CSIP5304 - Fuzzy Logic & Evolutionary Computing
Weekly Exercises 2 - Evolutionary Computing
MSc Artificial Intelligence, 2023-2024

Lecturer: Dr. Zacharias Anastassi.

Jonathan Atiene.

OUTLINE

Section 1 - Question

You must solve two problems, one unconstrained and one constrained.
Unconstrained Problem
Select one problem from the GA/PSO/DE presentation slides, i.e.

- *the Rosenbrock function,*
- *the Himmelblau function,*
- *the Rastrigin function, or*
- *the Ackley function.*

Wherever you can select the number of variables, i.e. Rosenbrock or Rastrigin function, select nVar = 2.
Constrained Problem
Select one problem from the Constraint Handling Techniques presentation slides, i.e. one of the four mathematical constrained problems, namely:

- *the Constrained Himmelblau problem,*
- *the Constrained Rosenbrock problem,*
- *the Constrained g06 CEC 2006 problem, or*
- *the Constrained g08 CEC 2006 problem,*

or one of the three real-world problems, namely:

- *the Milling Process problem,*
- *the Welded Beam Design problem, or*
- *the Two-Bar Truss Design problem.*

Select one constraint-handling technique from the homonymous slides, i.e.

- *the Static Penalty,*
- *the Dynamic Penalty, or*
- *Deb's Approach.*

Note: If you select one of the four mathematical constrained problems, the objective function must be different from the objective function of the unconstrained problem you have selected.
Methods
Solve each problem with three different methods:

- *Real-Coded Genetic Algorithm,*
- *Particle Swarm Optimisation, and*
- *Differential Evolution method.*

For each of the six problem-method combinations, and for the specific combination of parameters values, operators, techniques, and variants, explained in section 3, find the minimum number of iterations needed so that the absolute error of the best solution is below a user-defined tolerance, e.g. 10^(-2). For the evaluation of the error, use the theoretical/reference global minimum of each problem, also provided in the slides. Repeat the experiment several times and take the median value of the number of iterations needed. For the problems that have two independent variables, i.e. the four unconstrained problems and the four constrained problems, provide some contour plots on the x-y plane, showcasing the progression between the stages of the evolution, at different number of generations.
The parameters, operators, techniques, and variants of the three methods are shown below.
Real-Coded Genetic Algorithm

- *Parameters: maximum iterations, population size, crossover probability, mutation probability, and any additional parameters used in the operators.*
- *Operators: selection, crossover, mutation, and survivor.*

You may use the analysis and results from Weekly Exercises 1, if appropriate, for the operators that may be common with the Binary-Coded Genetic Algorithm, e.g. selection and survivor, and further investigate the other operators and parameter values in terms of performance.
Particle Swarm Optimisation

- *Parameters: maximum iterations, population size, inertia coefficient, personal acceleration coefficient, social acceleration coefficient, and any additional parameters used in the dynamic/advanced techniques.*
- *Dynamic techniques: damping factor for the inertia coefficient.*
- *Advanced techniques: constriction coefficients.*

You may use the analysis and results from Weekly Exercises 2, if appropriate.
Differential Evolution method

- *Parameters: maximum iterations, population size, scaling factor, crossover probability and its range, and any additional parameters used in the variants.*

- *Variants utilising x/y/z strategies.*

You may use the analysis and results from Weekly Exercises 3, if appropriate.
Effect of parameters, operators, techniques, and variants

- *For each of the three methods, report the effect of different values of parameters, operators, techniques, and variants, which are mentioned in the previous paragraphs, and state the combination for the most efficient combination.*
- *Use at least two different operators, techniques, and variants for each type when comparing (four operator types for RGA, two techniques for PSO, two x/y/z strategies for DE).*
- *If you observe behaviours that differ between the unconstrained and the constrained problems, state the differences.*
- *Demonstrate why this combination is the most efficient for each problem (or both).*
- *Determine which parameters, operators, techniques, and variants have the highest impact on the efficiency, and which have the lowest.*
- *Determine the effect of the constraint handling technique parameters and examine whether the constraints of the final best solution are satisfied or not.*

Comparison, Interpretation and Conclusions

- *For each problem, compare the three most efficient methods of each type (one RGA, one PSO and one DE). Compare the efficiencies for the tolerance which you based your analysis on.*
- *How well the best methods (without re-tweaking the parameters) perform if we change the tolerance? Interpret your findings.*
- *How does the constraint handling parameter affect the results?*
- *Report on the common and different behaviours between the methods.*
- *Interpret the results, draw conclusions, and provide recommendations for further improvement.*

Important notes

Create a report that answers the above questions.
As a guide you should aim to make the report around 15 sides of A4 (using Arial font and minimum size 11) and between 2000-3000 words, including appendices. We will not impose a penalty for reports that are too long or too short, but this is intended to be a technical report and it should present the work in a concise but detailed manner.
Use headings for sections and subsections with bold typeface, clearly stating the content, e.g. "RGA – Constrained Himmelblau problem" for a section heading, or "$\mu+\lambda$ and μ, λ Selection Operators" for a subsection heading.
The code, output, if used, and some additional graphs can be included in an appendix, however the main part of the project, which includes the text and the main graphs, should independently provide the main points.
The modified code should be copied as text, as opposed to an image, in the report and also as separate .m files.
The graphs should be clear and have titles, axis labels, legends etc.
The provided code should be considered as a good starting point, and is subject to improvement, especially on the documentation. Improve the readability (e.g. add some comments in commands, variables, and functions to explain what they do, even those included in the original code).
Ensure that the code has optimum performance (e.g. avoid calculating the same formulas more than once where not necessary).
Make the programme as parameterised as possible. This means that any constant should be defined in the main script file once, and then only the variable name should be passed to the functions.

Section 2.1 – Introduction to Evolutionary Algorithms

Evolutionary Algorithms are computational methods that follow the same principle as Darwin's theory of evolution, this algorithm uses parameters like the population of the organism, successive number of generation counts, mutations and crossovers (Sherstnev, Pavel 2024), The world today is made up of species that fit perfectly into their inhabitants, an unexplainable solution as to how they evolved to this point and how they morphed into having the best attribute to inhabit their ecosystem.

Evolutionary Algorithms mimic this principle by helping us find the best population over several generations by removing the unwanted species through the process of selection and elimination and reproducing the best species through the biological process of crossovers and exposing these populations to mutations which can either be a positive or negative effect to the organisms objective function.

In the development of an Evolutionary Algorithm, there is a lot of abstraction and randomness as to the best way to stimulate ideas like selection, what a random population consist of, the degree of mutation, how to apply the concepts of competition among the organisms, this led to more detailed research and the discovery of several approach and methods such as the roulette wheel selection, tournament selection and different criteria for elimination, types of crossover (Stańczak, Jarosław 2024). We will be exploring this information here and also be tuned to find the most optimal parameters or procedure to attain a near-optimal solution.

Evolutionary Algorithms are optimisation algorithms, in present-day engineering problems have been mustered up to the point where speed, productivity and efficiency are important points to be tackled in most problems (Nurmuhammed *et al,* 2024) and as such a lot of inspiration has been taken from observations around is the Honey Badger Algorithm which was inspired by the behaviour of honey badgers in search for food (A.Hashim *et al.* 2022). More computing problems with a higher number of variables and multiple dimensions, like using the Honey Badger Algorithm for optimising the nodes in wireless sensor networks (Nguyen et al. 2023) these algorithms not only help us get faster solutions but also be able to get near accurate solutions.

Section 2.2 - Generic Flow of Evolution Algorithm

To understand the generic steps for implementing Evolutionary Algorithms we must first understand the flow of evolution, here we identify a generic approach with which all organisms have evolved and try to formulate a compute theory or implementation for this process.

Variation, inheritance, reproduction (crossover), Elimination, Selection and gradual selection of favourable traits over time are some of the notable points Darwin (1859) pointed out in his early work in the book on the theory of evolution. A computational flow for evolution will be selecting a random population, and defining the operators and variables, we then impose a selection method to choose only the best-performing organisms, this combination and continuous selection over a while then leads to better adaptation of the organisms (Eiben, 2015).

A typical Evolutionary flow: (Eiben, 2015).

1. **Random Initial Population**: this population contains a random sample of the good solutions or bad solutions,
2. **Fitness Value**: the fitness value of each of the population, this is a measure of the objective function, depending on the aim of the function we can ascertain the quality of the fitness value for an optimization problem that aims at minimizing the cost function.
3. **Selection Procedure:** This report will explore certain selection procedures which will be used to determine the parents for the next offspring

4. **CrossOver and Mutation:** CrossOver and Mutation are the next steps after Selection, Mutation introduces environmental factors that show the effect of environmental interactions with the organism this is randomly distributed depending on the mutation operator. Crossover involves producing offspring, the fitness value for the new offspring is then computed
5. **Selection Procedure:** The members of the population are selected, we will also be implementing several selection procedures in this write-up.
6. **Elimination:** The organisms not selected are then eliminated and the process is repeated until the successive number of generations is attained, or a termination criteria is set

```
function e = EvolutionComputingAlgorithm()
    population = InitilizePopualation()
    for i = 1:NumberOfGenerations
        fitnessValue = FitnessFunction(population)
        selectedParents = selectionProcess(population, fitnessValue)
        children = crossoverProcess(selectedParents, crossoverOperator)
        children = mutationProcess(children, mutationOperator)
        children = FitnessFunction(children)
        population = selectionCriteria(selectedParents, children)
    end
end
```

Figure 1 shows a pseudo-code that shows the step-by-step flow of the evolution Algorithm.

Section 2.3 - Classification of Evolutionary algorithms

Evolutionary algorithms are classified based on differences in the structure and the type of evolutionary concept they implement, we will be dealing with only 3 classes of Evolutionary Algorithms in this course work namely

Genetic Algorithms: This is an evolutionary algorithm that imitates the genetic makeup of organisms, it implements the biological process of evolution on the genetic level and aims to find the fittest solution for evolution (Carr, J. 2014).

Genetic algorithms (GAs) date back to the 1960s; it was coined by Holland and his partners at the University of Michigan. The goal was not to design an algorithm but to compute and conduct a formal study on how adaptations occur (Mitchel, M. 1999).

In GAs, there is a fitness function, an objective function, chromosome population, selection of the fittest chromosomes to make up the parent, crossover, and mutation to produce new offspring (Carr, J. 2014).

A genetic algorithm begins with a randomly chosen set of chromosomes, which serves as the first generation (initial population), and simultaneously checks each chromosome and selects the best for the next cycle until we reached convergence or the point of global minimum. Genetic algorithms in parallel check many points in the solution space, optimize the points and provide several optimum parameters instead of a single solution, and has proven to be efficient when tested on several problems (Carr, J. 2014).

Advantages of GAs

GAs work with a pool of candidate solutions, increasing the chance of finding the best solution. Less Prone to Local Optima: They are less likely to get stuck on a suboptimal solution and can overcome "hills" in the search space. Less Complex: GAs don't need complex mathematical functions (derivatives) to work. They only rely on a fitness function you define to measure how good a solution is. They work well with noisy, non-linear problems where traditional methods struggle. GAs can be used in many situations because they

don't rely on specific mathematical properties of the problem. You can adjust GAs to prioritize either finding the absolute best solution (accuracy) or finding a good solution quickly (efficiency). (Carr, J. 2014).

Particle Swarm Optimization:
In 1995 James Kennedy and Russell Eberhart, introduced the term Particle Swarm Optimization, it is an optimization technique that is based on the behaviour of particles in a swarm, it is inspired by the collective behaviour of bees and birds, where elements comes together to find the best solution (Jones, A. 2023).

General PSO Algorithm: PSO follows a slightly different approach than genetic also algorithm, because it deals with motion and velocity, when particles move together in a swarm the aim is to find a solution in the swarm they do this by finding the local best particle in this population and having other particle moving towards the current local best.

```
function pso_algorithm()
    % Initialize swarm population
    population = initializeSwarmPopulation();  % Note: Capitalize function names
    gbest = Inf;  % Initialize gbest to positive infinity
    for i = 1:number_of_iterations  % Adjust loop syntax
        lbest = evaluate(population);
        if lbest < gbest
            gbest = lbest; % Use semi-colons to suppress output
        end
        velocity = update_velocity(i);
        population = update_position(population);  % Assume element-wise operations
    end
end
```

Figure 2 shows a pseudo-code that shows the step-by-step flow of the particle swarm optimization Algorithm.

Real Genetic Algorithm Parameters:

Parameters	Description	Value
costFunction	The function to be minimized or maximized “min one”	@(x) MinOne(x)
numberVariables	Number of decision variables in the problem	100
maximumIteration	Maximum number of iterations for the algorithm	100
populationSize	Number of individuals in each generation	100
beta	Parameter used in selection pressure for probability calculation	1
crossOverProbability	Probability of crossover occurring between two c	1
mutationProbability	Probability of mutation occurring in an individual	0.02
selectionMethod	Method used for selecting individuals for reproduction	RouletteWheel

Table 1 shows a parameters, description and default values of the Real Genetic Algorithm

Differential Evolution:

Differential Evolution (DE) is an Optimization Algorithm that is based on a search technique for finding the most powerful parameter variables by using standard evolutionary computing. (S. Das and P. N., 2011). Unlike other Evolution algorithms, DE seems to slightly deviate in its Mutation phase, at this point we randomly select vectors called the mutant vector for a specific generation to be perturbed by a target vector that is undergoing mutation

for that specific population and space, together these two vectors form the mutant vector which is the result of the mutation, after mutation, selection, etc.

Symbol	Description	Value
Test Function	The Rosenbrock Test Function will be used to test the Unconstrained problem and the Himmelblau Function will be used to work on the constrained optimization problem	
Constraints	Inequality constraints ensures that all solutions are within the feasible search space as defined by the problem.	X >= 0
Optimal Solution	The theoretical or known optimal solution value is set to 517.063. This value is utilized to assess the accuracy and convergence of the Himmelblau algorithm.	517.063
Number of Generation	Number of generations needed for the evolution.	100
nPop	Population size is the number of vectors evaluated in each generation	50
Beta_min	Lower bound of scaling factor this range allows for mutation of the perturbing factors	0.2
beta_max	Upper bound of the scaling factor	0.8
pCR	Crossover probability	0.2
tolerance	Tolerance value	10^-2
Static Penalty Value	Static penalty value	10

Table 2 default parameter and description of parameters

Particle Swarm Optimization:

The Particle Swarm Optimization (PSO) Algorithm Code: The PSO algorithm as shown in the code was used to evaluate the global best, best cost function and convergence of the algorithm, against several parameters as shown below Other parameters will be introduced later in this report.

Parameters	Description	Value
numberVariables	Number of decision variables in the problem	2
maximumIteration	Maximum number of iterations for the algorithm	100
populationSize	This is the size of the population in the swarm	50
inertiaCoefficient	The inertia coefficient is a factor that influences the velocity of each particl swarm.	1
wdamp	Parameter that reduces the coefficient of the inertia as it goes through the iteration.	1
tol	determines the acceptable level of accuracy for the solution.	10^-2
velocityControl	controls how drastically particles can change their position in each iteration	0.2

Table 3 shows a parameters, description and default values of the particle swarm optimization algorithm.

Section 2.4 - Objective Functions and Optimization functions

Objective function is a means to maximize or minimize a value,

Optimization functions:

Algorithms are iterative processes computed to solve a problem, they can be expressed as equations, and the convergence of these equations is mostly controlled by factors and parameters defined by the context

of the problem (Xin-She Yang, 2019.). Optimization functions are necessary in algorithms; they are useful in achieving quicker convergence, reducing the time of execution, and even minimising the results of variables. A profound example is minimising the surface area of a given volume, this is an example of a minimization Optimization function.

Generic structure of Optimization functions:

maximise/minimise $f(x), x = (x, x1, \ldots xi)$ subject to

$b(x) = 0\ (j = 1,2,\ldots,Jn),$

$d(x) \leq 0\ (k = 1,2,\ldots,Kn)$

The generic structure of optimization functions involves a goal to either maximize or minimize the objective function f(x). The vector x consists of variables xi that we seek to optimize.

The objective function is subject to a set of constraints:

1. **Equality constraints:** b(x) = 0 for j = 1, 2, ..., Jn. These constraints impose a set of equations that the variables x must satisfy.

2. **Inequality constraints:** d(x) ≤ 0 for k = 1, 2, ..., Kn. These constraints impose conditions that the variables x must adhere to, ensuring that certain values are not exceeded.

The goal is to find values for the variable x that satisfy the constraints while optimizing the objective function. This involves finding the optimal point that achieves the best possible value of the objective function within the boundaries set by the constraints. Constrained and Unconstrained Optimization functions are the types of optimization functions, it is unconstrained if constraints are defined and unconstrained if the constraints are not defined (Xin-She Yang, 2019.).

Section 2.5 - Constraint Handling techniques

There are 5 popular techniques for handling constraints in evolutionary computation, they include penalty functions, Deb's feasibility rules, stochastic ranking, constrained method and gradient-based repair. (Victor H. *et al.,* 2021). In the course of this report, we will be focusing more on the penalty functions.

Penalty functions:

Penalty functions have been the most common way to include constraints in evolutionary algorithms and mathematical programming since the 1940s. This is because they are simple and efficient. Penalty functions modify the fitness landscape by adding a penalty value to the objective value of each infeasible individual (Victor H. *et al.,* 2021).

*penalty_function (x, c) = f(x) + r * sum(max(0, g(x)))*

where:

- f(x) is the objective function
- g(x) is the constraint function
- r is a positive penalty parameter

It is suggested that the value of r be kept very low and then increased gradually over time, just above the limit where the found solutions are infeasible, that is called the minimum penalty rule (Victor H. *et al.,* 2021)

This is done because in a minimization problem if r is too low leaving the constrained space might be difficult, compared to move the particles to the feasible region.

Types of Penalty Functions:

We have several types of penalty functions, Death Penalty, Static Penalty and Dynamic Penalty among the others will be our main focus in this report. In death penalty function methods, individuals who violate any one of the constraints are completely rejected, and no information is needed from infeasible solutions and this makes the solution very efficient (B. Tessema and G. G. Yen, 2006). In static penalty it is calculated as the summation of the constraint violation added to the objective function value, the penalty parameter is constant throughout the iteration. Lastly, it is called a dynamic penalty if the current generation count influences the penalty value (B. Tessema and G. G. Yen, 2006).

Section 2.6 - Benchmark Functions

These are objective functions used to compare the performance of multi-modal optimization problems, it is important because the best solution may not always be the best due to various constraints. Multi-modal problems aim to find good local minima and select the best solution among the potential solutions. Evolution algorithm is one of the most common choices for solving multi-modal problems due to the use of population and multiple variables (B.Y. Qu, *et al.*, 2016).

In the course of this report, I will be using two major test functions, Rosenbrock and Himmelblau function

Rosenbrock function: is a non-convex function, which is used as a efficiency test problem for optimization algorithms, it helps find the lowest or highest values of that function.
Defined mathematically as $f(x, y) = 100 (y - x 2) + (1 - x) 2$

Himmelblau Function it is a multi modal function, that has it has 4 similar local minima
Defined mathematically as $f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$

Experiment Setup:
This experiment will be conducted in a MATLAB environment, the algorithm will be executed 10 consecutive times and plotted on a graph as a single line chart, the median value will be taken as the accepted optimal result of the benchmark, we will also show convergence on a contour plot for each of our benchmark, **"TolMin"** will be referred to as the number of iteration needed to reach the tolerance value.

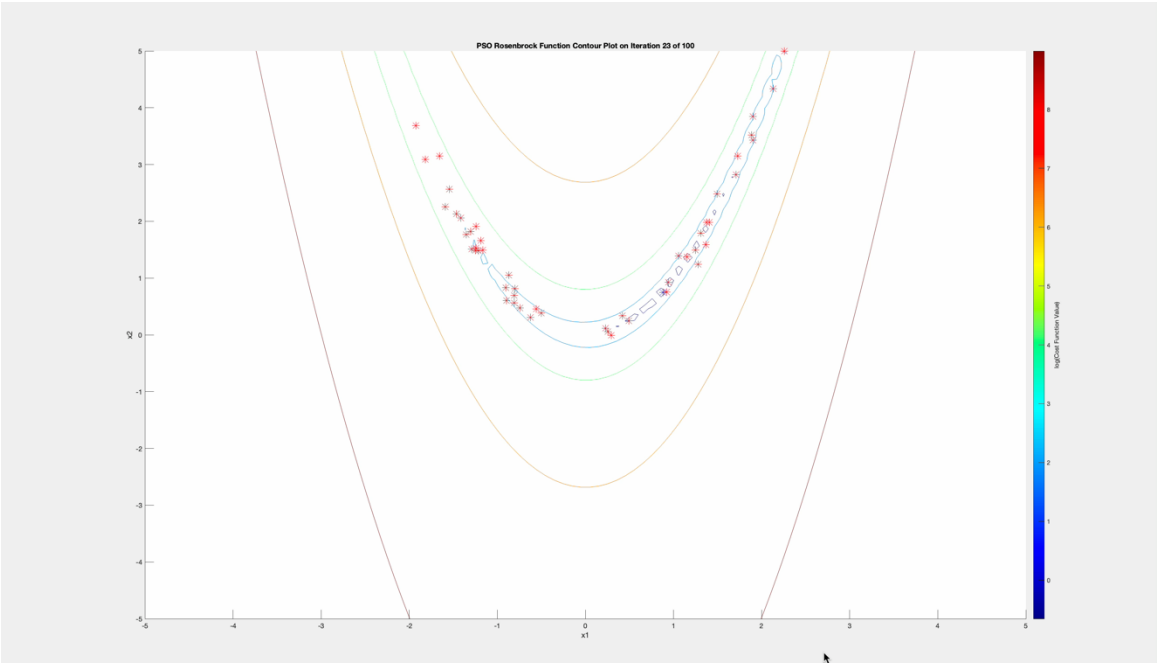
Section 3 – LAB REPORTS

Section 3.1 – RGA Bench Mark

Section 3.1.1 - Default Parameters based on slides:

Parameters	Description	Value
testFunction	The optimization function methods to test the effect of the algorithm	@(x) Rosenbrock (x)
numberVariables	Number of decision variables in the problem	2
maximumIteration	Maximum number of iterations for the algorithm	200
populationSize	Number of individuals in each generation	40
crossOverProbability	Probability of crossover occurring between two c	1
mutationProbability	Probability of mutation occurring in an individual	0.5
Cross Over Operator	SBX crossover operator (nc)	15
mutation Operator	Polynomial mutation Operator (nm)	20
Elimination Operator	Strategy for Elimination	(n + γ) - strategy
Selection Operator	Binary tournament selection operator	

Table 4 showing the benchmark of the default parameters as shown on the slide.



Section 3.1.2 - Default Benchmark according to the parameters on the slide:

This is the default benchmark using the default operators from the slide. Using the default parameters above in **Table 4** it showed a first min convergence value of 180 we see values ranging as low as 66 to as high as 192, it shows that this algorithm works properly to find the global minimum point.

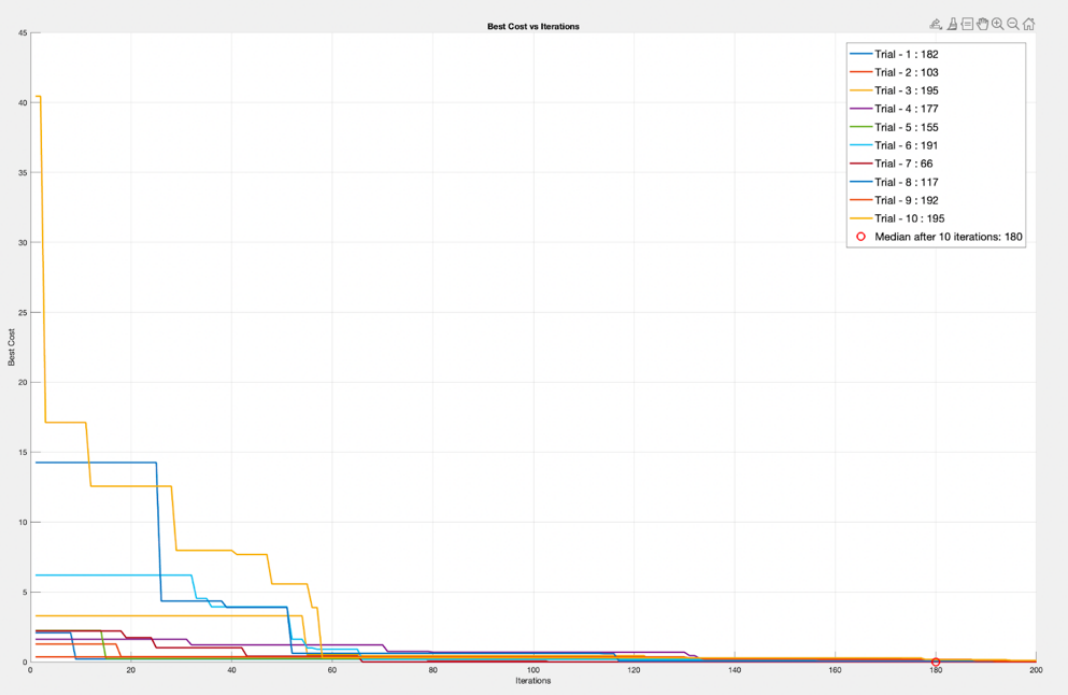


Figure 3 showing the benchmark of the default parameters as shown on the slide.

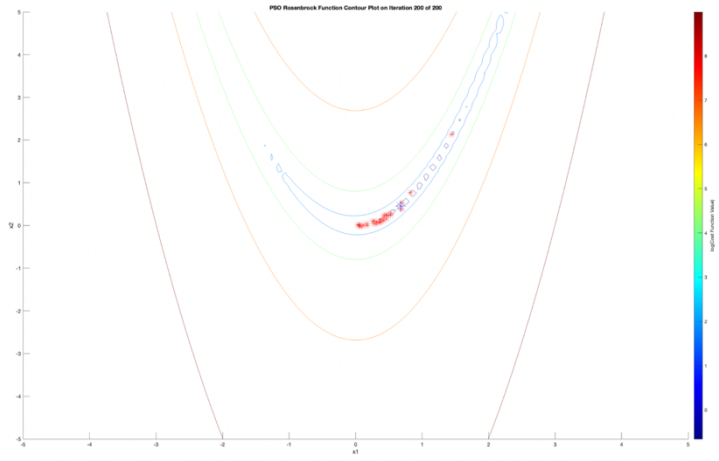


Figure 4 showing contour plot for the RGA un-constrained Rosenbrock function

The contour plot in Fig.4 shows the contour plot for the Rosenbrock function, here we can see that. The particles found their way to the point of global minimum of the Rosenbrock Plot.

It is also noticeable that there is no total convergence since all particles did not exactly converge to the exact point of the current global best particle.

Section 3.1.2 - Parameter tuning to find optimal parameters for these values.

maximumIteration	50	100	250	500	1000
populationSize	50	100	250	500	1000

crossOverProbability	0	0.25	0.6	0.75	1
mutationProbability	0	0.25	0.6	0.75	1
Cross over operator	5	10	15	30	40
Mutation operator	5	10	15	30	40

Table 5 finding the best parameter combination

I will be tuning these parameters in table 5 with the aim to find the best fit combination, my choice of values ranges from 50 - 1000 for the maxIteration, and same for population size, we do not repeat the values of the from the optimal parameters because we already have a bench mark showing those values.

Section 3.1.3 - Results from tuning

The table 6 below we will be deducing patterns from the different parameters and choose the optimal value, the values shows the first minimum convergence generation.

maximumIteration		populationSize		crossOverProbability		mutationProbability		Cross over operator		Mutation operator	
50	49	50	179	0	123	0	152	5	178	5	10
100	60	100	195	0.25	146	0.25	155	10	200	10	92
250	242	250	58	0.6	148	0.6	131	15	105	15	100
500	483	500	155	0.75	148	0.75	94	30	112	30	185
1000	410	1000	195	1	196	1	Inf	40	107	40	178

Table 6 Results from parameter combination

Section 3.1.4 – Determining the most effective combination of parameters for RGA.

From the results i got above i decided to choose some promising values,

i. **Maximum Iteration:** The effect of iteration count shows a significant importance in the convergence of the plot, with lower values like 50 showed 49 and 100 showed 60 as the convergence point, while larger values like 500 – 1000 also showed larger values, 50 and 100 showed the lowest convergence point, it is certain that the min convergence point for this values did not converge properly, because of how close the min point was to the end of the iteration. Especially when compared to higher values like 1000 and 500.

   Reviewing the contour plot in figure 5 It is also obvious that most of the individuals in the solution did not attain convergence, as they did not get the global minimum point of the Rosenbrock function point. The choice here is therefore a 1000 iteration as it shows the best convergence and a convincing ratio that the Optimization Algorithm has achieved its true global minimum.

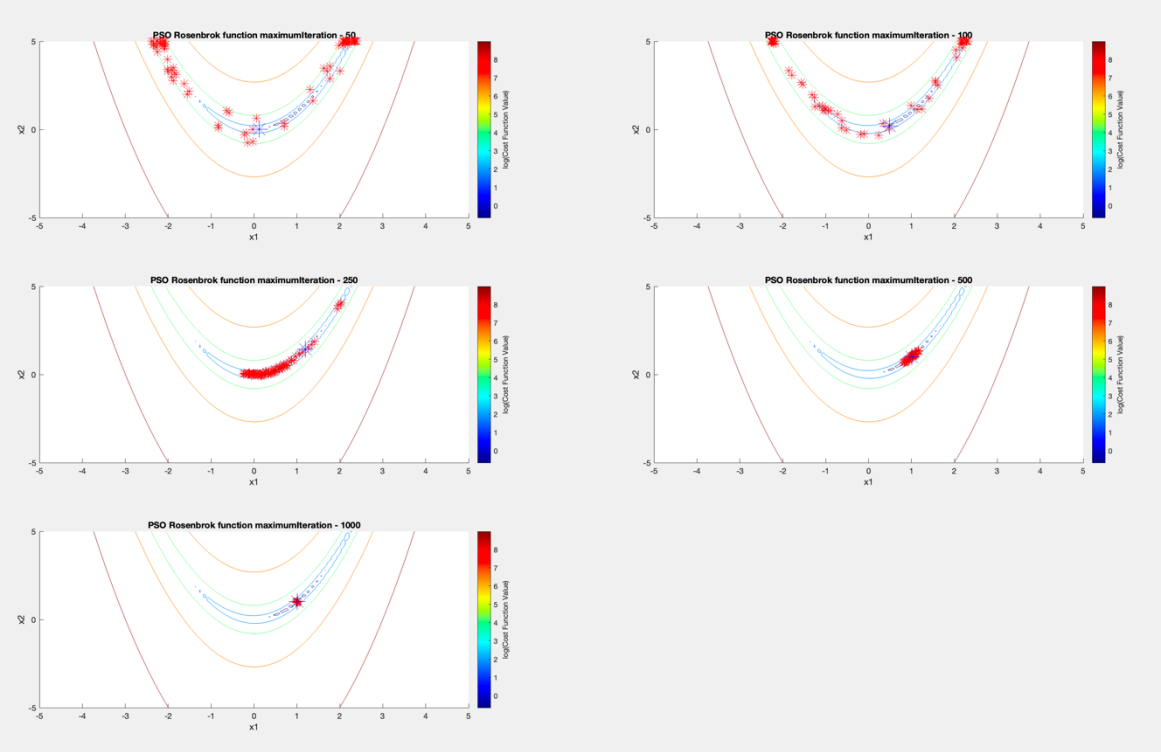


Figure 5 contour plot showing effect of maximum iteration count on RGA- Rosenbrock

ii. **Population Size:** The amount of population introduced to the algorithm affects level of correctness of the algorithm, when the population is too low, it means that we are limited to a few diverse populations set. On the contrary an increase in the population size can also lead to slower convergence rate, since this leads to increase computation power the results show a drastic reduction in min convergence rate when the population size was 250 and such I will be choosing this as my optimal value.

iii. **Crossover Probability** When there's no crossover, the performance seems to be best this might be due to premature convergence as there is not enough time to explore the right population and hence no divergence. The result also shows that for Low to Moderate Crossover (0.25 to 0.75): There was a steady increase in performance, it can also be deduced that a lower crossover probability leads to a stricter population and hence a premature convergence, when the crossover probability was at a crossover probability of 1 where every pair of parents undergoes crossover, there's a significant increase in TolMin. This implies that if every parent performed a crossover then, we would have a more diverse genetic pool to deal with and hence a longer generation to point out min convergence point, the figure 6 shows the contour plot and the reaction of the population to cross over probability at 0 there is no convergence but increased convergence with higher values.

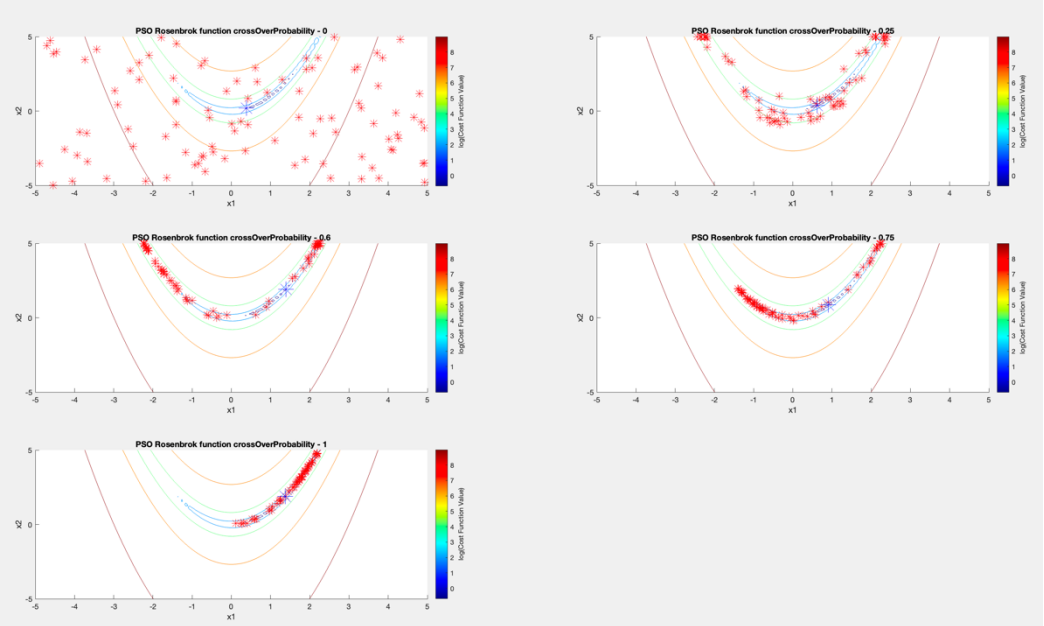


Figure 6 effect of crossover probability on RGA - Rosenbrock Function

iv. **Mutation Probability:** 0.6 highest performance before a significant drop,

v. **Crossover Operator Performance:** For the is values based on the results in **Table 3** I will be picking the best per 148 (best performance with chosen crossover probability)

vi. **Mutation Operator Performance:** 15 (highest stable performance)

The most effective combination will thus be

maximumIteration = 1000; populationSize = 250; crossOverProbability = 0.75; mutationProbability = 0.6; Cross over operator= 15; Mutation operator=15;

Section 3.1.5 - Benchmark of efficient Parameters Combination

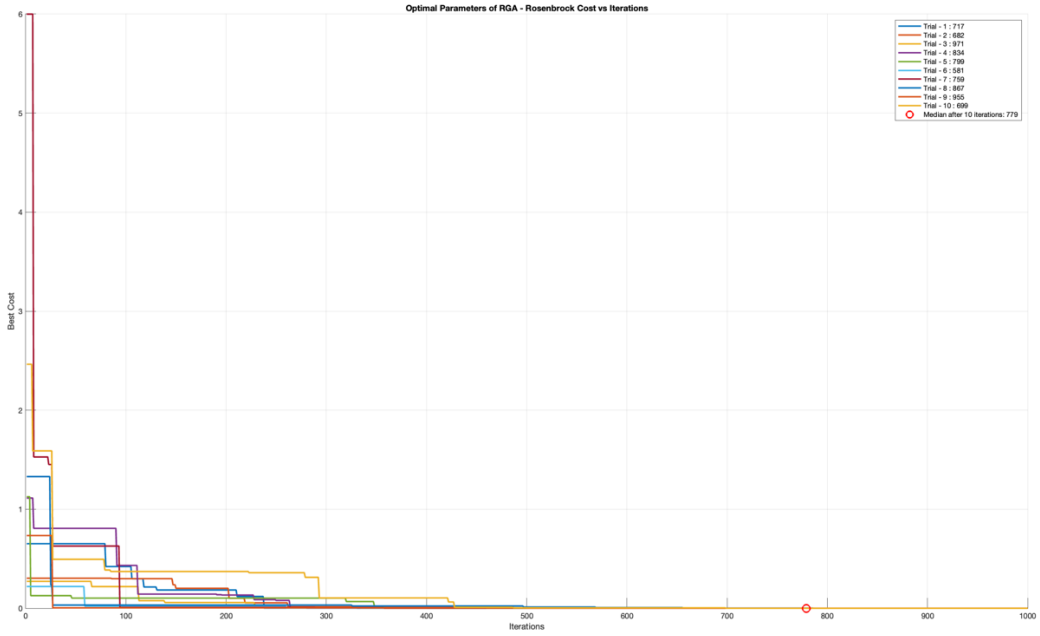
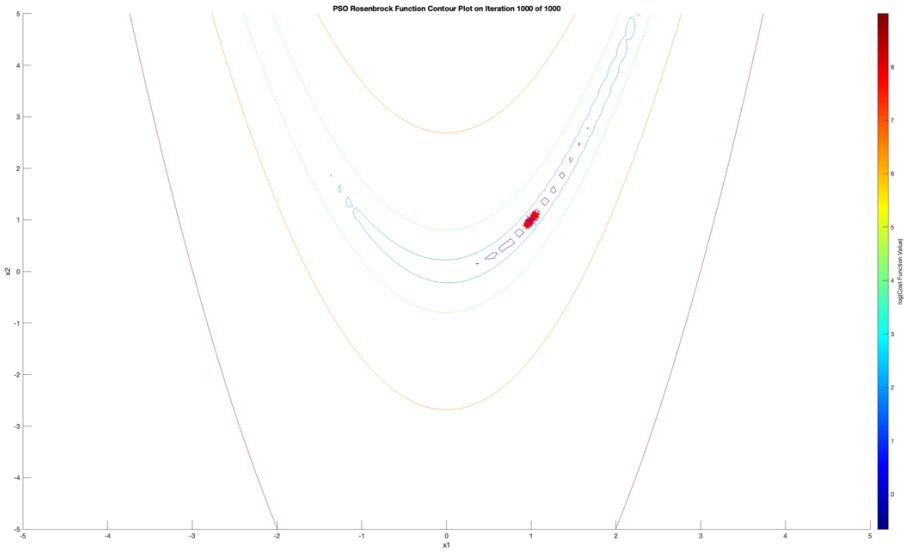


Figure 7 showing median benchmark of efficient combination



Section 3.1.6 – Determining the most effective Operators for RGA.

To determine the effect of operators on the unconstrained RGA we will be swapping out operators at different level and then taking the benchmark after 10 trials. The Operators for each level will be

1. **Effect of Linear Cross Over Operator vs SBX Operator:**

   - The Linear crossover operator involves producing offspring's that are weighted average of the combination of the parents, this does not allow for increased diversity because off springs are always within the bounds of the parents Genes, unlike the SBX cross over operator that maintains diversity hence allowing the off springs to fall outside the interval defined by the parents' genes.
   - The median benchmark for this operator as seen on the fig is 302, this shows a drastic reduction as when compared to SBX crossover operator, we got a maximum of 900 and a min of 89, this is relatively low compared to 779 which we got from the SBX operator.
   - It can be deduced that the Linear crossover operator needs less computational power when compared with the SBX operator, since we have less diversity, it takes less time to attain convergence.
   -

2. **Effect of Random mutation operator vs Polynomial Operator:**
   - The Random mutation operator This operator typically modifies the genes of the individuals by adding a random value, the random value is usually within a range, while the Polynomial operator is more sophisticated than random mutation because it is more random and is used to introduce more randomness in the mutation.
   - The result of this benchmark shows a median value of 738 as shown on fig. when compared to 779 from the optimal parameter,
   - A worthy point to note would be that while the Polynomial operator there was a good exploration of space compared to the exploitation where the Random mutation operator seems to be a better operation for exploitation problem.

3. **Effect of Roulette Wheel selection operator vs Tournament selection Operator:**
   - The Roulette Wheel selection operator uses a probabilistic approach to selection of individuals a roulette wheel where each individual has the same probability to be selected. A random spin of the wheel determines which individual is selected, while the Tournament Selection Operator involved choosing a tournament size and having individuals compete against themselves by comparing their fitness values.
   - The median value of this operator was 655 compared to the 779 of the Tournament Selection operator, while the Tournament selection spent more of the generation performing exploitation as it searched for the global best it was seen that that the Tournament Selection operator did more exploration and less exploitation since it took random values and then spends most of the time finding the global minimum point of the Rosenbrock function.

3. **Effect of Elimination (x, y) – strategy vs (x + y) – strategy :**

Section 3.1.7 – Constrained Himmelblau Optimization Function for RGA

Here we have an extra parameter to look out for this will be the results of optimizing parameters of the RGA constrained optimization. We will be using the Himmelblau Test function, Inequality Constraints, and the static penalty constraint handling technique. The constraints will are expressed mathematically as

$$\boldsymbol{C(x) = 26 \geq (x1 - 5)^2 + x2^2}$$

$$\boldsymbol{C(x) = 20 \geq 4x1 + x2}$$

$$\boldsymbol{C(x) = x1, x2 \geq 0}$$

Maximum Iteration		Population Size		Crossover Probability		Mutation Probability		Cross over operator		Mutation operator		Penalty	
50	37	50	170	0	0	0	156	5	137	5	188	50	180
100	80	100	199	0.25	192	0.25	169	10	193	10	162	100	162
250	229	250	191	0.6	126	0.6	135	15	136	15	172	150	188
500	304	500	160	0.75	169	0.75	197	30	159	30	118	200	178
1000	413	1000	48	1	178	1	34	40	177	40	18	250	199

Figure 8 effect of parameters on constrained RGA

Section 3.1.8 - Parameters, operators, techniques, and variants have the highest impact on the efficiency, and which have the lowest.

From the effective combination of

Section 3.2 - UnConstrained Particle Swarm Optimization Algorithm (uPSO)
Rosenbrock function is one of the several optimization benchmarks used in optimization problems. It's particularly useful in assessing how well an algorithm can escape local minima and how effectively it can converge to a global minimum in a complex landscape. It is usually symbolized as a parabolic convex graph.

The Particle Swarm Optimization (PSO) Algorithm Code:
The PSO algorithm shown in the code was used to evaluate the global best, best cost function and convergence of the algorithm, against several parameters as shown below Other parameters will be introduced later in this report.

Parameters	Description	Value
numberVariables	Number of decision variables in the problem	2
maximumIteration	Maximum number of iterations for the algorithm	100
populationSize	This is the size of the population in the swarm	50
inertiaCoefficient	The inertia coefficient is a factor that influences the velocity of each particl swarm.	1
wdamp	Parameter that reduces the coefficient of the inertia as it goes through the iteration.	1
tol	determines the acceptable level of accuracy for the solution.	10^-2
velocityControl	controls how drastically particles can change their position in each iteration	0.2

Results of the PSO implementation

Benchmark A

nVar	maxIt	popSize	inCoff	tol	Resul

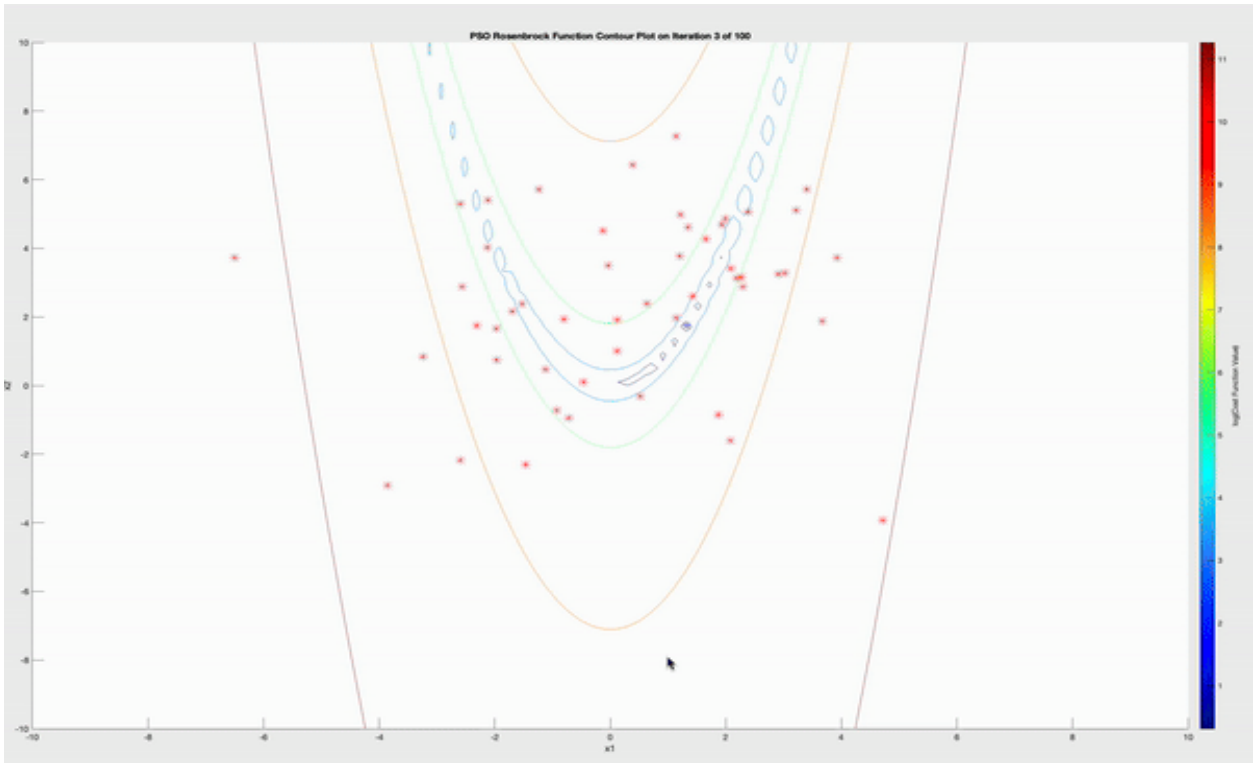


Fig 10d.1 Median of the best combination of optimal parameters

10.3.1 Results for Differential Evolution

10.3.1.1 Outline for Differential Evolution Method Report

In this section, I will mostly describe the effect of parameters on the Differential Evolution Algorithm and do a performance analysis of the effect of the parameters of Differential Evolution in an unconstrained space and within a constrained space. We will also be discussing parameters with the highest impact on the Algorithm while trying to solve the non-linear stochastic problem DE.

Developing an ideal optimization function can be challenging and occasionally unfeasible. Therefore, we improve the optimization function by imposing additional constraints. Optimizing this constrained setup provides an approximation close to the problem's actual solution while remaining feasible. Methods like penalty methods, projected gradient descent, and interior points are used to solve constrained optimization problems. In this context, I will employ the static penalty method, which involves adding a static penalty to vectors that do not conform to the feasible region of the constrained problem.

10.3.1.2 Generic Structure of the Algorithm

Step 1: Initialize the population: A random population of vectors

Step 2: Evolution Stage.

For each generation
We loop through all the population of vectors,
Mutate the vector using any of the variants eg. Rand/1/bin, Best/2/bin, etc.
Next is the crossover stage,
Selection Phase and then
Termination of the optimum criteria are met or the end of the generation

10.3.1.3 Variants

There are several variants used in the Differential Evolution Algorithm. They are classified as using (DE/x/y/z) strategy or pattern, the X signifies the selection criteria of the mutant vector, it can be random, or the best, and Y is the number of mutant vectors used in this scenario. Z is referred to as the cross-over method, this can be binomial, exponential, and so on.
Examples of variants include, these will be implemented and we will discuss the effect of these variants on constrained and unconstrained Optimization problems.

DE/Rand/1/bin which is our default mutation technique because of how often it is used (Centeno-Telleria *et al*., 2021)**,** means 1, Random Vector with the binary crossover technique.

$Gi = XGr1 + F * (XGr4 - XGr5)$

DE/Rand/2/bin 2 random mutation vectors, and the binary crossover operator.
$VGi = XGr1 + F \cdot (XGr2 - XGr3) + F \cdot (XGr4 - XGr5)$

DE/Best/2/bin Same as above but we will select the single best two vectors from the current population as a basis for mutation
$VGi = XGbest1 + XGbest + F \cdot (XGr1 - XGr2) + F \cdot (XGr3 - XGr4)$

The mutation co-efficient, F, influences the size of the search steps in the optimization process, with values usually ranging from 0 to 2 Typically, smaller F values are advantageous when the population is near the optimal value, whereas larger values are beneficial when the population is far from this value (Centeno-Telleria et al., 2021). We will be conducting experiments to determine the effect of these variants on the benchmark test function and also the effect of the different parameters.,

10.3.1.4 Implementation of the DE Algorithm

Unconstrained Problem: I choose to use the Rosenbrock function to test the performance of this algorithm,

Constrained Problem: I will be using the Himmelblau function with the static penalty technique for the penalty function.

10.3.2 Implementation Details.

Evaluating the performance of this algorithm will be done in a MATLAB environment, the benchmark needed to reach the minimum point will be the median of 10 consecutive runs this is because of the stochastic nature of the Genetic Algorithm problem and the performance metrics we would use are, the convergence rate on the contour plot and the min number of iteration need to reach convergence.

10.3.2.1 Code Enhancement:

The code was improved and helper functions like savePlot for saving images, BenchMark function for taking benchmarks, and the PlotParametersFunction, which plots the graph for the toning of the parameter.

10.3.3 Results and Analysis.

10.3.3.1 Line Charts The results of **Fig. 10.3.0** show a chart that illustrates the application of the Rosenbrock test function, on the differential evolution algorithm, the x-axis shows the number of iterations and the y-axis shows the “Best Cost”. From the graph, we can see a sharp decline in the best cost after about 5 iterations. On the chart, we can also note the “First Minimum Convergence” Label, this point, marks the best iteration before the minimum convergence was reached for that number of generations. This implies that the algorithm has reached a near-optimal solution by the 45th iteration. We can also see that, there are no subsequent improvements past the convergence point, this may imply that the algorithm might have become trapped in a local minimum or its possible to explore, with a greater generation or space for exploration.

10.3.3.2 Median

The Median plot shows a general, rapid initial convergence, and a sharp decrease in the best cost within the best few iterations, this shows that the algorithm is proficient, and in the presence of several different random populations, the plot still performs the median of this execution is shown on the Line plot in **Fig. 10.3.1**

10.3.3.3 Contour Plot

The figure in **Fig. 10.3.2** Contour plot shows all the iterations from the 1st generation down to the 100th, in the first generation we can observe the scattered positions of the candidates on the Rosenbrock plot which is a bowl-shaped line with the middle where the global optimum is located, the candidates closer to the middle valley has a lower fitness value, After each successive iteration, we can observe the migration of the particles towards the global minimum showing that the DE algorithm is perturbing vectors and converging them towards the global minimum of the Plot. At the end of the 100th iteration, it can be observed that even if most particles migrated to the global optimum point, even if they did not later converge to a single point this shows there is still room for optimization.

10.3.4 Performance Toning

To find the optimal parameters to get the optimal value for Rosenbrock and also get a faster convergence rate we will be varying the values of the default parameters to see the effect of each value, choosing values for parameters can be difficult, I decided to go with 6 different values, 3 higher and 3 lesser values compared to the default parameters. The results of the tuned values will be shown below. I got the contour plot and the line charts to carefully display the effect of each parameter on the min iteration count and the convergence on the contour plot.

Maximum Iteration	100	200	300	400	500	600
	48	27	4	43	36	42
Population	50	100	150	500	1000	1300
	31	8	14	36	27	10
Beta_min	0	0.2	0.4	0.6	0.8	1
	19	21	54	55	36	17
Beta_max	0	0.2	0.4	0.6	0.8	1
	42	Inf	22	36	46	16
Crossover Probability	0	0.2	0.4	0.6	0.8	1
	44	64	35	42	23	22

Table 10.3.1 Results showing the effect of parameters on the DE/Rand/1/bin unconstrained Rosenbrock

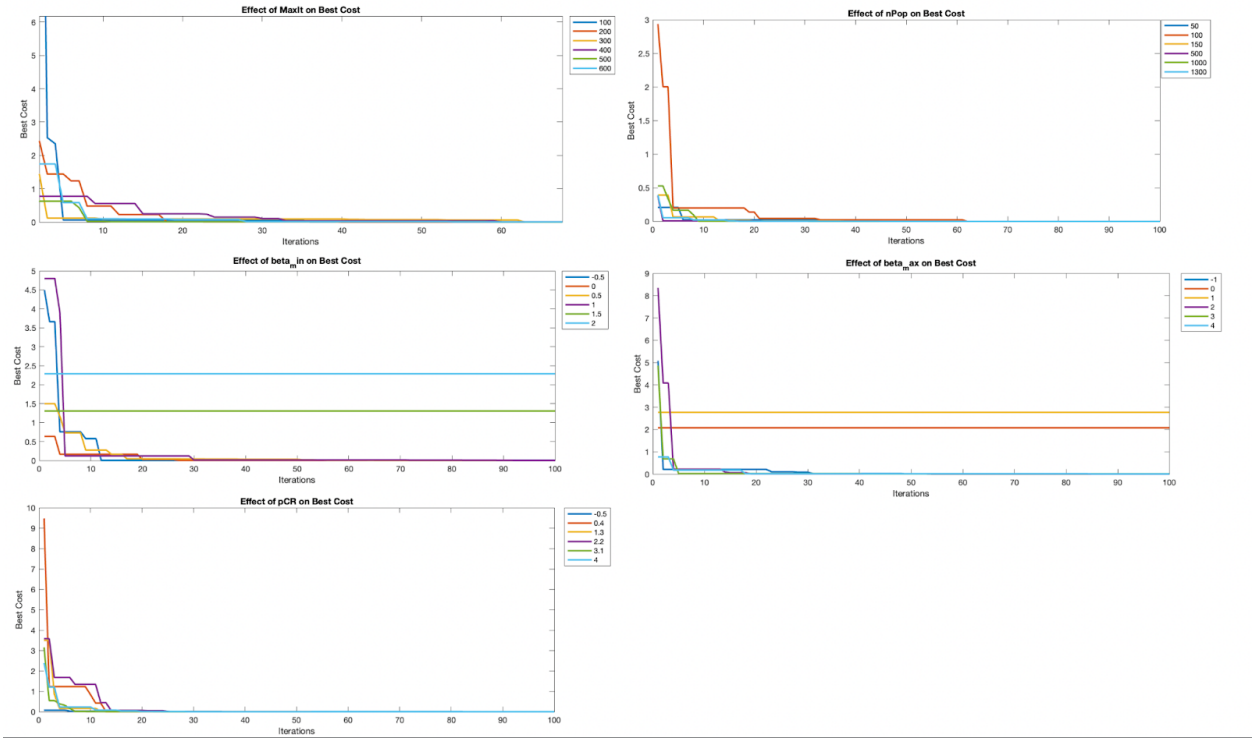


Fig.10.4.1 Effect of parameters on DE/Rand/1/bin

The figures in **10.4** show the contour plot and **fig. 10.4.1** above shows the line plot of the effect of parameters on the Algorithm, the line plot shows us a quantitative view and the result shown in the table in **10.3.1** while the

contour plot shows the reaction of the particles to changes in the parameters while migrating to the convergence point.

10.3.4.1 The Effect of Maximum Iteration Count / Number of Generation on DE/Rand/1/bin

From the results we saw a downward trend when increasing the maxIt value initially leads to faster convergence, we got the best value at max Iteration of 300, the algorithm reaches the first min threshold in just 4 iterations, which is the fastest among the displayed results.

Another obvious point noted is an increase in the number of iterations needed to reach the first convergence point, this implies that maxIt does not necessarily lead to faster convergence or improved best cost. This is evidenced by the increase in min convergence point from 4 to 43 and then fluctuating in the 40s as maxIt is increased from 300 to 600, this could be as a result of many reasons which include, the stochastic nature of the genetic algorithm, as the random population was used when conducting this experiment.,

From our results the algorithm performs best at a max It of 300, reaching the desired tolerance level with the fewest iterations. This suggests a good point to pick an optimal setting for this specific problem and algorithm configuration.

It is also possible that we can get better results over time if the algorithm runs for more iterations, but a larger max Iteration count implies more computational resources are used. If we cannot find a high computational decrease in the min convergence point it is not necessary to increase the maximum number of iterations as we don't see a continuous decrease.

Details of the contour plot can be found in **section 10.4** of the appendix, this plot shows the effect of this parameter on the overall exploitation and exploration of the Differential Evolution Algorithm, increasing the space and time has a great influence on the convergence point, an increase follows this in runtime.

10.3.4.2 The Effect of Population on DE/Rand/1/bin

I reduced the population size from the optimal value and noticed that after all the values we checked for optimal value, with the smallest population size, the algorithm took 31 iterations to reach the tolerance level, this can imply that fewer candidate solutions can slow the convergence, potentially due to limited diversity in the population, which might result in reduced exploration capabilities, increasing the Population (nPop = 100), we noticed that the performance improved significantly we needed only 8 iterations to reach the tolerance level. This suggests that a moderate increase in population size can enhance the exploration and exploitation balance, leading to faster convergence., and exploring other values like an Increase of the population to 150 resulted in more iterations (14) to achieve TolMin compared to a population of 100.

From the Results on the contour plot as shown in **section 10.4** we can deduce that increasing the population means an increase in the number of iterations needed to check for the global best. Increasing the numbers in cases like **1300** and **1000** did not necessarily mean we could find better solutions but instead introduced an excess unneeded population. Another evidence of this deduction can be seen on the contour plot we have almost similar convergence on the same spot even with an increase in population.

10.3.4.3 The Effect of beta_max and min on DE/Rand/1/bin

From the results on the table in **Table 10.3.1** at beta_min 1, TolMin drops to 17 iterations, which is the lowest among the provided values. While for For beta_max when it is set to 1, the TolMin decreases significantly to 16 iterations, which is the best performance for the beta_max settings provided.

These observations between the range of values checked for beta_min and beta_max indicate that the choice of beta_min and beta_max can affect the result of the DE algorithm as shown on the contour plot. The value of

these parameters at 1 yields the best performance as they require a few iterations to reach the tolerance threshold. This confirms that there is an optimal range for these parameters that allows the DE algorithm to effectively balance exploration and exploitation of the search space.

Also looking at the contour plot in **section (10.4)** for the values of beta_max, 0 and -1, there was no convergence of the individual population, it is also obvious that when the beta_max is reduced to 0.2, it shows that higher values may allow for too much diversity, leading to inefficiency in convergence as the search becomes more random.

However, I can tell a possible flaw in this procedure because not adjusting the beta max and beta_min simultaneously thus ensuring that beta max is always greater than beta min could affect the validity of the optimal solution. As in scenarios where that occurred, we saw that the particle did not move to convergence this is visible on the contour plot in **section(10.4).**

10.3.4.4 The Effect of PcR on DE/Rand/1/bin

The values of cross-over probability usually range from 0 - 1, 0 being that crossover does not occur and 1 that crossover occurs, the result shows a decrease in the tolerance minimum value when the crossover coefficient, was increased by a tiny bit when PcR is 1 we got the best convergence point,

When seen on the contour plot, it can be confirmed that we got the best convergence when the crossover probability was 1.

10.3.5 Optimal Parameter Values from DE/Rand/1/bin

Selecting Optimal values from the parameter tuning, we used both the quantitative results from the line chart and also the level of convergence of the Rosenbrock contour plot pair from the reasons clearly stated above during the tuning I decided to choose these values as the best combination for finding optimal results

Max. Iteration: 300, Population 100, beta_min 1, beta max 1, Probability of crossover 1. The median results of the optimal parameter values for DE/Rand/1/bin are shown in the plots below

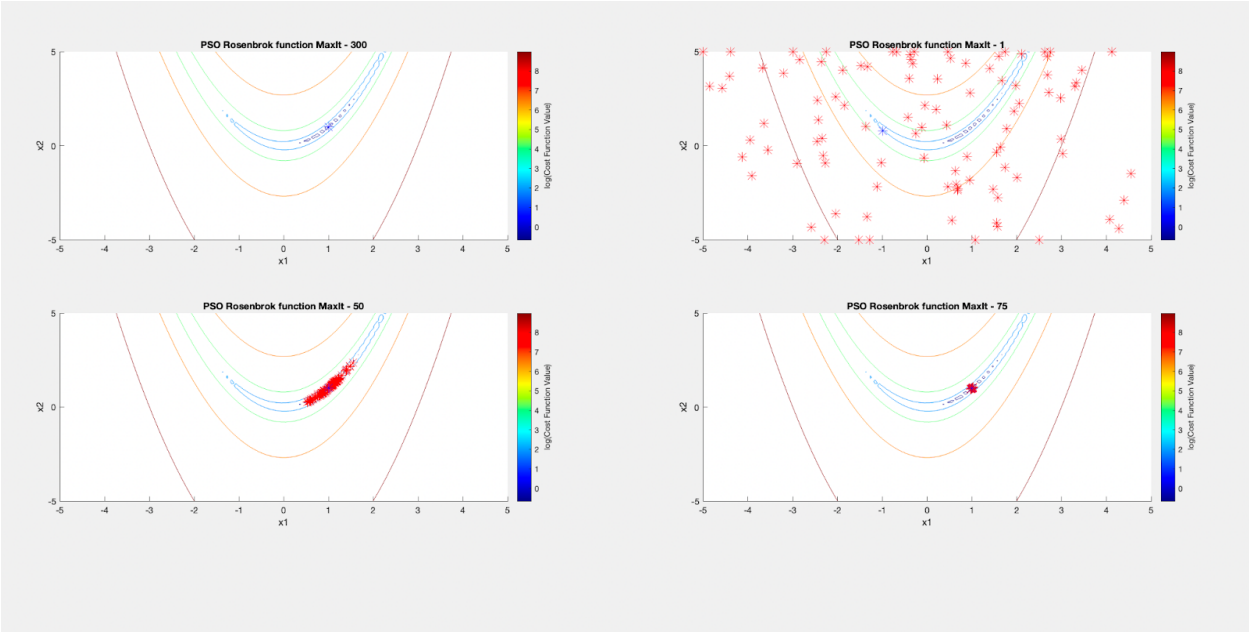


Fig. *showing the contour plot of the optimal parameter of choices of DE/Rand/1/bin.*

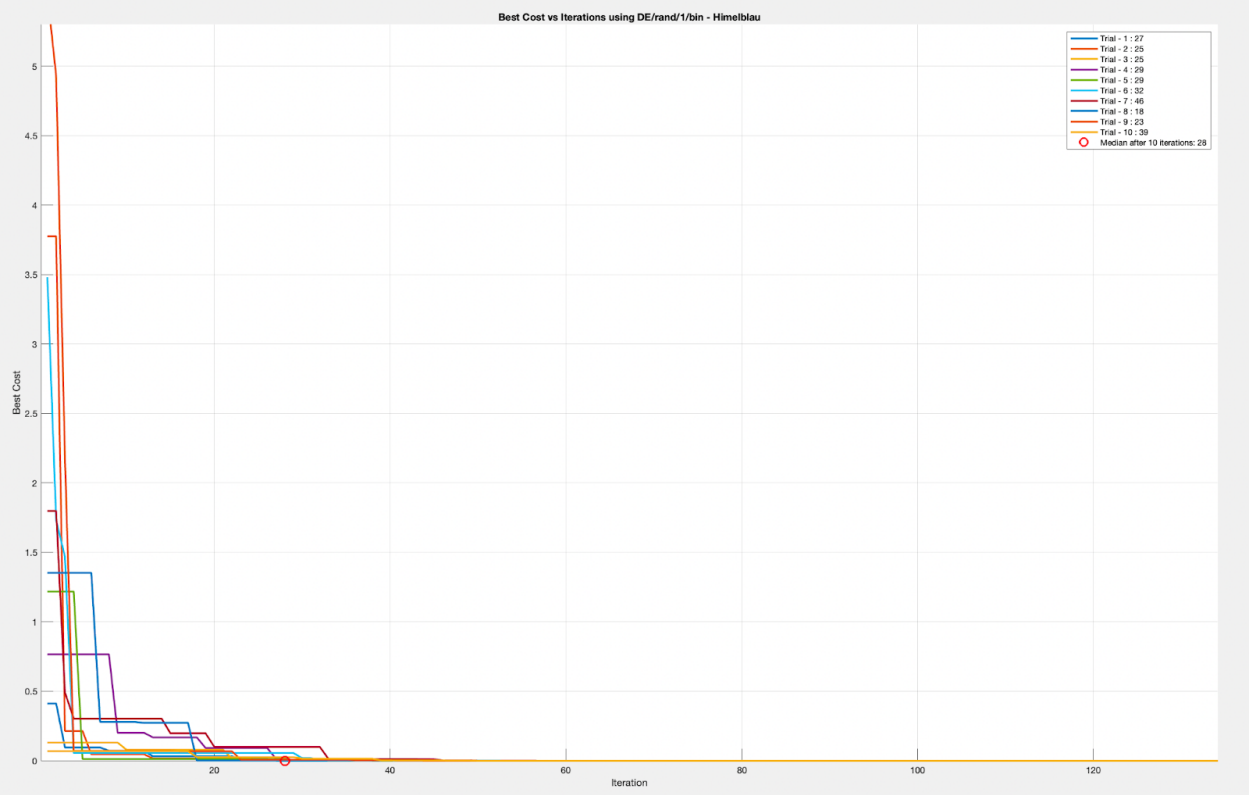
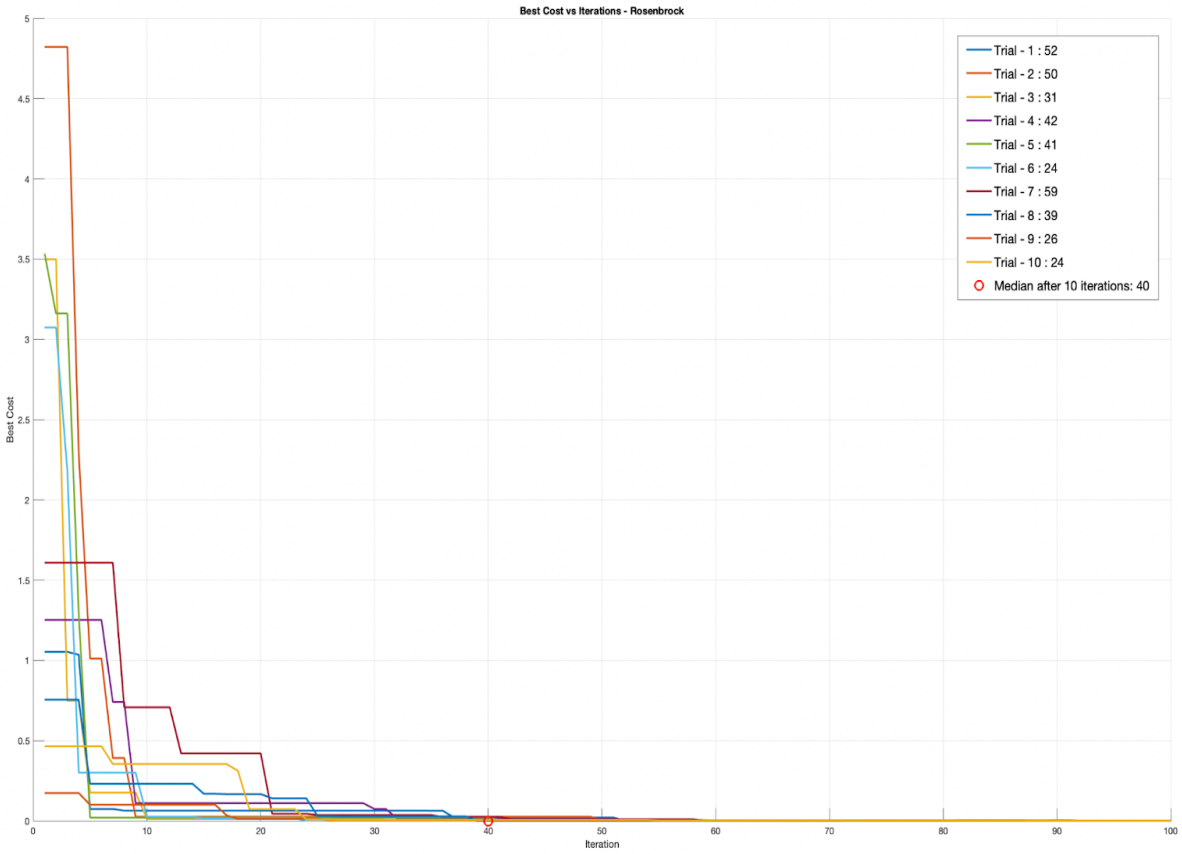


Fig. *showing the contour plot of the optimal parameter of choices of DE/Rand/1/bin*

10.3.5 Variant I - DE/Best/2/bin

As explained earlier this is a type of evolution strategy that involves taking the best individual in the population for perturbing, the best is then used to create the resultant vector, and then the binary crossover method.

10.3.5.1 Benchmark of DE/Best/2/bin



The plot from the results shows convergence to a minimum after about 20 - 30 iterations, the values for the iteration range as low as 26 and as high as 59. This shows that the algorithm is efficient and works as expected.

The contour fig. shown in **Fig. 10.3.5** shows the movement of particles from the first generation to the last generation., there was a convergence, towards the slope, of the global best of the Rosenbrock function.

Fig. *shows the benchmark of DE/best/2/bin with a median iteration count of 40.*

10.3.5.2 Effect of Parameters

I conducted a series of parameter tuning, using values very similar to the one on the **Rand/1/bin**

Maximum Iteration	100	200	300	400	500	600
	32	23	46	12	11	21
Population	50	100	150	500	1000	1300
	4	28	16	21	17	4

Beta_min	0	0.2	0.4	0.6	0.8	1
	10	17	7	24	41	64
Beta_max	0	0.2	0.4	0.6	0.8	1
	8	Inf	14	14	14	24
Crossover Probability	0	0.2	0.4	0.6	0.8	1
	33	57	16	17	10	11

The results here show the number of iterations needed to reach a tolerance minimum, for maximum iteration, we took consecutive values of 100 - 600, it was observed that 500 had the lowest value with 11 as the minimum iteration count needed to attain convergence, from the contour plot in **fig 10.3.5**. We can also observe that we begin to see a total convergence of all particles to the global minimal point on the Rosenbrock plot.

The effect of population as a parameter on the Algorithm shows a decline in the number of iterations needed to meet the

Table 10.3.1 Results showing the effect of parameters on the DE/Best/2/bin unconstrained Rosenbrock

min tolerance value it is clear that for a population count of **50 - 1300**, tol min was low, this shows a swift convergence to local optima, this can be a result of premature convergence without properly exploring the required space. For mid-range values from **100 - 1000**, we show a slower significant difference in the number of generations needed to meet the tolerance value. 150 can be chosen as the optimal value as the number of iterations required is the difference between 150 and 1000 since 1000 requires more resources in time and computing.

Effect of beta_min and beta_max, These values promote significant diversity within the population, For beta_min, the results from the table showed an increase in the successive number of iterations as the value increases from 0 to 1. When the value was 0.4, we saw a rapid convergence, suggesting that smaller scaling factors can effectively fine-tune solutions, I will be taking this value as my optimal point as we saw a great drop for the min convergence point. For beta max, the values from the table show a constant value of 14 for the beta_min values from 0.4 - 0.8, there is also an observed increase when the value was 1.
From the contour plot, in **Fig. 10.3.5** it can also be deduced that we had no convergence when the value was 0 but a perfect and steep convergence in values **0.4, 0.6, 0.8,** I will be using 0.8 as my optimal value.

Effect of cross-over probability, from the result, we see a constant decline in the iteration count needed to achieve min tolerance, when the value was 0 we got 33 this could be because the algorithm was not able to explore

10.3.5.3 Optimal Parameter Combination from DE/Best/2/bin

Max Iteration of 400, Population count of 150, beta_min 0.4, beta_max 0.6, cross-over probability of 0.8.

Results:

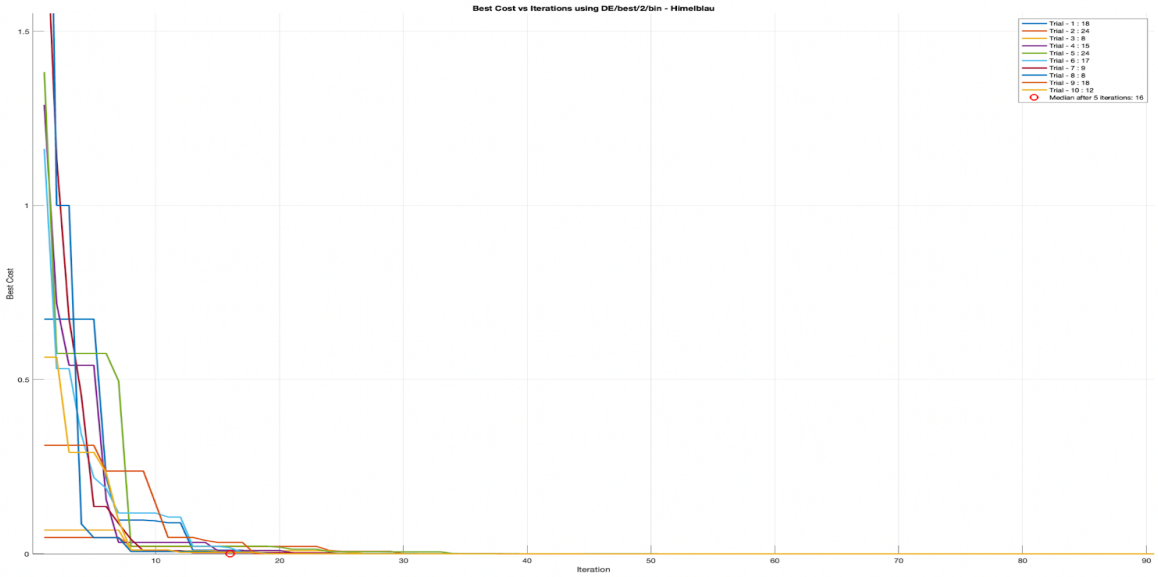
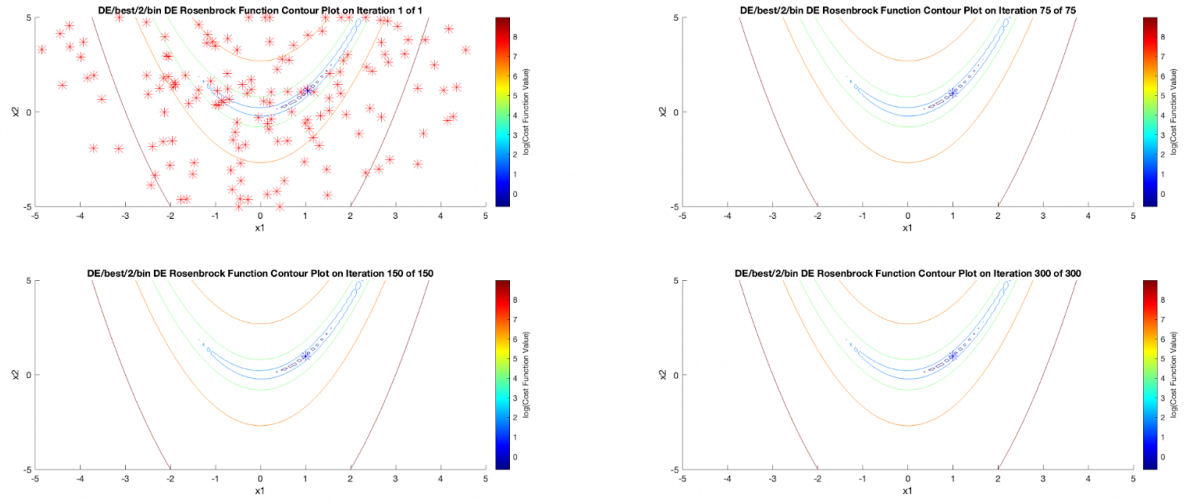


Table 10.3.1 Results showing the effect of parameters on the DE/Best/2/bin unconstrained Rosenbrock Function

The **fig** shows 10 trials for the Rosenbrock function of the DE/best/2/bin of, each line represents a trial, median for the 10 iterations, the minimum number of generations for each trial ranged from 8 - 24 showing a rapid convergence, this shows that the algorithm can find the areas of the search space with the global minimum values, after the initial steep drop we can observe a flattening curve, the nature of this algorithm uses the best individual to generate the trial vector and employing the binary crossover technique, This strategy appears effective for the Himmelblau function, given the rapid initial decline in best cost across trials. After 10 trials we obtained a median of 16.

The contour plot below shows the contour plot of the Rosenbrock function, here we see a convergence by the 75th iteration, as this is evident on the line chart in **Fig 10.3.5**



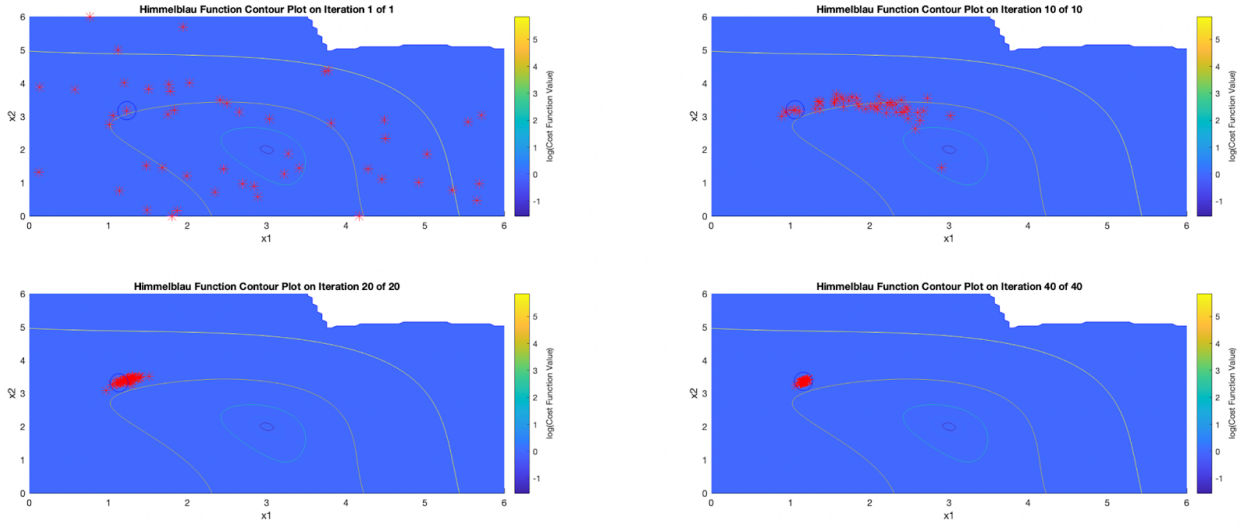
10.3.6 Constraint Handling using Himmeblau Function.

I will be using the static penalty method to handle this constrained optimization problem, this also uses the inequality conditions.

This Constrained problem was solved under the constraints of
(x 1-5)2 +i < 26 ,
4x1 +x22 < 20 ,
X1, X2 >= 0

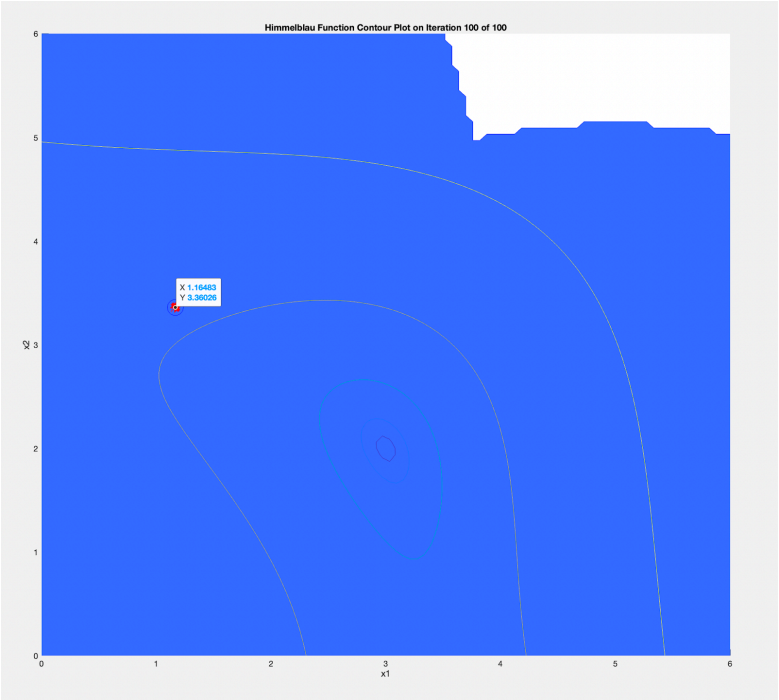
In addition to the optional parameters we will be introducing the penalty coefficient R = 10 the default as taken from the slide, the first constraint (C1) makes a circular plot that marks the feasible region, while the blue background color marks the infeasible region.

The plot shows the global best on the himmelblau function Plot as a dark blue circle, it also show the optimal point was in the infeasible region, the penalty coefficient is responsible for assigning a penalty value to every individual with a fitness value that falls in the infeasible region.



Using the default parameters as given on the slide we have a default plot of the constrained D.E (Differential Evolution) the default parameters can be found in the table below we will be using the **Rand/1/bin** variant and then find the optimal parameter using this variant.

When inspecting the results presented below, it becomes clear that the optimal point remains within the infeasible region, it is detected that the optimal value attained was a minimum value of 1 (this was used as the default values when we have no iteration point where fitness value meets the tolerance value). This signifies that although convergence occurred, it did not happen at the feasible region.



Plot showing the default values for 100 iterations.

10.3.6. Finding Optimal Parameters for DE/Rand/1/bin :

In the table below we will be deducing patterns from the different parameters and choosing the optimal value, the values show the first minimum convergence generation, the tables show in **Table 10.5.1** shows all the effect of different parameters on the constrained problem, it was observed that there was no change in the best cost fitness value and and the min generation count before tolerance value was seen to be infinite except the penalty parameter,. This exciting observation is the effect of static penalty on the Line chart and the contour plot of Himmelblau.

Static Penalty	Min Tolerance Value	Best Cost
10	Inf	189.0495
50	8	496.4747
100	32	516.6382
200	26	516.6382
250	35	516.638

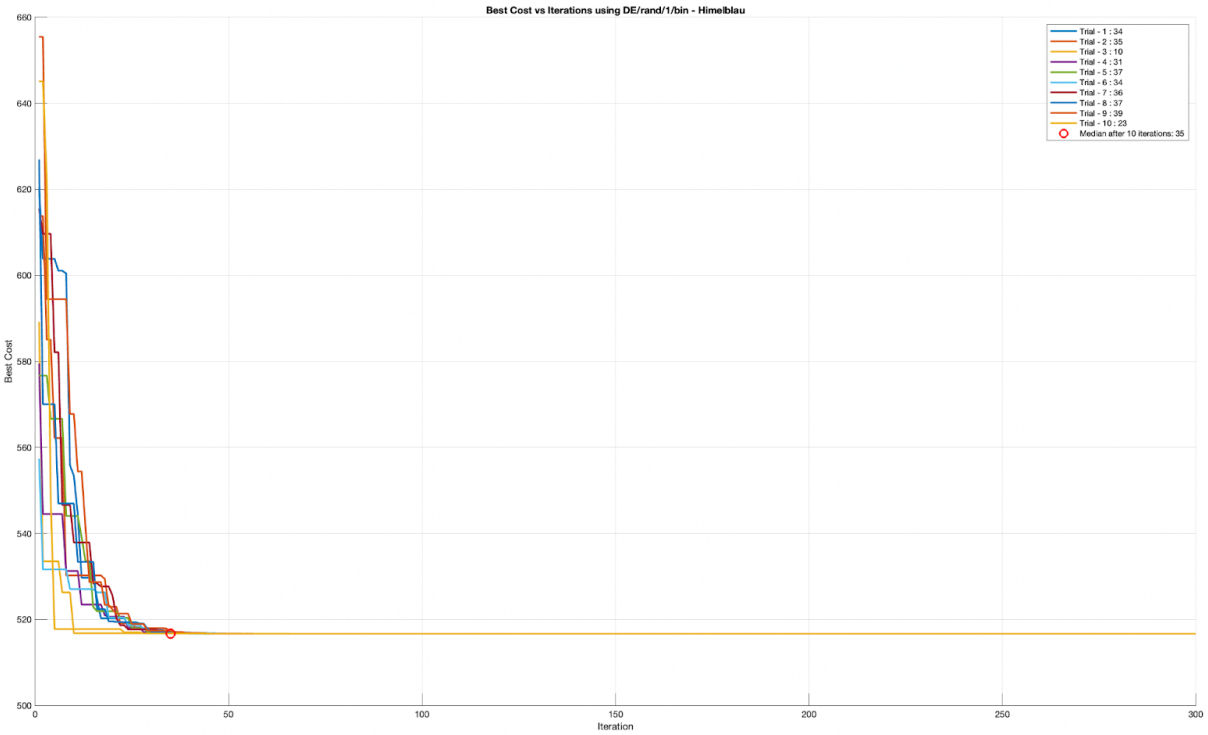
The Effect of this value can be seen on the contour plot in **Fig. 10.5.5** The 5 plot shows the value of static penalty for 10, 50, 100 , 200 and 250. When the penalty value was 10, the DE algorithm may not have been able

to penalize constraint violations, this will result in a final population that includes many infeasible solutions converging at the infeasible point since it was unable to move the population to the infeasible region

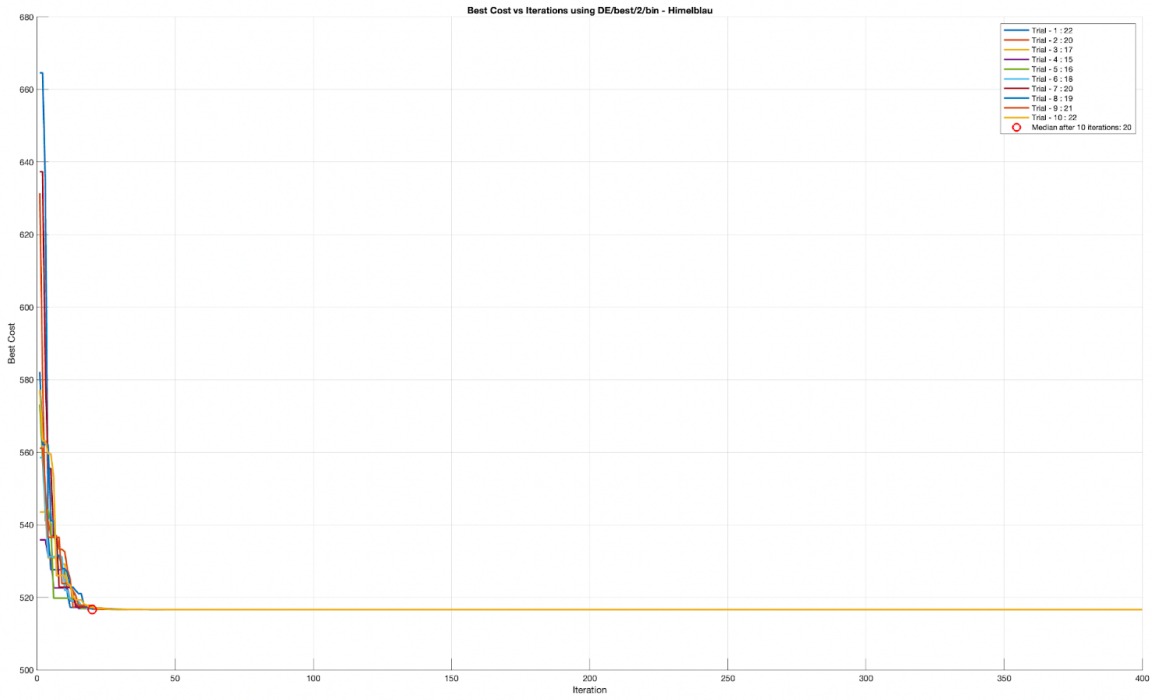
Table *10.5.5 showing effect of penalty on Himmelblau contour plot.*

As the penalty value increases (10 - 50), the algorithm moved the population from the infeasible regions, converging it towards feasible areas of the search space. Increasing the penalty of this values further pushed the global best the himmelblau function to optimum position this is seen in the contour plot of 100 - 250.

10.3.6. BechMark for constrained Himmeblau for DE/Rand/1/bin



Since we have optimal values from the constrained problem i will be using those optimal parameters and using a penalty coefficient of 100 to get my optimal solution. The result shows the median benchmark of 35. Convergence on the contour plot during successive generations can be found in **fig10.5.**



10.3.6. BechMark constrained Himmeblau for DE/Best/2/bin

The result shows the median benchmark of 20. Convergence on the contour plot during successive generations can be found in **fig10.5.2.**

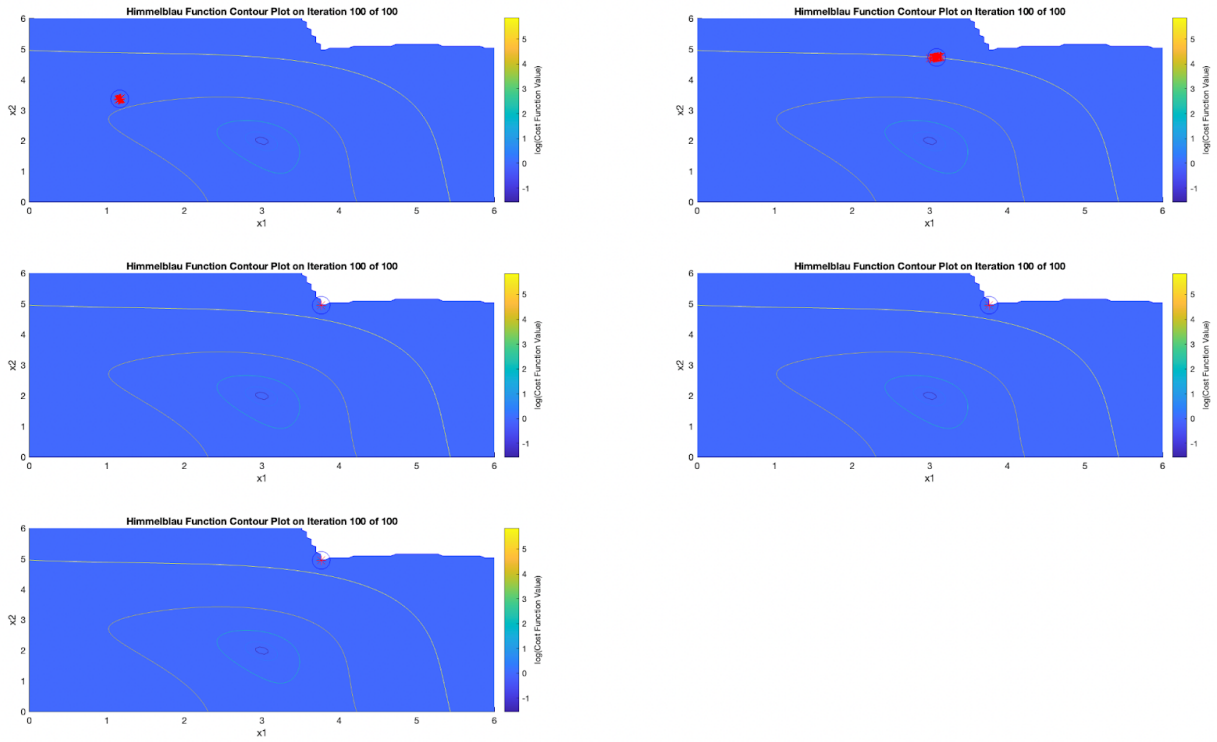


Fig. *10.5.5 showing effect of penalty on Himmelblau contour plot.*

10.3.6. Comparative Analysis

Tolerance Sensitivity Analysis of how changing the error tolerance affects DE performance.

Discussion & Conclusions

The experiments conducted above prove that the DE algorithm is efficient, in the constrianed and unconstrianed problem and improving the parameters can lead to better reaction of the algorithm and faster convergence to the global minimum, different parameters have varying effect on convergence point of the algorithm, but most times at the expense of higher computation.

Proceeding to do a comparative analysis of this experiment,

Test Function	Constrianed - Rosenbrock		Unconstrained – Himmelblau	
Evolution Variant	Rand/1/bin	Best/2/bin	Rand/1/bin	Best/2/bin
Median	The default median with the default parameters from the slide gave a TolMin of 44 afterter 10 trials the max was 100 and the minimum of 24. The tuned results how ever gave a median of 23 after 10 successive trials.	the default median, for the the default parameter as shown on the slide for this variant is 40, it shows to have lower values as compared to the Rand/1/bin algorithm, the trials show a max of 59 which was way lower than that of the Rand/1/bin and a minimum value of 24. the effect of the optimal selection of parameter on the plot shows a median of 16.	Median after introducing optimal parameter was 35 after 10 successive trials, the min being 10 and the max 39.	Median using the optimnal parameter value that reaches convergence in the optimnal point is 20 after 10 succesive trials. The min here was 15 and the max 22, showing a closer range of values.
Convergence	The results from the mean shows that the DE evolution strategy of the Rand/1/bin is an efficient, algorithm as it quickly finds a convergence average in the 44th generation out of 300 generations, this shows that the Rand/1/bin Evolution strategy is effective in finding the convergence.	Compared the charts on the Rand/1/bin, the Best/2/bin has a better median value , this can be because probably the best two vectors were selected to form the trail vector as such we where able to locate the local best faster then leading to a steeper convergence rate	The fitness value of the convergence point on this plot is seen to be arount 516.7, this is according to the values in the slide, the plot shows a steep decline within the first 50 generations, the evidence of the high fitness value is because of the addition of the penalty coefficient to particles outside the feasible areas.	Compared the Rand/1/Bin where a steep slope was observed before the 50th generation, we can see a steeper downward slope in the best/2/bin, also proving the superiority of the Best/2/bin algorithm over Rand/1/bin as it is able to find the global best solution faster.
Line / Contour Plot	The line Plot shows that Rand/1/bin is a better agorithm, when It comes to solving an exploration problem, as we can see A steep decline in as it converges to find its global minimum but as it gets towards the 0 point we can see more exploitation as the algorithm now finetunes all the promising points or region for the global minimum piont. This is evidence on Fig.10.5 where we have a steep decline of majority of the trials In the first 30 iterations, while the others are used to find the actual minimum point. The Best/2/bin algorithm prooves to be a better exploratiive convergence rate, compared to the Rand/1/bin but lesser exploitative caabilities. Its prooves that Best/2/bin is a better varaint of the algorithm for finding better convergence.		The same observation can be extended for the costrained operation, the Contor Plot showed faster convergence as at about the 30th iteration we can see that the best/2/bin already found the global minimum on the optimal point,.	
Most Effetive Parameter	All Paramters tend to have			

Recommendation

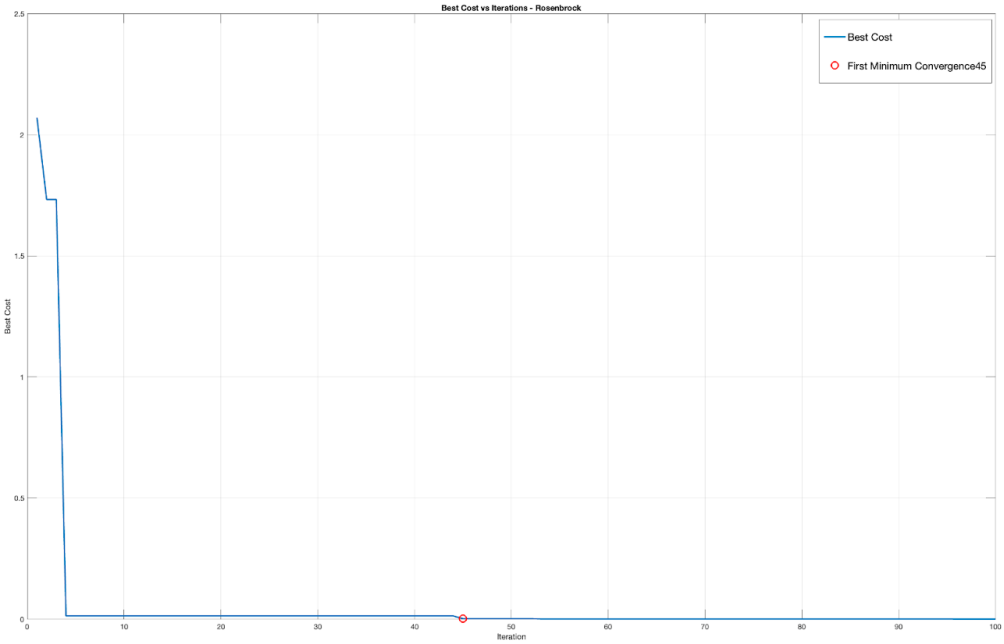


fig 10.3.1 The benchmark median plot after 10 consecutive executions.

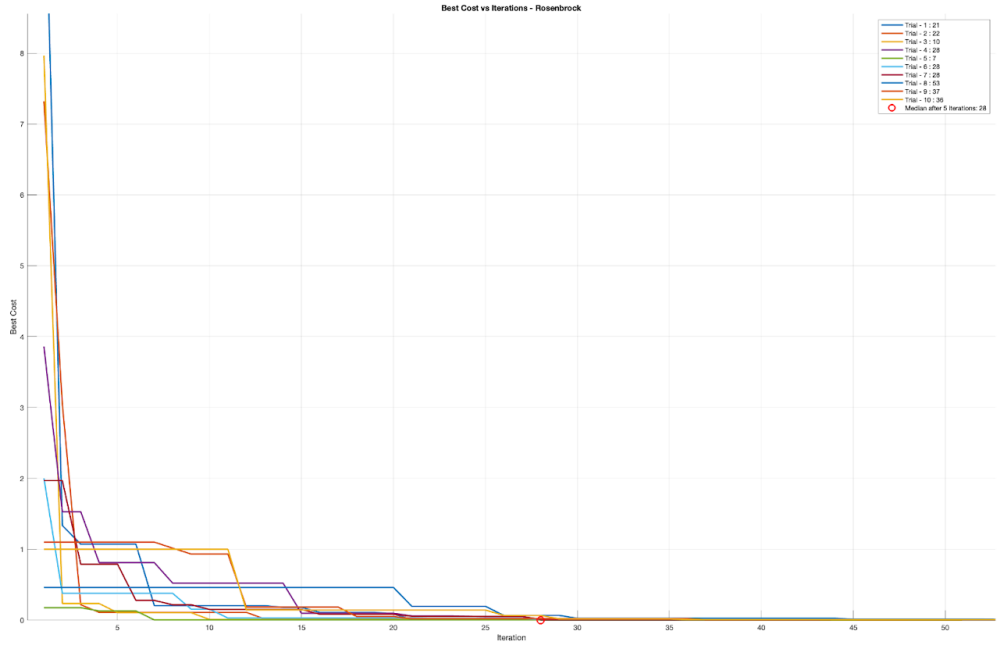


fig 10.3.1 The benchmark median plot after 10 consecutive executions.

Maximum Iteration	Value	100	200	300	400	500	600
	Best cost	189.04	189.04	189.04	189.04	189.04	189.04
	Min Tol value	Inf	Inf	Inf	Inf	Inf	Inf
Population	Value	50	100	150	500	1000	1300
	Best cost	189.05	189.05	189.05	189.05	189.05	189.05
	Min Tol value	Inf	Inf	Inf	Inf	Inf	Inf
Beta_min	Value	0	0.2	0.4	0.6	0.8	1
	Best cost	189.05	189.05	189.05	189.05	195.82	194.12
	Min Tol value	Inf	Inf	Inf	Inf	Inf	Inf
Beta_max	Value	0	0.2	0.4	0.6	0.8	1
	Best cost	192.2173	189.04	189.05	189.05	189.05	189.06
	Min Tol value	Inf	Inf	Inf	Inf	Inf	Inf
Crossover Probability	Value	0	0.2	0.4	0.6	0.8	1
	Best cost	Inf	Inf	Inf	Inf	Inf	Inf
	Min Tol value	189.0551	189.0494	189.0494	189.0494	189.0494	189.0494

Table 10.5.1 The benchmark median plot after 10 consecutive executions.

- Examination of constraint satisfaction in the final solutions
- **Comparative Analysis**
- **Method Comparison**: Compare DE's performance with other methods (Real-Coded Genetic Algorithm, Particle Swarm Optimization) on the same problems.
- **Tolerance Sensitivity**: Analysis of how changing the error tolerance affects DE performance.
- **Conclusions and Recommendations**
- **Summary of Findings**: Key insights and outcomes from the DE experiments.
- **Improvement Suggestions**: Recommendations for enhancing DE's effectiveness based on the study.
- **Future Work**: Potential areas for further research and exploration in optimization using DE.
- **Appendices**
- **A. Code Listings**: Complete DE code used in experiments.
- **B. Additional Graphs and Tables**: Supplementary visual data not included in the main report.
- **References**

References

Sherstnev, Pavel. (2024). The fittest: evolutionary machine learning in Python. ITM Web of Conferences. 59. 10.1051/itmconf/20245902020.

A.Hashim, Fatma & Houssein, Essam & Hussain, Kashif & Mabrouk, Mai & Al-Atabany, Walid. (2022). Honey Badger Algorithm: New Metaheuristic Algorithm for Solving Optimization Problems. Mathematics and Computers in Simulation. 192. 10.1016/j.matcom.2021.08.013.

Nurmuhammed, Mustafa & Akdag, Ozan & Karadag, Teoman. (2024). Modified Archimedes optimization algorithm for global optimization problems: a comparative study. Neural Computing and Applications. 1-32. 10.1007/s00521-024-09497-1.

Stańczak, Jarosław. (2024). Efficient Selection Methods in Evolutionary Algorithms. Computer Science. 25. 10.7494/csci.2024.25.1.5330.

Nguyen, Trong-The & Dao, Thi-Kien & Dao, Trinh-Dong & Nguyen, Vinh-Tiep. (2023). An Improved Honey Badger Algorithm for Coverage Optimization in Wireless Sensor Network. 網際網路技術學刊. 24. 363-377. 10.53106/160792642023032402015.

Darwin, C., 1859. On the Origin of Species. London: John Murray.

Eiben, A.E., 2015. Introduction to Evolutionary Computing. Springer.

An Introduction to Genetic Algorithms Jenna Carr May 16, 2014

An Introduction to Genetic Algorithms Mitchell Melanie A Bradford Book The MIT Press Cambridge, Massachusetts • London, England Fifth printing, 1999 First MIT Press paperback edition, 1998

Jones, Andrew. (2023). Particle Swarm Optimization. 10.13140/RG.2.2.32162.20163.

Xin-She Yang, Introduction to optimization, Xin-She Yang, Introduction to Algorithms for Data Mining and Machine Learning, Academic Press, 2019, Pages 1-18, ISBN -9780128172162 (https://www.sciencedirect.com/science/article/pii/B9780128172162000089)

Victor H. Cantú, Catherine Azzaro-Pantel, Antonin Ponsich, Constraint-handling techniques within differential evolution for solving process engineering problems, Applied Soft Computing, Volume 108, 2021, 107442, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2021.107442. (https://www.sciencedirect.com/science/article/pii/S1568494621003653)

B. Tessema and G. G. Yen, "A Self Adaptive Penalty Function Based Algorithm for Constrained Optimization," 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 2006, pp. 246-253, doi: 10.1109/CEC.2006.1688315. keywords: {Constraint optimization;Interference constraints;Genetic algorithms;Benchmark testing;Genetic engineering;Production;Costs;Search methods},

B.Y. Qu, J.J. Liang, Z.Y. Wang, Q. Chen, P.N. Suganthan, Novel benchmark functions for continuous multimodal optimization with comparative results, Swarm and Evolutionary Computation, Volume 26, 2016, Pages 23-34, ISSN 2210-6502, https://doi.org/10.1016/j.swevo.2015.07.003.

S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," in IEEE Transactions on Evolutionary Computation, vol. 15, no. 1, pp. 4-31, Feb. 2011, doi: 10.1109/TEVC.2010.2059031.

Centeno-Telleria M, Zulueta E, Fernandez-Gamiz U, Teso-Fz-Betoño D, Teso-Fz-Betoño A. Differential Evolution Optimal Parameters Tuning with Artificial Neural Network. *Mathematics*. 2021; 9(4):427. https://doi.org/10.3390/math9040427