EXPERIMENT 3.2

Student Name: Samuel	**UID: 24MAI10016**
Branch: AIML	**Section/Group: 24MAI-1**
Semester: 2	**Date of Performance: / /2025**
Subject Name: SC Lab	**Subject Code: 24CSH–668**

AIM: STUDY AND IMPLEMENT A DERIVATIVE-FREE OPTIMIZATION ON COMPLEX PROBLEMS

SOFTWARE USED : MATLAB

THEORY: Optimization problems often require the calculation of derivatives to find the best solution. However, in many real-world scenarios, derivative information is unavailable, expensive to compute, or unreliable due to noise and discontinuities. **Derivative-Free Optimization (DFO)** methods overcome this limitation by searching for optimal solutions without requiring gradient information. One such powerful DFO technique is **Particle Swarm Optimization (PSO)**, inspired by the collective behavior of birds and fish.

PSO is a population-based stochastic optimization algorithm where a swarm of particles explores the search space by updating their positions based on their own experiences and those of their neighbors. Each particle adjusts its movement using a combination of its current velocity, the best solution it has found so far (**personal best**), and the best solution found by the entire swarm (**global best**). The algorithm is highly effective for nonlinear, non-differentiable, and complex optimization problems such as house price prediction, robotic path planning, and feature selection in machine learning.

In this experiment, PSO is applied to **house price prediction**, where the goal is to optimize a model that predicts house prices based on input parameters such as area. The optimization process minimizes the error between predicted and actual prices, demonstrating the effectiveness of DFO in complex real-world problems.

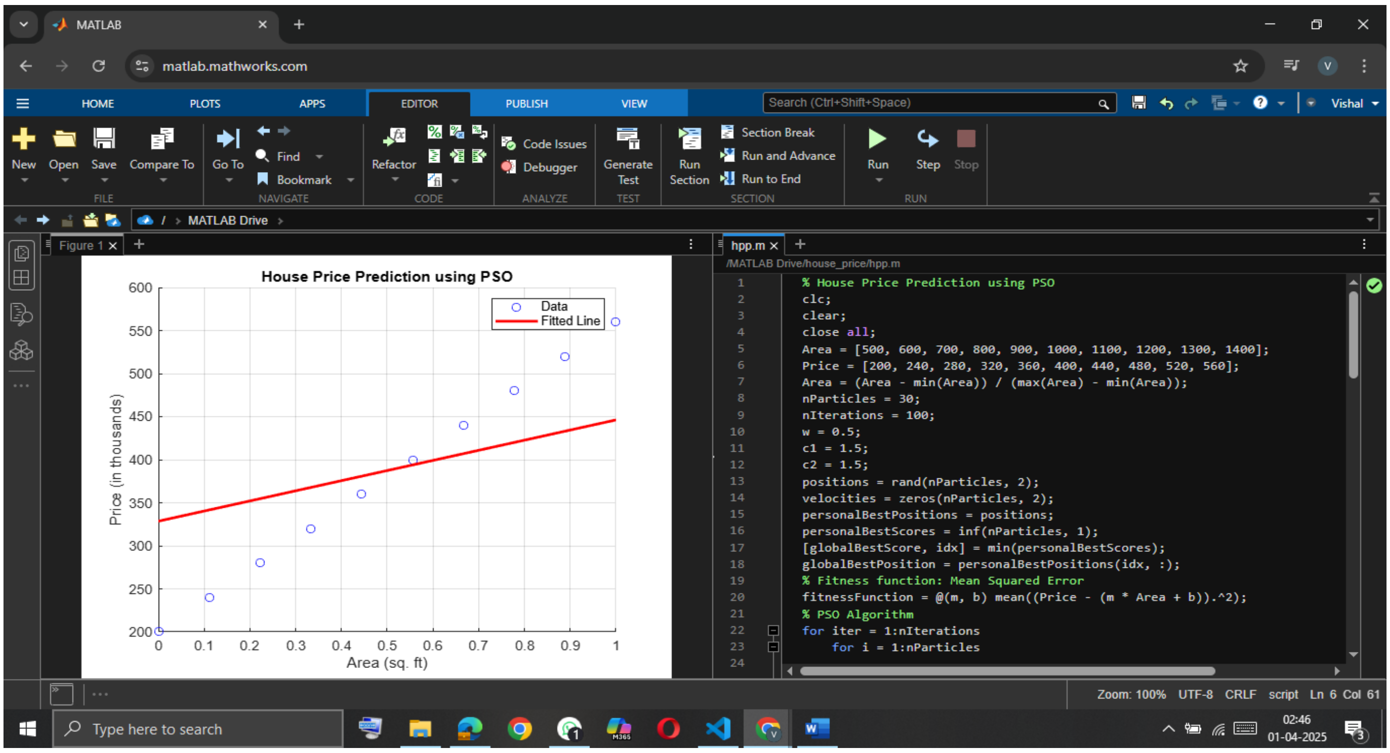
ALGORITHM:

1. **Initialize parameters:** Define the number of particles, maximum iterations, inertia weight, cognitive and social coefficients.
2. **Generate initial population:** Randomly initialize particle positions and velocities.
3. **Evaluate fitness:** Compute the objective function (error between predicted and actual values).
4. **Update personal and global bests:** Each particle updates its personal best if the current position is better. The global best is updated based on the best-performing particle.
5. **Update velocity and position:** Adjust particle velocities using the cognitive and social components and update positions accordingly.
6. **Repeat until convergence:** Continue iterations until the stopping condition is met (e.g., max iterations or minimal error).
7. **Output results:** Return the optimal parameters and visualize the results.

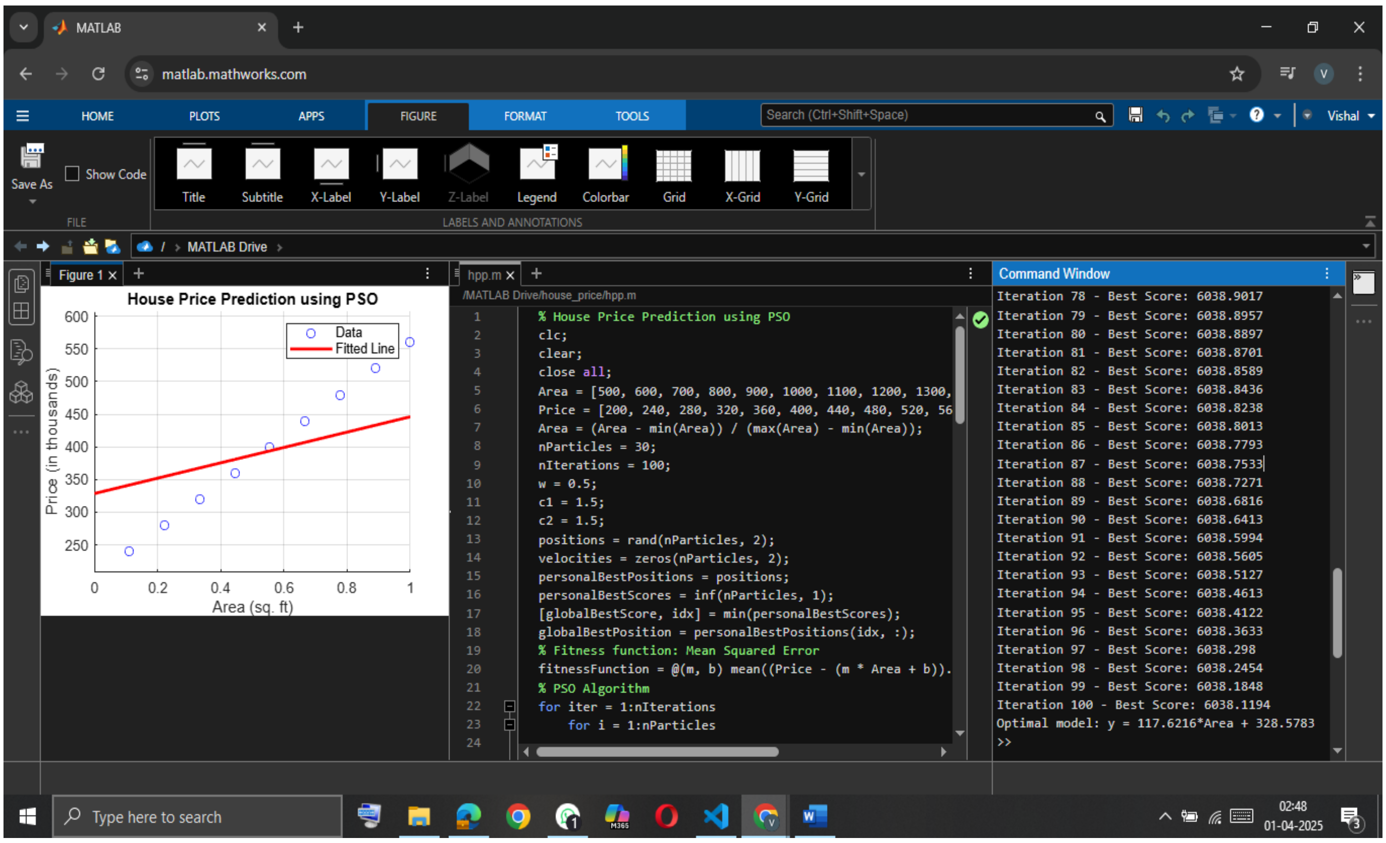
SOURCE CODE

```
% House Price Prediction using PSO
clc;
clear;
close all;
Area = [500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400];
Price = [200, 240, 280, 320, 360, 400, 440, 480, 520, 560];

Area = (Area - min(Area)) / (max(Area) - min(Area));
nParticles = 30;
nIterations = 100;
w = 0.5;
c1 = 1.5;
c2 = 1.5;


positions = rand(nParticles, 2);
velocities = zeros(nParticles, 2);
personalBestPositions = positions;
personalBestScores = inf(nParticles, 1);
[globalBestScore, idx] = min(personalBestScores);
globalBestPosition = personalBestPositions(idx, :);
% Fitness function: Mean Squared Error
fitnessFunction = @(m, b) mean((Price - (m * Area + b)).^2);
% PSO Algorithm
for iter = 1:nIterations
    for i = 1:nParticles

        m = positions(i, 1);
        b = positions(i, 2);
        fitness = fitnessFunction(m, b);
        if fitness < personalBestScores(i)
            personalBestScores(i) = fitness;
            personalBestPositions(i, :) = positions(i, :);
        end


        if fitness < globalBestScore
            globalBestScore = fitness;
            globalBestPosition = positions(i, :);
        end
    end


    disp(['Iteration ' num2str(iter) ' - Best Score: ' num2str(globalBestScore)]);
    drawnow;
    pause(0.1);
    for i = 1:nParticles
        velocities(i, :) = w * velocities(i, :) ...
                         + c1 * rand() * (personalBestPositions(i, :) - positions(i, :)) ...
                         + c2 * rand() * (globalBestPosition - positions(i, :));
        positions(i, :) = positions(i, :) + velocities(i, :);
    end
end
% Final model (m, b)
m_optimal = globalBestPosition(1);
b_optimal = globalBestPosition(2);
% Display the optimal model
disp(['Optimal model: y = ' num2str(m_optimal) '*Area + ' num2str(b_optimal)]);
% Plot the results
figure;
scatter(Area * (max(Area) - min(Area)) + min(Area), Price, 'bo');
hold on;
predictedPrices = m_optimal * Area + b_optimal;
plot(Area * (max(Area) - min(Area)) + min(Area), predictedPrices, 'r-', 'LineWidth', 2);
title('House Price Prediction using PSO');
xlabel('Area (sq. ft)');
ylabel('Price (in thousands)');
legend('Data', 'Fitted Line');
grid on;
```

SCREENSHOT OF OUTPUT:



```
% House Price Prediction using PSO
clc;
clear;
close all;
Area = [500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400];
Price = [200, 240, 280, 320, 360, 400, 440, 480, 520, 560];
Area = (Area - min(Area)) / (max(Area) - min(Area));
nParticles = 30;
nIterations = 100;
w = 0.5;
c1 = 1.5;
c2 = 1.5;
positions = rand(nParticles, 2);
velocities = zeros(nParticles, 2);
personalBestPositions = positions;
personalBestScores = inf(nParticles, 1);
[globalBestScore, idx] = min(personalBestScores);
globalBestPosition = personalBestPositions(idx, :);
% Fitness function: Mean Squared Error
fitnessFunction = @(m, b) mean((Price - (m * Area + b)).^2);
% PSO Algorithm
for iter = 1:nIterations
    for i = 1:nParticles
```



/MATLAB Drive/house_price/hpp.m

```
% House Price Prediction using PSO
clc;
clear;
close all;
Area = [500, 600, 700, 800, 900, 1000, 1100, 1200, 1300,
Price = [200, 240, 280, 320, 360, 400, 440, 480, 520, 56
Area = (Area - min(Area)) / (max(Area) - min(Area));
nParticles = 30;
nIterations = 100;
w = 0.5;
c1 = 1.5;
c2 = 1.5;
positions = rand(nParticles, 2);
velocities = zeros(nParticles, 2);
personalBestPositions = positions;
personalBestScores = inf(nParticles, 1);
[globalBestScore, idx] = min(personalBestScores);
globalBestPosition = personalBestPositions(idx, :);
% Fitness function: Mean Squared Error
fitnessFunction = @(m, b) mean((Price - (m * Area + b)).
% PSO Algorithm
for iter = 1:nIterations
    for i = 1:nParticles
```

```
Iteration 78 - Best Score: 6038.9017
Iteration 79 - Best Score: 6038.8957
Iteration 80 - Best Score: 6038.8897
Iteration 81 - Best Score: 6038.8701
Iteration 82 - Best Score: 6038.8589
Iteration 83 - Best Score: 6038.8436
Iteration 84 - Best Score: 6038.8238
Iteration 85 - Best Score: 6038.8013
Iteration 86 - Best Score: 6038.7793
Iteration 87 - Best Score: 6038.7533
Iteration 88 - Best Score: 6038.7271
Iteration 89 - Best Score: 6038.6816
Iteration 90 - Best Score: 6038.6413
Iteration 91 - Best Score: 6038.5994
Iteration 92 - Best Score: 6038.5605
Iteration 93 - Best Score: 6038.5127
Iteration 94 - Best Score: 6038.4613
Iteration 95 - Best Score: 6038.4122
Iteration 96 - Best Score: 6038.3633
Iteration 97 - Best Score: 6038.298
Iteration 98 - Best Score: 6038.2454
Iteration 99 - Best Score: 6038.1848
Iteration 100 - Best Score: 6038.1194
Optimal model: y = 117.6216*Area + 328.5783
>>
```

LEARNING OUTCOME:

1. **Understanding Derivative-Free Optimization (DFO):** Learn how PSO optimizes functions without requiring gradient information.
2. **Implementation of PSO in MATLAB:** Gain practical experience in implementing PSO for predictive modeling tasks.
3. **Applying PSO to Real-World Problems:** Understand how PSO can be used for house price prediction and other complex problems.
4. **Visualization and Interpretation of Results:** Learn how to analyze and visualize PSO optimization results effectively.