Performance Analysis of A Particle Swarm Optimization Algorithm Using Rosenbrock Function

Name: EC Assessment Week 2	**Author: Babu Pallam**
Date: 29/02/2024	Email Id.: P2849288@my365.dmu.ac.uk

Introduction

Among performance testing problems for optimization of algorithms Rosenbrock function is one of the well-known problems available today. This report uses Rosenbrock function intensively for the purpose of optimizing Particle Swarm Optimization (PSO) algorithm. The algorithm has been tested, which produces results in terms of number of iterations needed for the algorithm to find the optimum solution. The first section, Benchmark A, describes a basic PSO algorithm with the minimum iteration needed to find the optimum value to the Rosenbrock function. Following by that, next section, Benchmark B, discusses about the effect of parameters such as maximum iterations, population size, inertia coefficient, personal acceleration coefficient, social acceleration coefficient and others. A detail analysis of the resultant when one or more parameter value get effected has been presented in this section. Next, in Benchmark C, as an extension of the algorithm, some dynamic techniques have been applied in the code. Which include, implementation of some constriction coefficients that power the Inertia Coefficient (w), and dynamicity to the damping factor of w. The effects those values in the result have been described. Finally, the comparison of the above three benchmarks, A, B and C have been included, with a conclusion.

Benchmark A: Analysis of Algorithm for Theoretical Global Optimum

The problem selected for the optimization of PSO algorithm was Rosenbrock function with two variables as input. The function implemented is defined below.

$$\textbf{Minimise } f(x_1, x_2) = 100*(x_2-x_1^2)^2 + (1-x_1)^{2,}$$

$$\textbf{Bounds: } -10 <= x_1 <= 10 \textbf{ and } -10 <= x_2 <= 10$$

The PSO algorithm for this function has been successfully implemented and executed with the following parameter specification.

- Maximum iterations (MaxIt) = 500
- Population size (nPop) = 50
- Inertia coefficient (w) = 1
- Damping Ratio of Inertia Coefficient (wdamp) = 0.99
- Personal Acceleration Coefficient (c1) = 2
- Social Acceleration Coefficient (c2) = 2

The optimum solution of the function is zero, which is defined as f (1,1) = 0. Since the convergence takes significant number of iterations to reach this optimum value given, for performance testing, a tolerance value has been set, which is considered as an acceptable

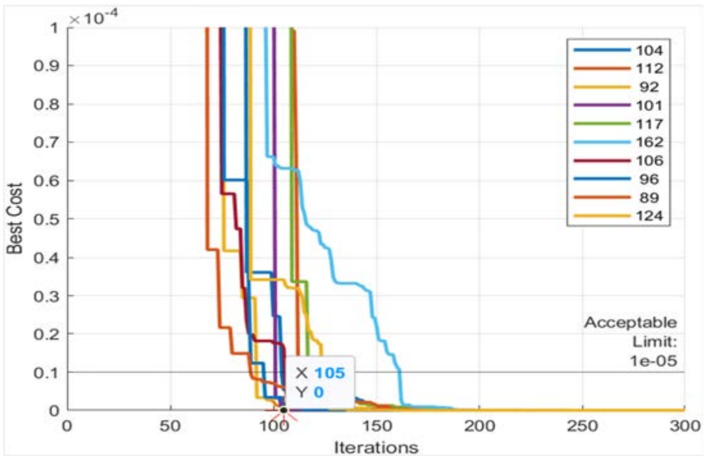


Figure 1

solution to the problem with possible maximum error. The tolerance value chosen is 10^{-5}. Keeping this value as static along with the other parameters, several experiments have been conducted to find the minimum iterations needed for the algorithm to reach the chosen tolerance value. The Figure 1 represents the graph, which was the result of ten consecutive experiments done, and the median value calculated on those results. As can be seen, the median value found is 105; this value is referred in the following sections as MedianBM1. The output produced has been provided in Appendix A along with MATLAB code.

To witness the convergence of the solutions of size nPop, contour plot feature has been used in MATLAB with a function named contour(). A few snapshots of the result are shown in Figure 2. As can be seen in this figure, in generation 1, the 50 solutions were spread across the solution space, all solution generated were distance apart. The red dot specifies the global best of that generation. In generation 200, more convergence can be seen since the points are were in a closed region. In 300th generation, the only two solutions can be seen slightly away from the global best solution. In 450th generation all 50 solutions were converged into the global best solution, in which the two variable values as can be seen are x=1 and y=1. The size and colour mentioned are part of the MATLAB code as part of the contour implementation, which can be ignored.

The next section takes gives the detailed analysis of how values of the parameters influence the result.

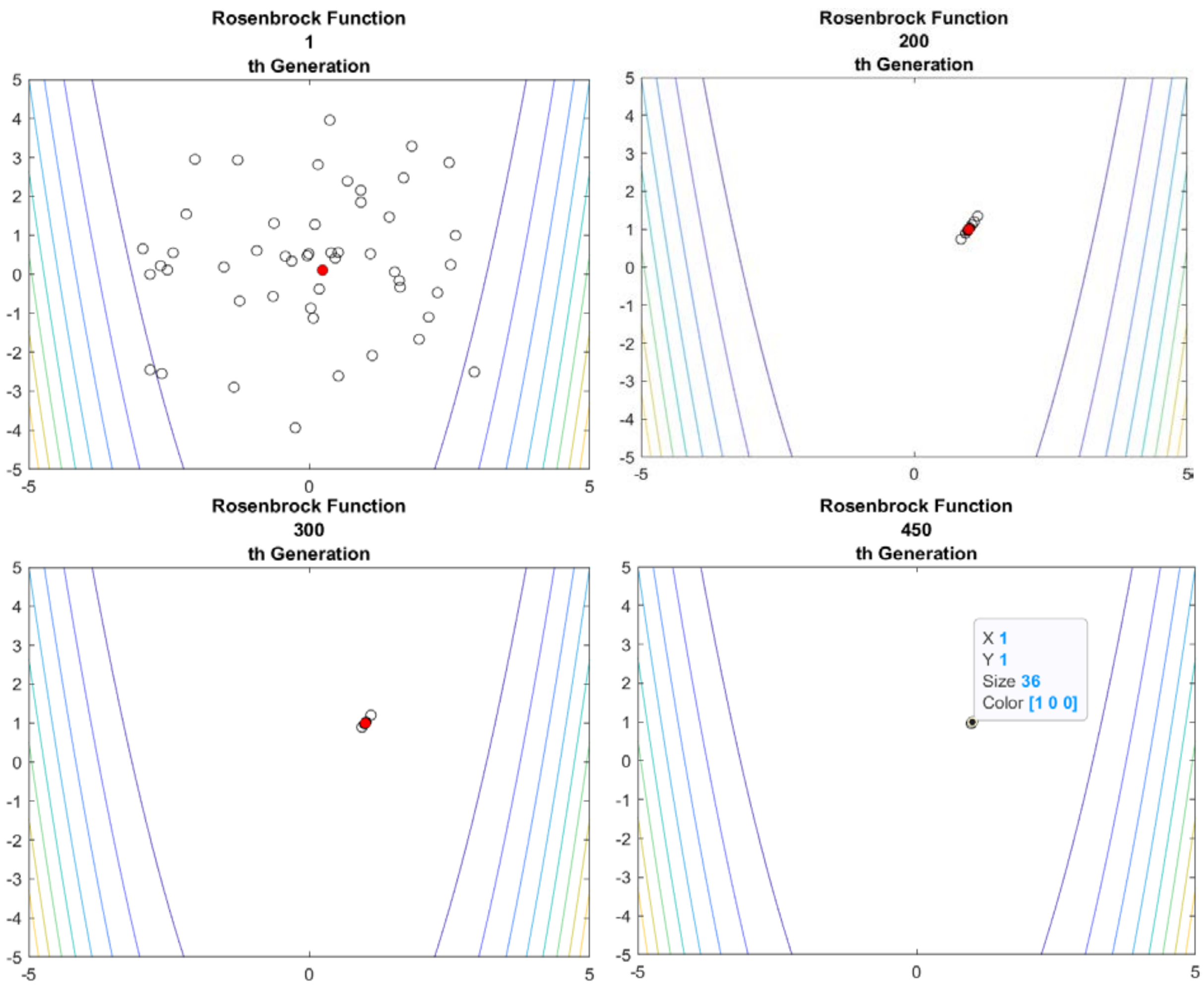


Figure 2

Benchmark B: Effect of Parameters

This section discusses about effect of the parameters which are mentioned in the above section. During the experimentation process, while changing some parameter values, the Benchmark A has been followed for values of rest. Two assumptions have been made for easy testing.

1. The acceptance solution value is 10^{-5}, which is known as Tolerance value. All testing has been carried out based on this value. Once the solution reaches this value, irrespective of the MaxIt, the test will be terminated with the resulted output.
2. For those tests which are not not resulted into the desired solution with MaxIt iterations, the result would be set into MaxIt. This has been done throughout the analysis for the purpose of presentation of result as graphs. Thereby, the result is 500 or more indicates for a test can be either the absolute or approximated result, but the performance analysis has been unaffected and not compromised during the whole experiments.

Several experiments have been carried out with the implemented code from which the Benchmark A has been deduced; it drew the below inferences. Figures and graphs mentioned in this section have been provided in Appendix B.

- In Rosenbrock function, MaxIt has a significant effect if the accuracy of the test is absolute with zero error. Since the tolerance value is set to 10^{-5}, the solution which reaches equals or less than this value can be considered as the best solution. So, throughout testing this value did remain constant. Figure 6 illustrates as a graph the results of ten experiments done with MaxIt ranges from 200 to 2000. The acceptance has been marked by a line parallel to the x axis. From the points, where the lines denoting different tests intersect with this acceptance line, all tests reach the required results with a count of iterations that is in the bounded range, in this 70 to 136, which includes the MedianBM1. Based on the analysis, it is easy to conclude that significant hike of MaxIt has no effect in the resulted outcome rather it may make a setback in term of performance, which is yet to be found.
- nPop greatly influences the result. nPop indicates the number of the solutions which are considered parallelly to find the best solution among those. The result of the test is provided in Figure 7. The result ranges from 36 to 95 which all are lower than the MedianBM1. 95 is resulted for nPop=50(, which is used in Benchmark A) and for nPop = 15000, the test converged into the acceptable solution in 36 iterations. From this, an observation can be made. As nPop increases the number of iterations needed to reach the result decreases. The probability of presence of the best solution is high in case of large population, thereby this result is worth to be noted.
- The values of w and wdamp also make a notable change in the result. Surf feature of MATLAB has been used to depict the result in 3D view, by changing values of w and wdamp during every test using MATLAB. The Figure 8 provides the plot. The w ranges from 1 to 10 and wdamp ranges from -1 to 1. The best result found during the test is 16, when w=7 and wdamp =0.2. For wdamp =1 and -1, the result is 500. In between -0.8 to +0.8 of wdamp, irrespective of w (within the test boundary) the results are around MedianBM1.
- While testing with different values of w and c1, a notable effect has been identified, especially for the values of c1. The Figure 9 provides the results of tests. For c1, which is lesser than 0 and greater than 3 would cause a steep increase in the results, which has been shows as yellow regions in the graph. The lowest value resulted which could be considered as best in these tests is when c1 = 2 and w=1, that is 76 iterations.
- Like testing with different values of w and c2, from range 1 to 10 for w and 0 to 10 for c2, as given in Figure 10, for values of c2 from 2 to 4, for various w's, it results into a value which can be considered as good, though not best. For rest of the c2 values, the result is 500 and higher which is not considerable. Another observation is that as w increases for c1 = 2, the result also increases. The result which could be considered as best found in these tests is 47, when w=0 and c2 = 2. But w = 0 creates an uncertainty in this problem since it will lead into a situation in which the inertia part of the velocity is obsolete, which must not be ignored.
- Effect of c1 and c2 together makes a change in the values. The Figure 11 provides the results. In which c1 and c2 varies from -2 to 2. For values higher than 0.5 in the range chosen for both c1 and c2, give the result which is near to MedeanBM1. For

rest, the result varies significantly except for few. The best result found in this is 49 when both c1 and c2 are 0.5.

In addition to that, another experiment has been conducted with the values that gave best results from the above observations. The result of this experiment has been provided in Figure 12. As per the result, 5, which is the median of results of ten repetitive tests made with the parameter values as MaxIt = 10; nPop = 15000; w = 7; wdamp = 0.2; c1 = 1; and c2 = 1; Detailed discussion of this test has been given in the Comparison section of this report.

Benchmark C: Effect of dynamic/advanced parameters

This section discusses about how the result got influenced when the algorithm has been extended by giving more dynamicity to parameters, such as w, and wdamp. To add dynamicity to w, few constrictions coefficient have been introduced to the algorithm. The resulted graphs and associated code mentioned in this section have been added in Appendix C. The logic behind calculating w has been shows in Equation 1 as MATLAB Code.

```
% Constriction Coefficients
kappa = 1;
phi1 = 2.05;
phi2 = 2.05;
phi = phi1 + phi2;
chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));
w=chi
```

Equation 1

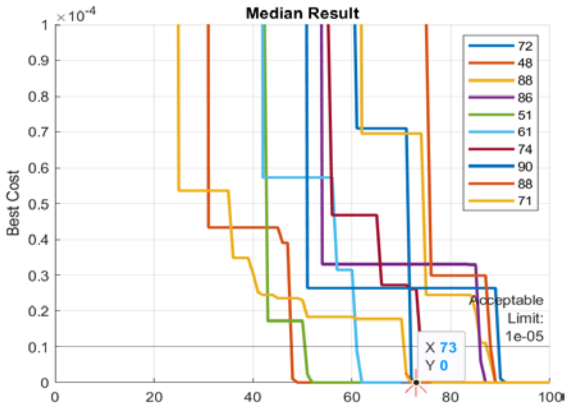


Figure 3

The tests have been carried out several times after the extension of the code. The median value of the result has been found for a sequence of execution of ten tests.

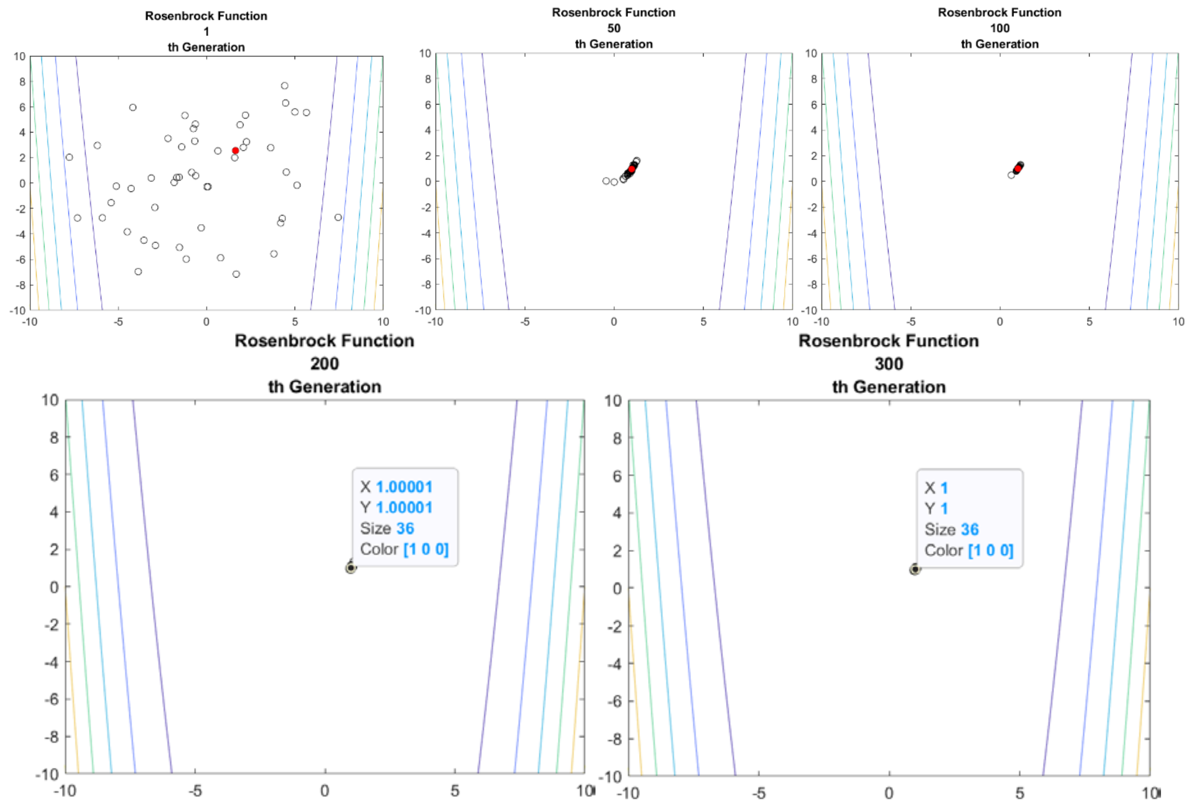


Figure 4

The median value found was 73, which is later referred as MedianBM3. Further, to understand the efficiency of the modified algorithm, with the help of contour features, graphs has been plotted. The result was notable. Figure 4 shows snapshot of 5 generations. In Benchmark A, a successful convergence has been seen in 450th generation, but here this has been witnessed at 300th generation itself. While comparing with Benchmark A, this is a significant improvement.

The observations drawn during the performance analysis has been discussed below.

- Based on Figure 13 and 14, MaxIt and nPop has the same effect as we have seen in Benchmark B. Same explanation of Benchmark B holds here.
- The experiment has done two times with different set of kappa values. From which two observations have been made. First observation is based on Figure 15. For value range in between 0.5 and 1, the result is increasing from 13 to 73. Further, for kappa = 0.55, the result is 13, which is the best result received during this batch of tests. when kappa=1, it results 73 which is same as medianBM3. Second observation is based on Figure 16. The kappa value ranges from 1 to 1.5. By close analysis, for kappa value higher than 1.2, the result is higher than 500 (reason is that the lines are absent the figure). Moreover, as kappa increases, the result also increases.
- To understand the effect of kappa and phi1, an experiment has been conducted based on different values for kappa and phi1. The results found has been given in

Figure 17. Based on this figure, the best result found during the test is 15 when kappa is 0.9 and phi1 is 3.

Further research has been done by adding dynamic nature to parameter wdamp since in the algorithm used so far has a fixed value for wdamp, which is equal to 1 throughout the optimization process. Equation 2 shows the implementation of w with flexible wdamp. Where wmin = 0.1 and wmax is the same w which has been calculated using the Equation 1. In addition to that it denoted the generation number and 'i' denoted index of particle.

```
% Calculating Inertia Coefficient
(w) using wmax and wmin
w = wmax - ((i/it)*(wmax-wmin));
```

Equation 2

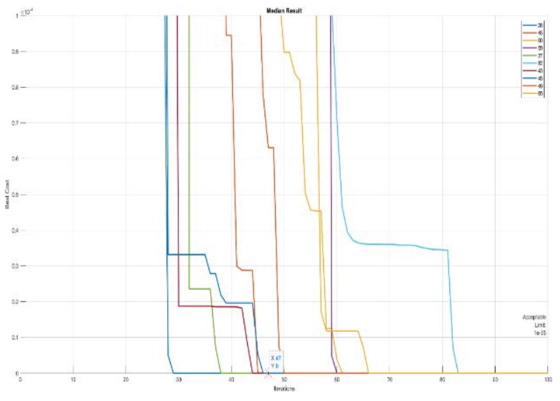


Figure 5

The median of result has been calculated in the same way how it has been carried out for Benchmark A, and the result found to be 47, which is given in Figure 5. With the further extended code, the experiment has been continued, and reached to the following observations. The figures and code mentioned below have been included in Appendix D.

- The test carried out for different values of kappa, and the result is like what have been found in the result shows in Figure 15. The result of this test has been provided in Figure 18. For values higher than 1 for kappa, a high in the result value has been observed. The best value we received here is when kappa=0.7
- As presented in Figure 19, the effect of kappa and phi1 with dynamic wdamp, is also arbitrary. For few combinations like kappa =0.9 and phi1 =9, the result is significantly higher, but for rest the values are changing in between 25 and 500. The best result received is 25 for kappa = 0.5 and phi1=1.
- The effect of wmin has also been analysed in the similar way, which is given in Figure 20. The test has been repeated for various values from wmin =-10 to wmin=30. The result also varies from 33 to 105. A gradual increase of result has been observed for a slight increase in wmin.

Comparison of Benchmark A, B and C

The following conclusions can be made while comparing the Benchmarks A, B, and C.

- The change in median values: MedianBM1 and MedianBM3 is worth to be noted. MedianBM1 is 105 and MedianBM3 is 73. From the result, it can be clearly concluded that by adding the dynamic technique for w, the result has been reduced by at least 30, which is considered because of the significant improvement in the algorithm. In addition to that by adding flexible wdamp into the algorithm, the median of the result found to be 47, which could also be claimed as an efficiency improvement in the algorithm since this value is much lower than MedianBM3.

- The best result found on Benchmark B and C has been given below:
  - Benchmark B: 16, when w=7 and wdamp =0.2 (Ref. Figure 8)
  - Benchmark C: 15 when kappa is 0.9 and phi1 is 3 (Ref. Figure 17)
  From above results, it is evident that the Benchmark C made an improvement in the algorithm since the result found is slightly better than 16, which has been resulted in Benchmark B.
- During the performance analysis of Benchmark B, an interesting test has been conducted, in which the result found to be 5. The result has been drawn as a graph in Appendix B, under section B7 as Figure 12. The code which produced this graph has been given under the same section. The result 5 is much lower compared to 15 and 16, resulted in benchmark C and B respectively. When analysing the parameter values behind resulting this outcome, it is apparent that the result is greatly influenced by the nPop. In this nPop has been set to 15000. That means that, 15000 randomly generated solutions are there as outcomes of Rosenbrock function at the first generation process itself, from which the best solution to be found. So, the probability of having the optimum solution is very high compared to the population with value of nPop is fifty. Thereby, though this result is more optimum, a gentle ignorance of this test has been performed. The importance of advanced research is needed to find how the value of nPop would affect the performance of algorithm in terms of resources and time in order to say this test result can be considered as an optimum result.

Conclusion

The Rosenbrock function which has been implemented using PSO algorithm has been successfully build and executed. The benchmark A, B and C has been reached with evidential, in the form of visual graphs and the MATLAB Codes. Using contours and surf, more visualized view of how the convergence happens and how the parameters influence the result have been analysed. The results of experiments done in the PSO algorithm have been discussed in the above sections. The best combination of parameters needed to get the optimum result in minimum number of iterations have been found. The requirement of comprehensive research has been detailed.

APPENDIX A

A1. MATLAB Code: RosenBrock Function:

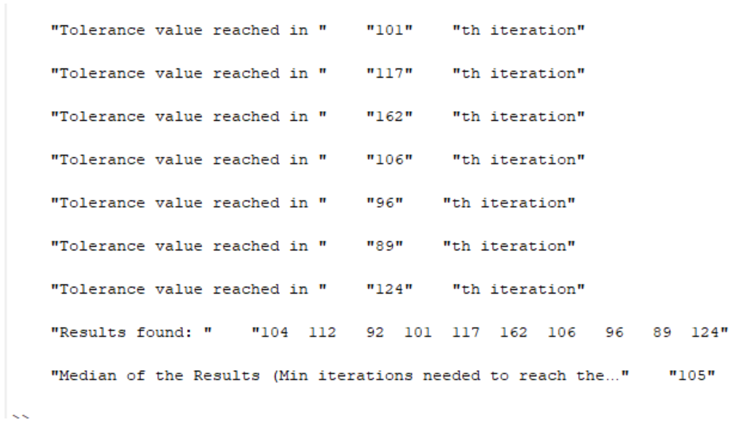
```
function z = RosenBrockFunction(x,y)
        %based on 5a slide no 12
     z =100*(y-x.^2).^2+(1-x).^2;
end
```

A2: MATLAB Code: The modified code of finding the Median of Minimum Iterations with The Acceptable Tolerance Value

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 300;        % Maximum Number of Iterations
params.nPop = 50;           % Population Size (Swarm Size)
params.w = 1;               % Intertia Coefficient
params.wdamp = 0.99;        % Damping Ratio of Inertia Coefficient
params.c1 = 2;              % Personal Acceleration Coefficient
params.c2 = 2;              % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Informatin
params.toleranceValue = 10^(-5); %tolerance value which can be accepted as solution with the maximum error allowed

%% Calling PSO - FOR MEDIAN CALCULATION
% Run 10 experiments and find the median of those results
numberOfIterations =10;
experimentsResults = zeros(1,numberOfIterations);
for i=1:numberOfIterations
    out = bm1PSO_RosenBrockFun_Median_Calculation(problem, params);
    experimentsResults(i) = out.minIterationForToleranceValue;
    %plot the value in the graph if exists otherwise plot it in a new graph
    hold on
    semilogy(out.BestCosts,"LineWidth",2)
    grid on;
    hold off
end
%display the result
disp(["Results found: " num2str(experimentsResults)])
%find the median
medianOfResults = median(experimentsResults);
disp(["Median of the Results (Min iterations needed to reach the tolerance value): " num2str(medianOfResults)])

%plot median value in the graph
hold on
semilogy(medianOfResults,0,"r*","MarkerSize",20)
hold off
%add graph attributes
xlabel('Iterations');
ylabel('Best Cost');
xlim([0,300]);
ylim([0, 10^-4])
yline(params.toleranceValue,'-',{'Acceptable','Limit:', num2str(params.toleranceValue) });
legend(num2str(experimentsResults'), 'location', 'northeast');
```

A3: Command line output produced after execution of the code



```
"Tolerance value reached in "    "101"    "th iteration"

"Tolerance value reached in "    "117"    "th iteration"

"Tolerance value reached in "    "162"    "th iteration"

"Tolerance value reached in "    "106"    "th iteration"

"Tolerance value reached in "    "96"    "th iteration"

"Tolerance value reached in "    "89"    "th iteration"

"Tolerance value reached in "    "124"    "th iteration"

"Results found: "    "104   112   92   101   117   162   106   96   89   124"

"Median of the Results (Min iterations needed to reach the…"    "105"
```

A4: MATLAB Code: Drawing Contour Plots under section- Benchmark A

MATLAB Code:

File name: bm1App_rosenBrockFun_ContourPlots.m

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 450;        % Maximum Number of Iterations
params.nPop = 50;          % Population Size (Swarm Size)
params.w = 1;              % Intertia Coefficient
params.wdamp = 0.99;       % Damping Ratio of Inertia Coefficient
params.c1 = 2;             % Personal Acceleration Coefficient
params.c2 = 2;             % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Informatin
params.toleranceValue = 10^(-15);

%Generation numbers needed to be plot as a graph
params.genNumber = [1,200,300,450];
%Call PSO functions
out = bm1PSO_rosenBrockFun_ContourPlots(problem, params);
```

File name: bm1PSO_rosenBrockFun_ContourPlots

```
function out = bm1PSO_rosenBrockFun_ContourPlots(problem, params)


%% Problem Definiton
CostFunction = problem.CostFunction;  % Cost Function
nVar = problem.nVar;        % Number of Unknown (Decision) Variables
VarSize = [1 nVar];         % Matrix Size of Decision Variables
VarMin = problem.VarMin;    % Lower Bound of Decision Variables
VarMax = problem.VarMax;    % Upper Bound of Decision Variables
%% Parameters of PSO
MaxIt = params.MaxIt;   % Maximum Number of Iterations
nPop = params.nPop;     % Population Size (Swarm Size)
w = params.w;           % Inertia Coefficient
wdamp = params.wdamp;   % Damping Ratio of Inertia Coefficient
c1 = params.c1;         % Personal Acceleration Coefficient
```

```
c2 = params.c2;         % Social Acceleration Coefficient
genNumber =params.genNumber;%generation index of which graph has to be ploted
% The Flag for Showing Iteration Information
ShowIterInfo = params.ShowIterInfo;
MaxVelocity = 0.2*(VarMax-VarMin);
MinVelocity = -MaxVelocity;
%% Initialization
% The Particle Template
empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Cost = [];
empty_particle.Best.Position = [];
empty_particle.Best.Cost = [];
% Create Population Array
particle = repmat(empty_particle, nPop, 1);
% Initialize Global Best
GlobalBest.Cost = inf;
% Initialize Population Members
for i=1:nPop
    % Generate Random Solution
    particle(i).Position = unifrnd(VarMin, VarMax, VarSize);
    % Initialize Velocity
    particle(i).Velocity = zeros(VarSize);%zeros([1,5])== [0 0 0 0 0]
    % Evaluation
    particle(i).Cost = CostFunction(particle(i).Position(1),particle(i).Position(2));
    % Update the Personal Best
    particle(i).Best.Position = particle(i).Position;
    particle(i).Best.Cost = particle(i).Cost;
    % Update Global Best
    if particle(i).Best.Cost < GlobalBest.Cost
        GlobalBest = particle(i).Best;
    end
end
% Array to Hold Best Cost Value on Each Iteration
BestCosts = zeros(MaxIt, 1);
%plot Control Variables
plotX = zeros(MaxIt,nPop);
plotY = zeros(MaxIt,nPop);
%% Main Loop of PSO
for it=1:MaxIt
        % scatter(1,1.^2,35,'ok','filled');  %testing -- working fine
    for i=1:nPop
        % Update Velocity
        particle(i).Velocity = w*particle(i).Velocity ...
            + c1*rand(VarSize).*(particle(i).Best.Position - particle(i).Position) ...
            + c2*rand(VarSize).*(GlobalBest.Position - particle(i).Position);
        % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity,MinVelocity);%if Velocity is lower than the min velocity
        particle(i).Velocity = min(particle(i).Velocity,MaxVelocity);%if velocity is greater than the max velocity
        % Update Position
        particle(i).Position = particle(i).Position + particle(i).Velocity;
        % Apply Lower and Upper Bound Limits
        particle(i).Position = max(particle(i).Position, VarMin);
        particle(i).Position = min(particle(i).Position, VarMax);
        % Evaluation
        particle(i).Cost = CostFunction(particle(i).Position(1),particle(i).Position(2));
        % 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;

            % Update Global Best
            if particle(i).Best.Cost < GlobalBest.Cost
                GlobalBest = particle(i).Best;
            end
        end
        %save the best soliution found in the iteration it of particle i
        %into plotX and plotY inorder to draw the graph
         plotX(it,i) = particle(i).Best.Position(1);
         plotY(it,i) = particle(i).Best.Position(2);

    end %end of nPop Iteration
    % % Store the Best Cost Value
    BestCosts(it) = GlobalBest.Cost;
    if ShowIterInfo
```

```
        disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCosts(it))]);
    end
    %plot the graph with the bestcost and globalbest received in the above
    %generation process
    if any(it == genNumber(:))
        %this line of code below is to  make sure that all graphs come into
        %seperate
        figure(it);
        % plot the contour graph
        %create mesh
        [X,Y] = meshgrid(VarMin:0.1:VarMax, VarMin:0.1:VarMax);
        %create the possible solution space with X and Y
        Z= 100*(Y-X.^2).^2+(1-X).^2;
        %draw contour
        contour(X,Y,Z);

        %mark the points in the contour graph for the iteration 'it'
        for loop = 1:nPop
            hold on;
            scatter(plotX(it,loop),plotY(it,loop),"k");
            title(['Rosenbrock Function' num2str(it) "th Generation"]);
            hold on
            scatter(GlobalBest.Position(1), GlobalBest.Position(2),"filled","o","r");
            hold off
        end
    end

    % Damping Inertia Coefficient
    w = w * wdamp;

end %end of it (MaxIt)
%%
out.pop = particle;
out.BestSol = GlobalBest;
out.BestCosts = BestCosts;
end
```

APPENDIX B

B1: Effect of Parameter: Maximum Iteration (MaxIt)

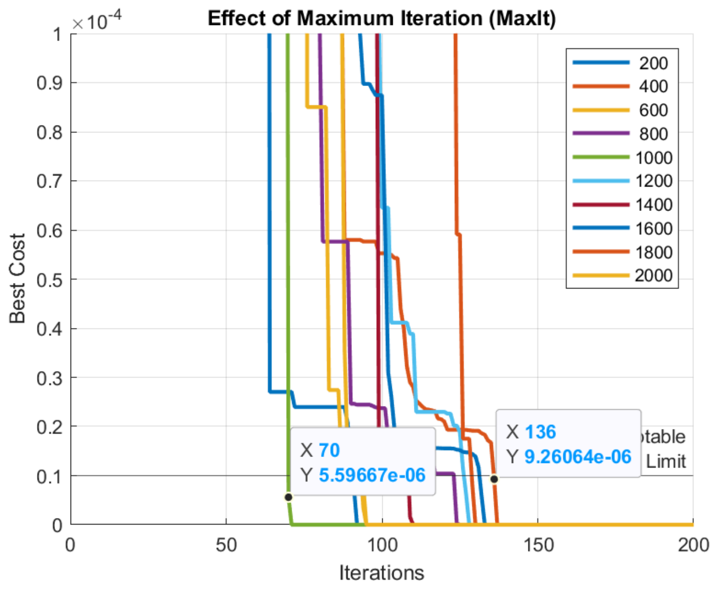


Figure 6

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO
params.nPop = 50;           % Population Size (Swarm Size)
params.w = 1;               % Intertia Coefficient
params.wdamp = 0.99;        % Damping Ratio of Inertia Coefficient
params.c1 = 2;              % Personal Acceleration Coefficient
params.c2 = 2;              % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Informatin

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Effect of Maximum Iteration (MaxIt)
%different number of 'MaxIt' values
maxItenVariation = [200:200:2000];

%Variable to store the number of Iterations take to reach the minimum
%value (In Min One Problem, it is zero)
numberOfIterations = numel(maxItenVariation);
%Run the same Binary GA 30 times
for i=1:numberOfIterations
    params.MaxIt = maxItenVariation(i);
    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    % plot the result into the existing graph
    hold on
    semilogy(out(i).BestCosts,"LineWidth",2);
    hold off
end
% Describing the attributes for the graph
title("Effect of Maximum Iteration (MaxIt)")
xlabel('Iterations');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a more close view
ylim([0 10^-4])
xlim([0 200]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to 10^-10
yline(10^-5,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(maxItenVariation.'), 'location', 'northeast');
```

B2: Effect of Parameter: Population Size/Swarm Size (nPop)

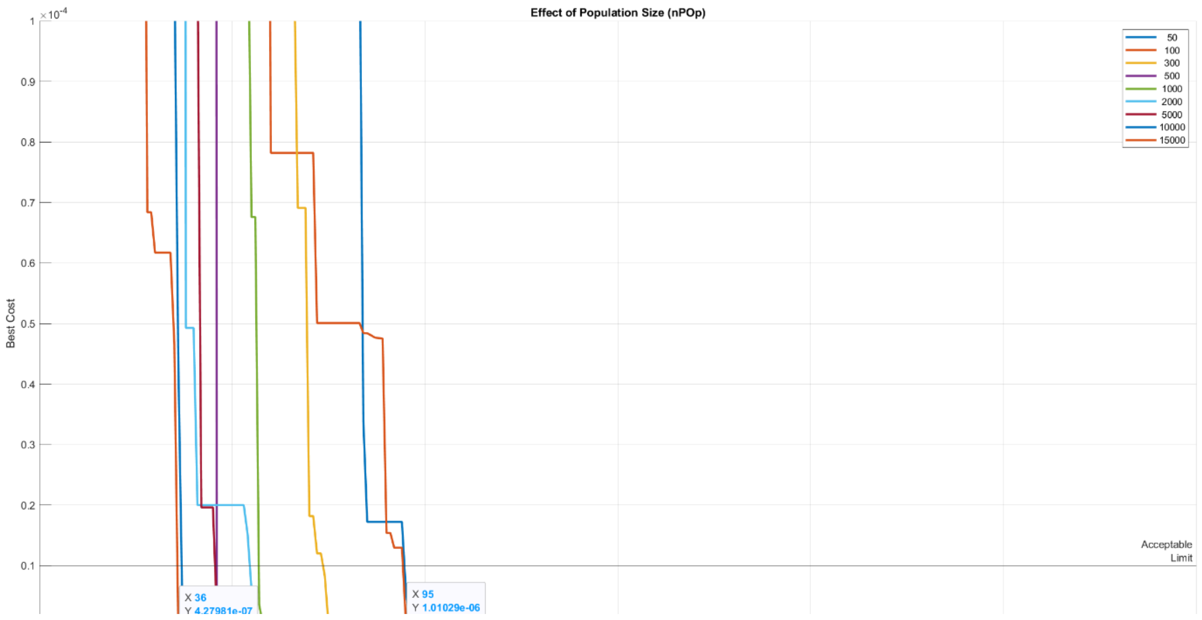


Figure 7

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;        % Number of Unknown (Decision) Variables
problem.VarMin =  -10;   % Lower Bound of Decision Variables
problem.VarMax =  10;    % Upper Bound of Decision Variables
```

```
params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Parameters of PSO

params.MaxIt = 300;        % Maximum Number of Iterations
params.w = 1;               % Intertia Coefficient
params.wdamp = 0.99;        % Damping Ratio of Inertia Coefficient
params.c1 = 2;              % Personal Acceleration Coefficient
params.c2 = 2;              % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Informatin

%% Effect of Population Size (nPop)

%different number of 'nPop' values
nPopVariation = [50, 100,300,500,1000,2000,5000,10000,15000];

%Variable to store the number of Iterations take to reach the minimum
%value (In Min One Problem, it is zero)
numberOfIterations = numel(nPopVariation);
iterationsInEachSolution = inf(1,numberOfIterations);
%Run the same Binary GA 30 times
for i=1:numberOfIterations
    params.nPop = nPopVariation(i);
    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    %filter number of non zeros, since adding one to this will give the
    %minimum iterations needed for the result
    iterationsInEachSolution(i)= nnz(out(i).BestCosts)+1;
    % plot the result into the existing graph
    hold on

semilogy(out(i).BestCosts,"LineWidth",2,'DisplayName',num2str(nPopVariation(i)));
    hold off
end
% Describing the attributes for the graph
title("Effect of Population Size (nPOp)")
xlabel('Iterations');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a more close view
ylim([0 10^-4])
xlim([0 300]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to 10^-5
yline(10^-5,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(nPopVariation.'), 'location', 'northeast');
```

B3: Effect of Parameters: Inertia Coefficient (w) & Damping Ratio of Inertia Coefficient (wdamp)

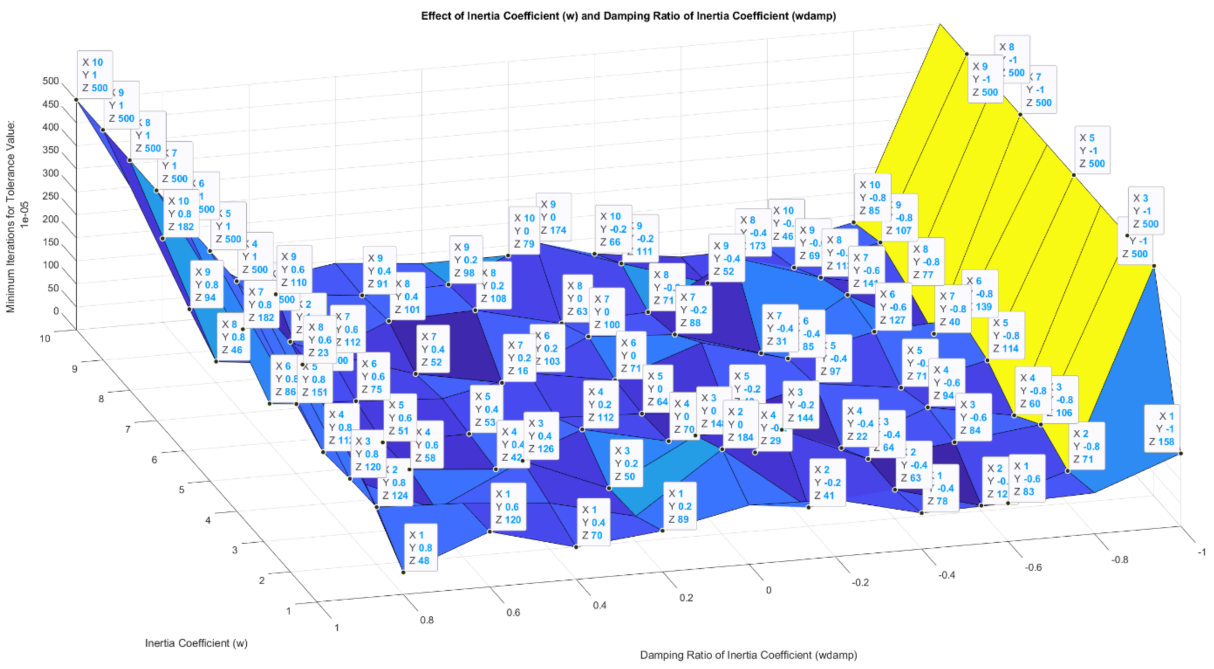


Figure 8

MATLAB Code:

```
clc; clear;

%% Problem Definiton
problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 500;        % Maximum Number of Iterations
params.nPop = 50;            % Population Size (Swarm Size)
params.c1 = 2;               % Personal Acceleration Coefficient
params.c2 = 2;               % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Information

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Effect of Inertia Coefficient (w) and Damping Ratio of Inertia Coefficient
(wdamp)

%save different values of w and wdamp inorder to do the experiment
wVariation = [1:10];
```

```
wdampVariation = [-1:0.2:1];


% size of variation arrays
sizeOfwVariation = numel(wVariation);
sizeOfwDampVariation = numel(wdampVariation);

% use two for loops to do the experiment with various combinations of w
% and wdamp
for i=1:sizeOfwVariation
     % Intertia Coefficient (w)
    params.w = wVariation(i);
    for j=1:sizeOfwDampVariation
        % Damping Ratio of Inertia Coefficient (wdamp)
        params.wdamp = wdampVariation(j);
        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end

% For ploting the result for performance analysis, An assumption has been made as
for those experimenents
% which have not reached the Tolerance value after the exicution of each
experiment can be considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfwVariation
    for j=1:sizeOfwDampVariation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end


% Describing the attributes for the graph
[X,Y] = meshgrid(wVariation,wdampVariation);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of Inertia Coefficient (w) and Damping Ratio of Inertia Coefficient
(wdamp)');
xlabel('Inertia Coefficient (w)');
ylabel('Damping Ratio of Inertia Coefficient (wdamp)');
zlabel(["Minimum Iterations for Tolerance Value: "
num2str(params.toleranceValue)]);
```

B4: Effect of Parameters: Inertia Coefficient (w) & Personal Acceleration Coefficient (c1)

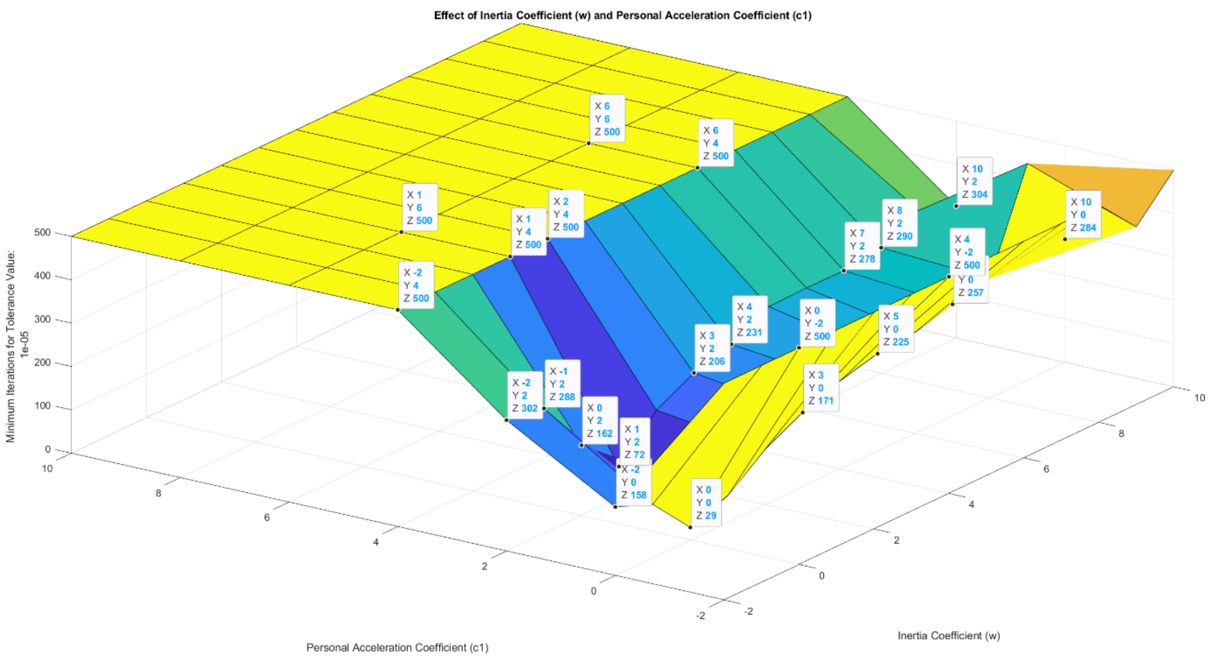


Figure 9

MATLAB Code:

```
clc; clear;

%% Problem Definiton
problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 500;        % Maximum Number of Iterations
params.nPop = 50;           % Population Size (Swarm Size)
params.wdamp = 0.99;        % Damping Ratio of Inertia Coefficient
params.c2 = 2;              % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Information

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Effect of Inertia Coefficient (w) and Personal Acceleration Coefficient (c1)

%save different values of w and c1 inorder to do the experiment
wVariation = [-2:10];
c1Variation = [-2:2:10];
```

```
% size of variation arrays
sizeOfwVariation = numel(wVariation);
sizeOfc1Variation = numel(c1Variation);

% use two for loops to do the experiment with various combinations of w
% and c1
for i=1:sizeOfwVariation
     % Intertia Coefficient
    params.w = wVariation(i);
    for j=1:sizeOfc1Variation
        % Personnel Acceleration Coefficient
        params.c1 = c1Variation(j);
        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end


% For ploting the result for performance analysis, An assumption has been made as
for those experiments
% which have not reached the Tolerance value after the exicution of each
experiment can be considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfwVariation
    for j=1:sizeOfc1Variation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end

% Describing the attributes for the graph
[X,Y] = meshgrid(wVariation,c1Variation);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of Inertia Coefficient (w) and Personal Acceleration Coefficient
(c1)');
xlabel('Inertia Coefficient (w)');
ylabel('Personal Acceleration Coefficient (c1)');
zlabel(["Minimum Iterations for Tolerance Value: "
num2str(params.toleranceValue)]);
```

B5: Effect of Parameters: Inertia Coefficient (w) & Social Acceleration Coefficient (c2)

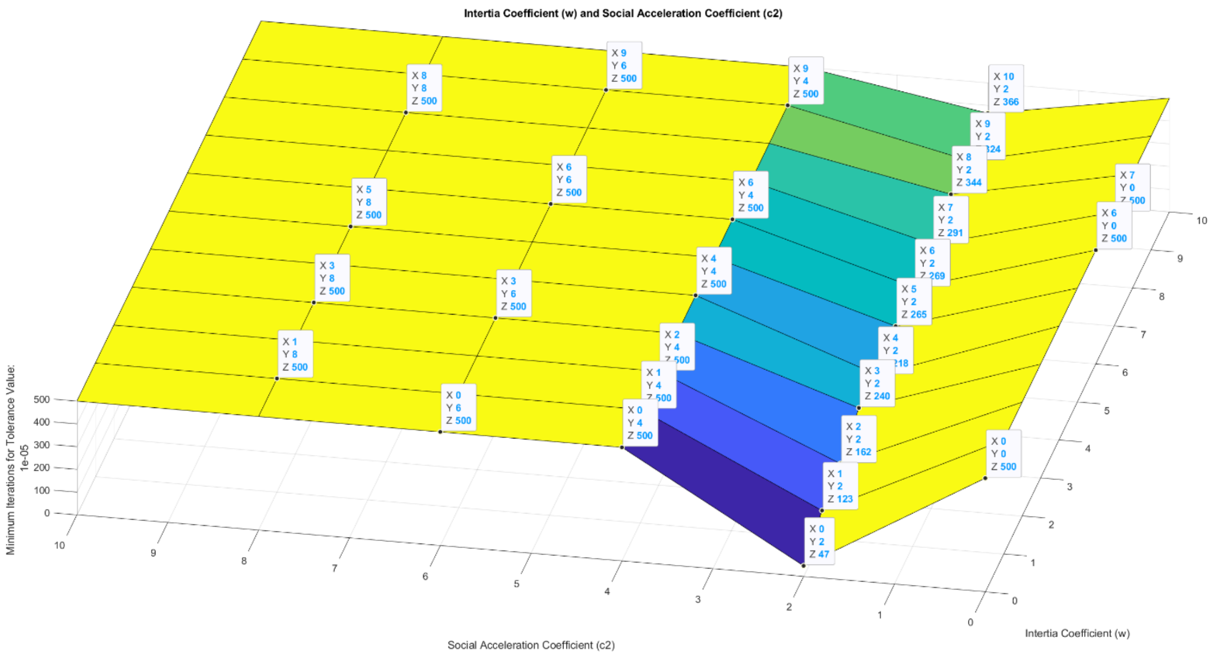


Figure 10

MATLAB Code:

```
clc; clear;

%% Problem Definiton
problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 500;        % Maximum Number of Iterations
params.nPop = 50;            % Population Size (Swarm Size)
params.c1 = 2;                % Social Acceleration Coefficient
params.wdamp = 0.99;         % Damping Ratio of Inertia Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Information

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Intertia Coefficient (w) and Social Acceleration Coefficient (c2)

%save different values of c1 and c2 inorder to do the experiment
wVariation = [0:10];
c2Variation = [0:2:10];

% size of variation arrays
sizeOfWVariation = numel(wVariation);
```

```
sizeOfc2Variation = numel(c2Variation);

% use two for loops to do the experiment with various combinations of w
% and c2
for i=1:sizeOfWVariation
     % Intertia Coefficient (w)
    params.w = wVariation(i);
    for j=1:sizeOfc2Variation
        % Social Acceleration Coefficient (c2)
        params.c2 = c2Variation(j);
        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end

% For ploting the result for performance analysis, An assumption has been made as
for those experimenents
% which have not reached the Tolerance value after the exicution of each
experiment can be considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfWVariation
    for j=1:sizeOfc2Variation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end

% Describing the attributes for the graph
[X,Y] = meshgrid(wVariation,c2Variation);
Z= transpose(Z)
surf(X,Y,Z)
title('Intertia Coefficient (w) and Social Acceleration Coefficient (c2)');
xlabel('Intertia Coefficient (w)');
ylabel('Social Acceleration Coefficient (c2)');
zlabel(["Minimum Iterations for Tolerance Value: "
num2str(params.toleranceValue)]);
```

B6: Effect of Parameters: Personal Acceleration Coefficient (c1) & Social Acceleration Coefficient (c2)

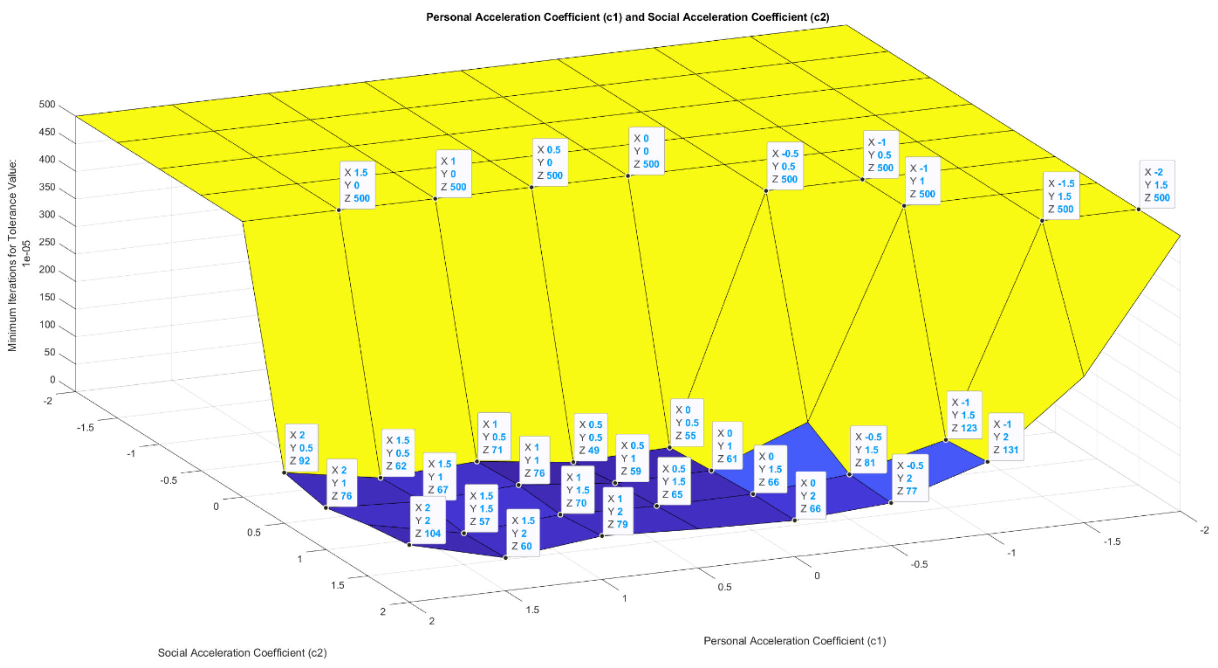


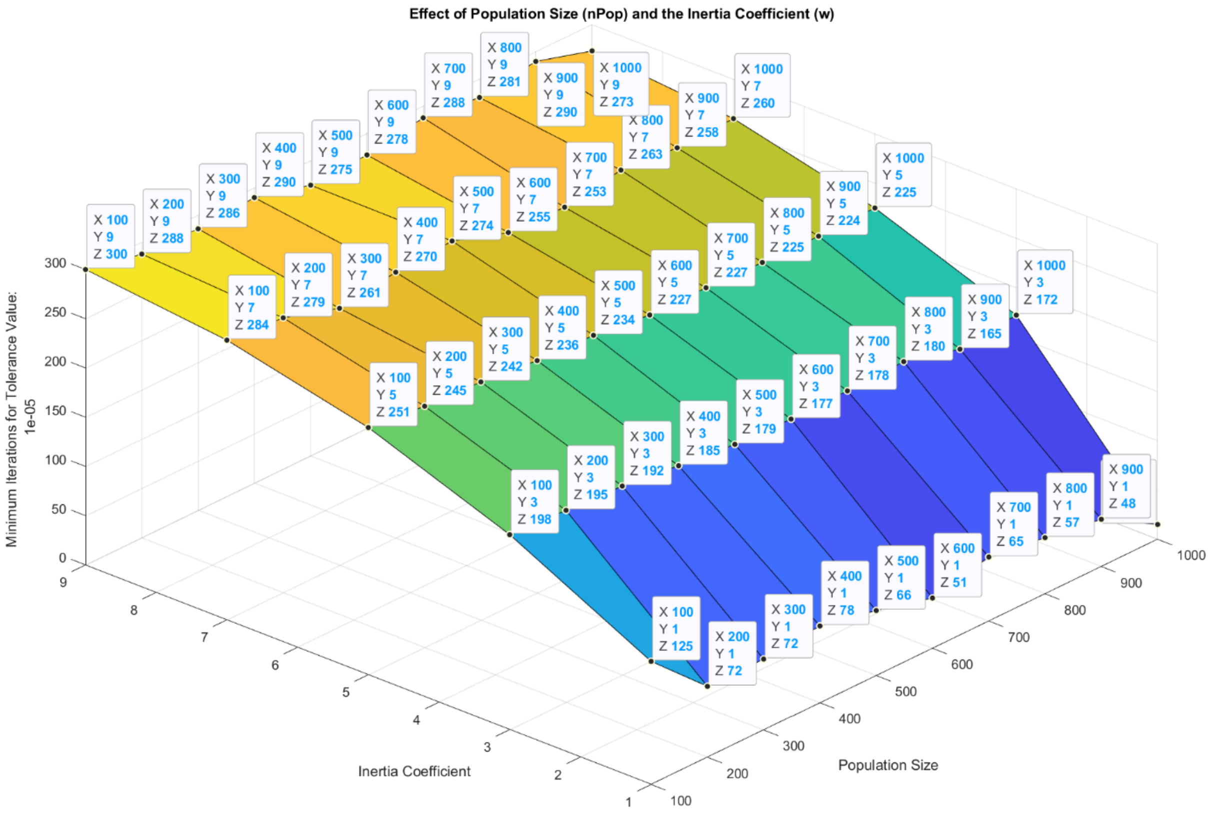
Figure 11

MATLAB Code:

```
clc; clear;

%% Problem Definiton
problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 500;        % Maximum Number of Iterations
params.nPop = 50;           % Population Size (Swarm Size)
params.w = 1;               % Intertia Coefficient
params.wdamp = 0.99;        % Damping Ratio of Inertia Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Information

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Personal Acceleration Coefficient (c1) and Social Acceleration Coefficient (c2)

%save different values of c1 and c2 inorder to do the experiment
c1Variation = [-2:0.5:2];
c2Variation = [-2:0.5:2];
```

```matlab
% size of variation arrays
sizeOfc1Variation = numel(c1Variation);
sizeOfc2Variation = numel(c2Variation);


% use two for loops to do the experiment with various combinations of c1
% and c2
for i=1:sizeOfc1Variation
     % Personal Acceleration Coefficient (c1)
    params.c1 = c1Variation(i);
    for j=1:sizeOfc2Variation
        % Social Acceleration Coefficient (c2)
        params.c2 = c2Variation(j);
        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end

% For ploting the result for performance analysis, An assumption has been made as
for those experimenents
% which have not reached the Tolerance value after the exicution of each
experiment can be considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfc1Variation
    for j=1:sizeOfc2Variation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end
% Describing the attributes for the graph
[X,Y] = meshgrid(c1Variation,c2Variation);
Z= transpose(Z)
surf(X,Y,Z)
title('Personal Acceleration Coefficient (c1) and Social Acceleration Coefficient
(c2)');
xlabel('Personal Acceleration Coefficient (c1)');
ylabel('Social Acceleration Coefficient (c2)');
zlabel(["Minimum Iterations for Tolerance Value: "
num2str(params.toleranceValue)]);
```

B4: Effect of Parameters: Population Size (nPop) & Inertia Coefficient (w)



MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 500;         % Maximum Number of Iterations
params.wdamp = 0.99;         % Damping Ratio of Inertia Coefficient
params.c1 = 2;               % Personal Acceleration Coefficient
params.c2 = 2;               % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Information

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Effect of Population Size (nPop) and the Inertia Coefficient (w)

%save different values of nPop and w inorder to do the experiment
nPopVariation = [100:100:1000];
wVariation = [1:2:10];
```

```
% size of variation arrays
sizeOfnPopVariation = numel(nPopVariation);
sizeOfwVariation = numel(wVariation);

% use two for loops to do the experiment with various combinations of nPop
% and w
for i=1:sizeOfnPopVariation
    % Population Size (Swarm Size)
    params.nPop = nPopVariation(i);
    for j=1:sizeOfwVariation
         % Intertia Coefficient
        params.w = wVariation(j);
        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end


% For ploting the result for performance analysis, An assumption has been made as
for those experiments
% which have not reached the Tolerance value after the exicution of each
experiment can be considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfnPopVariation
    for j=1:sizeOfwVariation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end

% Describing the attributes for the graph
[X,Y] = meshgrid(nPopVariation,wVariation);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of Population Size (nPop) and the Inertia Coefficient (w)')
xlabel('Population Size');
ylabel('Inertia Coefficient');
zlabel(["Minimum Iterations for Tolerance Value: "
num2str(params.toleranceValue)]);
```

B8: Best Result Reached in Benchmark B

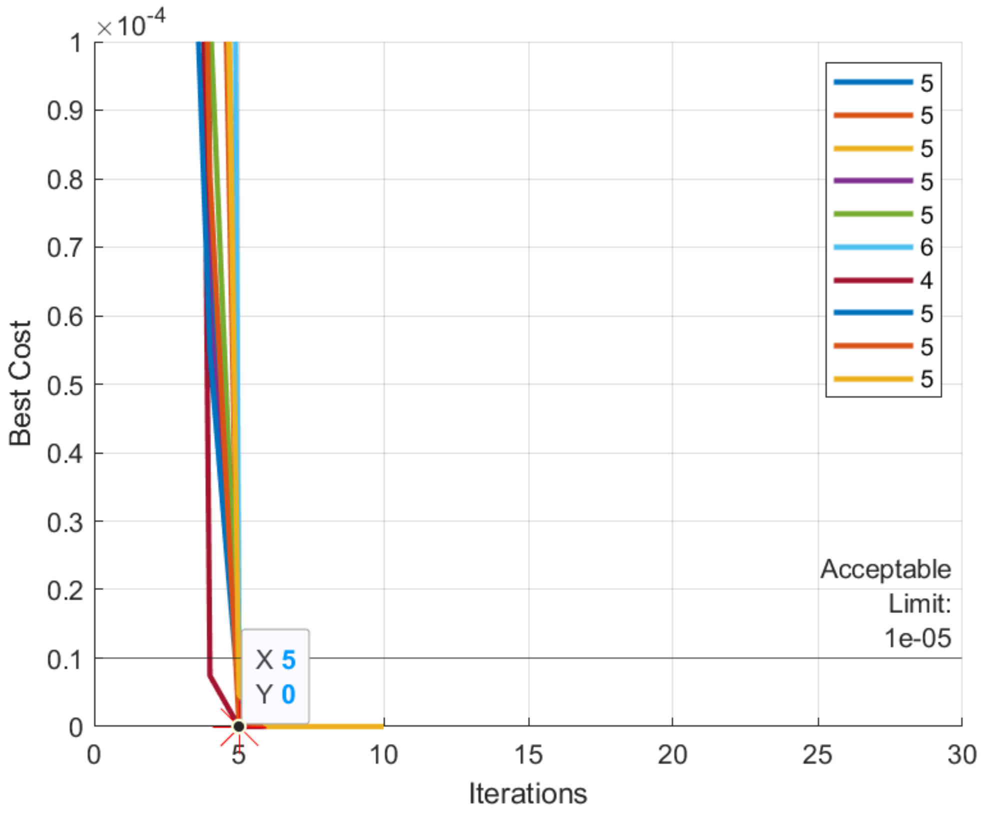


Figure 12

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin =  -10;  % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

%% Parameters of PSO

params.MaxIt = 10;         % Maximum Number of Iterations
params.nPop = 15000;            % Population Size (Swarm Size)
params.w = 7;                % Intertia Coefficient
params.wdamp = 0.2;         % Damping Ratio of Inertia Coefficient
params.c1 = 1;               % Personal Acceleration Coefficient
params.c2 = 1;               % Social Acceleration Coefficient
params.ShowIterInfo = false; % Flag for Showing Iteration Informatin
params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed

%% Calling PSO - FOR MEDIAN CALCULATION
% Run 10 experiments and find the median of those results
```

```
numberOfIterations =10;
experimentsResults = zeros(1,numberOfIterations);
for i=1:numberOfIterations
    out = pso_RosenBrockFunWithToleranceValue(problem, params);
    experimentsResults(i) = out.minIterationForToleranceValue;
    %plot the value in the graph if exists otherwise plot it in a new graph
    hold on
    semilogy(out.BestCosts,"LineWidth",2)
    grid on;
    hold off
end
disp(["Results found: " num2str(experimentsResults)])
medianOfResults = median(experimentsResults);
disp(["Median of the Results (Min iterations needed to reach the tolerance value):
" num2str(medianOfResults)])

%plot median value in the graph
hold on
semilogy(medianOfResults,0,"r*","MarkerSize",20)
hold off

xlabel('Iterations');
ylabel('Best Cost');
xlim([0,30]);
ylim([0, 10^-4])
yline(params.toleranceValue,'-',{'Acceptable','Limit:',
num2str(params.toleranceValue) });
legend(num2str(experimentsResults'), 'location', 'northeast');
```

B9: The PSO algorithm

This is the PSO algorithm used asa a function to implement the results in each section of benchmark B and benchmark C

File name: pso_RosenBrockFunWithToleranceValue.m

```
function out = pso_RosenBrockFunWithToleranceValue(problem, params)


%% Problem Definiton

CostFunction = problem.CostFunction;  % Cost Function

nVar = problem.nVar;        % Number of Unknown (Decision) Variables

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

VarMin = problem.VarMin;   % Lower Bound of Decision Variables
VarMax = problem.VarMax;    % Upper Bound of Decision Variables


%% Parameters of PSO

MaxIt = params.MaxIt;   % Maximum Number of Iterations
```

```
nPop = params.nPop;      % Population Size (Swarm Size)

w = params.w;            % Inertia Coefficient
wdamp = params.wdamp;    % Damping Ratio of Inertia Coefficient
c1 = params.c1;          % Personal Acceleration Coefficient
c2 = params.c2;          % Social Acceleration Coefficient

% The Flag for Showing Iteration Information
ShowIterInfo = params.ShowIterInfo;

% Define the velocity bounds
MaxVelocity = 0.2*(VarMax-VarMin);
MinVelocity = -MaxVelocity;

%Defining the paramters for Tolerance Value for Acceptace of the Result
%(Best Cost for the objective Function)
toleranceValue = params.toleranceValue;
minIterationForToleranceValue = Inf;

%% Initialization

% The Particle Template
empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Cost = [];
empty_particle.Best.Position = [];
empty_particle.Best.Cost = [];

% Create Population Array
particle = repmat(empty_particle, nPop, 1);

% Initialize Global Best
GlobalBest.Cost = inf;

% Initialize Population Members
for i=1:nPop

    % Generate Random Solution
    particle(i).Position = unifrnd(VarMin, VarMax, VarSize);

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

    % Evaluation
    particle(i).Cost =
CostFunction(particle(i).Position(1),particle(i).Position(2));

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

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

end

% Array to Hold Best Cost Value on Each Iteration
```

```
BestCosts = zeros(MaxIt, 1);


%% Main Loop of PSO

for it=1:MaxIt

    for i=1:nPop

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

        % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity, MinVelocity);
        particle(i).Velocity = min(particle(i).Velocity, MaxVelocity);

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

        % Apply Lower and Upper Bound Limits
        particle(i).Position = max(particle(i).Position, VarMin);
        particle(i).Position = min(particle(i).Position, VarMax);

        % Evaluation
        particle(i).Cost =
CostFunction(particle(i).Position(1),particle(i).Position(2));

        % 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;

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

        end

    end

    % Store the Best Cost Value
    BestCosts(it) = GlobalBest.Cost;

    %Compare the bestcost found in this iteration with the tolerance value
    if abs(BestCosts(it)) < toleranceValue
        disp(["Tolerance value reached in " num2str(it) "th iteration"]);
        minIterationForToleranceValue = it;
        %If Tolerance solution has been reached, return from the function -
        %this is done for the purpous of efficiency and readability
        out.pop = particle;
        out.BestSol = GlobalBest;
        out.BestCosts = BestCosts;
        out.minIterationForToleranceValue = minIterationForToleranceValue;
        % If the tolerance value has been reached, return from the function
        return;
    end

    % Display Iteration Information
    if ShowIterInfo
        disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCosts(it))]);
    end

    % Damping Inertia Coefficient
    w = w * wdamp;

end

out.pop = particle;
out.BestSol = GlobalBest;
out.BestCosts = BestCosts;
out.minIterationForToleranceValue = minIterationForToleranceValue;

end
```

APPENDIX C

C1: Effect of Parameter: Maximum Iteration (MaxIt)

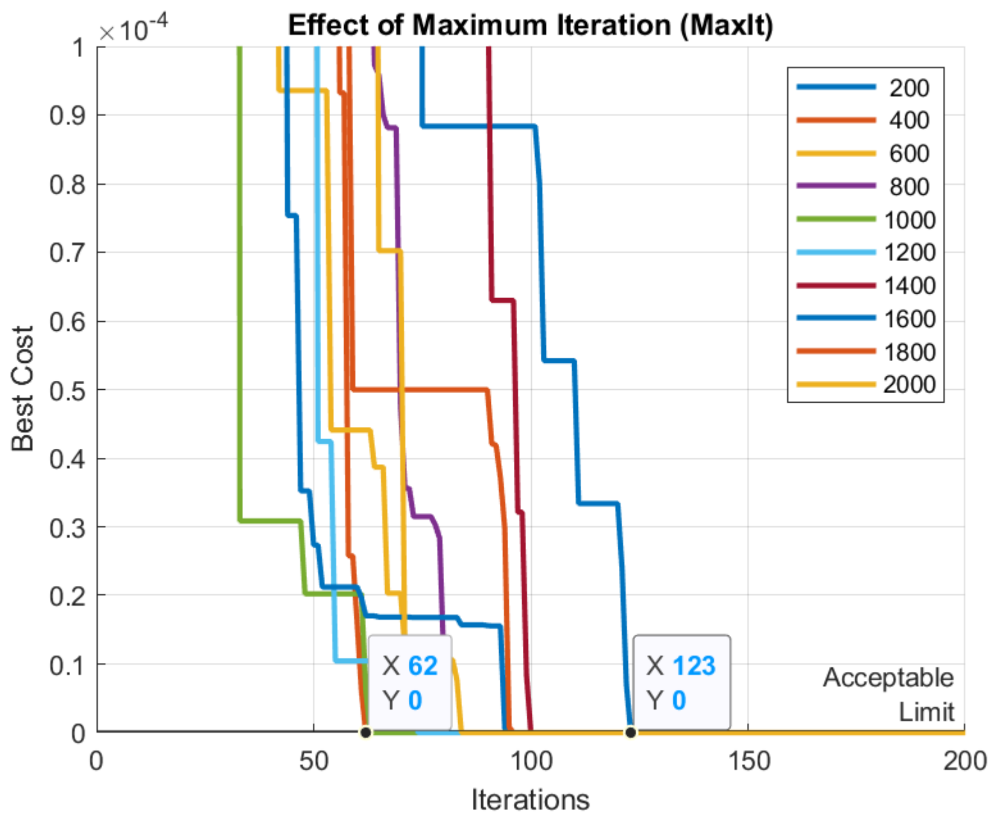


Figure 13

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin = -10;   % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables


%% Parameters of PSO

% Constriction Coefficients
kappa = 1;
phi1 = 2.05;
phi2 = 2.05;
phi = phi1 + phi2;
chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

params.nPop = 50;           % Population Size (Swarm Size)
params.w = chi;             % Intertia Coefficient
params.wdamp = 1;            % Damping Ratio of Inertia Coefficient
params.c1 = chi*phi1;        % Personal Acceleration Coefficient
params.c2 = chi*phi2;        % Social Acceleration Coefficient
params.ShowIterInfo = true; % Flag for Showing Iteration Information
params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed

%% Effect of Maximum Iteration (MaxIt)
%different number of 'MaxIt' values
maxItenVariation = [200:200:2000];

%Variable to store the number of Iterations take to reach the minimum
%value (In Min One Problem, it is zero)
numberOfIterations = numel(maxItenVariation);
%Run the same Binary GA 30 times
for i=1:numberOfIterations
    params.MaxIt = maxItenVariation(i);
    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    % plot the result into the existing graph
    hold on
    semilogy(out(i).BestCosts,"LineWidth",2);
    hold off
end
% Describing the attributes for the graph
title("Effect of Maximum Iteration (MaxIt)")
xlabel('Iterations');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a more close view
ylim([0 10^-4])
xlim([0 200]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to 10^-10
yline(10^-5,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(maxItenVariation.'), 'location', 'northeast');
```

C2: Effect of Parameter: Population Size/Swarm Size (nPop)

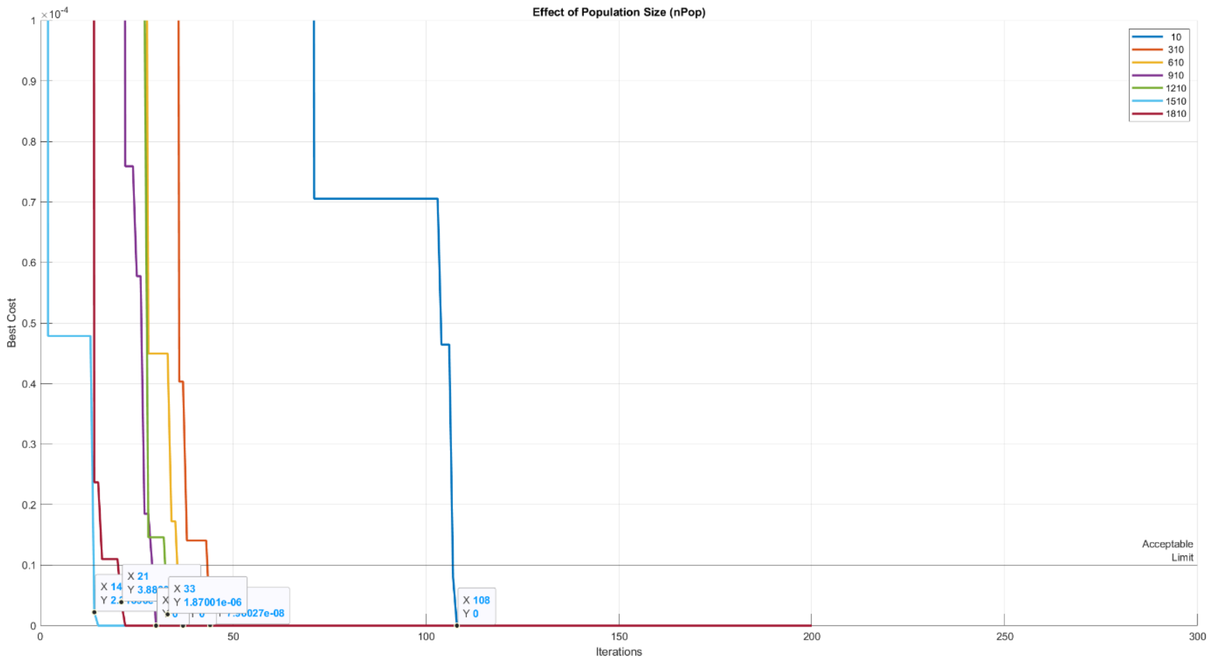


Figure 14

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;      % Number of Unknown (Decision) Variables
problem.VarMin = -10;   % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables


%% Parameters of PSO

% Constriction Coefficients
kappa = 1;
phi1 = 2.05;
phi2 = 2.05;
phi = phi1 + phi2;
chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

params.MaxIt = 200;        % Maximum Number of Iterations
params.w = chi;             % Intertia Coefficient
params.wdamp = 1;           % Damping Ratio of Inertia Coefficient
params.c1 = chi*phi1;       % Personal Acceleration Coefficient
params.c2 = chi*phi2;       % Social Acceleration Coefficient
params.ShowIterInfo = true; % Flag for Showing Iteration Information
params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed

%% Effect of Maximum Iteration (MaxIt)
```

```matlab
%different number of 'MaxIt' values
nPopVariation = [10:300:2000];

%Variable to store the number of Iterations take to reach the minimum
%value (In Min One Problem, it is zero)
numberOfIterations = numel(nPopVariation);
%Run the same Binary GA 30 times
for i=1:numberOfIterations
    params.nPop = nPopVariation(i);
    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    % plot the result into the existing graph
    hold on
    semilogy(out(i).BestCosts,"LineWidth",2);
    hold off
end
% Describing the attributes for the graph
title("Effect of Population Size (nPop)")
xlabel('Iterations');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a more close view
ylim([0 10^-4])
xlim([0 300]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to 10^-10
yline(10^-5,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(nPopVariation.'), 'location', 'northeast');
```

C3: Effect of Constriction Coefficient: Kappa

C3.A: For Kappa = {0.50, 0.51, 0.55, 0.7, 0.8, 0.9, 1}

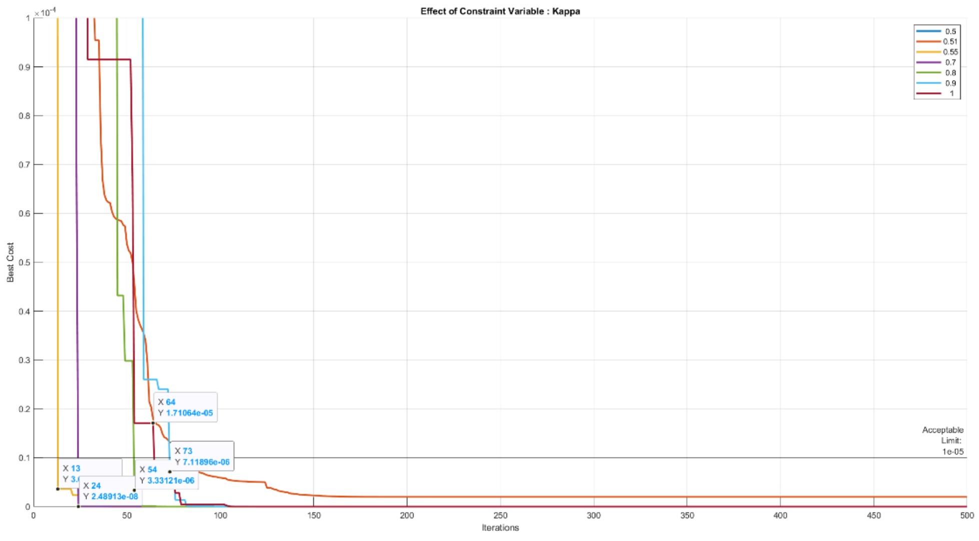


Figure 15

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin = -10;    % Lower Bound of Decision Variables
problem.VarMax =  10;    % Upper Bound of Decision Variables



params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Effect of Constraint Variable : Kappa

%different number of 'kappa' values
% kappaVariations = [1:0.05:1.5]; %values in between 1 and 1.5 -- B
kappaVariations = [0.50, 0.51, 0.55, 0.7, 0.8,0.9,1]; %values in between
% 0 and 1 C


%Variable to store the number of Iterations take to reach the minimum
%value (In Min One Problem, it is zero)
numberOfIterations = numel(kappaVariations);
%run the experiment for numberOfIterations times
for i=1:numberOfIterations
    % Parameters of PSO
    % Constriction Coefficients
    kappa = kappaVariations(i);
    phi1 = 2.05;
    phi2 = 2.05;
    phi = phi1 + phi2;
    chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

    params.MaxIt = 500;        % Maximum Number of Iterations
    params.w = chi;             % Intertia Coefficient
    params.wdamp = 1;           % Damping Ratio of Inertia Coefficient
    params.c1 = chi*phi1;       % Personal Acceleration Coefficient
    params.c2 = chi*phi2;       % Social Acceleration Coefficient
    params.ShowIterInfo = true; % Flag for Showing Iteration Information
    params.nPop = 50;           % Population Size (Swarm Size)

    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    % plot the result into the existing graph
    hold on

semilogy(out(i).BestCosts,"LineWidth",2,'DisplayName',num2str(kappaVariations(i)))
;
    hold off
end
% Describing the attributes for the graph
title("Effect of Constraint Variable : Kappa")
xlabel('Iterations');
ylabel('Best Cost');
% make the y limit to a narrow region to get a closer look
ylim([0 10^-4]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to the tolerance value
yline(params.toleranceValue,'-',{'Acceptable','Limit: '
num2str(params.toleranceValue)});
grid on;
legend(num2str(kappaVariations.'), 'location', 'northeast');
```

C3.B: For Kappa = {1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4, 1.45, 1.5}

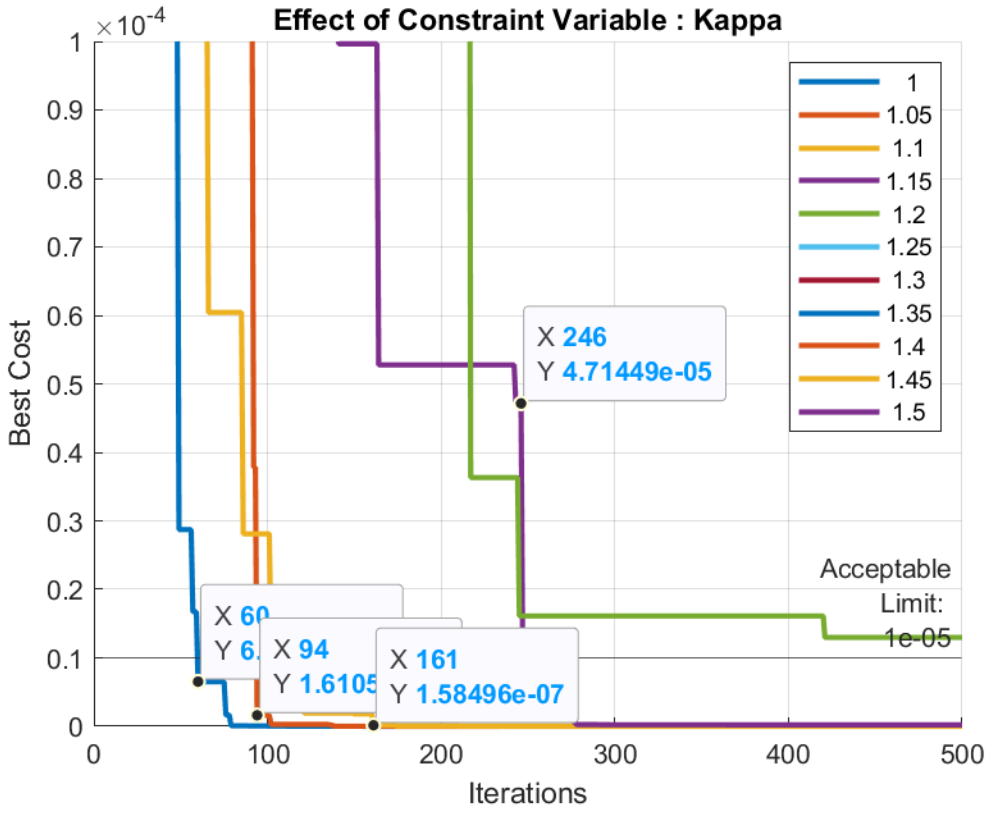


Figure 16

C4: Effect of Constriction Coefficients: Kappa & phi1

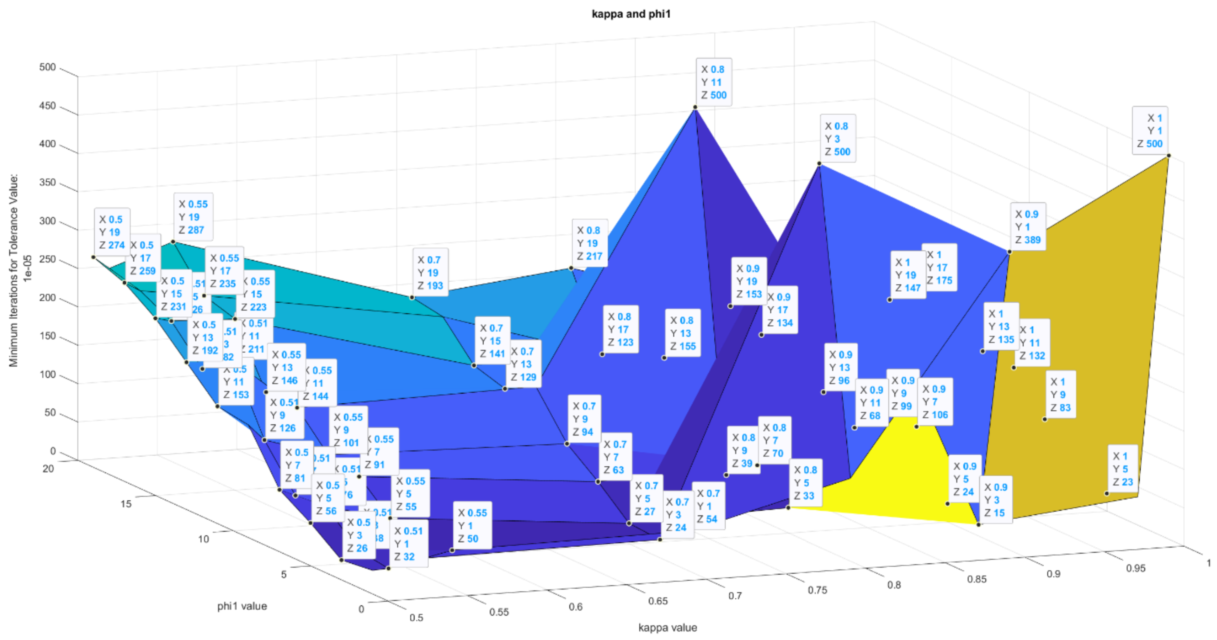


Figure 17

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin = -10;    % Lower Bound of Decision Variables
problem.VarMax =  10;    % Upper Bound of Decision Variables

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as
solution with the maximum error allowed
%% Effect of Constraint Variable : Kappa and phi1

%different number of 'kappa' values
% kappaVariations = [1:0.05:1.5]; %values in between 1 and 1.5 -- B
kappaVariations = [0.50, 0.51, 0.55, 0.7, 0.8,0.9,1]; %values in between
%different values of phi1
phi1Variations = 1:2:20;

% size of variation arrays
sizeOfkappaVariation = numel(kappaVariations);
sizeOfphi1Variation = numel(phi1Variations);


% use two for loops to do the experiment with various combinations of c1
% and c2
for i=1:sizeOfkappaVariation
     % Personal Acceleration Coefficient (c1)
    kappa = kappaVariations(i);
    for j=1:sizeOfphi1Variation
         phi1 = phi1Variations(j)
         phi2 = 2.05;
         phi = phi1 + phi2;
         chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

         params.MaxIt = 500;        % Maximum Number of Iterations
         params.w = chi;             % Intertia Coefficient
         params.wdamp = 1;           % Damping Ratio of Inertia Coefficient
         params.c1 = chi*phi1;       % Personal Acceleration Coefficient
         params.c2 = chi*phi2;       % Social Acceleration Coefficient
         params.ShowIterInfo = true; % Flag for Showing Iteration Information
         params.nPop = 50;           % Population Size (Swarm Size)

        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end

% For ploting the result for performance analysis, An assumption has been made as
for those experimenents
% which have not reached the Tolerance value after the exicution of each
experiment can be considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfkappaVariation
    for j=1:sizeOfphi1Variation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end
% Describing the attributes for the graph
[X,Y] = meshgrid(kappaVariations,phi1Variations);
Z= transpose(Z)
surf(X,Y,Z)
title('Personal Acceleration Coefficient (c1) and Social Acceleration Coefficient
(c2)');
xlabel('kappa value');
ylabel('phi1 value');
zlabel(["Minimum Iterations for Tolerance Value: "
num2str(params.toleranceValue)]);
```

C5: MATLAB Code: Drawing Contour Plots under section- Benchmark C

```
function out = bm1PSO_rosenBrockFun_ContourPlots(problem, params)


%% Problem Definiton

CostFunction = problem.CostFunction;  % Cost Function

nVar = problem.nVar;         % Number of Unknown (Decision) Variables

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

```
VarMin = problem.VarMin;  % Lower Bound of Decision Variables
VarMax = problem.VarMax;    % Upper Bound of Decision Variables


%% Parameters of PSO

MaxIt = params.MaxIt;   % Maximum Number of Iterations
nPop = params.nPop;     % Population Size (Swarm Size)

w = params.w;           % Inertia Coefficient
wdamp = params.wdamp;   % Damping Ratio of Inertia Coefficient

c1 = params.c1;         % Personal Acceleration Coefficient
c2 = params.c2;         % Social Acceleration Coefficient

genNumber =params.genNumber;%generation index of which graph has to be ploted

% The Flag for Showing Iteration Information
ShowIterInfo = params.ShowIterInfo;

MaxVelocity = 0.2*(VarMax-VarMin);
MinVelocity = -MaxVelocity;


%% Initialization

% The Particle Template
empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Cost = [];
empty_particle.Best.Position = [];
empty_particle.Best.Cost = [];

% Create Population Array
particle = repmat(empty_particle, nPop, 1);

% Initialize Global Best
GlobalBest.Cost = inf;

% Initialize Population Members
for i=1:nPop

    % Generate Random Solution
    particle(i).Position = unifrnd(VarMin, VarMax, VarSize);

    % Initialize Velocity
    particle(i).Velocity = zeros(VarSize);%zeros([1,5])== [0 0 0 0 0]

    % Evaluation
    particle(i).Cost = 
CostFunction(particle(i).Position(1),particle(i).Position(2));

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

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

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

end

% Array to Hold Best Cost Value on Each Iteration
BestCosts = zeros(MaxIt, 1);

%plot Control Variables
plotX = zeros(MaxIt,nPop);
plotY = zeros(MaxIt,nPop);
%% Main Loop of PSO
for it=1:MaxIt
        % scatter(1,1.^2,35,'ok','filled');  %testing -- working fine
    for i=1:nPop
        % Update Velocity
        particle(i).Velocity = w*particle(i).Velocity ...
            + c1*rand(VarSize).*(particle(i).Best.Position - particle(i).Position)
...
            + c2*rand(VarSize).*(GlobalBest.Position - particle(i).Position);

        % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity,MinVelocity);%if Velocity
is lower than the min velocity
        particle(i).Velocity = min(particle(i).Velocity,MaxVelocity);%if velocity
is greater than the max velocity

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

        % Apply Lower and Upper Bound Limits
        particle(i).Position = max(particle(i).Position, VarMin);
        particle(i).Position = min(particle(i).Position, VarMax);

        % Evaluation
        particle(i).Cost =
CostFunction(particle(i).Position(1),particle(i).Position(2));

        % 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;

            % Update Global Best
            if particle(i).Best.Cost < GlobalBest.Cost
                GlobalBest = particle(i).Best;
            end
        end
        %save the best soliution found in the iteration it of particle i
        %into plotX and plotY inorder to draw the graph
         plotX(it,i) = particle(i).Best.Position(1);
         plotY(it,i) = particle(i).Best.Position(2);

    end %end of nPop Iteration
    % % Store the Best Cost Value
    BestCosts(it) = GlobalBest.Cost;
    if ShowIterInfo
        disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCosts(it))]);
    end
    %plot the graph with the bestcost and globalbest received in the above
    %generation process
    if any(it == genNumber(:))
        %this line of code below is to  make sure that all graphs come into
        %seperate
        figure(it);
        % plot the contour graph
        %create mesh
        [X,Y] = meshgrid(VarMin:0.1:VarMax, VarMin:0.1:VarMax);
        %create the possible solution space with X and Y
        Z= 100*(Y-X.^2).^2+(1-X).^2;
        %draw contour
        contour(X,Y,Z);

        %mark the points in the contour graph for the iteration 'it'
        for loop = 1:nPop
            hold on;
            scatter(plotX(it,loop),plotY(it,loop),"k");
            title(['Rosenbrock Function' num2str(it) "th Generation"]);
            hold on
            scatter(GlobalBest.Position(1),
GlobalBest.Position(2),"filled","o","r");
            hold off
        end
    end

    % Damping Inertia Coefficient
    w = w * wdamp;

end %end of it (MaxIt)
%%
out.pop = particle;
out.BestSol = GlobalBest;
out.BestCosts = BestCosts;

end
```

APPENDIX D

D1: Effect of Constriction Coefficients: Kappa

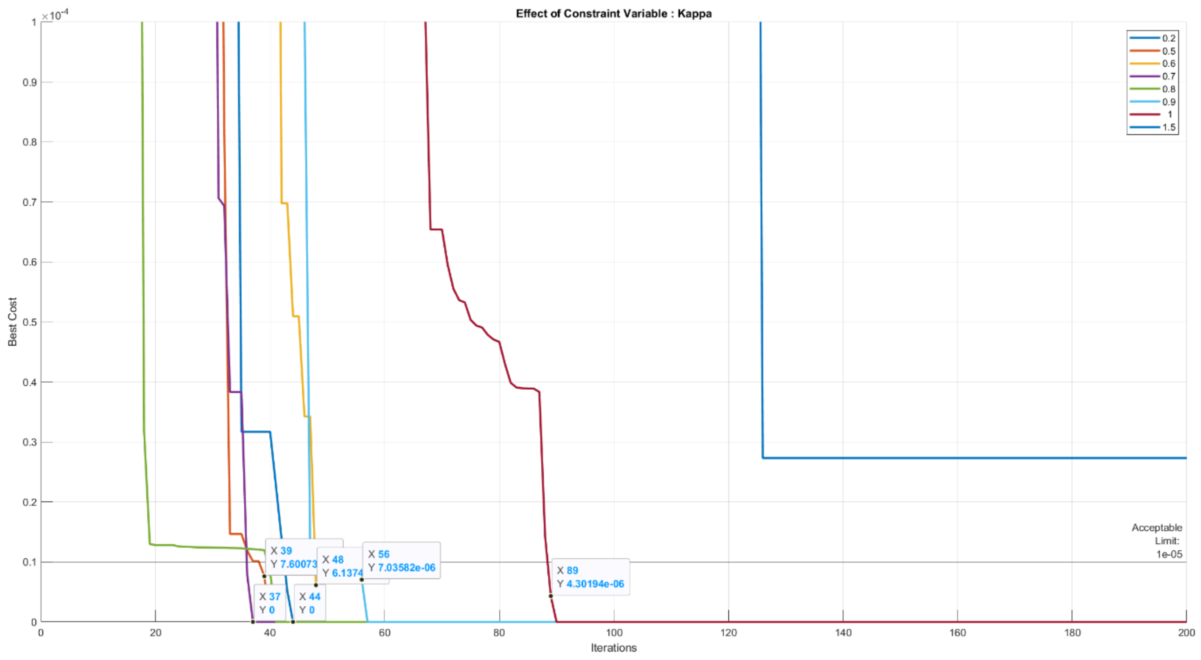


Figure 18

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;      % Number of Unknown (Decision) Variables
problem.VarMin = -10;   % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables



params.toleranceValue = 10^(-5); %tolerance value which can be accepted as solution with the
maximum error allowed
%% Effect of Constraint Variable : Kappa with dynamic damping of Inertia Coefficient

%different number of 'kappa' values
% kappaVariations = [1:0.05:1.5]; %values in between 1 and 1.5 -- B
kappaVariations = [0.2,0.50, 0.6 0.7, 0.8,0.9,1,1.5]; %values in between
% 0 and 1 C


%Variable to store the number of Iterations take to reach the minimum
%value (In Min One Problem, it is zero)
numberOfIterations = numel(kappaVariations);
%run the experiment for numberOfIterations times
for i=1:numberOfIterations
    % Parameters of PSO
    % Constriction Coefficients
    kappa = kappaVariations(i);
    phi1 = 2.05;
    phi2 = 2.05;
    phi = phi1 + phi2;
    chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

    params.MaxIt = 500;        % Maximum Number of Iterations
    %adding dynamici nature to both w (Inertia Coefficient) and wdamp (Damping
    %Ratio of Inertia Coefficient)
    params.wmax = chi;          % Maximum value of Intertia Coefficient
    params.wmin = 0.1;           % Minimum value of Intertia Coefficient
    params.c1 = chi*phi1;       % Personal Acceleration Coefficient
    params.c2 = chi*phi2;       % Social Acceleration Coefficient
    params.ShowIterInfo = true; % Flag for Showing Iteration Information
    params.nPop = 50;           % Population Size (Swarm Size)

    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    % plot the result into the existing graph
    hold on
    semilogy(out(i).BestCosts,"LineWidth",2,'DisplayName',num2str(kappaVariations(i)));
    hold off
end
% Describing the attributes for the graph
title("Effect of Constraint Variable : Kappa")
xlabel('Iterations');
ylabel('Best Cost');
% make the y limit to a narrow region to get a closer look
ylim([0 10^-4]);
xlim([0 200])
%draw a line parallel to x axis to find on which iteration the output
%reaches to the tolerance value
yline(params.toleranceValue,'-',{'Acceptable','Limit: ' num2str(params.toleranceValue)});
grid on;
legend(num2str(kappaVariations.'), 'location', 'northeast');
```

---

filename: pso_RosenBrockFunWithToleranceValue

```
function out = pso_RosenBrockFunWithToleranceValue(problem, params)


%% Problem Definiton

CostFunction = problem.CostFunction;  % Cost Function

nVar = problem.nVar;        % Number of Unknown (Decision) Variables

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

VarMin = problem.VarMin;          % Lower Bound of Decision Variables
VarMax = problem.VarMax;    % Upper Bound of Decision Variables


%% Parameters of PSO

MaxIt = params.MaxIt;   % Maximum Number of Iterations

nPop = params.nPop;     % Population Size (Swarm Size)
```

```
wmax = params.wmax;     % Maximum value of Inertia Coefficient
wmin = params.wmin;     % Minimum value of Inertia Coefficient
c1 = params.c1;         % Personal Acceleration Coefficient
c2 = params.c2;         % Social Acceleration Coefficient

% The Flag for Showing Iteration Information
ShowIterInfo = params.ShowIterInfo;

% Define the velocity bounds
MaxVelocity = 0.2*(VarMax-VarMin);
MinVelocity = -MaxVelocity;

%Defining the paramters for Tolerance Value for Acceptace of the Result
%(Best Cost for the objective Function)
toleranceValue = params.toleranceValue;
minIterationForToleranceValue = Inf;

%% Initialization

% The Particle Template
empty_particle.Position = [];
empty_particle.Velocity = [];
empty_particle.Cost = [];
empty_particle.Best.Position = [];
empty_particle.Best.Cost = [];

% Create Population Array
particle = repmat(empty_particle, nPop, 1);

% Initialize Global Best
GlobalBest.Cost = inf;

% Initialize Population Members
for i=1:nPop

    % Generate Random Solution
    particle(i).Position = unifrnd(VarMin, VarMax, VarSize);

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

    % Evaluation
    particle(i).Cost = CostFunction(particle(i).Position(1),particle(i).Position(2));

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

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

end

% Array to Hold Best Cost Value on Each Iteration
BestCosts = zeros(MaxIt, 1);


%% Main Loop of PSO
```

```
for it=1:MaxIt

    for i=1:nPop

        % Calculating Inertia Coefficient (w) using wmax and wmin
        w = wmax - ((i/it)*(wmax-wmin));

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

        % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity, MinVelocity);
        particle(i).Velocity = min(particle(i).Velocity, MaxVelocity);

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

        % Apply Lower and Upper Bound Limits
        particle(i).Position = max(particle(i).Position, VarMin);
        particle(i).Position = min(particle(i).Position, VarMax);

        % Evaluation
        particle(i).Cost = CostFunction(particle(i).Position(1),particle(i).Position(2));

        % 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;

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

        end

    end

    % Store the Best Cost Value
    BestCosts(it) = GlobalBest.Cost;

    %Compare the bestcost found in this iteration with the tolerance value
    if abs(BestCosts(it)) < toleranceValue
        disp(["Tolerance value reached in " num2str(it) "th iteration"]);
        minIterationForToleranceValue = it;
        %If Tolerance solution has been reached, return from the function -
        %this is done for the purpous of efficiency and readability
        out.pop = particle;
        out.BestSol = GlobalBest;
        out.BestCosts = BestCosts;
        out.minIterationForToleranceValue = minIterationForToleranceValue;
        % If the tolerance value has been reached, return from the function
        return;
    end

    % Display Iteration Information
```

```
        if ShowIterInfo
            disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCosts(it))]);
        end



    end

    out.pop = particle;
    out.BestSol = GlobalBest;
    out.BestCosts = BestCosts;
    out.minIterationForToleranceValue = minIterationForToleranceValue;

    end
```

D2: Effect of Constriction Coefficients: Kappa and phi1

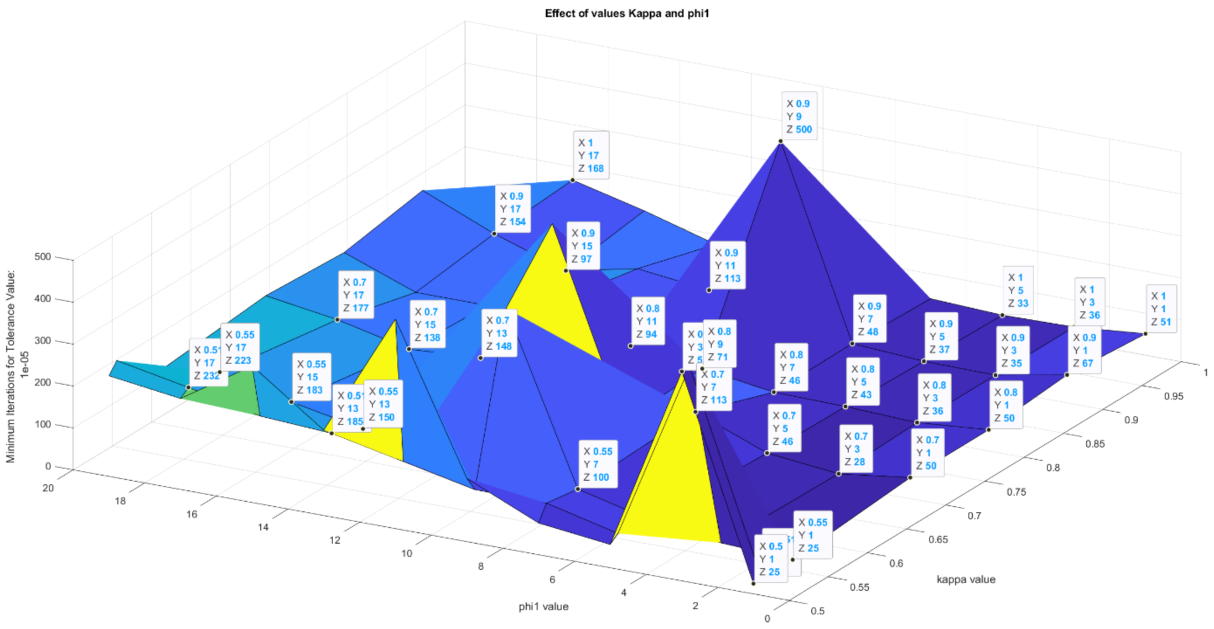


Figure 19

MATLAB Code:

```
clc; clear;

%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin = -10;   % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as solution with the
maximum error allowed
%% Effect of Constraint Variable : Kappa and phi1
```

```
%different number of 'kappa' values
% kappaVariations = [1:0.05:1.5]; %values in between 1 and 1.5 -- B
kappaVariations = [0.50, 0.51, 0.55, 0.7, 0.8,0.9,1]; %values in between
%different values of phi1
phi1Variations = 1:2:20;

% size of variation arrays
sizeOfkappaVariation = numel(kappaVariations);
sizeOfphi1Variation = numel(phi1Variations);


% use two for loops to do the experiment with various combinations of c1
% and c2
for i=1:sizeOfkappaVariation
     % Personal Acceleration Coefficient (c1)
    kappa = kappaVariations(i);
    for j=1:sizeOfphi1Variation
         phi1 = phi1Variations(j)
         phi2 = 2.05;
         phi = phi1 + phi2;
         chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

         params.MaxIt = 500;        % Maximum Number of Iterations
       %adding dynamici nature to both w (Inertia Coefficient) and wdamp (Damping
       %Ratio of Inertia Coefficient)
       params.wmax = chi;          % Maximum value of Intertia Coefficient
       params.wmin = 0.1;            % Minimum value of Intertia Coefficient
       params.c1 = chi*phi1;       % Personal Acceleration Coefficient
         params.c2 = chi*phi2;       % Social Acceleration Coefficient
         params.ShowIterInfo = true; % Flag for Showing Iteration Information
         params.nPop = 50;           % Population Size (Swarm Size)

        out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
        Z(i,j)= out(i).minIterationForToleranceValue;
    end
end

% For ploting the result for performance analysis, An assumption has been made as for those
experimenents
% which have not reached the Tolerance value after the exicution of each experiment can be
considered as the one
% which needed more than MaxIt, so MaxIt value replaces as result
% where "Inf" - (infinite) is assigned.
for i=1:sizeOfkappaVariation
    for j=1:sizeOfphi1Variation
        if Z(i,j) == Inf
            %assigning MaxIt as result for Inf value
            Z(i,j) = params.MaxIt;
        end
    end
end
% Describing the attributes for the graph
[X,Y] = meshgrid(kappaVariations,phi1Variations);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of values Kappa and phi1');
xlabel('kappa value');
ylabel('phi1 value');
zlabel(["Minimum Iterations for Tolerance Value: " num2str(params.toleranceValue)]);
```

D3: Effect of Constriction Coefficients: wmin

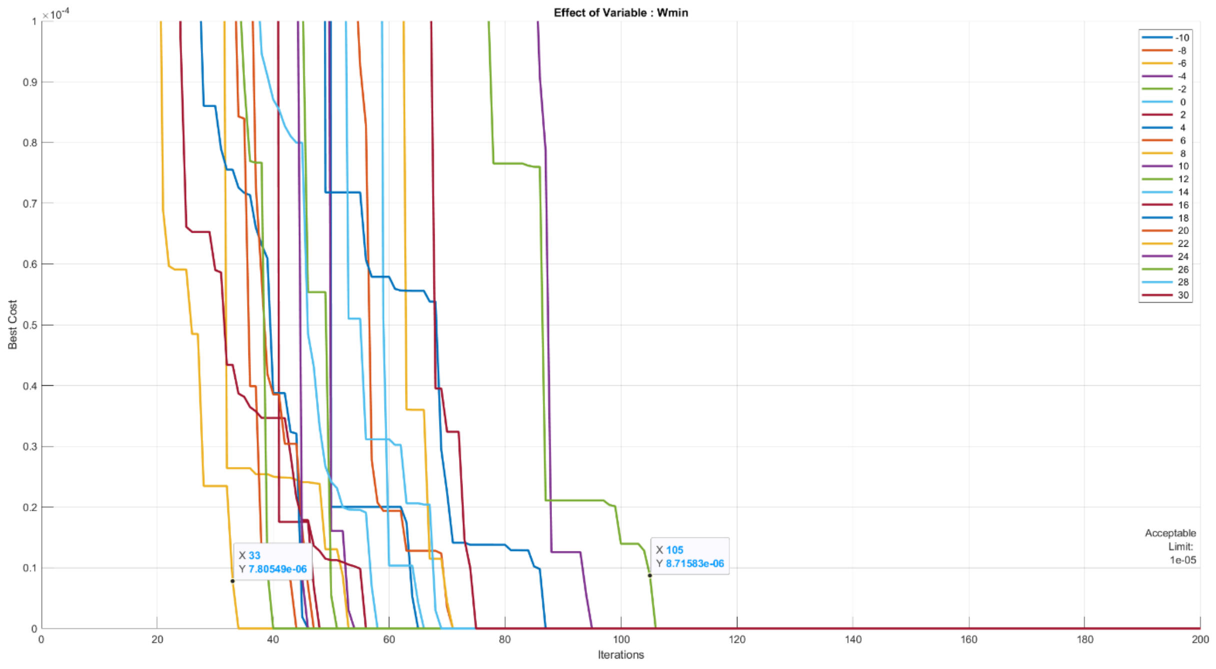


Figure 20

MATLAB Code:

```
%% Problem Definiton

problem.CostFunction = @(x,y) RosenBrockFunction(x,y);  % Cost Function
problem.nVar = 2;       % Number of Unknown (Decision) Variables
problem.VarMin = -10;   % Lower Bound of Decision Variables
problem.VarMax =  10;   % Upper Bound of Decision Variables


%% Parameters of PSO

% Constriction Coefficients
kappa = 1;
phi1 = 2.05;
phi2 = 2.05;
phi = phi1 + phi2;
chi = 2*kappa/abs(2-phi-sqrt(phi^2-4*phi));

params.MaxIt = 1000;        % Maximum Number of Iterations
params.nPop = 50;           % Population Size (Swarm Size)
%adding dynamici nature to both w (Inertia Coefficient) and wdamp (Damping
%Ratio of Inertia Coefficient)
params.wmax = chi;          % Maximum value of Intertia Coefficient
params.wmin = 0.1;           % Minimum value of Intertia Coefficient
params.c1 = chi*phi1;       % Personal Acceleration Coefficient
params.c2 = chi*phi2;       % Social Acceleration Coefficient
params.ShowIterInfo = true; % Flag for Showing Iteration Information

params.toleranceValue = 10^(-5); %tolerance value which can be accepted as solution with the
maximum error allowed

%% Effect of Constraint Variable : Wmin

%different number of 'wmin' values
wminVariations = [-10:2:30];

numberOfIterations = numel(wminVariations);
%run the experiment for numberOfIterations times
for i=1:numberOfIterations
    out(i) = pso_RosenBrockFunWithToleranceValue(problem, params);
    % plot the result into the existing graph
    hold on
    semilogy(out(i).BestCosts,"LineWidth",2,'DisplayName',num2str(wminVariations(i)));
    hold off
end
% Describing the attributes for the graph
title("Effect of Variable : Wmin")
xlabel('Iterations');
ylabel('Best Cost');
% make the y limit to a narrow region to get a closer look
ylim([0 10^-4]);
xlim([0 200])
%draw a line parallel to x axis to find on which iteration the output
%reaches to the tolerance value
yline(params.toleranceValue,'-',{'Acceptable','Limit: ' num2str(params.toleranceValue)});
grid on;
legend(num2str(wminVariations.'), 'location', 'northeast');
```