Table of Contents

Particle Swarm Optimisation

Name: Ventrapragada Sai Shravani

PRN:17070123120

Batch: G-5 (E&TC)

```
% Theory:
%
% Copyright (c) 2015, Yarpiz (www.yarpiz.com)
% All rights reserved. Please read the "license.txt" for license
 terms.
%
% Project Code: YPAP115
% Project Title: Path Planning using PSO in MATLAB
% Publisher: Yarpiz (www.yarpiz.com)
%
% Developer: S. Mostapha Kalami Heris (Member of Yarpiz Team)
%
% Contact Info: sm.kalami@gmail.com, info@yarpiz.com
%

clc;
clear;
close all;
```

Problem Definition

```
model=CreateModel();

model.n=3;  % number of Handle Points

CostFunction=@(x) MyCost(x,model);    % Cost Function

nVar=model.n;       % Number of Decision Variables

VarSize=[1 nVar];   % Size of Decision Variables Matrix

VarMin.x=model.xmin;          % Lower Bound of Variables
```

```
VarMax.x=model.xmax;          % Upper Bound of Variables
VarMin.y=model.ymin;          % Lower Bound of Variables
VarMax.y=model.ymax;          % Upper Bound of Variables
```

PSO Parameters

```
MaxIt=100;          % Maximum Number of Iterations

nPop=150;           % Population Size (Swarm Size)

w=1;                % Inertia Weight
wdamp=0.98;         % Inertia Weight Damping Ratio
c1=1.5;             % Personal Learning Coefficient
c2=1.5;             % Global Learning Coefficient

% % Constriction Coefficient
% phi1=2.05;
% phi2=2.05;
% phi=phi1+phi2;
% chi=2/(phi-2+sqrt(phi^2-4*phi));
% w=chi;                % Inertia Weight
% wdamp=1;              % Inertia Weight Damping Ratio
% c1=chi*phi1;          % Personal Learning Coefficient
% c2=chi*phi2;          % Global Learning Coefficient

alpha=0.1;
VelMax.x=alpha*(VarMax.x-VarMin.x);    % Maximum Velocity
VelMin.x=-VelMax.x;                    % Minimum Velocity
VelMax.y=alpha*(VarMax.y-VarMin.y);    % Maximum Velocity
VelMin.y=-VelMax.y;                    % Minimum Velocity
```

Initialization

```
% Create Empty Particle Structure
empty_particle.Position=[];
empty_particle.Velocity=[];
empty_particle.Cost=[];
empty_particle.Sol=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];
empty_particle.Best.Sol=[];

% Initialize Global Best
GlobalBest.Cost=inf;

% Create Particles Matrix
particle=repmat(empty_particle,nPop,1);

% Initialization Loop
for i=1:nPop

    % Initialize Position
    if i > 1
```

```
            particle(i).Position=CreateRandomSolution(model);
        else
            % Straight line from source to destination
            xx = linspace(model.xs, model.xt, model.n+2);
            yy = linspace(model.ys, model.yt, model.n+2);
            particle(i).Position.x = xx(2:end-1);
            particle(i).Position.y = yy(2:end-1);
        end

        % Initialize Velocity
        particle(i).Velocity.x=zeros(VarSize);
        particle(i).Velocity.y=zeros(VarSize);

        % Evaluation
        [particle(i).Cost,
     particle(i).Sol]=CostFunction(particle(i).Position);

        % Update Personal Best
        particle(i).Best.Position=particle(i).Position;
        particle(i).Best.Cost=particle(i).Cost;
        particle(i).Best.Sol=particle(i).Sol;

        % Update Global Best
        if particle(i).Best.Cost<GlobalBest.Cost

            GlobalBest=particle(i).Best;

        end

    end

    % Array to Hold Best Cost Values at Each Iteration
    BestCost=zeros(MaxIt,1);
```

PSO Main Loop

```
    for it=1:MaxIt

        for i=1:nPop

            % x Part

            % Update Velocity
            particle(i).Velocity.x = w*particle(i).Velocity.x ...
                + c1*rand(VarSize).*(particle(i).Best.Position.x-
particle(i).Position.x) ...
                + c2*rand(VarSize).*(GlobalBest.Position.x-
particle(i).Position.x);

            % Update Velocity Bounds
            particle(i).Velocity.x = max(particle(i).Velocity.x,VelMin.x);
            particle(i).Velocity.x = min(particle(i).Velocity.x,VelMax.x);
```

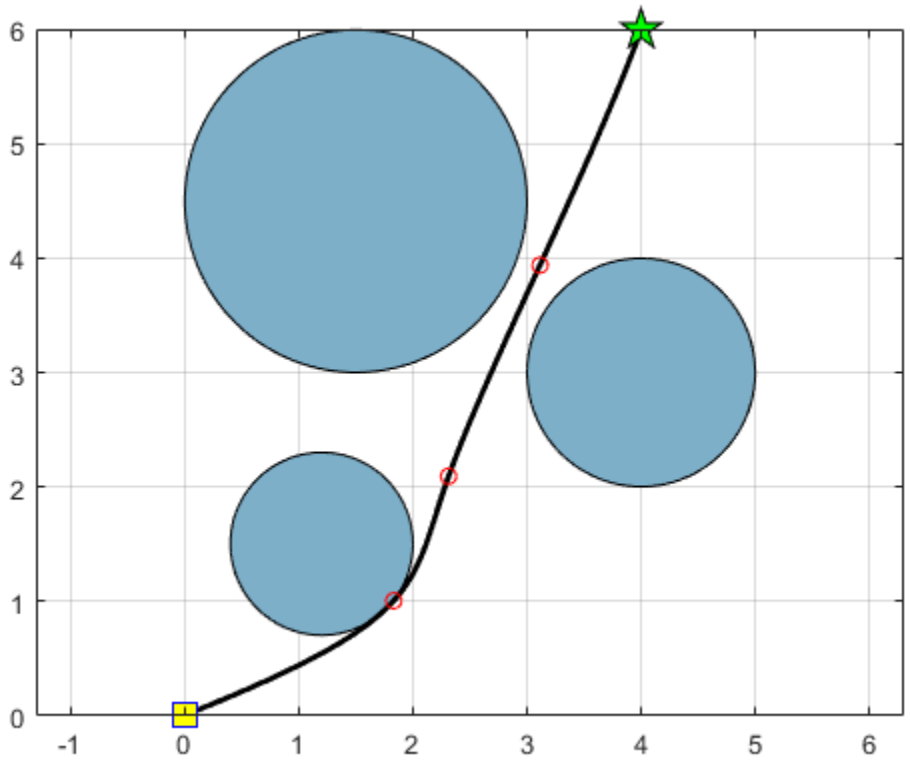
```
        % Update Position
        particle(i).Position.x = particle(i).Position.x +
 particle(i).Velocity.x;

        % Velocity Mirroring
        OutOfTheRange=(particle(i).Position.x<VarMin.x |
 particle(i).Position.x>VarMax.x);
        particle(i).Velocity.x(OutOfTheRange)=-
particle(i).Velocity.x(OutOfTheRange);

        % Update Position Bounds
        particle(i).Position.x = max(particle(i).Position.x,VarMin.x);
        particle(i).Position.x = min(particle(i).Position.x,VarMax.x);

        % y Part

        % Update Velocity
        particle(i).Velocity.y = w*particle(i).Velocity.y ...
            + c1*rand(VarSize).*(particle(i).Best.Position.y-
particle(i).Position.y) ...
            + c2*rand(VarSize).*(GlobalBest.Position.y-
particle(i).Position.y);

        % Update Velocity Bounds
        particle(i).Velocity.y = max(particle(i).Velocity.y,VelMin.y);
        particle(i).Velocity.y = min(particle(i).Velocity.y,VelMax.y);

        % Update Position
        particle(i).Position.y = particle(i).Position.y +
 particle(i).Velocity.y;

        % Velocity Mirroring
        OutOfTheRange=(particle(i).Position.y<VarMin.y |
 particle(i).Position.y>VarMax.y);
        particle(i).Velocity.y(OutOfTheRange)=-
particle(i).Velocity.y(OutOfTheRange);

        % Update Position Bounds
        particle(i).Position.y = max(particle(i).Position.y,VarMin.y);
        particle(i).Position.y = min(particle(i).Position.y,VarMax.y);

        % Evaluation
        [particle(i).Cost,
 particle(i).Sol]=CostFunction(particle(i).Position);

        % Update Personal Best
        if particle(i).Cost<particle(i).Best.Cost

            particle(i).Best.Position=particle(i).Position;
            particle(i).Best.Cost=particle(i).Cost;
            particle(i).Best.Sol=particle(i).Sol;

            % Update Global Best
            if particle(i).Best.Cost<GlobalBest.Cost
```

```
                GlobalBest=particle(i).Best;
            end

        end


    end

    % Update Best Cost Ever Found
    BestCost(it)=GlobalBest.Cost;

    % Inertia Weight Damping
    w=w*wdamp;

    % Show Iteration Information
    if GlobalBest.Sol.IsFeasible
        Flag=' *';
    else
        Flag=[', Violation = ' num2str(GlobalBest.Sol.Violation)];
    end
    disp(['Iteration ' num2str(it) ': Best Cost = '
 num2str(BestCost(it)) Flag]);

    % Plot Solution
    figure(1);
    PlotSolution(GlobalBest.Sol,model);
    pause(0.01);

end
```

```
Iteration 1: Best Cost = 20.1906 *
Iteration 2: Best Cost = 18.0118 *
Iteration 3: Best Cost = 12.6177 *
Iteration 4: Best Cost = 12.6177 *
Iteration 5: Best Cost = 12.0777 *
Iteration 6: Best Cost = 12.0777 *
Iteration 7: Best Cost = 10.6058 *
Iteration 8: Best Cost = 10.6058 *
Iteration 9: Best Cost = 10.431 *
Iteration 10: Best Cost = 9.0554 *
Iteration 11: Best Cost = 8.8534 *
Iteration 12: Best Cost = 8.7036 *
Iteration 13: Best Cost = 8.7036 *
Iteration 14: Best Cost = 8.7036 *
Iteration 15: Best Cost = 8.7036 *
Iteration 16: Best Cost = 8.4936 *
Iteration 17: Best Cost = 8.4936 *
Iteration 18: Best Cost = 7.9976 *
Iteration 19: Best Cost = 7.9976 *
Iteration 20: Best Cost = 7.9976 *
Iteration 21: Best Cost = 7.9976 *
Iteration 22: Best Cost = 7.8597 *
Iteration 23: Best Cost = 7.61 *
Iteration 24: Best Cost = 7.61 *
```

```
Iteration 25: Best Cost = 7.61 *
Iteration 26: Best Cost = 7.61 *
Iteration 27: Best Cost = 7.61 *
Iteration 28: Best Cost = 7.61 *
Iteration 29: Best Cost = 7.5881 *
Iteration 30: Best Cost = 7.5881 *
Iteration 31: Best Cost = 7.5881 *
Iteration 32: Best Cost = 7.5881 *
Iteration 33: Best Cost = 7.5881 *
Iteration 34: Best Cost = 7.5826 *
Iteration 35: Best Cost = 7.5826 *
Iteration 36: Best Cost = 7.5826 *
Iteration 37: Best Cost = 7.5805 *
Iteration 38: Best Cost = 7.5805 *
Iteration 39: Best Cost = 7.5787 *
Iteration 40: Best Cost = 7.5756 *
Iteration 41: Best Cost = 7.5734 *
Iteration 42: Best Cost = 7.5734 *
Iteration 43: Best Cost = 7.569, Violation = 1.2477e-06
Iteration 44: Best Cost = 7.569, Violation = 1.2477e-06
Iteration 45: Best Cost = 7.5688 *
Iteration 46: Best Cost = 7.5672 *
Iteration 47: Best Cost = 7.5672 *
Iteration 48: Best Cost = 7.5668 *
Iteration 49: Best Cost = 7.5663 *
Iteration 50: Best Cost = 7.5662 *
Iteration 51: Best Cost = 7.5655 *
Iteration 52: Best Cost = 7.5655 *
Iteration 53: Best Cost = 7.5654 *
Iteration 54: Best Cost = 7.5654 *
Iteration 55: Best Cost = 7.5649 *
Iteration 56: Best Cost = 7.5649 *
Iteration 57: Best Cost = 7.5649 *
Iteration 58: Best Cost = 7.5649 *
Iteration 59: Best Cost = 7.5649 *
Iteration 60: Best Cost = 7.5648 *
Iteration 61: Best Cost = 7.5645 *
Iteration 62: Best Cost = 7.5643 *
Iteration 63: Best Cost = 7.5639 *
Iteration 64: Best Cost = 7.5639 *
Iteration 65: Best Cost = 7.5638 *
Iteration 66: Best Cost = 7.5637 *
Iteration 67: Best Cost = 7.5637 *
Iteration 68: Best Cost = 7.5637 *
Iteration 69: Best Cost = 7.5636 *
Iteration 70: Best Cost = 7.5635 *
Iteration 71: Best Cost = 7.5634 *
Iteration 72: Best Cost = 7.5634 *
Iteration 73: Best Cost = 7.5634 *
Iteration 74: Best Cost = 7.5634 *
Iteration 75: Best Cost = 7.5634 *
Iteration 76: Best Cost = 7.5634 *
Iteration 77: Best Cost = 7.5634 *
Iteration 78: Best Cost = 7.5631 *
```

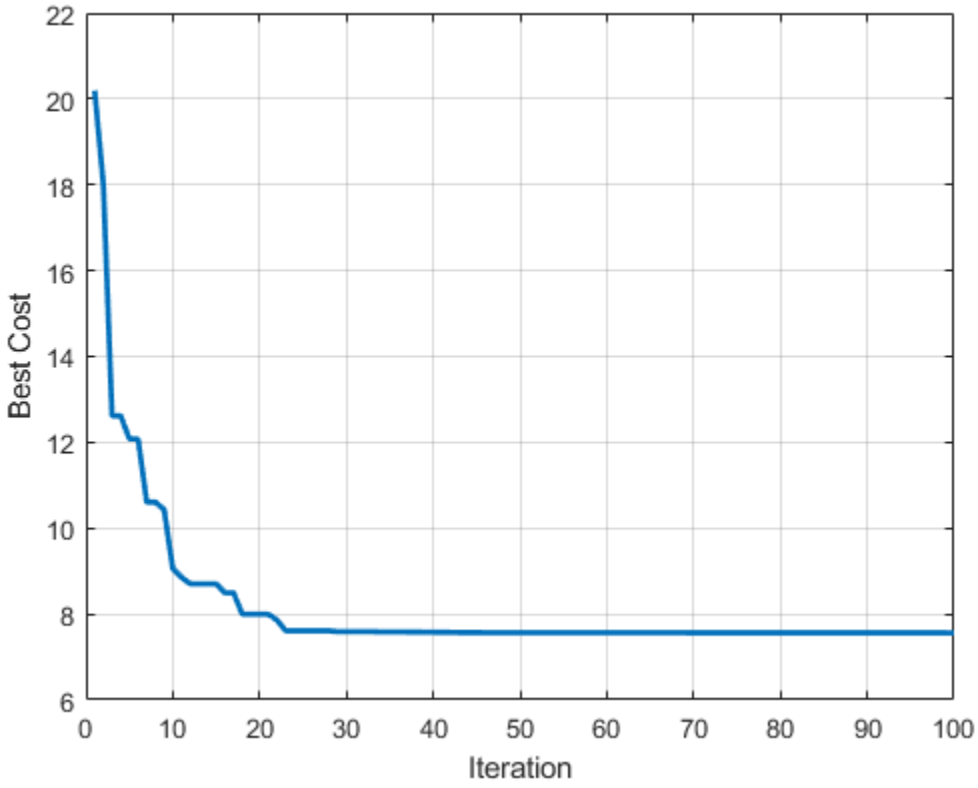
```
Iteration 79: Best Cost = 7.5631 *
Iteration 80: Best Cost = 7.563 *
Iteration 81: Best Cost = 7.563 *
Iteration 82: Best Cost = 7.5627 *
Iteration 83: Best Cost = 7.5627 *
Iteration 84: Best Cost = 7.5626 *
Iteration 85: Best Cost = 7.5626 *
Iteration 86: Best Cost = 7.5625 *
Iteration 87: Best Cost = 7.5625 *
Iteration 88: Best Cost = 7.5623 *
Iteration 89: Best Cost = 7.5623 *
Iteration 90: Best Cost = 7.5623, Violation = 4.8611e-08
Iteration 91: Best Cost = 7.5623, Violation = 4.8611e-08
Iteration 92: Best Cost = 7.5622 *
Iteration 93: Best Cost = 7.562 *
Iteration 94: Best Cost = 7.562 *
Iteration 95: Best Cost = 7.5619 *
Iteration 96: Best Cost = 7.5618 *
Iteration 97: Best Cost = 7.5618 *
Iteration 98: Best Cost = 7.5618 *
Iteration 99: Best Cost = 7.5617 *
Iteration 100: Best Cost = 7.5616 *
```



Results

```
figure;
```

```
plot(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;
```



Conclusion:

When iterations were 500 the model's best cost becomes stagnant and therefore goes into the case of overfitting. When the iterations are reduced to 100 the best cost is still varying and therefore provides with a faster convergence, lower time taken and an optimised path and therefore avoids pre mature convergence.

Published with MATLAB® R2020a