```
    clc
    clear all
    close all
    function Z = objective_function(O)
    % Objective function definition without constraints

    x = O(1);
    y = O(2);

    % Calculate the objective function value
    Z = (1 - x)^2 + 100 * (y - x^2)^2;
end
    %% Problem Definition
    nVar = 2; % Number of variables

    %% PSO Parameters
    MaxIter = 220; % Maximum number of iterations
    nPop = 100; % Population size
    w = 0.7; % Inertia weight
    % d = 0.9; % Daping ratio of inertia
    c1 = 1.2; % Acceleration coefficient for personal best
    c2 = 1.2; % Acceleration coefficient for global best

    %% Initialization
    x0.position = [];
    x0.velocity = [];
    x0.fitness = [];
    x0.best.position = [];
    x0.best.fitness = [];

    % Create a population
    x = repmat(x0, nPop, 1);
    global_best.fitness = inf;

    % Generate initial population
    for i = 1:nPop
        x(i).position = rand(1, nVar) * 3 - 1.5; % Random solutions in a reasonable↙
range (e.g., [-1.5, 1.5])
        x(i).velocity = zeros([1 nVar]); % Initial velocity
        x(i).fitness = objective_function(x(i).position); % Calculate fitness
        x(i).best.position = x(i).position; % Update local best
        x(i).best.fitness = x(i).fitness; % Initialize local best fitness

        if x(i).fitness < global_best.fitness
            global_best.position = x(i).position; % Update global best position
            global_best.fitness = x(i).fitness; % Update global best fitness
        end
    end
```

```
    %% Main PSO Loop
    for iter = 1:MaxIter
        for i = 1:nPop
            % Update velocity using PSO formula
            r1 = rand(); % Random number for cognitive component
            r2 = rand(); % Random number for social component

            x(i).velocity = w * x(i).velocity ...
                          + c1 * r1 * (x(i).best.position - x(i).position) ...
                          + c2 * r2 * (global_best.position - x(i).position);

            % Update position based on velocity
            x(i).position = x(i).position + x(i).velocity;

            % Evaluate fitness of new position
            x(i).fitness = objective_function(x(i).position);

            % Update personal best if current fitness is better
            if x(i).fitness < x(i).best.fitness
                x(i).best.position = x(i).position;
                x(i).best.fitness = x(i).fitness;

                % Update global best if necessary
                if x(i).fitness < global_best.fitness
                    global_best.position = x(i).position;
                    global_best.fitness = x(i).fitness;
                end
            end
        end

        % Optionally display progress or store data here (e.g., plot or log)
        disp(['Iteration ' num2str(iter) ': Best Fitness = ' num2str(global_best.↙
fitness)]);
    end

    %% Final Output
    disp(['Global Best Position: ' num2str(global_best.position)]);
    disp(['Global Best Fitness: ' num2str(global_best.fitness)]);

%% Plotting the Objective Function as a 3D Surface Plot

% Create a grid of values for plotting the objective function surface.
[xGrid, yGrid] = meshgrid(linspace(-1.5, 1.5, 400), linspace(-0.5, 2.5, 400));
ZGrid = arrayfun(@(x,y) objective_function([x,y]), xGrid, yGrid);

% Create the surface plot.
figure;
surf(xGrid, yGrid, ZGrid); % Surface plot of the objective function.
shading interp; % Interpolated shading for smooth color transitions.
```

```
colorbar; % Add color bar to indicate values.
hold on;

% Plot the optimal point found by PSO.
plot3(global_best.position(1), global_best.position(2), global_best.fitness, 'ro',↙
'MarkerSize', 8, 'LineWidth', 2);
title('3D Surface Plot of Objective Function Z');
xlabel('x');
ylabel('y');
zlabel('Objective Function Value (Z)');
legend('Objective Function Surface', 'Optimal Point');
hold off;
```