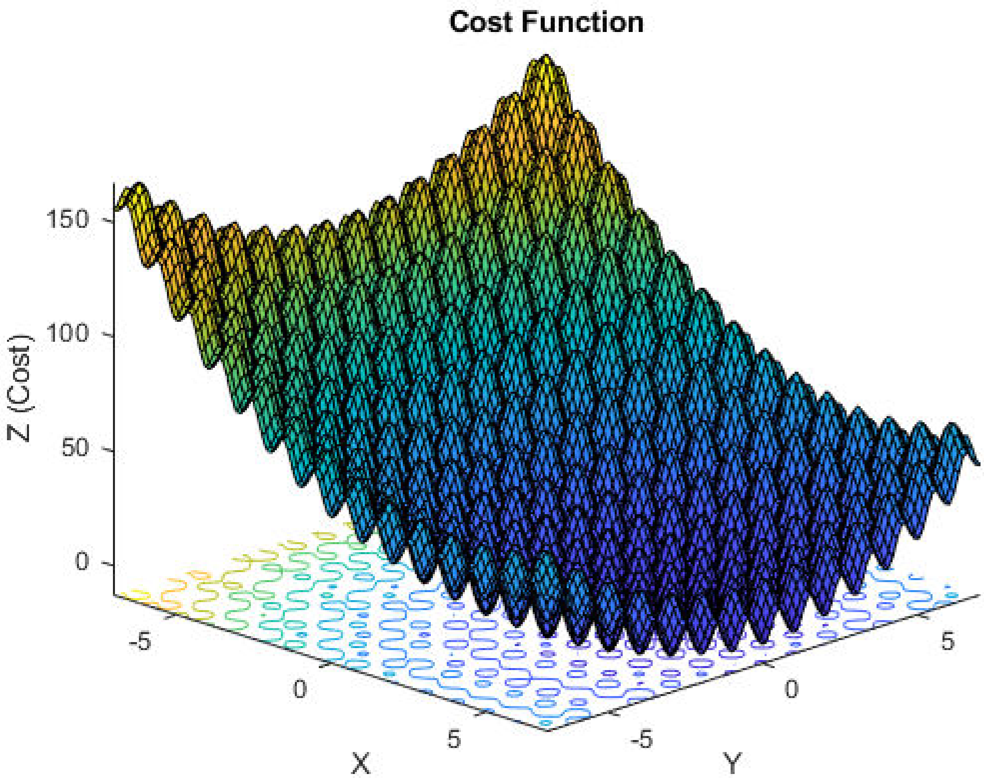
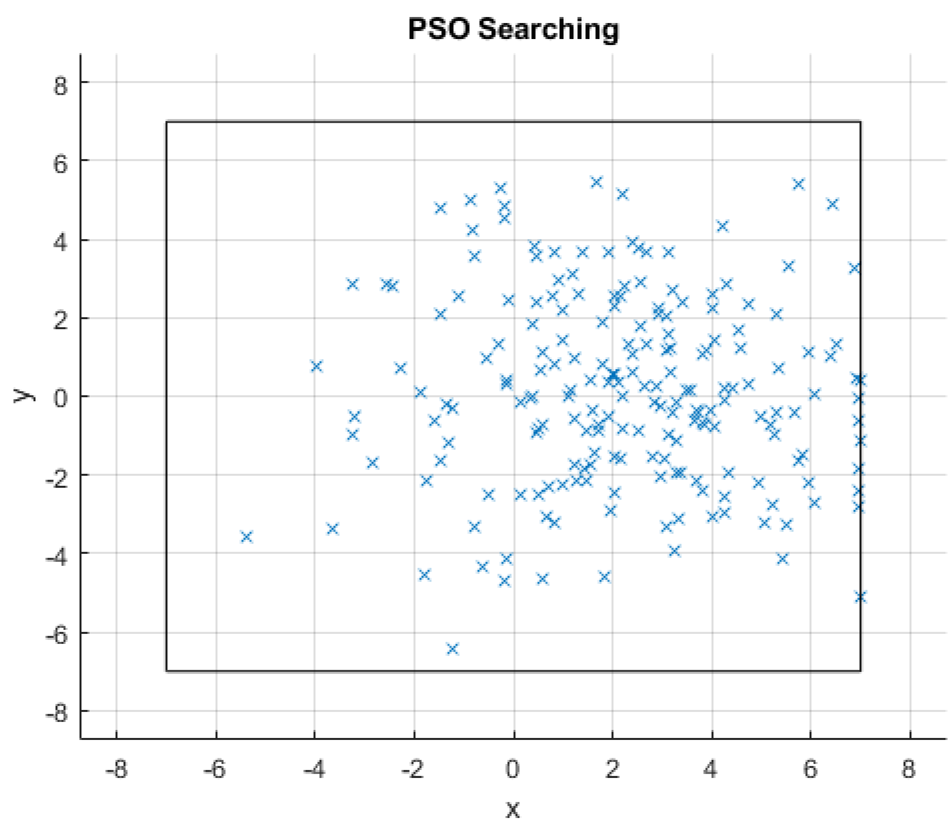
Table of Contents

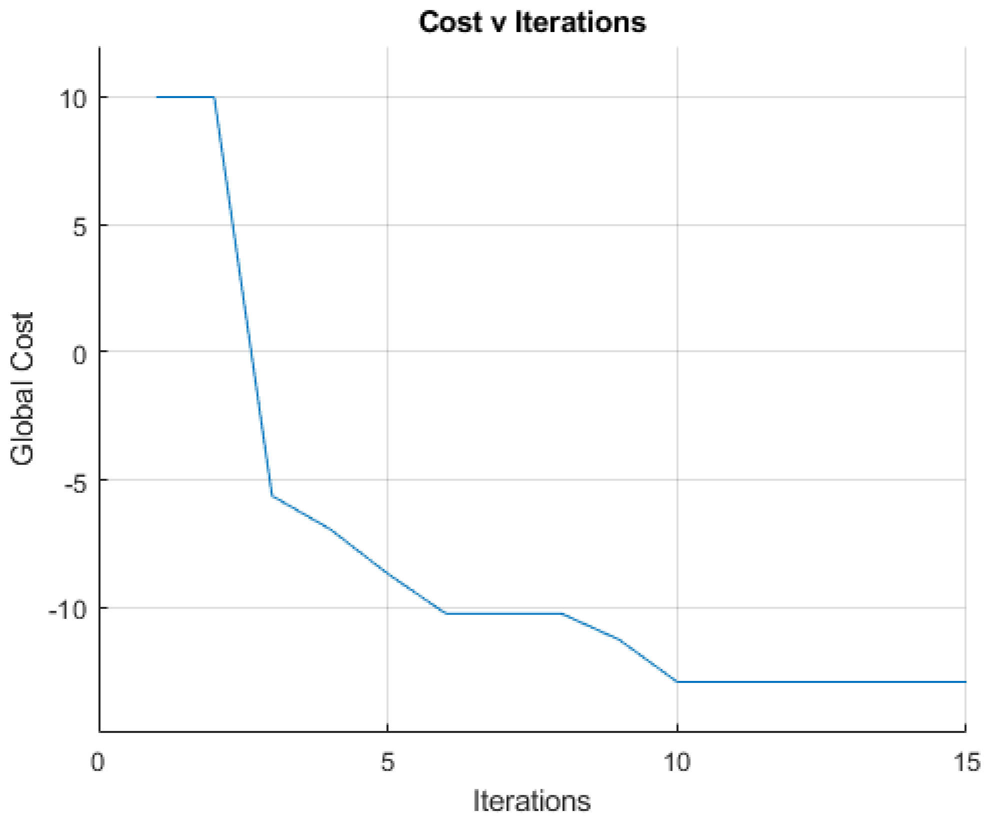
Hw 1

```
%Due 9/27/24
%Jack Vranicar
%jjv20@fsu.edu

%{
    Implement the PSO algorithm to minimize the following function
    f(x,y) = (x - 4)^2 - 7 * cos(2pi(x - 4)) + y2 - 7cos(2piy)
    with x E [-7 7] and y E [-7 7]
%}

%https://www.mathworks.com/matlabcentral/answers/66763-generate-random-
numbers-in-range-from-0-8-to-4

%r = a + (b-a).*rand(100,1); from source above

clc
clearvars
format compact
close all

% Call the PSO function
global_cost_vars = my_PSO;
```

Best cost of -12.92 found at (4.01, 0.97)


Cost Function
Z (Cost)
150
100
50
0
-5
0
5
X
5
0
-5
Y


PSO Searching
8
6
4
2
0
-2
-4
-6
-8
y
-8
-6
-4
-2
0
2
4
6
8
x




```
function [Params] = PSOParams()

%Number of Particles
Params.N = 200;

%Search Boundaries
Params.ya = -7;
Params.yb = 7;
Params.xa = -7;
Params.xb = 7;

%Initializing particl name matrix
Params.paricle_names = zeros(1, Params.N);

%Initializing global values
Params.G.best_cost = 10;
Params.G.best_pos = [0 0];

%Setting alpha values, and alpha1 max and min
Params.alpha_1_max = 10;
Params.alpha_1_min = 1;
Params.alpha_1_delta = 0.5; %Chnage in alpha 1 each try

Params.alpha_2 = 3;
Params.alpha_3 = 6.5;
```

```
%Setting time variables (seconds)
Params.dt = 0.01;
Params.max_time = 20;

%Setting maximum velocity
Params.max_vel = 10;

%Initializing global cost matrix
Params.global_cost(1) = Params.G.best_cost;

%Plotting
Params.plotBool = 1; %1 for plots, 0 for no plots
Params.plotResolution = 1;%How many frames to advance from the previous one

end



function [G] = my_PSO()

%% Paramater Initialization

%Get the parameters from the PSOParams function
Params = PSOParams();

%% 1.1- Plot Function

% Plotting the provided cost function (if desired)
if Params.plotBool

    xvals = linspace(Params.xa, Params.xb, 100);
    yvals = linspace(Params.ya, Params.yb, 100);

    [X, Y] = meshgrid(xvals, yvals);

    Z = hwFn(X, Y);

    figure()
        hold on
        surfc(X, Y, Z);
        view(45, 25);
        xlabel("X")
        ylabel("Y")
        zlabel("Z (Cost)")
        title("Cost Function")
        hold off
end

%% 1.2: PSO Iterations

%Generating initial positions and velcities
for j = 1:Params.N
```

```
    % Random initial position
    rand_x = Params.xa + (Params.xb-Params.xa)*rand(1);
    rand_y = Params.ya + (Params.yb-Params.ya)*rand(1);

    % Creating each particle name
    particle_name = string(strcat('p', num2str(j)));
    particle_names(j) = particle_name;

    % Random Velocity
    p.(particle_name).vel = [2*rand(1) - 1, 2* rand(1) - 1];

    % Assigning random position to the particle
    p.(particle_name).pos = [rand_x, rand_y];

    % Arbitrarily assigning personal best for the particle
    p.(particle_name).best_cost = 1000;
    p.(particle_name).best_pos = p.(particle_name).pos;

    % Storing these positions in an array
    positions(j, :, 1) = p.(particle_name).pos; %third dimension is time

end

%% Picking variables out of the Params struct

%Global vals
global_cost = Params.global_cost;
G.best_cost = Params.G.best_cost;
G.best_pos = Params.G.best_pos;

%Time vars
dt = Params.dt;
max_time = Params.max_time;

%Alphas
alpha_1_max = Params.alpha_1_max;
alpha_1_delta = Params.alpha_1_delta;
alpha_1_min = Params.alpha_1_min;
alpha_2 = Params.alpha_2;
alpha_3 = Params.alpha_3;
max_vel = Params.max_vel;

%Number of particles
N = Params.N;

%% Values that won't change w/ dif simulations

global_improvement = 1;
sim_time = 0;
tick = 1;
alpha_1 = alpha_1_max;

%% Running the PSO
while((G.best_cost ~= 0) && (sim_time <= max_time) && global_improvement)
```

```
    sim_time = sim_time + dt; %simulating passage of time (for change in
position)
    tick = tick + 1; %Counting number of dt seconds passed

    %Storing all the best cost to plot later
    global_cost(tick) = G.best_cost;

    % Linearlly decrease alpha 1 until a certain point each loop
    if (alpha_1 > alpha_1_min)
        alpha_1 = alpha_1 - alpha_1_delta;
    end

    % Iterating on N number of particles
    for i = 1:N

        % Retrieve N particle name
        particle_name = particle_names(i);

        % Retrieve it's position
        pos = p.(particle_name).pos;

        % Calculate cost
        new_cost = hwFn(pos(1), pos(2));

        %Old personal best cost
        old_PB_cost = p.(particle_name).best_cost;

        %Storing cost if it's better than old PB
        if(abs(new_cost) < abs(old_PB_cost))
            p.(particle_name).best_cost = new_cost;
            p.(particle_name).best_pos = pos;
        end

        %Storing cost if it's better than old global best
        if(new_cost < G.best_cost)
            G.best_cost = new_cost;
            G.best_pos = pos;
        end

        %Determing future velocity
        cur_vel = p.(particle_name).vel;
        cur_pos = pos;
        best_pos = p.(particle_name).best_pos;
        g_best_pos = G.best_pos;
        new_vel = alpha_1*cur_vel + alpha_2*rand(1)*(best_pos-cur_pos) +
alpha_3*rand(1)*(g_best_pos-cur_pos);
        new_pos = (new_vel - cur_vel) * dt + cur_pos;

        %Setting bounds on position and velocity
        if ((abs(new_vel(1)) < max_vel) && abs(new_vel(2)) < max_vel)
            p.(particle_name).vel = new_vel;
        end

        if ((abs(new_pos(1)) <=7) && (abs(new_pos(2))<=7))
```

```
            p.(particle_name).pos = new_pos;
        end

        %Storing the positions into a Nx2xtick matrix
        positions(i,:, tick) = p.(particle_name).pos;

    end

    %Setting up a condition to stop looking if cost stops improving
    if (tick>5)
        if global_cost(tick-5) == global_cost(tick)
            global_improvement = 0;
        end
    end
end

fprintf("\n\nBest cost of %.2f found at (%.2f, %.2f)\n\n", G.best_cost,
G.best_pos(1), G.best_pos(2))


%% 1.3 Plotting/ Animating

% Determine if you want to plot results
plotBool = Params.plotBool;

% Resolution is how many frames n to plot; 1:n:end
plotResolution = Params.plotResolution;

if plotBool
    %Plotting particle posns (x,y) only
    pause(1);
    figure()
        hold on
        title("PSO Searching")
        xlabel('x')
        ylabel('y')
        grid on

    % This allows for the positions plot to animate
    for z = 1:plotResolution:tick
        cla
        plot(positions(:, 1, z), positions(:, 2, z), 'x');
        hold on
        axis([Params.xa*1.25 Params.xb*1.25 Params.ya*1.25 Params.yb*1.25])
        rectangle('position',[Params.xa, Params.ya, (Params.xb- Params.xa),
(Params.yb - Params.ya)])
        pause(.1);

    end
    hold off

    %Plotting the global cost vs time (ticks)
    max_cost = max(global_cost);
    min_cost = min(global_cost);
```

```
    pause(1)

    figure()
        hold on
        title("Cost v Iterations")
        xlabel("Iterations");
        ylabel("Global Cost")
        grid on

    % This allows for the global cost plot to animate
    for z = 1:plotResolution:tick
        cla
        plot(1:z, global_cost(1:z));
        axis([0 tick (min_cost-2) (max_cost+2)]);
        pause(.5);


    end

    hold off
end


end



function z = hwFn(x, y)

    z = (x-4).^2 - 7*cos(2*pi*(x-4)) + y.^2 -7*cos(2*pi*y);

end
```

Published with MATLAB® R2024a