# Performance Analysis of Differential Evolution with Himmelblau Function

**Name:** EC Assessment Week 3 **Author:** Babu Pallam
**Date:** 07/03/2024 **Email Id.:** P2849288@my365.dmu.ac.uk

## Introduction

Testing optimization algorithms is a crucial part to increase the performance in terms of speed, resources, and accuracy. Choosing the best testing function is one of the major steps in that process. Himmelblau function is one well known function available today for testing under mathematical optimization. This function provides a lot of challenges to optimization problems. It is a multi-modal problem with one local maximum and four local minima. This report uses minimization of Himmelblau function with two variables as a benchmark problem for optimization of classic Differential Evolution (DE) Algorithm. The algorithm has been implemented using software tool named MATLAB. The code has been tested on different parameter values and modified techniques and different variations of differential algorithm. Then finally, a performance analysis and comparison of the modification done has been detailed, along with several remarks where further research is required.

## Benchmark A: Towards Theoretical Global Minimum

The problem chosen as a benchmark is Himmelblau Function (Himmelblau, D.M., 1972. Applied Nonlinear Programming, McGraw-Hill.) with two variables, which is defined as follows,

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

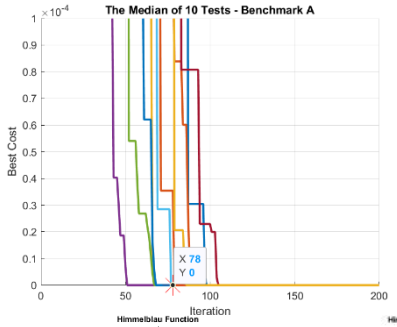
$$-5 \leq x \leq 5 \ \ and \ -5 \leq y \leq 5$$

Under minimization, four local minima are available for this problem, which are as listed below.

1. $f(3,2) = 0$
2. $f(-3.779310, -3.283186) = 0$
3. $f(-2.805118, 3.283186) = 0$
4. $f(3.584458, -1.848126)) = 0$

The DE Algorithm implemented for performance analysis was classic DE variant(Storn, R. & Price, K., 1997. Differential evolution, 11(4), pp.341–359.), i.e. “DE rand/1/bin” variant. Where rand denotes the base vector, 1 denotes the number of vector differences in production of mutate vector and bin refers to binomial distribution, which uses uniform crossover for crossover operation. The code for solving Himmelblau function has been implemented using DE mentioned above in MATLAB (see Appendix A A2) with the parameter values those has been set as follows, maximum iterations (MaxIt)=1000, population size (nPop)=50, bounds for scaling factor (beta) - beta_min =0.2 and beta_max =0.8, and crossover probability (pCR) = 0.2. During testing, a tolerance value has been set to $10^{-5}$ for the solution, i.e. this value can be considered as the minimum acceptable solution to Himmelblau problem during the evolution process with maximum possible error.

To find a theoretical optimum of the minimum number of iterations needed to reach this tolerance value given, the code has been tested several times. Figure 1 shows the results of ten tests (ten lines in the figure). The median value obtained is 78, which is marked with a red star. This median value has been referred as MedianBM1 in later sections. See Appendix A A3 for the code.




Further, how does the convergence happen has been analysed visually using MATLAB's contour feature. The Figure 2 provides snapshots of five outcomes during the generation process. This is the result of the code (see Appendix A4) that has been designed in such a way that it plots and then terminate the process during the generation process when the actual optimum value of Himmelblau function is reached, which is zero. The points indicate in the plot are the pairs, since it's a two-variable problem, of which the solution can be formed. At generation 1, the points were widely populated across the population space, but at generation 30, the points were clustering into four. The reason for this phenomenon is analysed. This could be because of Himmelblau function's four local optimum minima. As generation goes the convergence has been witnessed. At 262nd iteration, the algorithm found the optimum solution when, in (x, y) pair, x=3 and y=2, which indicates that the population united into a single solution. For code see in Appendix A A4.

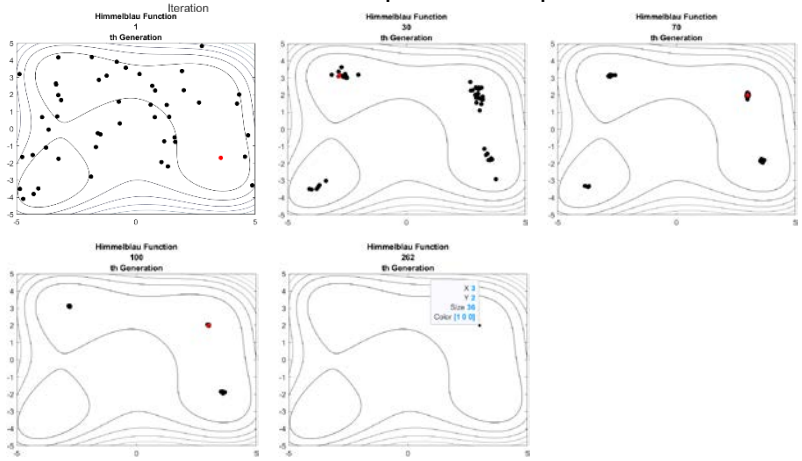



***Figure 2***

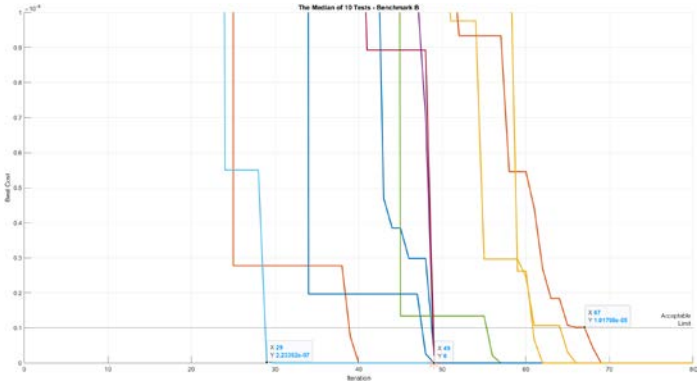
## Benchmark B: Effect of Parameters

As part of optimization, the code that implements the classic DE, DE rand/1/bin, with Himmelblau function, has been tested further in search of a specific combination of parameters which may influence the algorithm's efficiency. To do that, the parameters such as, MaxIt, nPop, beta, and pCR, have been taken into consideration, and then examined the effects of those in the result, which is the minimum iterations needed to reach the acceptance value (tolerance value) of the solution. The outcomes of those experiments were impressive. The main observations from the study have been noted below.

- It is found that, with the existence of tolerance value, here it is $10^{-5}$ , the values of MaxIt does not have a significant effect after a certain value. Figure 10 (see Appendix B B1) stands out associated result. All the outcomes were situated around the MedianBM1. As can be clearly seen in Figure 2, the union of solutions has been taken place within 100 generations, and rest 143 generations are to reach the optimum solution for all the selected solution samples.

- The value of nPop has a phenomenal effect in the outcome. As shown in Figure 11(see Appendix B B2), for smaller values of nPop, the result was larger than the MedianBM1. The value of nPop at least need to be 4, for which the result had a steep rise to more than 1000. When nPop=10, the result found to be 123, which is higher than MedianBM1. A steady decline of the result has been observed only when nPop higher than 1000. For nPop =2000, the result found to be 21, which is the lower value found in the result. What is striking in this study is that for values of nPop, 5000 and 10000, the result found to be more than 21. From this it can be concluded like this. There is always a limit for nPop value. After that, for higher nPop size, the result would also go higher.
- pCR values also influence the result dramatically. Figure 12 (see Appendix B B3) provides details of outcomes (as lines) of the tests where pCR varies from 0 to 1. For pCR = 0, the result was 292, which is higher than the MedianBM1. As pCR increases, a striking improvement can be seen. For pCR =1, It takes only 36 iterations to achieve the desired result, which is a good improvement as compared to MedianBM1.
- The beta bound's effects are also worthful to be noticed. The tests have been performed on different values of beta. Figure 13.1(see Appendix B B4) shows what was resulted for beat_min and beta_max ranging from 0 to 1. An improved result noticed when beta_min=0.2 and beta_max=1.2, which was 26. This value is the best value found in these tests. For beta_min = 0 with other values of beta_max, the result was more than 1000, which is inequitable while comparing with MedianBM1. In general, for higher values of beta bounds, the result was higher 100, which is also not optimum. In addition to that, a few tests have been done for more wider bounds. Figure 13.2 (see Appendix B B4) shows graph of the resulted outcomes. When the boundary gets wider, the iterations need to reach the desired output also goes high, it shows an inverse effect.

Based on the test results, every parameter has an importance in refining the results; further, it is found that one parameter has influence on one another. For instance, if pCR is low, then MaxIt must be higher to get a desired output. The median of best results, after parameter tuning, has been given in Figure 3 as a graph. As given, the results after parameter tuning, includes MaxIt = 80, nPop= 100, beta_min= 0.2, beta_max = 0.8 and pCR = 1. The median determined as 42. This result is impressive, which is almost half of the MedianBM1.



```
12  %% DE Parameters
13
14  params.MaxIt = 80;        %
15  params.nPop = 100;
16  params.beta_min = 0.2;
17  params.beta_max = 1.2;
18  params.pCR = 1;           %
```

*Figure 3*

# Benchmark C: Effect of Variants

The benchmark B has been pointing to optimize the performance of the algorithm by parameter tuning. In this section, effect of variants has been discussed. To do that, two variants of DE has been considered: 1. DE/best/1/bin variant and 2. DE/best/2/bin variant. In first variant, best of the current population has been taken as base vector for the generation of mutate vector. Second variant is a modified version of first variant, in such a way that the two vector differences are considered instead of one used in classic DE ( DE rand/1/n).

## DE/best/1/bin variant

The code implemented with this variant has been tested several times. The median of the result has been calculated, which is found to be 17. This median is referred to as **MedianBM3V1**. The contour plots have been plotted to observe the convergence. Figure 5 portraits it. At 83rd iteration the algorithm reached to its optimum value when x = 3.584458 and y = -1.848126, which is another solution for optimal minimum of the Himmelblau function.

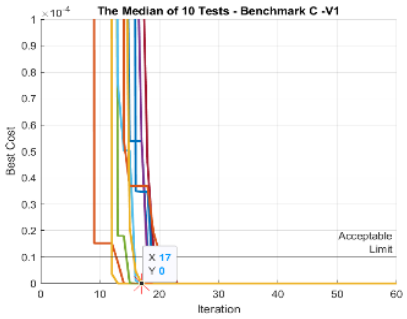



***Figure 4***

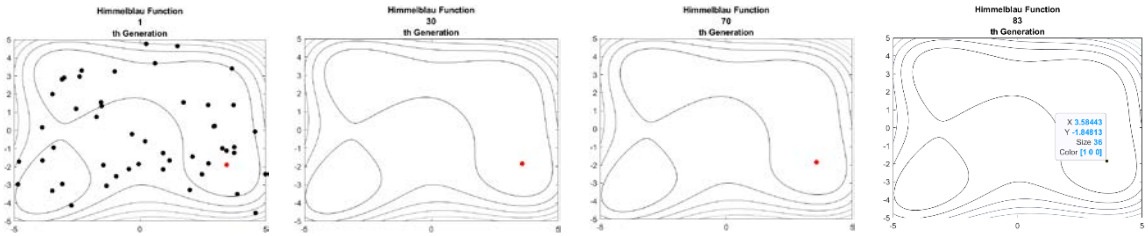



***Figure 5***

To see the effect of parameters in this variant, couple of tests have been carried out. The change in parameters has been recorded in graphs (see Appendix C C1). The effect of parameters follows the similar pattern as which has been observed in Benchmark B. From the Figures 14, 15, 16 and 17, (see Appendix C C1B, C1C, C1D and C1E) the best result found for the parameters are as given below.

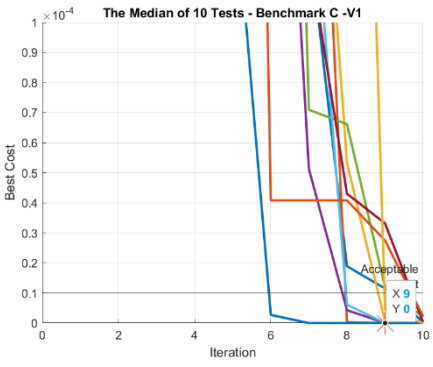



***Figure 6***

1. From Figure 15, 8 has been resulted for nPop=5000
2. From Figure 16, 8 has been resulted for pCR =1
3. From Figure 17, 9 has been resulted for beta_min =0.2 and beta_max=1.2

Based on the tuned values of parameters and appropriate value of MaxIt, ten tests have been performed and median has been calculated. As shown in figure 9, median value found as 9.

## DE best/2/bin variant

The median value of the result has been calculated after the implementation of the DE best/2/bin variant. The median value is 35, as shown in Figure 7. This value is referred as **MedianBM3V2.** As given in Figure 8, the evolution process has been observed using contour plots, which resulted the optimum value in 119$^{th}$ generation.

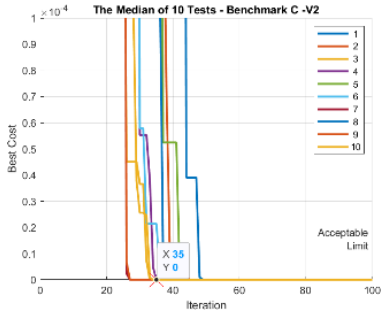



***Figure 7***

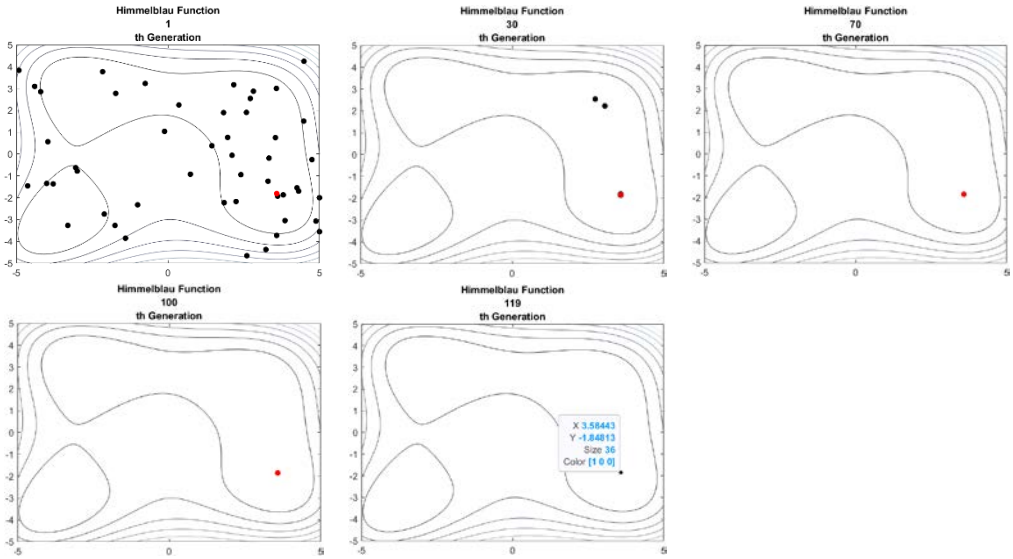



***Figure 8***

To find the parameter's effect, the tests have been performed in the same way how that has been done with other two variants (see Appendix C2 C2A) . Based on the results given in Figure 18, 19, 20 and 21 (see Appendix C C2B, C2C, C2D and C2E), the best values of each parameter for lower result has been listed below.

- From Figure 19, 23 has been resulted, when nPop=10000
- From Figure 20, 18 has been resulted, when pCR = 1
- From Figure 21, 9 has been resulted, for beta_min=02, and beta_max =1.2

Based on the tuned values of parameters and appropriate value of MaxIt, ten tests have been performed and median has been calculated. As shown in figure 9, the value found as 29.

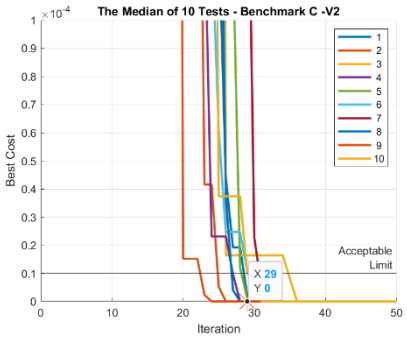



***Figure 9***

## Comparison: Benchmark A, B, and C

The code has been tested with different parameters with different variants of DE on same Himmelblau function. The observations drawn out of an extensive analysis and comparison of results are presented below as a table.

| | **Contour Plot Convergence to Optimum Value (zero)** | **Before Tuning** | | **After Tuning** | |
|---|---|---|---|---|---|
| | | **Parameter Specification** | **Median** (iterations reach the tolerance value set) | **Parameter Specification** | **Median** (iterations reach the tolerance value set) |
| **DE rand/1/bin** | 262 | MaxIt = 1000<br>nPop = 50<br>beta_min = 0.2<br>beta_max = 0.8<br>pCR = 0.2 | 78 | MaxIt = 80<br>nPop = 100<br>beta_min = 0.2<br>beta_max = 1.2<br>pCR = 1 | 42 |
| **DE best/1/bin** | 83 | MaxIt = 1000<br>nPop = 50<br>beta_min = 0.2<br>beta_max = 0.8<br>pCR = 0.2 | 17 | MaxIt = 50<br>nPop = 100<br>beta_min = 0.2<br>beta_max = 1.2<br>pCR = 1 | 9 |
| **DE best/2/bin** | 119 | MaxIt = 1000<br>nPop = 50<br>beta_min = 0.2<br>beta_max = 0.8<br>pCR = 0.2 | 35 | MaxIt = 50<br>nPop = 5000<br>beta_min = 0.2<br>beta_max = 1.2<br>pCR = 1 | 29 |

- It is striking while compare the contour graphs plotted for each variant of DE. Following observations were found to be interesting.
    - From the above table, the convergence in the evolution process had a great improvement while changing the variant. With DE rand/1/bin, it took 262 generations to attain the optimum result, but with DE best/1/bin, at 119$^{th}$ generation itself attained the optimum result.
    - The progress of convergence into a single cluster with the first variant (associated with Figure 2) found to be a linear function with a negative slope because, algorithm to reach the optimum value makes a gradual progress. Even at 100$^{th}$ generation, the solution space has been clustered into three regions. The union of those into single optimum solution could be seen only after that.

      In contrast to that, for the other two variants convergence process to a single cluster follows exponential function with a decreasing slop. For instance, in Figure 5, by 30$^{th}$ generation itself made all the solution into a single cluster, then rest of the generations are to find the optimum solution from the cluster.
- While comparing the median values, there is a significant change observed after parameter tuning in all three variants of DE. For instance, with DE rand/1/bin, 78 was the median after parameter tuning, but with optimization it has been reduced to 42, which could be considered as a great improvement made in the efficiency of algorithm.
- With every variant, when a high nPop value, the result found to be lower than the median value, see it in Figures 11, 15 and 19. But considering nPop at higher values greater than 2000 found to be inefficient in terms of time consumption. For the first two variants nPop has been considered as 100, but

for the third variant, DE best/2/bin, nPop value set to the value which is found optimum while checking effect of parameters., i.e. 5000. Even with high nPop, there is not much difference observed, rather a marginal decrease in the output result.

# Conclusion

DE algorithm solving Himmelblau function has been implemented successfully. As benchmark A, median of the minimum number of iterations required to reach the tolerance value set has been calculated. With the help of contour feature of MATLAB, the visualization of contour plot has been made. In benchmark B, the effect of parameters has been analysed, and noticed the observations. Further, in benchmark C, two variants of DE have been implemented and the parameter tuning has been done to see the improvement. Then at the end, a detailed comparison of the result found in benchmark A, B and C has been detailed along with remarks.

# References


- Storn, R. & Price, K., 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4), pp.341–359.
- Himmelblau, D.M., 1972. *Applied Nonlinear Programming*, McGraw-Hill, ISBN: 0070289212

# APPENDIX A

Appendix A provides the graphs and codes related to Benchmark A.

## A1. Himmelblau Function

```
% Himmelblau function
 function z = HimmelblauFunction(x,y)

    z = (x.^2 + y - 11).^2 + (x + y.^2 - 7).^2;
end
```

## A2. Differential Algorithm Implementation

```
%% DE /rand/1/bin
function out = DE(problem, params)
CostFunction = problem.CostFunction; % Cost Function
VarSize = [1 problem.nVar];     % Decision Variables Matrix Size
VarMin = problem.VarMin;          % Lower Bound of Decision Variables
VarMax = problem.VarMax;            % Upper Bound of Decision Variables
toleranceValue = problem.toleranceValue; % tolerance value at which the solution is acceptable with
the maximum error possible

%% DE Parameters

MaxIt = params.MaxIt;       % Maximum Number of Iterations
nPop = params.nPop;          % Population Size
beta_min = params.beta_min;     % Lower Bound of Scaling Factor (0)
beta_max = params.beta_max;     % Upper Bound of Scaling Factor (2)
pCR = params.pCR;          % Crossover Probability


%% Initialization

empty_individual.Position = [];
empty_individual.Cost = [];

BestSol.Cost = inf;

pop = repmat(empty_individual, nPop, 1);
%sample solution space
for i = 1:nPop
   %sample creation
   pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
   %sample solution creation
   pop(i).Cost = CostFunction(pop(i).Position(1),pop(i).Position(2));

   if pop(i).Cost<BestSol.Cost
      BestSol = pop(i);
   end

end
%inilialize the BestCost of all generations into zeros
BestCost = zeros(MaxIt, 1);
%% DE Main Loop

for it = 1:MaxIt
```

```
for i = 1:nPop
    %ith individual
    x = pop(i).Position;
    %create a random arrangement (reorder the population numbers)
    A = randperm(nPop);
    %Note: we can do this with rand, but we want all numbers different

    %remove the i index from this since we don't want that to
    %be target vector
    A(A == i) = [];

    a = A(1);   % Target Vector index
    b = A(2);   % Random Vector 1 index
    c = A(3);   % Random Vector 2 index

    % Mutant Vector
    beta = unifrnd(beta_min, beta_max, VarSize);

    %y is the mutant vector
    y = pop(a).Position + beta.*(pop(b).Position - pop(c).Position);
    y = max(y, VarMin);
    y = min(y, VarMax);

    % Trial Vector (Crossover between Target Vector x and Mutant Vector y)
    z = zeros(size(x));
    j0 = randi([1 numel(x)]);
    for j = 1:numel(x)
        if j == j0 || rand <= pCR
            z(j) = y(j);%from the mutant vector
        else
            z(j) = x(j);%from the target vector
        end
    end

    NewSol.Position = z;
    NewSol.Cost = CostFunction(NewSol.Position(1),NewSol.Position(2));

    % Selection
    if NewSol.Cost<pop(i).Cost
        pop(i) = NewSol;
        if pop(i).Cost<BestSol.Cost
            BestSol = pop(i);
        end
    end

end

% Update Best Cost
BestCost(it) = BestSol.Cost;

% Show Iteration Information
disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);

% Comparing the best cost with the tolerance value
% This code has been added for improving efficiency of the code
if BestCost(it) <= toleranceValue
    disp(['Tolerance value has been reached at generation: ' num2str(it)]);
    out.BestCost = BestCost;
    out.BestSol =BestSol;
    out.minIterationToReachToleranceValue = it;
```

```
        return;
    end
end
out.BestCost = BestCost;
out.BestSol =BestSol;
%assing number of iterations to reach minimum tolerance value as MaxIt since the generations of
MaxIt has
%not reached to the required result, this is done as part of plotting the
%graph without any error
out.minIterationToReachToleranceValue =MaxIt;
end
```

## A3. Median Calculation

Part of the modified code to find the median of the result that is the minimum number of iterations required to reach the tolerance value which has been set to ensure the convergence at the end of evolutionary process is given below.

```
%% DE /rand/1/bin
%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters

params.MaxIt = 1000;       % Maximum Number of Iterations
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE
% number of iterations required
noOfTests = 10;
% array to store the result of each test, initially all assign to infinity
minIterationsFromEachTest = inf(1,noOfTests);
%test the experiment for noOfTests times
for i=1:noOfTests
    out = DE(problem, params);
    minIterationsFromEachTest(i)=out.minIterationToReachToleranceValue;
    %plot the graph for this test result
    hold on
    plot(out.BestCost,"LineWidth",2)
    hold off
end

%% Show Results
%Calculate Median value
medianOfAllSolutions = median(minIterationsFromEachTest);
```

## A4. Contour Plots

The contour plots provide a better visualization about the convergence. The code which implements the contour plots have been given below. Note that the tolerance value

has been set to zero, the optimum solution. The process has been set to go till tolerance value with a bound of 1000 iterations maximum.

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 0; % tolerance value at which the solution is acceptable with the maximum error possible

%% DE Parameters

params.MaxIt = 1000;       % Maximum Number of Iterations
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%Generation numbers needed to be plot as a graph
params.genNumber = [1,30,70,100];

%% Calling DE
out = DE_ForContour(problem, params);
```

---

*DeForContour* Function has been defined as:

```
%% DE /rand/1/bin
function out = DE_ForContour(problem, params)
CostFunction = problem.CostFunction; % Cost Function
VarSize = [1 problem.nVar];     % Decision Variables Matrix Size
VarMin = problem.VarMin;          % Lower Bound of Decision Variables
VarMax = problem.VarMax;          % Upper Bound of Decision Variables
toleranceValue = problem.toleranceValue; % tolerance value at which the solution is acceptable with the maximum error possible

%% DE Parameters

MaxIt = params.MaxIt;       % Maximum Number of Iterations
nPop = params.nPop;         % Population Size
beta_min = params.beta_min;     % Lower Bound of Scaling Factor (0)
beta_max = params.beta_max;     % Upper Bound of Scaling Factor (2)
pCR = params.pCR;          % Crossover Probability
genNumber =params.genNumber;%generation index of which graph must be ploted

%% Initialization

empty_individual.Position = [];
empty_individual.Cost = [];

BestSol.Cost = inf;

pop = repmat(empty_individual, nPop, 1);
%sample solution space
for i = 1:nPop
```

```
    %sample creation
    pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
    %sample solution creation
    pop(i).Cost = CostFunction(pop(i).Position(1),pop(i).Position(2));

    if pop(i).Cost<BestSol.Cost
        BestSol = pop(i);
    end

end
%inilialize the BestCost of all generations into zeros
BestCost = zeros(MaxIt, 1);

%plot Control Variables
plotX = zeros(MaxIt,nPop);
plotY = zeros(MaxIt,nPop);
%% DE Main Loop

for it = 1:MaxIt

    for i = 1:nPop
        %ith individual
        x = pop(i).Position;
        %create a random arrangement (reorder the population numbers)
        A = randperm(nPop);
        %Note: we can do this with rand, but we want all numbers different

        %remove the i index from this since we don't want that to
        %be target vector
        A(A == i) = [];

        a = A(1);   % Target Vector index
        b = A(2);   % Random Vector 1 index
        c = A(3);   % Random Vector 2 index

        % Mutant Vector (Mutation)
        beta = unifrnd(beta_min, beta_max, VarSize);

        %y is the mutant vector
        y = pop(a).Position + beta.*(pop(b).Position - pop(c).Position);
        y = max(y, VarMin);
        y = min(y, VarMax);

        % Trial Vector (Crossover between Target Vector x and Mutant Vector y)
        z = zeros(size(x));
        j0 = randi([1 numel(x)]);
        for j = 1:numel(x)
            if j == j0 || rand <= pCR
                z(j) = y(j);%from the mutant vector
            else
                z(j) = x(j);%from the target vector
            end
        end

        NewSol.Position = z;
        NewSol.Cost = CostFunction(NewSol.Position(1),NewSol.Position(2));

        % Selection
        if NewSol.Cost<pop(i).Cost
            pop(i) = NewSol;
```

```
            if pop(i).Cost<BestSol.Cost
                BestSol = pop(i);
            end
        end
        %save the best solution found in the iteration it of particle i
        %into plotX and plotY to draw the graph
         plotX(it,i) = pop(i).Position(1);
         plotY(it,i) = pop(i).Position(2);
    end

    % Update Best Cost
    BestCost(it) = BestSol.Cost;

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);


    %plot the graph with the newly updated solutions and best received in the above
    %generation process( the drawing code has been seperated from this into
    %another file named Draw.m)
    %check with the toleranc value to make sure that the graph has been
    %ploted for the optimum solution (0) here tolerance value =0 has been
    %set
     if BestCost(it) <= toleranceValue
        disp(['Tolerance value has been reached at generation: ' num2str(it)]);
        genNumber =[it];
        DrawTheContourPlot
        %Once optimum has been reached no need to run the code again so
        %we terminate the exicution at here
        out.BestCost = BestCost;
        out.BestSol =BestSol;
        return;
     else
         DrawTheContourPlot
     end

end

out.BestCost = BestCost;
out.BestSol =BestSol;
end
```

---

The definitions of *DrawTheContourPlot* is given below.

```
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
        colormap(bone)
        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),"filled","k" );
        title(['Himmelblau Function' num2str(it) "th Generation"]);
        hold on
        scatter(BestSol.Position(1), BestSol.Position(2),"filled","o","r");
        hold off
    end
end
```

---

# APPENDIX B

Appendix B includes graphs and codes related to Benchmark B. Each section contains the graph and MATLAB code of several tests performed. Note that the graphs follow global identifiers as figure number for easy referencing.

## B1. Effect of Parameter: MaxIt

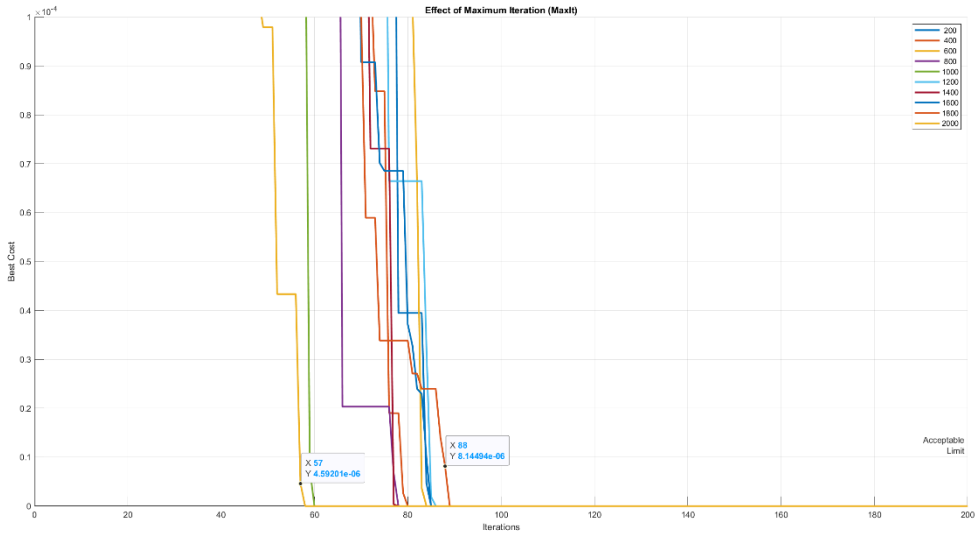



***Figure 10***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for varying MaxIt To see the Efffect
%different number of 'MaxIt' values
maxItenVariation = [200:200:2000];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(maxItenVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
```

```
    %Assign value to MaxIt
    params.MaxIt = maxItenVariation(i);
    out= DE(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"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 closer view
ylim([0 10^-4])
xlim([0 200]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(maxItenVariation.'), 'location', 'northeast');
```

# B2. Effect of Parameter: nPop

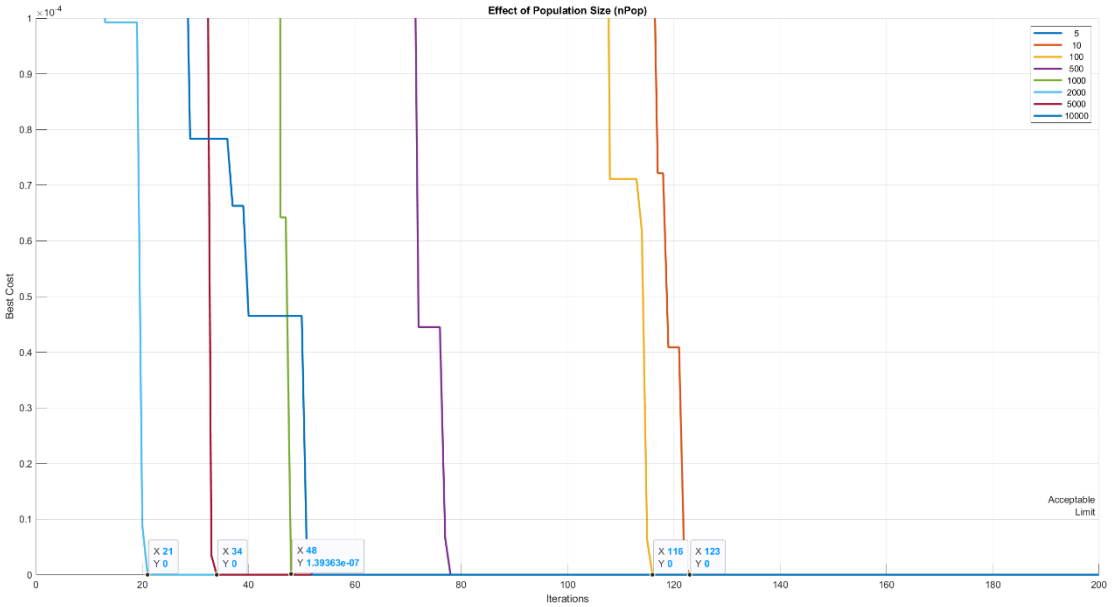



***Figure 11***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
```

```
problem.VarMax = 5;              % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;       % Maximum Number of Iterations
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for different nPop values to see the effect of it in the Result
%different number of 'nPop' values
nPopVariation = [5,10,100,500,1000,2000,5000,10000];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(nPopVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %Assign value to nPop
    params.nPop = nPopVariation(i);
    out= DE(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"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 closer view
ylim([0 10^-4])
xlim([0 200]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(nPopVariation.'), 'location', 'northeast');
```

# B3. Effect of Parameter: pCR

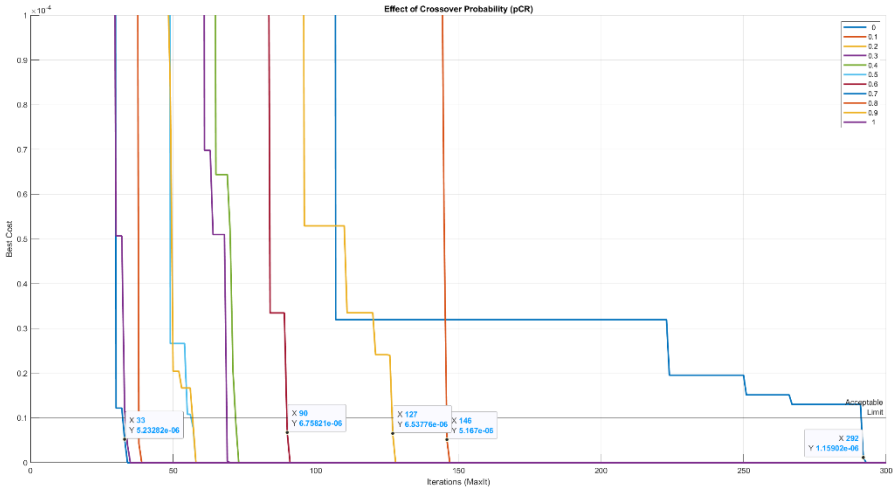



***Figure 12***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
params.beta_min = 0.2;    % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;    % Upper Bound of Scaling Factor (2)
%% Calling DE for different pCR values to see the effect of it in the Result
%different number of 'pCR' values
pCRVariation = [0:0.1:1];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(pCRVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %assign value to pCR
    params.pCR = pCRVariation(i);
    out= DE(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"LineWidth",2);
    hold off
end
```

```
% Describing the attributes for the graph
title("Effect of Crossover Probability (pCR)")
xlabel('Iterations (MaxIt)');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a closer view
ylim([0 10^-4])
xlim([0 300]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(pCRVariation.'), 'location', 'northeast');
```

# B4. Effect of Parameter: beta (beta_min & beta_max)

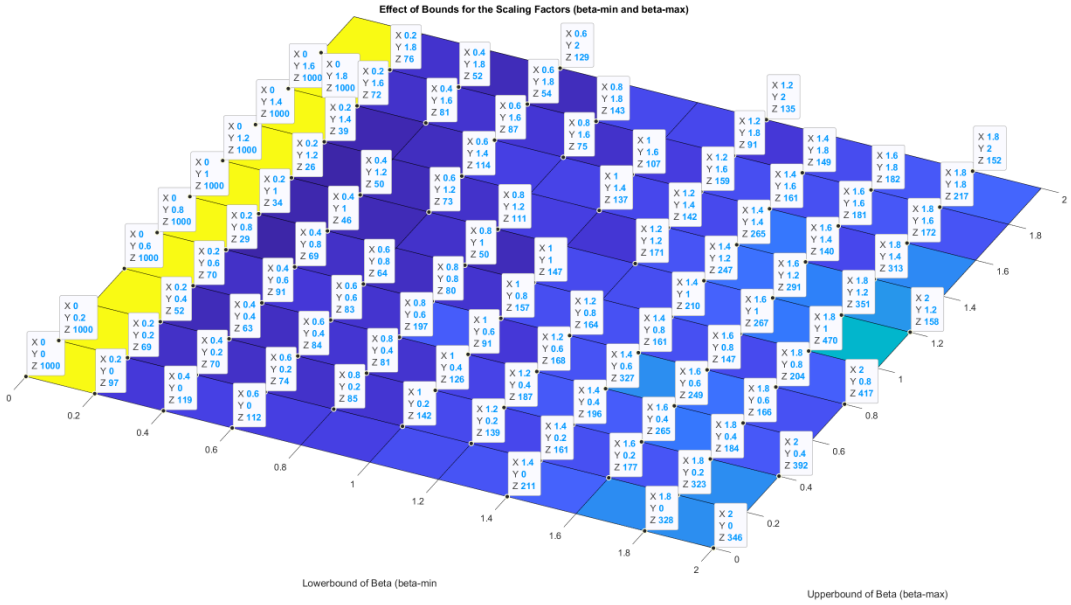



***Figure 13 1***

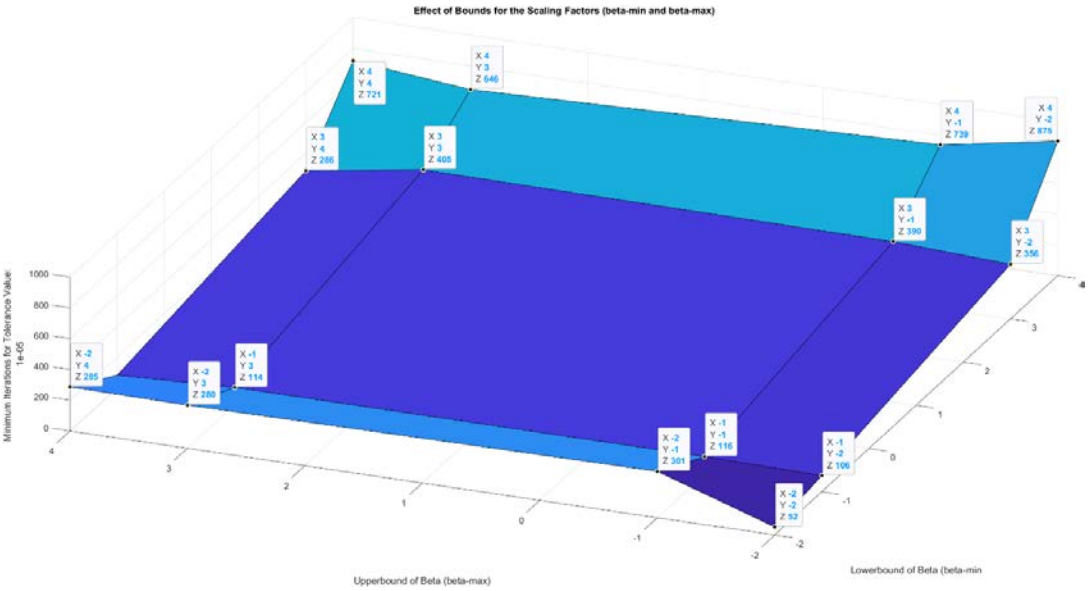



***Figure 13 2***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters

params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
params.pCR = 0.2;         % Crossover Probability (in between 0 and 1)
%% Effect of beta_min and beta_max in the result

%save different values of beta_min and beta_max
boundVariations = [0:0.2:2];% for figure A
% boundVariations = [-2,-1,3,4];% for figure-B

% size of variation arrays for the loop generation
sizeOfbetaMinVariation = numel(boundVariations);
sizeOfbetaMaxVariation = numel(boundVariations);
```

```
% use two for loops to do the experiment with various combinations
for i=1:sizeOfbetaMaxVariation
    % Lowerbound for scaling factor-beta (bata_min)
    params.beta_min = boundVariations(i);
    for j=1:sizeOfbetaMaxVariation
        % Upperbound for scaling factor-beta (bata_max)
        params.beta_max = boundVariations(i);
        out(i) = DE(problem, params);
        Z(i,j)= out(i).minIterationToReachToleranceValue;
    end
end

% Describing the attributes for the graph
[X,Y] = meshgrid(boundVariations,boundVariations);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of Bounds for the Scaling Factors (beta-min and beta-max)')
xlabel('Lowerbound of Beta (beta-min');
ylabel('Upperbound of Beta (beta-max)');
zlabel(["Minimum Iterations for Tolerance Value: " num2str(problem.toleranceValue)]);
```

# APPENDIX C

## C1. DE best/1/bin Variant

### C1A: MATLAB Code: Implementation of Algorithm

The algorithm of DE variant best/1/bin has been provided below as a function named "DE_V1()".

**MATLAB Code:**

```
%% DE /best/1/bin
function out = DE(problem, params)
CostFunction = problem.CostFunction; % Cost Function
VarSize = [1 problem.nVar];     % Decision Variables Matrix Size
VarMin = problem.VarMin;          % Lower Bound of Decision Variables
VarMax = problem.VarMax;          % Upper Bound of Decision Variables
toleranceValue = problem.toleranceValue; % tolerance value at which the solution is acceptable with
the maximum error possible

%% DE Parameters

MaxIt = params.MaxIt;       % Maximum Number of Iterations
nPop = params.nPop;         % Population Size
beta_min = params.beta_min;     % Lower Bound of Scaling Factor (0)
beta_max = params.beta_max;     % Upper Bound of Scaling Factor (2)
pCR = params.pCR;         % Crossover Probability


%% Initialization

empty_individual.Position = [];
empty_individual.Cost = [];

BestSol.Cost = inf;

pop = repmat(empty_individual, nPop, 1);
%sample solution space
for i = 1:nPop
   %sample creation
   pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
   %sample solution creation
   pop(i).Cost = CostFunction(pop(i).Position(1),pop(i).Position(2));

   if pop(i).Cost<BestSol.Cost
      BestSol = pop(i);
   end

end
%inilialize the BestCost of all generations into zeros
BestCost = zeros(MaxIt, 1);

%% DE Main Loop

for it = 1:MaxIt
   %assign the best solution as best solution in the current generation
   %which is used as base vector for mutate vector formation
   bestSolutionInPreviousGen = BestSol;
   %process
```

```
for i = 1:nPop
    %ith individual
    x = pop(i).Position;
    %create a random arrangement (reorder the population numbers)
    A = randperm(nPop);
    %Note: we can do this with rand, but we want all numbers different

    %remove the i index from this since we dont want that to
    %be target vector
    A(A == i) = [];

    a = A(1);   % Random Vector 1 index
    b = A(2);   % Random Vector 2 index

    % Mutant Vector (Mutation)
    beta = unifrnd(beta_min, beta_max, VarSize);

    %y is the mutant vector - best is used as base vector
    y = bestSolutionInPreviousGen.Position + beta.*(pop(a).Position - pop(b).Position);
    y = max(y, VarMin);
    y = min(y, VarMax);

    % Trial Vector (Crossover between Target Vector x and Mutant Vector y)
    z = zeros(size(x));
    j0 = randi([1 numel(x)]);
    for j = 1:numel(x)
        if j == j0 || rand <= pCR
            z(j) = y(j);%from the mutant vector
        else
            z(j) = x(j);%from the target vector
        end
    end

    NewSol.Position = z;
    NewSol.Cost = CostFunction(NewSol.Position(1),NewSol.Position(2));

    % Selection
    if NewSol.Cost<pop(i).Cost
        pop(i) = NewSol;
        if pop(i).Cost<BestSol.Cost
            BestSol = pop(i);
        end
    end

end

% Update Best Cost
BestCost(it) = BestSol.Cost;

% Show Iteration Information
disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);

if BestCost(it) <= toleranceValue
    disp(['Tolerance value has been reached at generation: ' num2str(it)]);
    out.BestCost = BestCost;
    out.BestSol =BestSol;
    out.minIterationToReachToleranceValue = it;
    return;
end
```

```
end
out.BestCost = BestCost;
out.BestSol =BestSol;
%assing number of iterations to reach minimum tolerance value as MaxIt since the generations of
MaxIt has
%not reached to the required result, this is done as part of plotting the
%graph without any error
out.minIterationToReachToleranceValue =MaxIt;
end
```

## C1B. Effect of Parameter: MaxIt

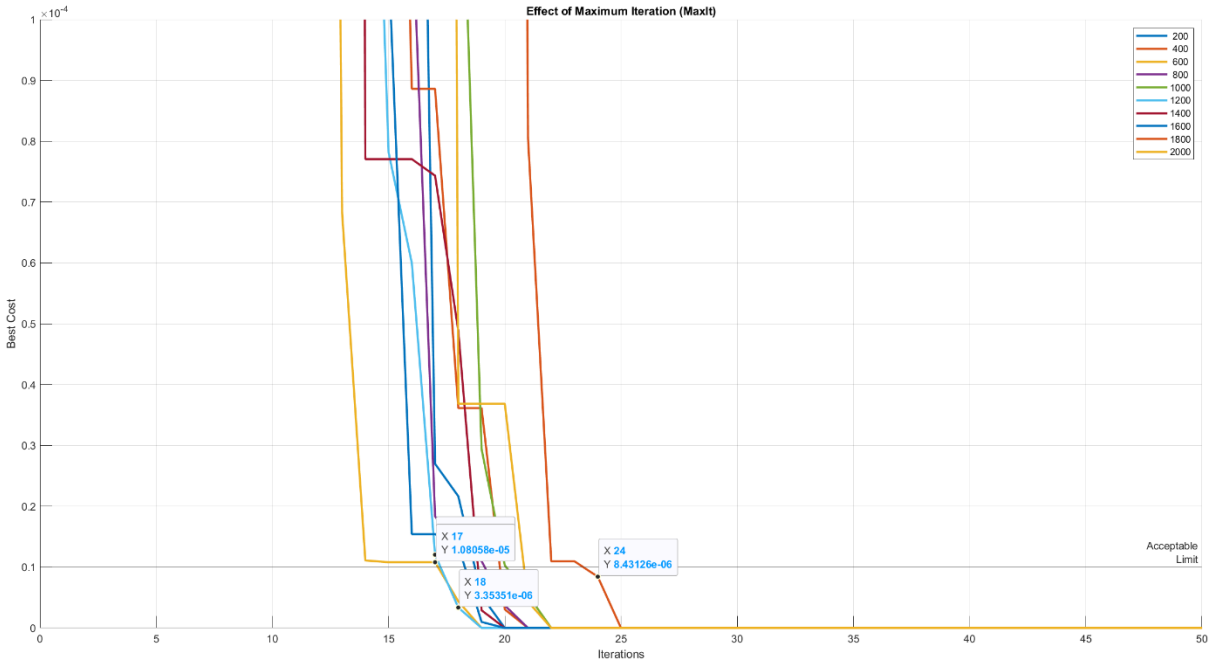



***Figure 14***

**MATLAB Code:**

```
%% DE /best/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for varying MaxIt To see the Efffect
%different number of 'MaxIt' values
```

```
maxItenVariation = [200:200:2000];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(maxItenVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %Assign value to MaxIt
    params.MaxIt = maxItenVariation(i);
    out= DE_V1(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"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 closer view
ylim([0 10^-4])
xlim([0 50]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(maxItenVariation.'), 'location', 'northeast');
```

## C1C. Effect of Parameter: nPop

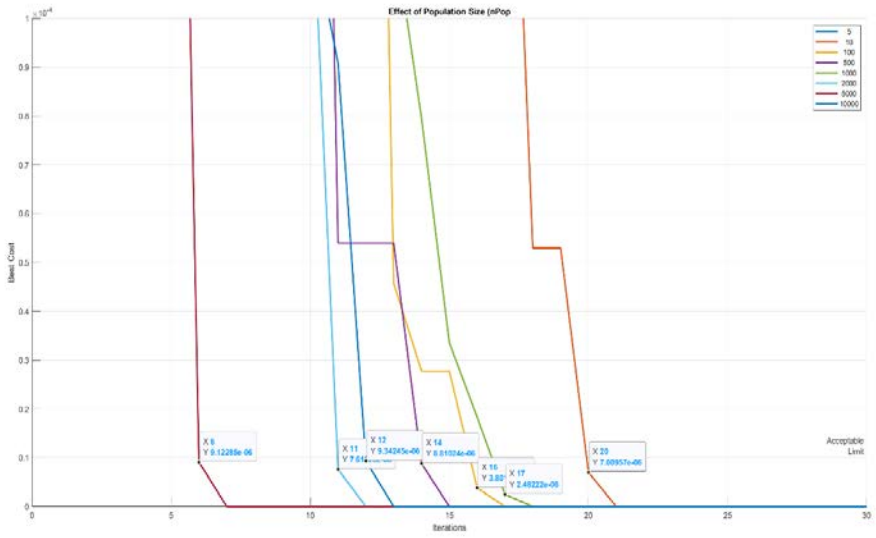



***Figure 15***

**MATLAB Code:**

```
%% DE /best/1/bin
clc; clear;
```

```
%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;       % Maximum Number of Iterations
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for different nPop values to see the effect of it in the Result
%different number of 'nPop' values
nPopVariation = [6,10,100,500,1000,2000,5000,10000];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(nPopVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %Assign value to nPop
    params.nPop = nPopVariation(i);
    out= DE_V1(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"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 closer view
ylim([0 10^-4])
xlim([0 50]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(nPopVariation.'), 'location', 'northeast');
```

## C1D. Effect of Parameter: pCR

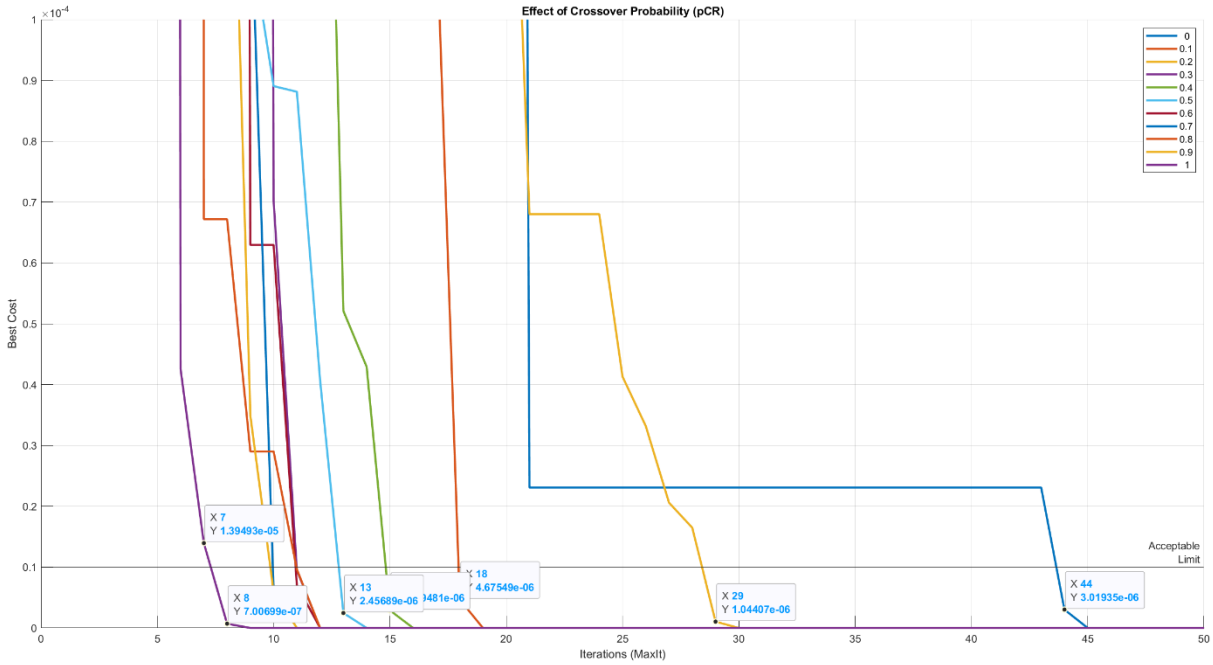



***Figure 16***

**MATLAB Code:**

```
%% DE /best/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
params.beta_min = 0.2;    % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;    % Upper Bound of Scaling Factor (2)
%% Calling DE for different pCR values to see the effect of it in the Result
%different number of 'pCR' values
pCRVariation = [0:0.1:1];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(pCRVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %assign value to pCR
    params.pCR = pCRVariation(i);
    out= DE_V1(problem, params);
```

```
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"LineWidth",2);
    hold off
end

% Describing the attributes for the graph
title("Effect of Crossover Probability (pCR)")
xlabel('Iterations (MaxIt)');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a closer view
ylim([0 10^-4])
xlim([0 50]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(pCRVariation.'), 'location', 'northeast');
```

## C1E. Effect of Parameter: beta (beta_min & beta_max)

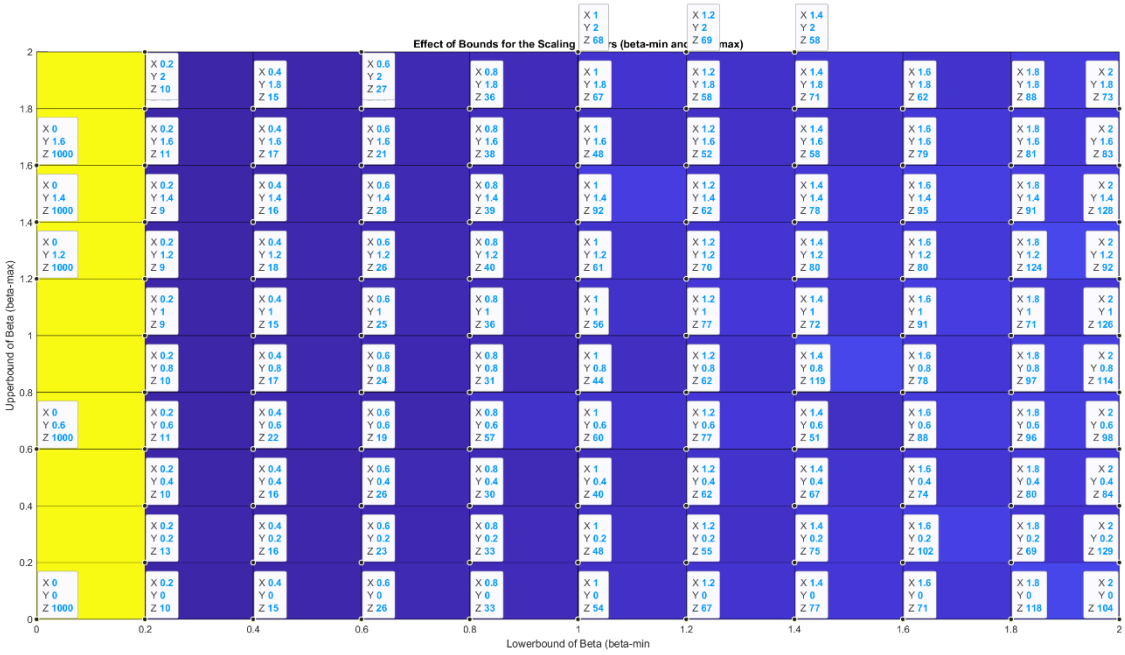



***Figure 17***

**MATLAB Code:**

```
%% DE /best/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
```

```
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters

params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
params.pCR = 0.2;         % Crossover Probability (in between 0 and 1)
%% Effect of beta_min and beta_max in the result

%save different values of beta_min and beta_max
boundVariations = [0:0.2:2];% for figure A

% size of variation arrays for the loop generation
sizeOfbetaMinVariation = numel(boundVariations);
sizeOfbetaMaxVariation = numel(boundVariations);

% use two for loops to do the experiment with various combinations
for i=1:sizeOfbetaMaxVariation
   % Lowerbound for scaling factor-beta (bata_min)
   params.beta_min = boundVariations(i);
   for j=1:sizeOfbetaMaxVariation
      % Upperbound for scaling factor-beta (bata_max)
      params.beta_max = boundVariations(i);
      out(i) = DE_V1(problem, params);
      Z(i,j)= out(i).minIterationToReachToleranceValue;
   end
end

% Describing the attributes for the graph
[X,Y] = meshgrid(boundVariations,boundVariations);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of Bounds for the Scaling Factors (beta-min and beta-max)')
xlabel('Lowerbound of Beta (beta-min');
ylabel('Upperbound of Beta (beta-max)');
zlabel(["Minimum Iterations for Tolerance Value: " num2str(problem.toleranceValue)]);
```

## C1F. MATLAB Code: Plotting the Contour

```
%% DE /best/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 0; % tolerance value at which the solution is acceptable with the maximum
error possible

%% DE Parameters

params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
```

```
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%Generation numbers needed to be plot as a graph
params.genNumber = [1,30,70,100];

%% Calling DE
out = DE_V1_ForContour(problem, params);
```

---

function DE_V1_ForContour() is defined as:

```
%% DE /best/1/bin
function out = DE_V1_ForContour(problem, params)
CostFunction = problem.CostFunction; % Cost Function
VarSize = [1 problem.nVar];     % Decision Variables Matrix Size
VarMin = problem.VarMin;            % Lower Bound of Decision Variables
VarMax = problem.VarMax;            % Upper Bound of Decision Variables
toleranceValue = problem.toleranceValue; % tolerance value at which the solution is acceptable with
the maximum error possible

%% DE Parameters

MaxIt = params.MaxIt;       % Maximum Number of Iterations
nPop = params.nPop;         % Population Size
beta_min = params.beta_min;     % Lower Bound of Scaling Factor (0)
beta_max = params.beta_max;     % Upper Bound of Scaling Factor (2)
pCR = params.pCR;          % Crossover Probability
genNumber =params.genNumber;%generation index of which graph must be ploted

%% Initialization

empty_individual.Position = [];
empty_individual.Cost = [];

BestSol.Cost = inf;

pop = repmat(empty_individual, nPop, 1);
%sample solution space
for i = 1:nPop
   %sample creation
   pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
   %sample solution creation
   pop(i).Cost = CostFunction(pop(i).Position(1),pop(i).Position(2));

   if pop(i).Cost<BestSol.Cost
      BestSol = pop(i);
   end

end
%inilialize the BestCost of all generations into zeros
BestCost = zeros(MaxIt, 1);

%plot Control Variables
plotX = zeros(MaxIt,nPop);
plotY = zeros(MaxIt,nPop);

%% DE Main Loop
for it = 1:MaxIt
   %assign the best solution as best solution in the current generation
   %which is used as base vector for mutate vector formation
```

```
bestSolutionInPreviousGen = BestSol;
%process
for i = 1:nPop
    %ith individual
    x = pop(i).Position;
    %create a random arrangement (reorder the population numbers)
    A = randperm(nPop);
    %Note: we can do this with rand, but we want all numbers different

    %remove the i index from this since we dont want that to
    %be target vector
    A(A == i) = [];

    a = A(1);   % Random Vector 1 index
    b = A(2);   % Random Vector 2 index

    % Mutant Vector (Mutation)
    beta = unifrnd(beta_min, beta_max, VarSize);

    %y is the mutant vector - best is used as base vector
    y = bestSolutionInPreviousGen.Position + beta.*(pop(a).Position - pop(b).Position);
    y = max(y, VarMin);
    y = min(y, VarMax);

    % Trial Vector (Crossover between Target Vector x and Mutant Vector y)
    z = zeros(size(x));
    j0 = randi([1 numel(x)]);
    for j = 1:numel(x)
        if j == j0 || rand <= pCR
            z(j) = y(j);%from the mutant vector
        else
            z(j) = x(j);%from the target vector
        end
    end

    NewSol.Position = z;
    NewSol.Cost = CostFunction(NewSol.Position(1),NewSol.Position(2));

    % Selection
    if NewSol.Cost<pop(i).Cost
        pop(i) = NewSol;
        if pop(i).Cost<BestSol.Cost
            BestSol = pop(i);
        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) = pop(i).Position(1);
     plotY(it,i) = pop(i).Position(2);
end

% Update Best Cost
BestCost(it) = BestSol.Cost;

% Show Iteration Information
disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);


%plot the graph with the newly updated solutions and best received in the above
%generation process( the drawing code has been seperated from this into
```

```
    %another file named Draw.m)
    %check with the toleranc value to make sure that the graph has been
    %ploted for the optimum solution (0) here tolerance value =0 has been
    %set
     if BestCost(it) <= toleranceValue
        disp(['Tolerance value has been reached at generation: ' num2str(it)]);
        genNumber =[it];
        DrawTheContourPlot
        %Once optimum has been reached no need to run the code again so
        %we terminate the exicution at here
        out.BestCost = BestCost;
        out.BestSol =BestSol;
        return;
     else
         DrawTheContourPlot
     end

end

out.BestCost = BestCost;
out.BestSol =BestSol;
end
```

---

The *DrawTheContourPlot* is defined as:

```
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
        colormap(bone)
        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),"filled","k" );
            title(['Himmelblau Function' num2str(it) "th Generation"]);
            hold on
            scatter(BestSol.Position(1), BestSol.Position(2),"filled","o","r");
            hold off
        end
    end
```

# C2. DE best/2/bin Variant

## C2A: MATLAB Code: Implementation of Algorithm

The algorithm of DE variant best/1/bin has been provided below as a function named “DE_V1()”.

**MATLAB Code:**

```
%% DE /best/2/bin
function out = DE_ForContour(problem, params)
CostFunction = problem.CostFunction; % Cost Function
VarSize = [1 problem.nVar];     % Decision Variables Matrix Size
VarMin = problem.VarMin;            % Lower Bound of Decision Variables
VarMax = problem.VarMax;             % Upper Bound of Decision Variables
toleranceValue = problem.toleranceValue; % tolerance value at which the solution is acceptable with
the maximum error possible

%% DE Parameters

MaxIt = params.MaxIt;       % Maximum Number of Iterations
nPop = params.nPop;          % Population Size
beta_min = params.beta_min;     % Lower Bound of Scaling Factor (0)
beta_max = params.beta_max;     % Upper Bound of Scaling Factor (2)
pCR = params.pCR;          % Crossover Probability
genNumber =params.genNumber;%generation index of which graph must be ploted

%% Initialization

empty_individual.Position = [];
empty_individual.Cost = [];

BestSol.Cost = inf;

pop = repmat(empty_individual, nPop, 1);
%sample solution space
for i = 1:nPop
    %sample creation
    pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
    %sample solution creation
    pop(i).Cost = CostFunction(pop(i).Position(1),pop(i).Position(2));

    if pop(i).Cost<BestSol.Cost
        BestSol = pop(i);
    end

end
%inilialize the BestCost of all generations into zeros
BestCost = zeros(MaxIt, 1);

%plot Control Variables
plotX = zeros(MaxIt,nPop);
plotY = zeros(MaxIt,nPop);
%% DE Main Loop
for it = 1:MaxIt
    %assign the best solution as best solution in the current generation
    %which is used as base vector for mutate vector formation
    bestSolutionInPreviousGen = BestSol;
    %process
    for i = 1:nPop
        %ith individual
        x = pop(i).Position;
        %create a random arrangement (reorder the population numbers)
        A = randperm(nPop);
        %Note: we can do this with rand, but we want all numbers different

        %remove the i index from this since we dont want that to
        %be target vector
        A(A == i) = [];
```

```
        a = A(1);   % Random Vector 1 index
        b = A(2);   % Random Vector 2 index
        c = A(3);  % Random Vector 3 index
        d = A(4);   %random Vector 4 index
        % Mutant Vector (Mutation)
        beta = unifrnd(beta_min, beta_max, VarSize);

        %y is the mutant vector - best is used as base vector
        y = bestSolutionInPreviousGen.Position + beta.*(pop(a).Position + pop(b).Position -
pop(c).Position - pop(d).Position);
        y = max(y, VarMin);
        y = min(y, VarMax);

        % Trial Vector (Crossover between Target Vector x and Mutant Vector y)
        z = zeros(size(x));
        j0 = randi([1 numel(x)]);
        for j = 1:numel(x)
            if j == j0 || rand <= pCR
                z(j) = y(j);%from the mutant vector
            else
                z(j) = x(j);%from the target vector
            end
        end

        NewSol.Position = z;
        NewSol.Cost = CostFunction(NewSol.Position(1),NewSol.Position(2));

        % Selection
        if NewSol.Cost<pop(i).Cost
            pop(i) = NewSol;
            if pop(i).Cost<BestSol.Cost
                BestSol = pop(i);
            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) = pop(i).Position(1);
         plotY(it,i) = pop(i).Position(2);
    end

    % Update Best Cost
    BestCost(it) = BestSol.Cost;

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);


    %plot the graph with the newly updated solutions and best received in the above
    %generation process( the drawing code has been seperated from this into
    %another file named Draw.m)
    %check with the toleranc value to make sure that the graph has been
    %ploted for the optimum solution (0) here tolerance value =0 has been
    %set
     if BestCost(it) <= toleranceValue
        disp(['Tolerance value has been reached at generation: ' num2str(it)]);
        genNumber =[it];
        DrawTheContourPlot
        %Once optimum has been reached no need to run the code again so
        %we terminate the exicution at here
```

```
        out.BestCost = BestCost;
        out.BestSol =BestSol;
        return;
    else
        DrawTheContourPlot
    end

end

out.BestCost = BestCost;
out.BestSol =BestSol;
end
```

## C2B. Effect of Parameter: MaxIt

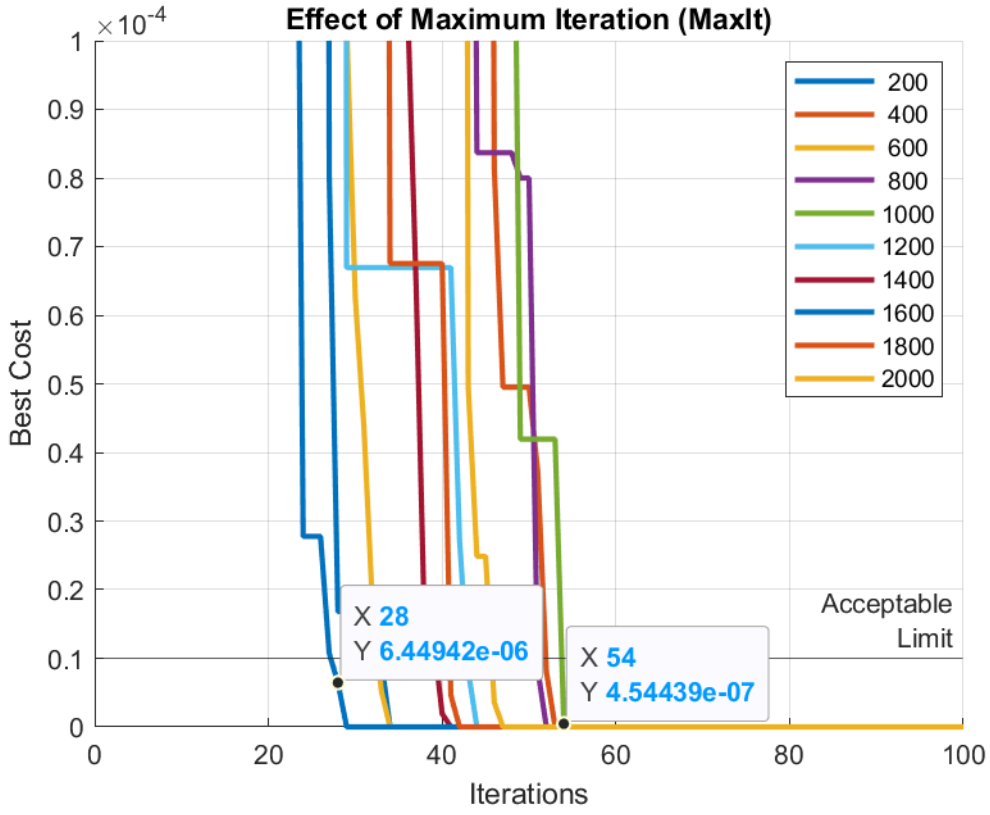



***Figure 18***

**MATLAB Code:**

```
%% DE /best/2/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for varying MaxIt To see the Efffect
%different number of 'MaxIt' values
maxItenVariation = [200:200:2000];
```

```
%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(maxItenVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %Assign value to MaxIt
    params.MaxIt = maxItenVariation(i);
    out= DE_V2(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"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 closer view
ylim([0 10^-4])
xlim([0 100]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(maxItenVariation.'), 'location', 'northeast');
```

## C2C. Effect of Parameter: nPop

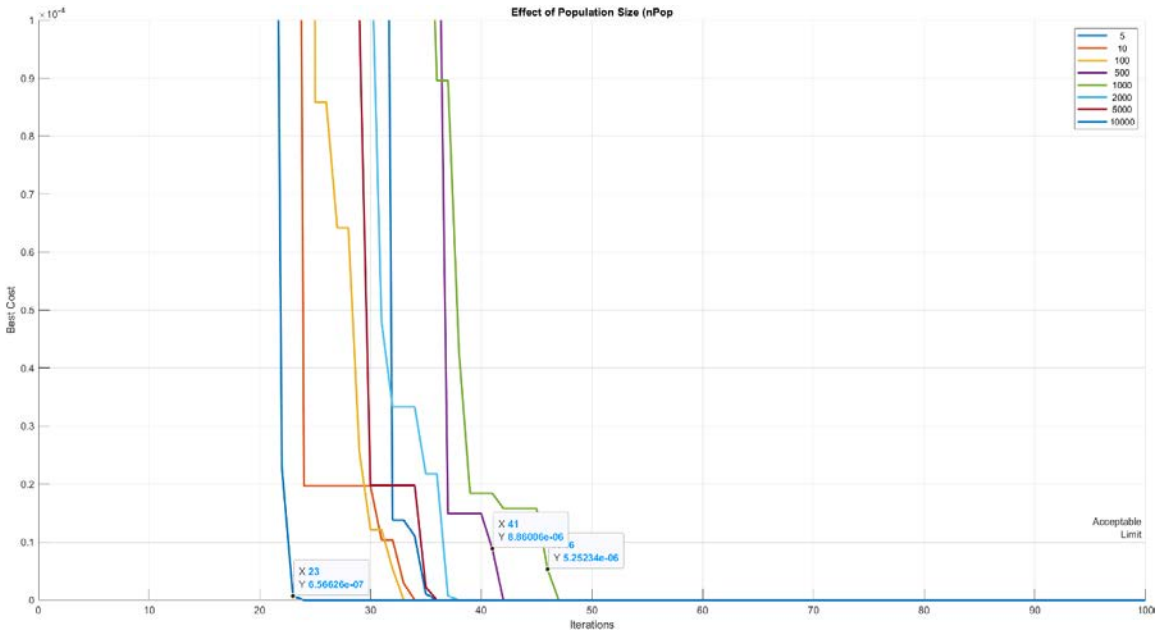



***Figure 19***

**MATLAB Code:**

```
%% DE /best/2/bin
clc; clear;
```

```
%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;              % Number of Decision Variables
problem.VarMin = -5;           % Lower Bound of Decision Variables
problem.VarMax = 5;            % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;       % Maximum Number of Iterations
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for different nPop values to see the effect of it in the Result
%different number of 'nPop' values
nPopVariation = [5,10,100,500,1000,2000,5000,10000];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(nPopVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %Assign value to nPop
    params.nPop = nPopVariation(i);
    out= DE_V2(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"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 closer view
ylim([0 10^-4])
xlim([0 100]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(nPopVariation.'), 'location', 'northeast');
```

## C2D. Effect of Parameter: pCR

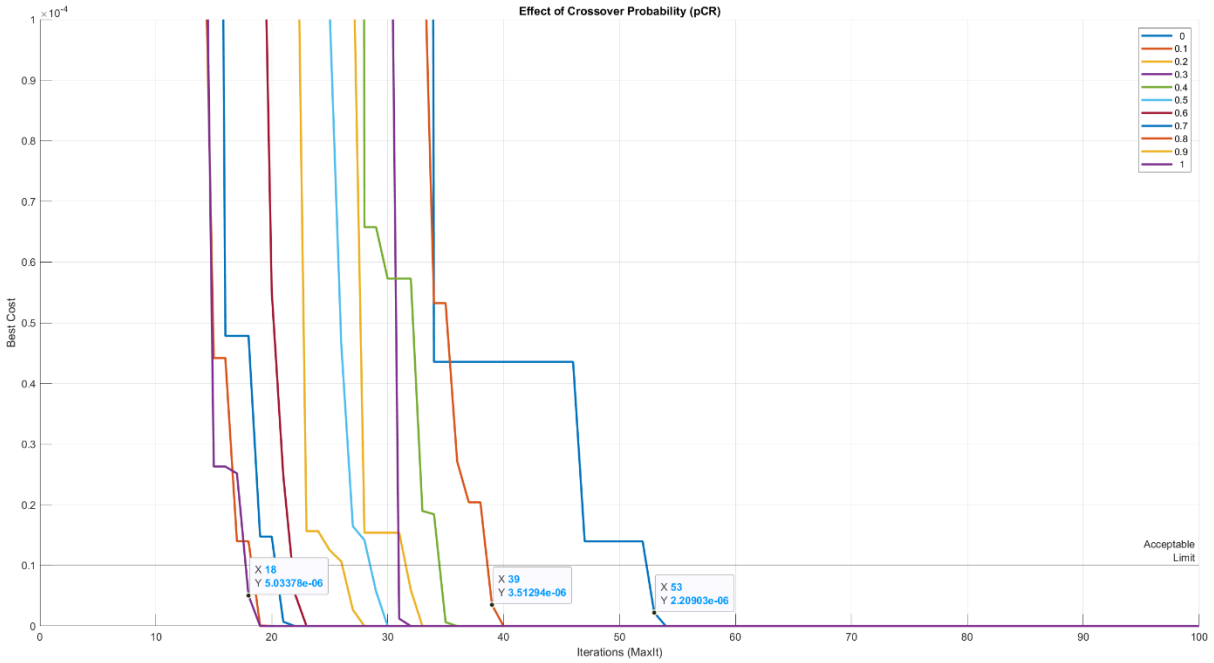



***Figure 20***

**MATLAB Code:**

```
%% DE /best/2/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
params.beta_min = 0.2;    % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;    % Upper Bound of Scaling Factor (2)
%% Calling DE for different pCR values to see the effect of it in the Result
%different number of 'pCR' values
pCRVariation = [0:0.1:1];

%Variable to store the number of Iterations take to reach the tolerance
%value
numberOfIterations = numel(pCRVariation);
%repeat the test for numberOfIterations times
for i=1:numberOfIterations
    %assign value to pCR
    params.pCR = pCRVariation(i);
    out= DE_V2(problem, params);
    % plot the result into the existing graph else create a new graph
```

```
    hold on
    semilogy(out.BestCost,"LineWidth",2);
    hold off
end

% Describing the attributes for the graph
title("Effect of Crossover Probability (pCR)")
xlabel('Iterations (MaxIt)');
ylabel('Best Cost');
%for the purpous of seeing the change in each experiment, xlim is used to
%get a closer view
ylim([0 10^-4])
xlim([0 100]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str(pCRVariation.'), 'location', 'northeast');
```

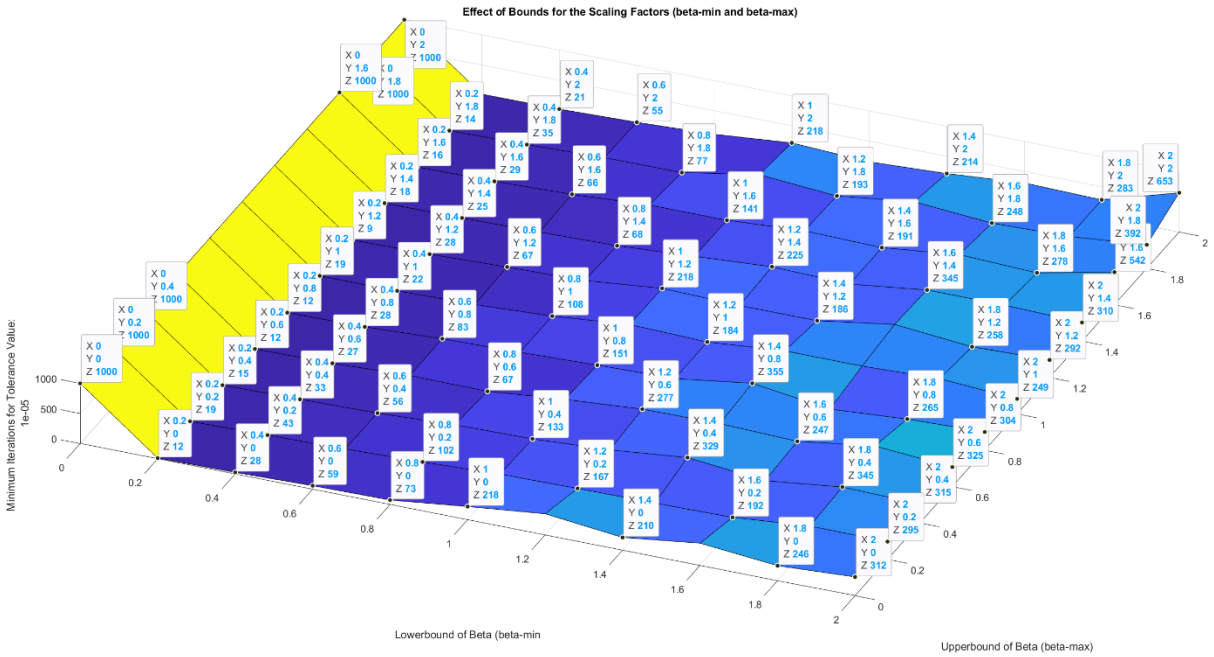



***Figure 21***

## C2E. Effect of Parameter: beta (beta_min & beta_max)

**MATLAB Code:**

```
%% DE /best/2/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
```

```
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is acceptable with the
maximum error possible

%% DE Parameters
params.MaxIt = 1000;      % Maximum Number of Iterations
params.nPop = 50;         % Population Size
params.pCR = 0.2;         % Crossover Probability (in between 0 and 1)

%% Effect of beta_min and beta_max in the result

%save different values of beta_min and beta_max
boundVariations = [0:0.1:1];% for figure A
% boundVariations = [1:0.2:2];% for figure B

% size of variation arrays for the loop generation
sizeOfbetaMinVariation = numel(boundVariations);
sizeOfbetaMaxVariation = numel(boundVariations);

% use two for loops to do the experiment with various combinations
for i=1:sizeOfbetaMaxVariation
   % Lowerbound for scaling factor-beta (bata_min)
   params.beta_min = boundVariations(i);
   for j=1:sizeOfbetaMaxVariation
      % Upperbound for scaling factor-beta (bata_max)
      params.beta_max = boundVariations(i);
      out(i) = DE_V2(problem, params);
      Z(i,j)= out(i).minIterationToReachToleranceValue;
   end
end

% Describing the attributes for the graph
[X,Y] = meshgrid(boundVariations,boundVariations);
Z= transpose(Z)
surf(X,Y,Z)
title('Effect of Bounds for the Scaling Factors (beta-min and beta-max)')
xlabel('Lowerbound of Beta (beta-min');
ylabel('Upperbound of Beta (beta-max)');
zlabel(["Minimum Iterations for Tolerance Value: " num2str(problem.toleranceValue)]);
```

## C2 F. MATLAB Code: Plotting the Contour

```
%% DE /best/2/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.VarMin = -5;          % Lower Bound of Decision Variables
problem.VarMax = 5;           % Upper Bound of Decision Variables
problem.toleranceValue = 0; % tolerance value at which the solution is acceptable with the maximum
error possible

%% DE Parameters

params.MaxIt = 1000;      % Maximum Number of Iterations
```

```
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%Generation numbers needed to be plot as a graph
params.genNumber = [1,30,70,100];

%% Calling DE
out = DE_V2_ForContour(problem, params);
```

---

DE_V2_ForContour() is defined as:

```
%% DE /best/2/bin
function out = DE_ForContour(problem, params)
CostFunction = problem.CostFunction; % Cost Function
VarSize = [1 problem.nVar];     % Decision Variables Matrix Size
VarMin = problem.VarMin;           % Lower Bound of Decision Variables
VarMax = problem.VarMax;            % Upper Bound of Decision Variables
toleranceValue = problem.toleranceValue; % tolerance value at which the solution is acceptable with
the maximum error possible

%% DE Parameters

MaxIt = params.MaxIt;       % Maximum Number of Iterations
nPop = params.nPop;         % Population Size
beta_min = params.beta_min;     % Lower Bound of Scaling Factor (0)
beta_max = params.beta_max;     % Upper Bound of Scaling Factor (2)
pCR = params.pCR;          % Crossover Probability
genNumber =params.genNumber;%generation index of which graph must be ploted

%% Initialization

empty_individual.Position = [];
empty_individual.Cost = [];

BestSol.Cost = inf;

pop = repmat(empty_individual, nPop, 1);
%sample solution space
for i = 1:nPop
   %sample creation
   pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
   %sample solution creation
   pop(i).Cost = CostFunction(pop(i).Position(1),pop(i).Position(2));

   if pop(i).Cost<BestSol.Cost
      BestSol = pop(i);
   end

end
%inilialize the BestCost of all generations into zeros
BestCost = zeros(MaxIt, 1);

%plot Control Variables
plotX = zeros(MaxIt,nPop);
plotY = zeros(MaxIt,nPop);
%% DE Main Loop
for it = 1:MaxIt
   %assign the best solution as best solution in the current generation
   %which is used as base vector for mutate vector formation
   bestSolutionInPreviousGen = BestSol;
```

```
    %process
    for i = 1:nPop
        %ith individual
        x = pop(i).Position;
        %create a random arrangement (reorder the population numbers)
        A = randperm(nPop);
        %Note: we can do this with rand, but we want all numbers different

        %remove the i index from this since we dont want that to
        %be target vector
        A(A == i) = [];

        a = A(1);   % Random Vector 1 index
        b = A(2);   % Random Vector 2 index
        c = A(3);  % Random Vector 3 index
        d = A(4);   %random Vector 4 index
        % Mutant Vector (Mutation)
        beta = unifrnd(beta_min, beta_max, VarSize);

        %y is the mutant vector - best is used as base vector
        y = bestSolutionInPreviousGen.Position + beta.*(pop(a).Position + pop(b).Position -
pop(c).Position - pop(d).Position);
        y = max(y, VarMin);
        y = min(y, VarMax);

        % Trial Vector (Crossover between Target Vector x and Mutant Vector y)
        z = zeros(size(x));
        j0 = randi([1 numel(x)]);
        for j = 1:numel(x)
            if j == j0 || rand <= pCR
                z(j) = y(j);%from the mutant vector
            else
                z(j) = x(j);%from the target vector
            end
        end

        NewSol.Position = z;
        NewSol.Cost = CostFunction(NewSol.Position(1),NewSol.Position(2));

        % Selection
        if NewSol.Cost<pop(i).Cost
            pop(i) = NewSol;
            if pop(i).Cost<BestSol.Cost
                BestSol = pop(i);
            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) = pop(i).Position(1);
         plotY(it,i) = pop(i).Position(2);
    end

    % Update Best Cost
    BestCost(it) = BestSol.Cost;

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);


    %plot the graph with the newly updated solutions and best received in the above
```

```
    %generation process( the drawing code has been seperated from this into
    %another file named Draw.m)
    %check with the toleranc value to make sure that the graph has been
    %ploted for the optimum solution (0) here tolerance value =0 has been
    %set
     if BestCost(it) <= toleranceValue
        disp(['Tolerance value has been reached at generation: ' num2str(it)]);
        genNumber =[it];
        DrawTheContourPlot
        %Once optimum has been reached no need to run the code again so
        %we terminate the exicution at here
        out.BestCost = BestCost;
        out.BestSol =BestSol;
        return;
     else
         DrawTheContourPlot
     end

end

out.BestCost = BestCost;
out.BestSol =BestSol;
end
```

---

*DrawTheContourPlot* is defined as:

```
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
        colormap(bone)
        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),"filled","k" );
            title(['Himmelblau Function' num2str(it) "th Generation"]);
            hold on
            scatter(BestSol.Position(1), BestSol.Position(2),"filled","o","r");
            hold off
        end
    end
```

# APPENDIX D

## D1: DE rand/1/bin: Different Bounds for the input variables (x and y)

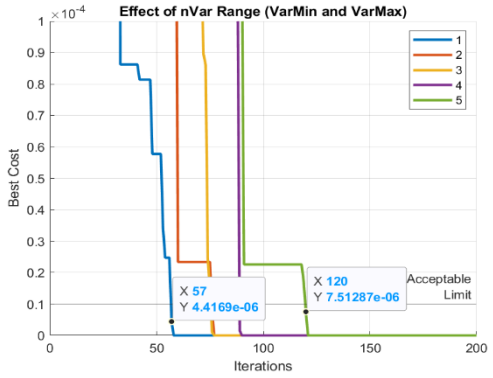



***Figure 22***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is
acceptable with the maximum error possible

%% DE Parameters
params.MaxIt =1000;
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for various Range of nVar
numberofRangeVariations =5;
for i=1:numberofRangeVariations
    %performing the test
    problem.VarMin = -1 * (i*10); % Lower Bound of Decision Variables
    problem.VarMax = 1 *(i*);  % Upper Bound of Decision Variables
    out= DE(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"LineWidth",2);
    hold off
end
```

```
% Describing the attributes for the graph
title("Effect of nVar Range (VarMin and VarMax)")
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 tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str([1:5].'), 'location', 'northeast');
```

## D2: DE best/1/bin: Different Bounds for the input variables (x and y)

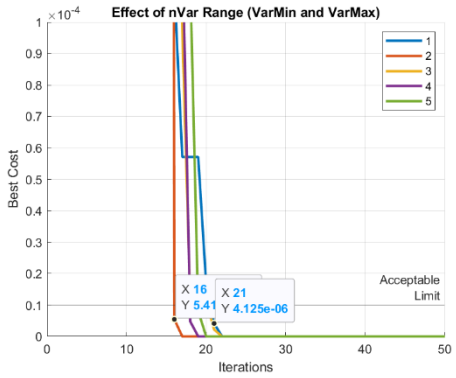



***Figure 23***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition

problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is
acceptable with the maximum error possible

%% DE Parameters
params.MaxIt =1000;
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for various Range of nVar
```

```
numberofRangeVariations =5;
for i=1:numberofRangeVariations
    %performing the test
    problem.VarMin = -1 * (i*10); % Lower Bound of Decision Variables
    problem.VarMax = 1 *(i*10);  % Upper Bound of Decision Variables
    out= DE_V1(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"LineWidth",2);
    hold off
end

% Describing the attributes for the graph
title("Effect of nVar Range (VarMin and VarMax)")
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 50]);
%draw a line parallel to x axis to find on which iteration the output
%reaches to tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str([1:5].'), 'location', 'northeast');
```

## D1: DE best/2/bin: Different Bounds for the input variables (x and y)

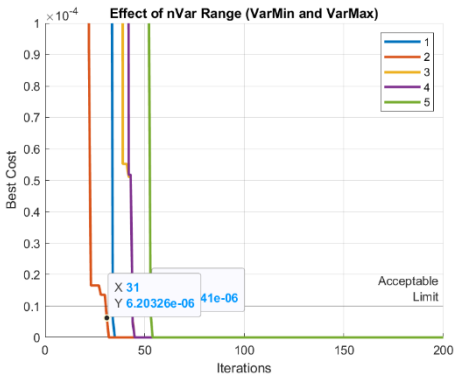



***Figure 24***

**MATLAB Code:**

```
%% DE /rand/1/bin
clc; clear;

%% Problem Definition
```

```
problem.CostFunction = @(x,y) HimmelblauFunction(x,y);    % Cost Function
problem.nVar = 2;             % Number of Decision Variables
problem.toleranceValue = 10^-5; % tolerance value at which the solution is
acceptable with the maximum error possible

%% DE Parameters
params.MaxIt =1000;
params.nPop = 50;          % Population Size
params.beta_min = 0.2;     % Lower Bound of Scaling Factor (0)
params.beta_max = 0.8;     % Upper Bound of Scaling Factor (2)
params.pCR = 0.2;          % Crossover Probability
%% Calling DE for various Range of nVar
numberofRangeVariations =5;
for i=1:numberofRangeVariations
    %performing the test
    problem.VarMin = -1 * (i*10); % Lower Bound of Decision Variables
    problem.VarMax = 1 *(i*10);  % Upper Bound of Decision Variables
    out= DE_V2(problem, params);
    % plot the result into the existing graph else create a new graph
    hold on
    semilogy(out.BestCost,"LineWidth",2);
    hold off
end

% Describing the attributes for the graph
title("Effect of nVar Range (VarMin and VarMax)")
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 tolerance value
yline(problem.toleranceValue,'-',{'Acceptable','Limit'});
grid on;
legend(num2str([1:5].'), 'location', 'northeast');
```