,1)$ is a random number between 0 and 1.
6. Update Particle Position:
    - For each particle, update its position using the formula:

$xi=xi+vix_i = x_i + v_i$

where:

    - xix_i is the position of the particle,
    - viv_i is the updated velocity.
7. Evaluate Fitness:
    - Calculate the fitness value of each particle based on its new position using the objective function.
8. Update Personal Best (pBest):
    - If the current fitness is better than the personal best fitness, update the particle’s personal best (pBest).
9. Update Global Best (gBest):

- If any particle has a better fitness than the global best, update the global best (gBest).

10. Check Termination Condition:
    - If the maximum number of generations is reached or the global best fitness value stops improving, stop the algorithm.
11. Return the Optimal Solution:
    - Output the global best position (gBest) and its fitness value as the solution

SOURCE CODE:

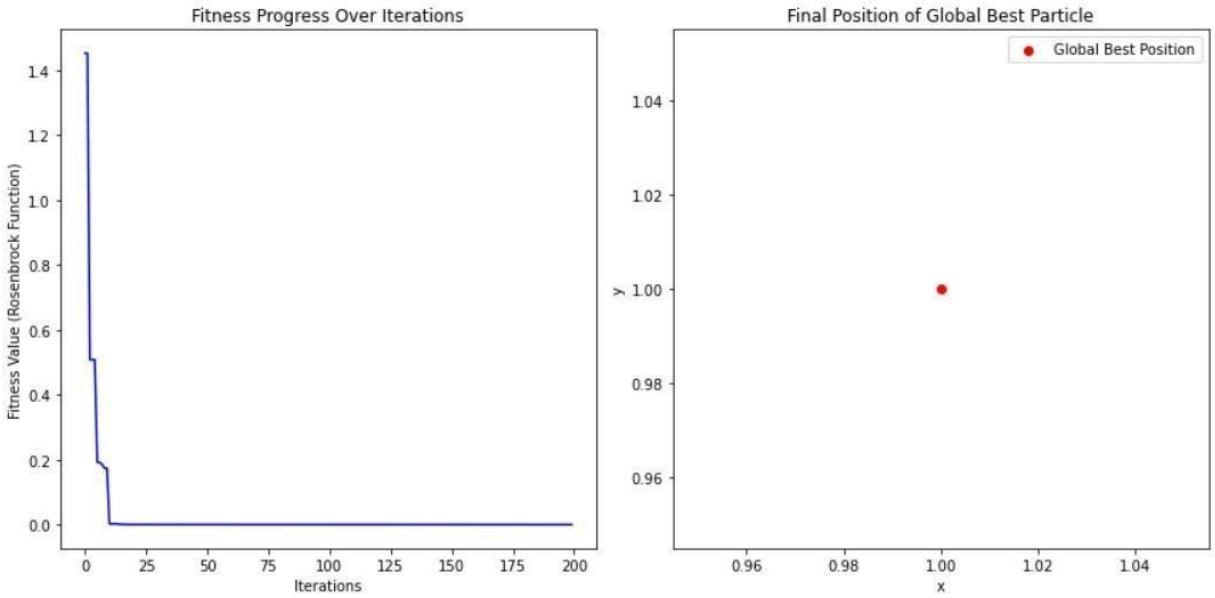
```
import numpy as np
import matplotlib.pyplot as plt

# Rosenbrock (Banana) function
def rosenbrock(x, y, a=1, b=100):
    return (a - x)**2 + b * (y - x**2)**2

# Particle Swarm Optimization (PSO)
def particle_swarm_optimization(rosenbrock_func, n_particles=30, n_iterations=100,
lb=-5, ub=5):
    # Initialize the positions and velocities of particles
    positions = np.random.uniform(lb, ub, (n_particles, 2))  # Position matrix (x, y)
    velocities = np.random.uniform(-1, 1, (n_particles, 2)) # Velocity matrix (vx, vy)
    personal_best = positions.copy()  # Best known position for each particle
    personal_best_scores = np.array([rosenbrock_func(p[0], p[1]) for p in positions])
    global_best = personal_best[np.argmin(personal_best_scores)]  # Global best
position
    global_best_score = np.min(personal_best_scores)

    # Store the fitness values for each iteration
    fitness_progress = []

    # PSO parameters
    w = 0.5  # Inertia weight
    c1 = 1.5 # Cognitive coefficient
    c2 = 1.5  # Social coefficient

    # PSO iterations
```

```
    for iteration in range(n_iterations):
        # Update velocities and positions of particles
        for i in range(n_particles):
            # Update velocity
            velocities[i] = (w * velocities[i] +
                             c1 * np.random.random() * (personal_best[i] - positions[i]) +
                             c2 * np.random.random() * (global_best - positions[i]))

            # Update position positions[i]
            += velocities[i]
            positions[i] = np.clip(positions[i], lb, ub)  # Ensure particles stay within
bounds

            # Evaluate fitness of the new position
            fitness = rosenbrock_func(positions[i][0], positions[i][1])

            # Update personal best if the current position is better
            if fitness < personal_best_scores[i]:
                personal_best[i] = positions[i]
                personal_best_scores[i] = fitness

        # Update global best
        current_global_best_score = np.min(personal_best_scores)
        if current_global_best_score < global_best_score:
            global_best_score = current_global_best_score
            global_best = personal_best[np.argmin(personal_best_scores)]

        # Store the global best fitness value
        fitness_progress.append(global_best_score)

    return global_best, global_best_score, fitness_progress

# Run PSO to optimize the Rosenbrock function
best_position, best_score, fitness_progress =
particle_swarm_optimization(rosenbrock, n_particles=30, n_iterations=200)

# Print the results
```

```
print(f"Best Position: {best_position}")
print(f"Best Score (Fitness): {best_score}")

# Plotting the results
plt.figure(figsize=(12, 6))

# Plot the fitness progress over iterations
plt.subplot(1, 2, 1)
plt.plot(fitness_progress, color='blue', label='Global Best Fitness')
plt.title("Fitness Progress Over Iterations")
plt.xlabel("Iterations")
plt.ylabel("Fitness Value (Rosenbrock Function)")

# Plot the convergence in the 2D space (final positions of particles)
plt.subplot(1, 2, 2)
plt.scatter(best_position[0], best_position[1], color='red', label='Global Best Position')
plt.title("Final Position of Global Best Particle")
plt.xlabel("x")
plt.ylabel("y") plt.legend()

plt.tight_layout() plt.show()
```

SCREENSHOT OF OUTPUTS:

```
Best Position: [1. 1.]
Best Score (Fitness): 0.0
```



LEARNING OUTCOMES

- Learned how PSO simulates the behavior of particles moving through the search space, adjusting their positions and velocities to find the global optimum.
- Gained hands-on experience in applying PSO to solve the Rosenbrock (Banana) function, which is a common benchmark for optimization algorithms.
- Understood how to track the fitness progression over iterations, and observed the convergence of the algorithm toward the optimal solution.
- Developed the ability to visualize the optimization process through plots, showing both the fitness progress over iterations and the final particle positions in the search space.