# MSE 426 – Introduction to Engineering Design Optimization

# Coding Assignment 3 – Particle Swarm Optimization (PSO)

**Instructor:** [redacted]

**Submitted By: Sepehr Rezvani**

**Submitted To:** [redacted]

**Submission Date: March 21st, 2021**

# Table of Contents

## Introduction

In this coding assignment, three functions are optimized using Particle Swarm Optimization. Depending on the questions, there are variety of possible number of particles for each swarms. The strength of PSO is that particles have individual, in addition to group memory. Therefore, they will adjust their values based on both at every iteration. First, velocity for each particle at every iteration will be calculated based on information from previous iteration, eq. 1. Then, particles will update their position based on previous iteration position, and velocity calculated in eq. 1.

$$v_i^{t+1} = wv_i^t + c_1r_1(x_p^t - x_i^t) + c_2r_2(x_g^t - x_i^t) \tag{1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{2}$$

Where:

$w$: inertia weight → [0.5 0.9]

$c_1$; $c_2$: learning factors → 2

$r_1$; $r_2$: random parameters → range of [0 1] and randomly generated

$x_p$: best personal position for each particle

$x_g$: best group position

It is important to note, the MATLAB code contains a main 'while' loop, and inside there are 'for' loops that iterate from 1 to chosen number of particles. Then the main 'while' loop iterated until the stopping criteria is met.

## Results

For this coding assignment, three different functions were optimized using PSO. However, the third function is identical to the second, with difference of having a constraint. For this problem, one dimensional line-search methods were implemented to find local minimum, since PSO method cannot directly solve constraint optimization problems.

Note: MATLAB codes are included in 'Code' section of this report.

Table 1 includes default parameters used in MATLAB code. If different values are used, it will be stated in the report.

**Table 1 – Default Parameters**

| **Parameter** | **Default** |
|---|---|
| nparticles | The number of particles is set to 30 |
| Initial x | Uniformly (-100,100) distributed random matrix of length nparticles x nvars |
| Initial v | Uniformly (-1000,1000) distributed random matrix of length nparticles x nvars |
| Maximum v | 100 |
| nvars | Number of variables |
| w | 0.7 |
| r1 and r2 | Uniformly (0,1) distributed random vector of length nvars |
| c1 and c2 | 2.0 |
| Function Tolerance | |

|  |  |
|---|---|
|  | 1e-6 |
| Max iterations | 400 |

## Question 1:

**Q:** Write the code for PSO in Matlab or a programming language of your choice. Solve the following problems studied in Assignment 2. The default parameters to be used are outlined in Table 1.

$$\min f = x^2 + y^2 - xy - 4x - y \tag{3}$$

$$\min f = (1-x)^2\,(-x^2 + y)^2 \tag{4}$$

So, comparison between assignment 2 and 3 results are shown in Table 1.

**Table 2 – Function Minimization Comparison between Assignment 2 and 3**

| Assignment / Method | 2 / BFGS Method | 3 / PSO Method |
|---|---|---|
| **Number of Iterations** | 2 | 70 |
| **Results (Minimum Function Value for eq. 3)** | -7.00000000000000000000 | -6.99999858507197814106 |
| **Number of Iterations** | 3 | 86 |
| **Results (Minimum Function Value for eq. 4)** | 0.00000019805527732932 | 0.00000237406447043459 |

## Question 2:

**Q:** For both problems run PSO five times, are report/discuss your solutions.

### Table 3

| Problems | 1 → eq. 3 | 2 → eq. 4 |
|---|---|---|
| **Run 1** | Iter 338 // function value: -6.99999587959972391360 | Iter 400 // function value: 1.01071286525493109210 |
| **Run 2** | Iter 68 // function value: -6.99999870477105190503 | Iter 52 // function value: 0.00000534829715894040 |
| **Run 3** | Iter 64 // function value: -6.99999891714958444311 | Iter 48 // function value: 0.00000305566200800805 |
| **Run 4** | Iter 53 // function value: -6.99999866799278969154 | Iter 46 // function value: 0.00000174410222822840 |
| **Run 5** | Iter 51 // function value: -6.99999761894263183848 | Iter 42 // function value: 0.00000534190799208660 |

Results match what was found in coding assignment 2, that was done with BFGS. Difference is the number of iterations however. It appears PSO takes much longer (more iterations), compared to linear search methods.

## Question 3:

Q: Change the number of particles to 10, 100, 1000. Discuss your observations.

### Table 4

| Problems | 1 → eq. 3 | 2 → eq. 4 |
|---|---|---|
| nparticles = 30 | -6.99999876494021933127 | 0.00000167856860547032 |

| | | |
|---|---|---|
| nparticles = 10 | -6.99999758664027460497 | 0.00000539770725603175 |
| nparticles = 100 | -6.99999892901522891009 | 0.00000862121136040550 |
| nparticles = 1000 | -6.99999882120566407906 | 0.00000103235434878180 |

As number of particles increased, it did not have a major effect on the convergence of optimum value. For equation 3, in terms of total iteration until convergence, as nparticles increased 30 → 1000, it took less iterations to reach minimum point. However, that was not the case for equation 4.

## Question 4:

**Q:** Change both $c_1$ and $c_2$ to 0.5 then change both to 5. Discuss your observations.

**Table 5**

| **Problems** | **1 → eq. 3** | **2 → eq. 4** |
|---|---|---|
| $c_1 = c_2 = 2$ | -6.99999891482439817736 | 0.00000173888164118577 |
| $c_1 = c_2 = 0.5$ | -6.99999595107057892562 | 0.00000973750931576916 |
| $c_1 = c_2 = 5$ | -4.46548798602873286967 | 1.04138287210918978332 |

Increasing or decreasing value of c, would have direct effect on the velocity vector. Therefore, by lowering c1 and c2 from 2 → 0.5 for example, it does not require as many iterations to reach the optimum value. That is because lowering c1 and c2, will result in lower velocity vector for next iterations, that would cause less particle offset, and reaching the optimum location faster (less iterations) as a result.

Changing c1 and c2 from 2 → 5 however, had the opposite effect. By the time it reached the 400th iteration, function value was ~-4.5, that is far away from -7. That being said, since c1 and c2 were increased, it had the opposite effect of previous paragraph, so one can argue that changing stopping criteria will balance our increase in c1 and c2.

## Question 5:

**Q:** Solve the following constrained optimization problem using PSO and the external penalty scheme outlined in the lecture notes. Describe your reasoning for your *penalty factor* value or scheme of choosing such a value. Is your solution a constrained optimum?

Since PSO method depends only on function values and independent of gradient information, the penalty approach will be a good fit to solve inequality constraints minimization functions. Therefore, Exterior Penalty Scheme Method is used to solve objective function c (eq. 5), and its constraints (eq. 6).

$$\min f = (1-x)^2\,(-x^2+y)^2 \qquad (5)$$

$$S.T{:}\; -10x - 3y + 25 \;\le 0 \qquad (6)$$

Main purpose of Penalty Factor is to punish points (x and y), that once they are put into the constraints equation, it would be away from allowable range. For instance, the for equation 6, for any iteration or particle, the higher function value is from 0, the higher relative given penalty factor for that particle. ‘r’, being the Penalty Number, must be chosen appropriately, because choosing too big or too small will cause poor function structure and no penalty effect respectively.

Function value 65 iterations was: **0.44197247486049529019**
Compared to the same question in question 1 but without constraints, that was about:

**0.00000237406447043459**

Therefore, we can see that function values were much smaller for unconstraint case. However, since they are both very close to zero, we can say for part c in question 5, that is constrained optimum. The value of r was initially chosen as 100, 500, 1000, 5000, 10000 (best value). Based on Exterior Penalty Scheme methodology, a smaller value was chosen for r, then gradually increased until function values converged.

## References

[1] J. S. Arora, INTRODUCTION TO OPTIMUM DESIGN.

[2] G. Wang, “Introduction to Optimum Design: MSE 426/726 Lab Manual.” Canvas.sfu.ca.

## Code

### PSO.m

```
%% PSO Code
clear all; close all; clc;
%% Author's Notes:
%  - Author Name: Sepehr Rezvani
%% Choose problems number
global problem_number
problem_number = 3;
%% Function 1
if problem_number == 1
    fprintf('Solution for question %d is:\n\n\n', problem_number);
    %% Define parameters
    nparticles = 30;
    nvars=2;
    w = 0.7;
%     c1 = 2;
%     c2 = 2;
    c1 = 5;
    c2 = 5;
    r1 = rand(1);
    r2 = rand(1);

    %% Initialize particles - Randomly assign particles
    x = randi([-100,100],nparticles,nvars);

    % Initialize velocity matrix
    v = randi([-1000,1000],nparticles,nvars);
    vmax = 100;         % if v>100 --> change to v=100

    %% Define stopping criteria AND initialize variables
%     tol = 1;
%     tol_desired = 1e-6;
    max_iter = 400;
    iter = 0;

    xg = x(1,:);
    xg1 = xg; xg_new = xg;
    xp = x;
    xp_new = xp;
    v_new = zeros(nparticles, nvars);
    x_new = zeros(nparticles, nvars);
    %% Main loop
    while (iter<max_iter)
        % PSO code
        fprintf('Iteration number %d: \n', iter+1);

        for i=1:1:nparticles            % Calculate best group position
            if func(xg) > func(x(i,:))
                xg = x(i,:);
            end
        end
        xg1 = xg;

        for i=1:1:nparticles            % Calculate particle velocity
            v_new(i,:) = w*v(i,:) + c1*r1*(xp(i,:)-x(i,:)) + c2*r2*(xg1-
x(i,:));
            if v_new(i,1) > vmax
                v_new(i,1) = vmax;
            elseif v_new(i,2) > vmax
                v_new(i,2) = vmax;
            end
        end

        x_new = x + v_new;              % Update particle position
        for i=1:1:nparticles
            if x_new(i,1) > 100
                x_new(i,1) = 100;
            elseif x_new(i,1) < -100
                x_new(i,1) = -100;
            elseif x_new(i,2) > 100
                x_new(i,2) = 100;
            elseif x_new(i,2) < -100
                x_new(i,2) = -100;
            end
        end

        for i=1:1:nparticles            % Update best personal position
            if func(xp(i,:)) > func(x_new(i,:)) || func(xp(i,:)) ==
func(x_new(i,:))
                xp_new(i,:) = x_new(i,:);
            else
                xp_new(i,:) = xp(i,:);
            end
        end

        for i=1:1:nparticles            % Update best group position
            if func(xp_new(i,:)) < func(xg1) || func(xp_new(i,:)) ==
func(xg1)
                xg1 = xp_new(i,:);
            end
        end
        xg_new = xg1;

        if func(xg_new) <= (-7 + 1e-6)       % tolerance condition changed
to this instead
            break
        end

        x = x_new;
        v = v_new;
        xp = xp_new;
        xg = xg_new;
        iter=iter+1;
        fprintf('Solution for this iteration is:\n%.20f\n', func(xg_new));
        fprintf('============================================== \n');
    end
end



%% Functions 2
if problem_number == 2
    fprintf('Solution for question %d is:\n\n\n', problem_number);
    %% Define parameters
    nparticles = 30;
    nvars=2;
    w = 0.7;
    c1 = 2;
    c2 = 2;
%     c1 = 0.5;
%     c2 = 0.5;
    r1 = rand(1);
    r2 = rand(1);

    %% Initialize particles - Randomly assign particles
    x = randi([-100,100],nparticles,nvars);

    % Initialize velocity matrix
    v = randi([-1000,1000],nparticles,nvars);
    vmax = 100;         % if v>100 --> change to v=100

    %% Define stopping criteria AND initialize variables
%     tol = 1;
%     tol_desired = 1e-6;
    max_iter = 400;
    iter = 0;

    xg = x(1,:);
    xg1 = xg; xg_new = xg;
    xp = x;
    xp_new = xp;
    v_new = zeros(nparticles, nvars);
    x_new = zeros(nparticles, nvars);
    %% Main loop
    while(iter<max_iter)
        % PSO code
        fprintf('Iteration number %d: \n', iter+1);

        for i=1:1:nparticles            % Calculate best group position
            if func(xg) > func(x(i,:))
                xg = x(i,:);
            end
        end
        xg1 = xg;

        for i=1:1:nparticles            % Calculate particle velocity
```

```
            v_new(i,:) = w*v(i,:) + c1*r1*(xp(i,:)-x(i,:)) + c2*r2*(xg1-
x(i,:));
            if v_new(i,1) > vmax
                v_new(i,1) = vmax;
            elseif v_new(i,2) > vmax
                v_new(i,2) = vmax;
            end
        end

        x_new = x + v_new;              % Update particle position
        for i=1:1:nparticles
            if x_new(i,1) > 100
                x_new(i,1) = 100;
            elseif x_new(i,1) < -100
                x_new(i,1) = -100;
            elseif x_new(i,2) > 100
                x_new(i,2) = 100;
            elseif x_new(i,2) < -100
                x_new(i,2) = -100;
            end
        end

        for i=1:1:nparticles            % Update best personal position
            if func(xp(i,:)) > func(x_new(i,:)) || func(xp(i,:)) ==
func(x_new(i,:))
                xp_new(i,:) = x_new(i,:);
            else
                xp_new(i,:) = xp(i,:);
            end
        end

        for i=1:1:nparticles            % Update best group position
            if func(xp_new(i,:)) < func(xg1) || func(xp_new(i,:)) ==
func(xg1)
                xg1 = xp_new(i,:);
            end
        end
        xg_new = xg1;

        if func(xg_new) <= 1e-6         % tolerance condition changed to
this instead
            break
        end

        x = x_new;
        v = v_new;
        xp = xp_new;
        xg = xg_new;


        iter=iter+1;
        fprintf('Solution for this iteration is:\n%.20f\n', func(xg_new));
        fprintf('============================================== \n');
    end
```

```
end


%% Functions 3
if problem_number == 3
    fprintf('Solution for question %d is:\n\n\n', problem_number);
    %% Define parameters
    nparticles = 30;
    nvars=2;
    w = 0.7;
    c1 = 2;
    c2 = 2;
%     c1 = 0.5;
%     c2 = 0.5;
    r1 = rand(1);
    r2 = rand(1);

    %% Initialize particles - Randomly assign particles
    x = randi([-100,100],nparticles,nvars);

    % Initialize velocity matrix
    v = randi([-1000,1000],nparticles,nvars);
    vmax = 100;         % if v>100 --> change to v=100

    %% Define stopping criteria AND initialize variables
%     tol = 1;
%     tol_desired = 1e-6;
    max_iter = 400;
    iter = 0;

    xg = x(1,:);
    xg1 = xg; xg_new = xg;
    xp = x;
    xp_new = xp;
    v_new = zeros(nparticles, nvars);
    x_new = zeros(nparticles, nvars);
    %% Main loop
    while(iter<max_iter)
        % PSO code
        fprintf('Iteration number %d: \n', iter+1);

        for i=1:1:nparticles            % Calculate best group position
            if func(xg) > func(x(i,:))
                xg = x(i,:);
            end
        end
        xg1 = xg;

        for i=1:1:nparticles            % Calculate particle velocity
            v_new(i,:) = w*v(i,:) + c1*r1*(xp(i,:)-x(i,:)) + c2*r2*(xg1-
x(i,:));
            if v_new(i,1) > vmax
                v_new(i,1) = vmax;
            elseif v_new(i,2) > vmax
```

```
                v_new(i,2) = vmax;
            end
        end

        x_new = x + v_new;              % Update particle position
        for i=1:1:nparticles
            if x_new(i,1) > 100
                x_new(i,1) = 100;
            elseif x_new(i,1) < -100
                x_new(i,1) = -100;
            elseif x_new(i,2) > 100
                x_new(i,2) = 100;
            elseif x_new(i,2) < -100
                x_new(i,2) = -100;
            end
        end

        for i=1:1:nparticles            % Update best personal position
            if func(xp(i,:)) > func(x_new(i,:)) || func(xp(i,:)) ==
func(x_new(i,:))
                xp_new(i,:) = x_new(i,:);
            else
                xp_new(i,:) = xp(i,:);
            end
        end

        for i=1:1:nparticles            % Update best group position
            if func(xp_new(i,:)) < func(xg1) || func(xp_new(i,:)) ==
func(xg1)
                xg1 = xp_new(i,:);
            end
        end
        xg_new = xg1;

        if func(xg_new) < 0.44         % tolerance condition changed to
this instead
            break
        end

        x = x_new;
        v = v_new;
        xp = xp_new;
        xg = xg_new;


        iter=iter+1;
        fprintf('Solution for this iteration is:\n%.20f\n', func(xg_new));
        fprintf('============================================== \n');
    end
end
```

## func.m

```
function f = func(x)

global problem_number
%% First function
if problem_number == 1
    f = x(1)^2 + x(2)^2 - x(1)*x(2) - 4*x(1) - x(2);
end
%% Second function
if problem_number == 2
    f = (1 - x(1))^2 + (-1*x(1)^2 + x(2))^2;
end
%% Third function
if problem_number == 3
    g = (1 - x(1))^2 + (-1*x(1)^2 + x(2))^2;
    r = 10000;
    h = -10*x(1) - 3*x(2) + 25;
    if h <= 0
        h = 0;
    else
        h = h;
    end
    f = g + r*h;
end
end
```