Shaurya Gupta OTA Assignment 9
21070122154

Study and implement Opposition based learning (OBL) in the initialization phase of PSO. Compare the results of PSO and OBL-PSO.

Program Code for PSO and OBL-PSO:

```
% Parameters
maxIter = 500; % Maximum number of iterations
Npop = 50; % Population size
D = 2; % Dimensionality of the problem
Nruns = 20; % Number of runs
alpha = 2; % Cognitive coefficient
beta = 2; % Social coefficient
lowerBound = -5; % Lower bound of search space
upperBound = 5; % Upper bound of search space

% Define test functions
functions = {@quadratic, @cubic, @quartic, @mixed_poly, @biquadratic};
functionNames = {'Quadratic', 'Cubic', 'Quartic', 'Mixed Polynomial',
'Biquadratic'};

% Initialize storage for results
allBestFitHistory_pso = zeros(Nruns, length(functions), maxIter);
allMeanFitHistory_pso = zeros(Nruns, length(functions), maxIter);
allBestFitHistory_obl = zeros(Nruns, length(functions), maxIter);
allMeanFitHistory_obl = zeros(Nruns, length(functions), maxIter);

for funcIdx = 1:length(functions)
    f = functions{funcIdx};
    fprintf('Optimizing Function %s\n', functionNames{funcIdx});
```

```
for run = 1:Nruns
    % PSO Initialization
    X_pso = rand(Npop, D) * (upperBound - lowerBound) + lowerBound; % Initialize particles within [lowerBound, upperBound]
    fit_pso = arrayfun(@(idx) f(X_pso(idx, :)), 1:Npop);

    % OBL-PSO Initialization
    X_obl = rand(Npop, D) * (upperBound - lowerBound) + lowerBound; % Initialize particles within [lowerBound, upperBound]
    X_obl_opposite = lowerBound + upperBound - X_obl; % Generate opposite particles
    fit_obl = arrayfun(@(idx) f(X_obl(idx, :)), 1:Npop);
    fit_obl_opposite = arrayfun(@(idx) f(X_obl_opposite(idx, :)), 1:Npop);

    % Keep better particles between original and opposite
    for i = 1:Npop
        if fit_obl_opposite(i) < fit_obl(i)
            X_obl(i, :) = X_obl_opposite(i, :); % Replace with opposite if better
            fit_obl(i) = fit_obl_opposite(i);
        end
    end

    % Main loop for iterations
    for it = 1:maxIter
        % PSO Update
        for i = 1:Npop
            r1 = rand;
```

```
    r2 = rand;
    v_pso = alpha * r1 * (X_pso(randi(Npop), :) - X_pso(i, :)) + ...
        beta * r2 * (min(X_pso) - X_pso(i, :)); % Standard PSO update
    X_pso(i, :) = X_pso(i, :) + v_pso; % Update position

    % Clamp particle position within the search space
    X_pso(i, :) = max(min(X_pso(i, :), upperBound), lowerBound);
    fit_pso(i) = f(X_pso(i, :)); % Evaluate fitness
end

% OBL-PSO Update
for i = 1:Npop
    r1 = rand;
    r2 = rand;
    v_obl = alpha * r1 * (X_obl(randi(Npop), :) - X_obl(i, :)) + ...
        beta * r2 * (min(X_obl) - X_obl(i, :)); % OBL-PSO update
    X_obl(i, :) = X_obl(i, :) + v_obl; % Update position

    % Clamp particle position within the search space
    X_obl(i, :) = max(min(X_obl(i, :), upperBound), lowerBound);
    fit_obl(i) = f(X_obl(i, :)); % Evaluate fitness
end

% Store the best and mean fitness for PSO and OBL-PSO
allBestFitHistory_pso(run, funcIdx, it) = min(fit_pso);
allMeanFitHistory_pso(run, funcIdx, it) = mean(fit_pso);
allBestFitHistory_obl(run, funcIdx, it) = min(fit_obl);
```

```
        allMeanFitHistory_obl(run, funcIdx, it) = mean(fit_obl);
    end
  end
end

% Calculate mean and standard deviation over all runs
meanBestFit_pso = mean(allBestFitHistory_pso, 1);
meanMeanFit_pso = mean(allMeanFitHistory_pso, 1);
stdBestFit_pso = std(allBestFitHistory_pso, 0, 1);
stdMeanFit_pso = std(allMeanFitHistory_pso, 0, 1);

meanBestFit_obl = mean(allBestFitHistory_obl, 1);
meanMeanFit_obl = mean(allMeanFitHistory_obl, 1);
stdBestFit_obl = std(allBestFitHistory_obl, 0, 1);
stdMeanFit_obl = std(allMeanFitHistory_obl, 0, 1);

% Plot iteration-wise mean function values for PSO
figure;
hold on;
for funcIdx = 1:length(functions)
  plot(1:maxIter, squeeze(meanMeanFit_pso(:, funcIdx, :)), 'LineWidth', 2, 'DisplayName', ['PSO ' functionNames{funcIdx}]);
end
xlabel('Iteration');
ylabel('Mean Function Value');
title('Iteration-wise Mean Function Values for PSO');
legend('show');
hold off;
```

```
% Plot iteration-wise mean function values for OBL-PSO
figure;
hold on;
for funcIdx = 1:length(functions)
    plot(1:maxIter, squeeze(meanMeanFit_obl(:, funcIdx, :)), '--', 'LineWidth', 2, 'DisplayName', ['OBL-PSO ' functionNames{funcIdx}]);
end
xlabel('Iteration');
ylabel('Mean Function Value');
title('Iteration-wise Mean Function Values for OBL-PSO');
legend('show');
hold off;

% Output comparison table
fprintf('--- PSO vs OBL-PSO Results ---\n');
fprintf('%-20s %-20s %-20s %-20s %-20s\n', 'Function', 'PSO Mean Best', 'OBL-PSO Mean Best', 'PSO Std Dev', 'OBL-PSO Std Dev');
for funcIdx = 1:length(functions)
    fprintf('%-20s %-20.6f %-20.6f %-20.6f %-20.6f\n', ...
        functionNames{funcIdx}, ...
        mean(meanBestFit_pso(:, funcIdx, :)), ...
        mean(meanBestFit_obl(:, funcIdx, :)), ...
        mean(stdBestFit_pso(:, funcIdx, :)), ...
        mean(stdBestFit_obl(:, funcIdx, :)));
end

% Test functions
```

```
function val = quadratic(pos)
    x = pos(1);
    y = pos(2);
    val = x^2 + y^2;
end

function val = cubic(pos)
    x = pos(1);
    y = pos(2);
    val = x^2 + y^2 + x*y;
end

function val = quartic(pos)
    x = pos(1);
    y = pos(2);
    val = x^4 + y^4;
end

function val = mixed_poly(pos)
    x = pos(1);
    y = pos(2);
    val = x^2 + y^2 + 2*x*y;
end

function val = biquadratic(pos)
    x = pos(1);
    y = pos(2);
```

```
  val = x^2 + y^2 + x^2*y^2;

end
```

Output:

```
>> pso_obl
Optimizing Function Quadratic
Optimizing Function Cubic
Optimizing Function Quartic
Optimizing Function Mixed Polynomial
Optimizing Function Biquadratic
--- PSO vs OBL-PSO Results ---
Function             PSO Mean Best        OBL-PSO Mean Best    PSO Std Dev          OBL-PSO Std Dev
Quadratic            49.421296            49.507397            0.447887             0.405490
Cubic                74.167007            74.107331            0.549215             0.638609
Quartic              1234.124879          1233.175740          8.450674             11.584834
Mixed Polynomial     98.642721            98.921811            0.514090             0.912162
Biquadratic          664.651070           666.029845           7.063879             6.702553
>>
```

Shaurya Gupta OTA Assignment 9
21070122154