Team #: 6

Members	Name	ID
Team Leader	بدر فيصل عبد الرؤوف عبد الحليم	20210219
2nd Member	يوسف الأمجد رأفت	20211066
3rd Member	احمد ايمن كامل مختار	20210030
4th Member	مارتينا سليمان سامي	20210704
5th Member	يوسف محمد مجدي كامل	20211102
6th Member	يوسف ياسر يوسف محمد	20211117

Github Link: **https://github.com/Badr-Faisal/EA-deliverables**

Project idea: Our project idea is An Genetic Algorithms and Differential Evolution for Function Optimization.

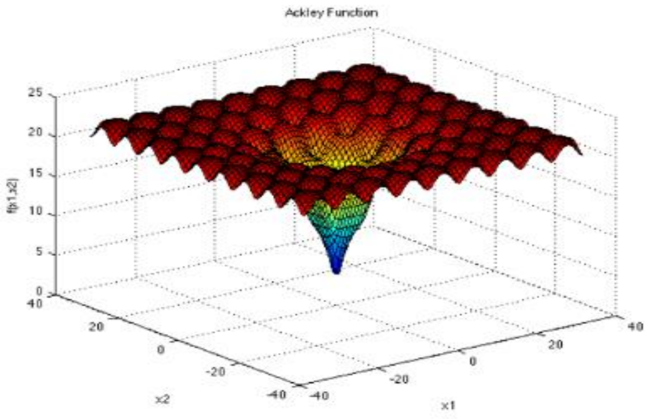
The goal: is to minimize these functions by finding the global minimum.

Overview :

- Implement (and compare) – independently – a Genetic Algorithm and a Differential Evolution Algorithm to find the global minimum of 5 benchmark optimization functions. You may select any 5 benchmark optimization functions from the following list:
  - **https://www.sfu.ca/~ssurjano/optimization.html**
- The five functions that were chosen :
  - Ackley Function
  - Bukin Function N. 6
  - Cross-in-Tray Function
  - Drop-Wave Function
  - Eggholder Function

- **ACKLEY FUNCTION DESCRIPTION**:

ACKLEY FUNCTION



$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

- Dimensions: d

- The Ackley function is widely used for testing optimization algorithms. In its two-dimensional form, as shown in the plot above, it is characterized by a nearly flat outer region, and a large hole at the center The function poses a risk for optimization algorithms, particularly hill climbing algorithms, to be trapped in one of its many local minima.

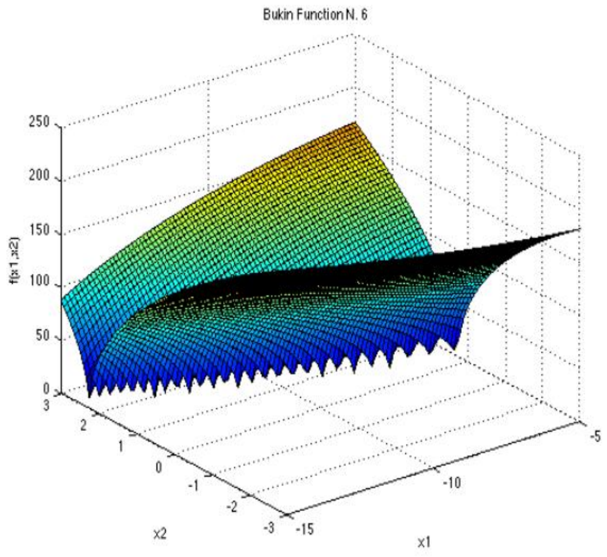
- Recommended variable values are: a = 20, b = 0.2 and c = 2π.

- Input Domain:
  The function is usually evaluated on the hypercube $x_i \in [-32.768, 32.768]$, for all i = 1, …,d although it may also be restricted to a smaller domain.

- Global Minimum:
  $$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, \ldots, 0)$$

• BUKIN FUNCTION N. 6 DESCRIPTION:

BUKIN FUNCTION N. 6

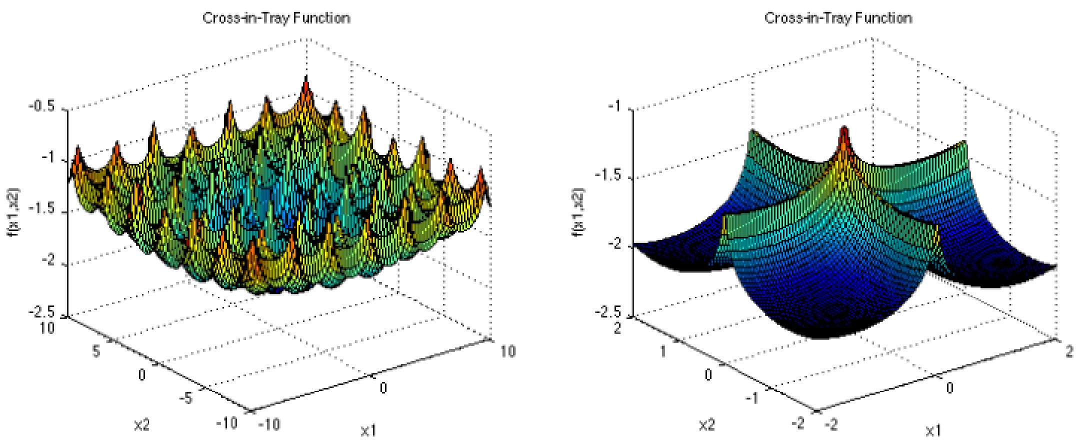


$$f(\mathbf{x}) = 100\sqrt{\left|x_2 - 0.01x_1^2\right|} + 0.01|x_1 + 10|$$

- Dimensions: 2
- The sixth Bukin function has many local minima, all of which lie in a ridge.
- Input Domain:
  The function is usually evaluated on the rectangle $x_1 \in [-15, -5]$, $x_2 \in [-3, 3]$.
- Global Minimum:
  $f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (-10, 1)$

- **CROSS-IN-TRAY FUNCTION DESCRIPTION:**

CROSS-IN-TRAY FUNCTION

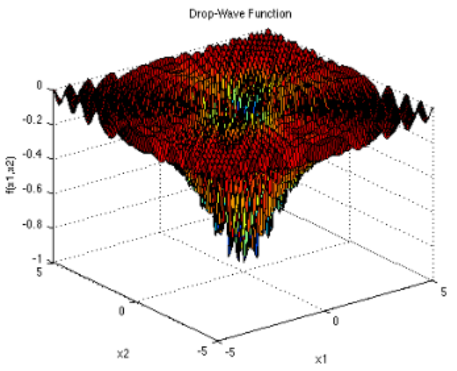


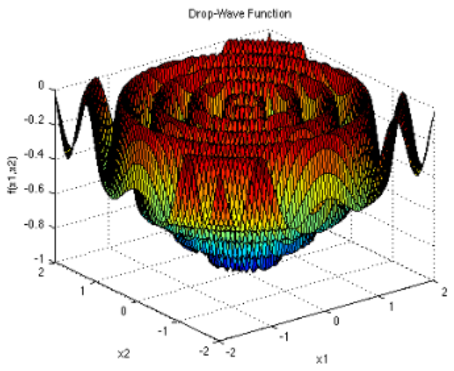
$$f(\mathbf{x}) = -0.0001\left(\left|\sin(x_1)\sin(x_2)\exp\left(\left|100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|\right)\right|+1\right)^{0.1}$$

- Dimensions: 2
- The Cross-in-Tray function has multiple global minima. It is shown here with a smaller domain in the second plot, so that its characteristic "cross" will be visible.
- Input Domain:
  The function is usually evaluated on the square $x_i \in [-10, 10]$, for all i = 1, 2.
- Global Minima:
  $f(\mathbf{x}^*) = -2.06261$, at $\mathbf{x}^* = (1.3491, -1.3491), (1.3491, 1.3491), (-1.3491, 1.3491)$ and $(-1.3491, -1.3491)$

• DROP-WAVE FUNCTION DESCRIPTION:

DROP-WAVE FUNCTION





$$f(\mathbf{x}) = -\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5(x_1^2+x_2^2)+2}$$

- Dimensions: 2

- The Drop-Wave function is multimodal and highly complex. The second plot above shows the function on a smaller input domain, to illustrate its characteristic features.

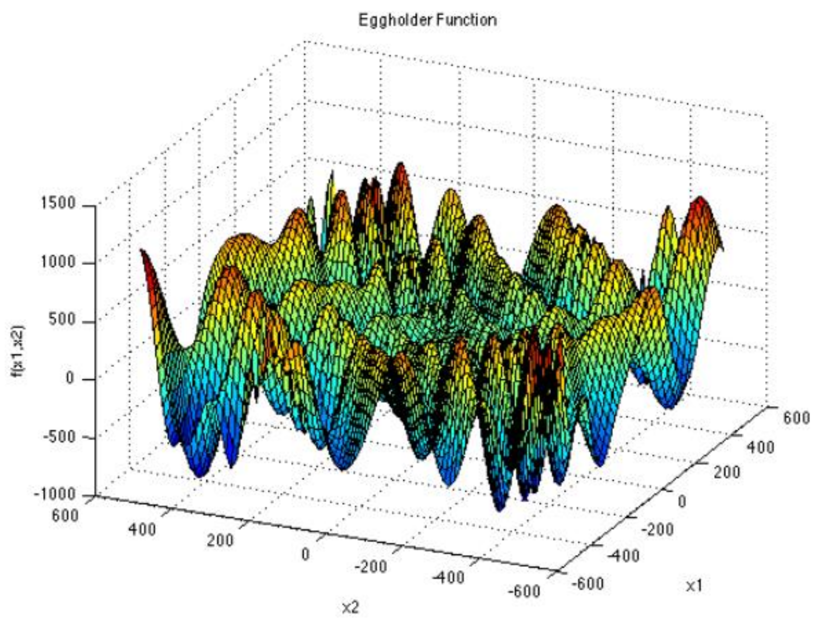
- Input Domain:
  The function is usually evaluated on the square $x_i \in [-5.12, 5.12]$, for all i = 1, 2.

- Global Minimum:

$f(\mathbf{x}^*) = -1$, at $\mathbf{x}^* = (0,0)$

• EGGHOLDER FUNCTION DESCRIPTION:

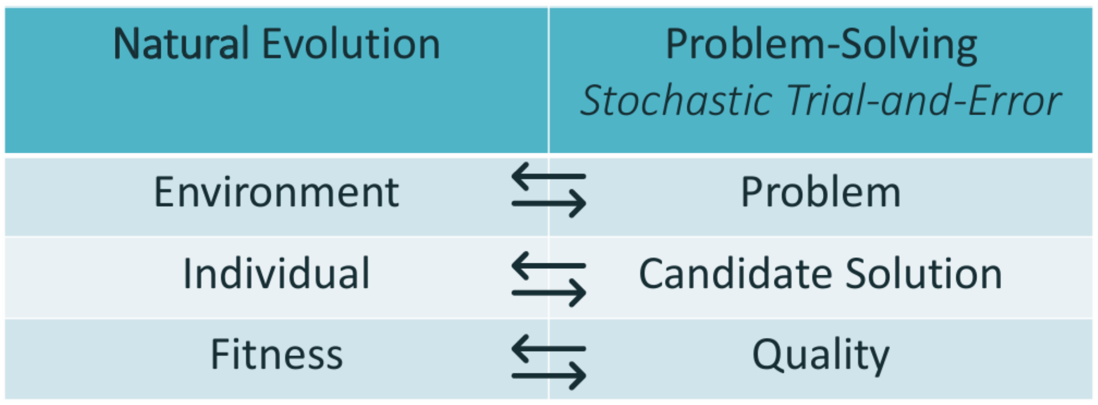
EGGHOLDER FUNCTION



$$f(\mathbf{x}) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1 \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right)$$

- Dimensions: 2
- The Eggholder function is a difficult function to optimize, because of the large number of local minima.
- Input Domain:
  The function is usually evaluated on the square $x_i \in [-512, 512]$, for all i = 1, 2.
- Global Minimum:
  $f(\mathbf{x}^*) = -959.6407, \text{ at } \mathbf{x}^* = (512, 404.2319)$

- **Overview about the Genetic algorithm** :



Natural Evolution		Problem-Solving *Stochastic Trial-and-Error*
Environment	⇆	Problem
Individual	⇆	Candidate Solution
Fitness	⇆	Quality

- In computer science and operations research, a **genetic algorithm** (**GA**) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection , Some examples of GA applications include optimizing decision trees for better performance, solving sudoku puzzles, hyperparameter optimization, causal inference, etc.
- Optimization problems

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, organisms, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits (also called bit set or bit string) Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found or the distribution of the sampling probability tuned to focus in those areas of greater interest.[5]

Selection

Main article: Selection (genetic algorithm)

During each successive generation, a portion of the existing population is selected to reproduce for a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem-dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Genetic operators

Main articles: Crossover (genetic algorithm) and Mutation (genetic algorithm)

The next step is to generate a second generation population of solutions from those selected, through a combination of genetic operators: crossover (also called recombination), and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests that more than two "parents" generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally, the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search.

Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms

It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed. An adequate population size ensures sufficient genetic diversity for the problem at hand, but can lead to a waste of computational resources if set to a value larger than required.

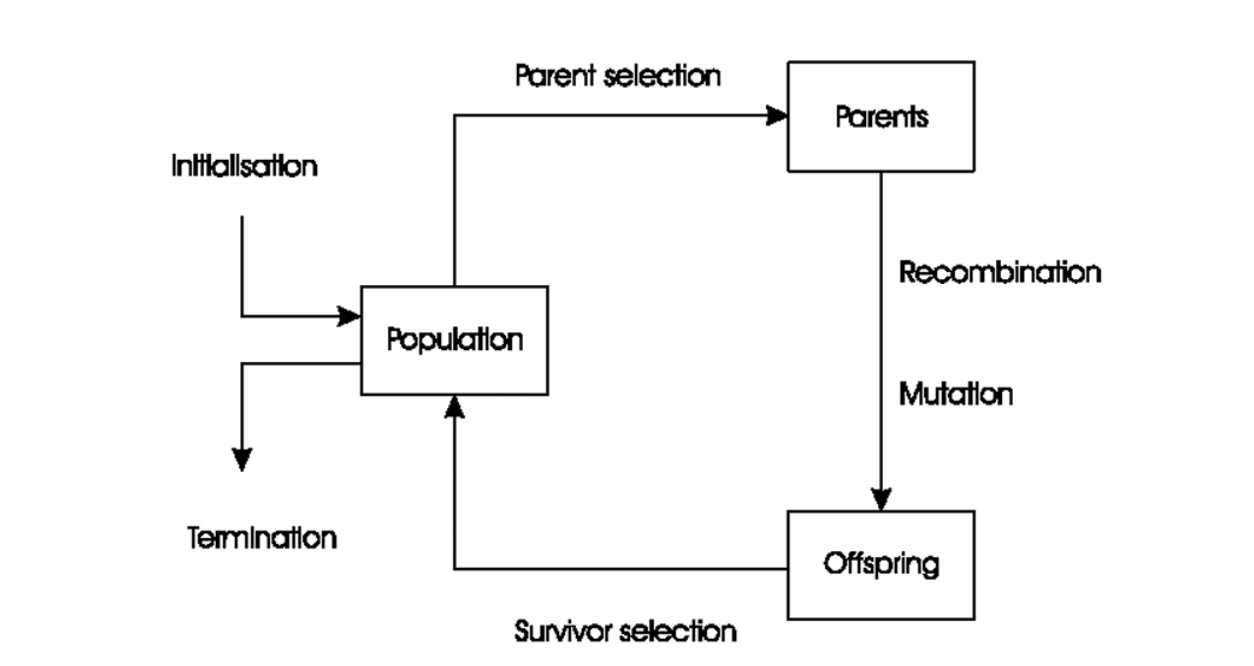
Heuristics

In addition to the main operators above, other heuristics may be employed to make the calculation faster or more robust. The speciation heuristic penalizes crossover between candidate solutions that are too similar; this encourages population diversity and helps prevent premature convergence to a less optimal solution.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Combinations of the above



This information was sourced from Wikipedia's website. The article provided valuable insights into [GA], and its content has been summarized to support our discussion.

- ## Overview about the Differential evolution:

In evolutionary computation, **differential evolution** (**DE**) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the optimized problem and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require the optimization problem to be differentiable, as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way, the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

Storn and Price introduced Differential Evolution in 1995. Books have been published on theoretical and practical aspects of using DE in parallel computing, multiobjective optimization, constrained optimization, and the books also contain surveys of application areas. Surveys on the multi-faceted research aspects of DE can be found in journal articles.

Parameter selection

The choice of DE parameters **NP**, **CR** and **F** can have a large impact on optimization performance. Selecting the DE parameters that yield good performance has therefore been the subject of much research. Rules of thumb for parameter selection were devised by Storn et al. and Liu and Lampinen. Mathematical convergence analysis regarding parameter selection was done by Zaharie

Algorithm

A basic variant of the DE algorithm works by having a population of candidate solutions (called agents). These agents are moved around in the search-space by using simple mathematical formulae to combine the positions of existing agents from the population. If the new position of an agent is an improvement then it is accepted and forms part of the population, otherwise the new position is simply discarded. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

- Choose the parameters NP >= 4 , CR belongs to [0,1], and F belongs to[0,2]

    - NP is the population size, i.e. the number of candidate agents or "parents"; a typical setting is 10n.
    - The parameter CR belongs to [0,1] is called the crossover probability and the parameter F belongs to [0,2] is called the differential weight. Typical settings are

      CR = 0.9 and **F = 0.8**

    - Optimization performance may be greatly impacted by these choices; see below

- Initialize all agents x with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:

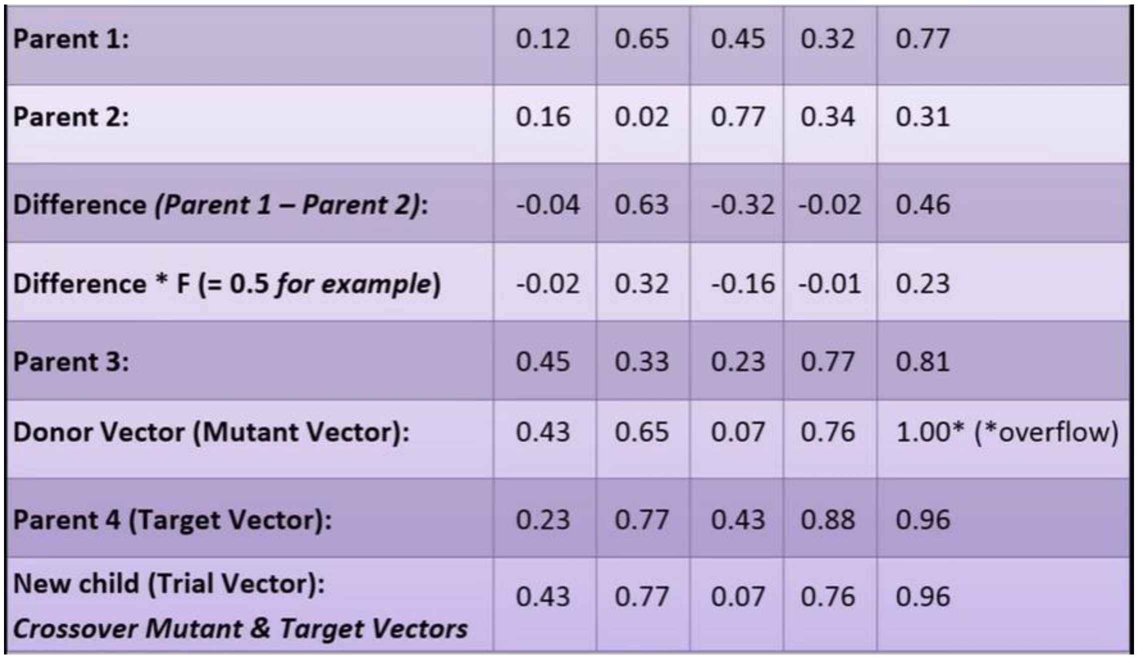
    - For each agent x in the population do:

- Pick three agents a , b and c from the population at random, they must be distinct from each other as well as from agent x (a is called the "base" vector)
- Pick a random index R belongs { 1 , … , n } where n is the dimensionality of the problem being optimized.
- Compute the agent's potentially new position y = { y1 , … , yn }as follows:
    - For each i belongs to {1,…,n}, pick a uniformly distributed random number ri tends to U (0,1)

If ri < CR or i = R then set yi = ai + F x (bi – ci) otherwise set yi = xi. (Index position R is replaced for certain.)

If f(y) <= f(x) then replace the agent x in the population with the improved or equal candidate solution y

- Pick the agent from the population that has the best fitness and return it as the best found candidate solution.



Parent 1:	0.12	0.65	0.45	0.32	0.77
Parent 2:	0.16	0.02	0.77	0.34	0.31
Difference *(Parent 1 – Parent 2)*:	-0.04	0.63	-0.32	-0.02	0.46
Difference * F (= 0.5 *for example*)	-0.02	0.32	-0.16	-0.01	0.23
Parent 3:	0.45	0.33	0.23	0.77	0.81
Donor Vector (Mutant Vector):	0.43	0.65	0.07	0.76	1.00* (*overflow)
Parent 4 (Target Vector):	0.23	0.77	0.43	0.88	0.96
New child (Trial Vector): ***Crossover Mutant & Target Vectors***	0.43	0.77	0.07	0.76	0.96

<u>This information was sourced from Wikipedia's website. The article provided valuable insights into [DE], and its content has been summarized to support our discussion.</u>

Main functionalities:

Optimization Algorithms: The code implements two optimization algorithms: Genetic Algorithm and Differential Evolution

Benchmark Functions: It provides several benchmark functions commonly used to evaluate optimization algorithms, such as Ackley, Bukin6, Cross-in-Tray, Drop-Wave, and Egg-Holder

Constraint Handling: The algorithms include constraint handling using penalty methods to handle constraints on the optimization variables

Evolution Process: Both algorithms perform evolution over a specified number of generations, involving selection, crossover, and mutation operations

Result Printing: The code prints the optimal solution and fitness value obtained by each algorithm for each benchmark function

Convergence Plotting: There is a provision to plot the convergence of fitness values over generations

- ## Similar Applications in Market:

    - PyGMO(Python Parallel Global Multiobjective Optimizer): PyGMO is a scientific library providing a large number of optimization problems and algorithms, including evolutionary algorithms like Genetic Algorithm and Differential Evolution

    - CMA-ES(Covariance Matrix Adaptation Evolution Strategy): CMA-ES is a popular evolutionary algorithm for continuous optimization. While different from genetic algorithms and differential evolution, it serves a similar purpose of finding optimal solutions for complex problems

- Optunity: Optunity is a library for solving optimization problems in Python. It provides a unified interface for different optimization algorithms, including genetic algorithms and differential evolution

- DEAP(Distributed Evolutionary Algorithms in Python): DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It offers a wide range of evolutionary algorithms and benchmark functions similar to those implemented in this code

• Literature review:

- “**A Comprehensive Survey on Differential Evolution Algorithm and it’s Applications” by P.Kannan et al.(2016):** This survey provides an in-depth review of Differential Evolution (DE) algorithm and its various variants. It covers DE's theoretical foundations, algorithmic details, applications across different domains, and comparative studies with other optimization algorithms.

- **"Genetic Algorithms: A Review and Perspective" by T. Bäck et al. (1997):** This paper offers a comprehensive review of Genetic Algorithms (GAs), focusing on their historical development, key concepts, variations, applications, and challenges. It provides insights into the strengths and weaknesses of GAs in comparison to other optimization techniques.

- **"A Survey of Constraint Handling Techniques in Evolutionary Computation Methods" by C. C. Ocampo-Martinez et al. (2015):** This survey paper discusses various constraint handling techniques employed in evolutionary computation methods, including penalty-based approaches like the one implemented in the provided code. It reviews different penalty methods, their advantages, and limitations, along with recommendations for choosing appropriate techniques based on problem characteristics.

- **"Benchmark Functions for the CEC'2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization" by A. L. M. E. Chaves et al. (2013):** This paper presents a collection of benchmark functions specifically designed for evaluating multimodal optimization algorithms, which are commonly used in evolutionary algorithms like Genetic Algorithm and Differential Evolution. It discusses the properties of these benchmark functions and their relevance in assessing the performance of optimization algorithms.

- **"Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces" by R. Storn and K. Price (1997):** This seminal paper introduces Differential Evolution (DE) as a simple yet efficient heuristic for global optimization in continuous spaces. It presents the basic concepts of DE, including mutation, crossover, and selection strategies, along with algorithmic variations and applications in various fields. The paper provides experimental results demonstrating the effectiveness of DE compared to other optimization algorithms.

- **"Genetic Algorithms for Real Parameter Optimization" by K. Deb et al. (1991):** This paper focuses on the application of Genetic Algorithms (GAs) specifically for real-parameter optimization problems. It introduces the concept of real-coded GAs, where solutions are represented as real-valued vectors instead of binary strings. The paper discusses various aspects of real-coded GAs, including encoding schemes, crossover and mutation operators, and convergence analysis. It also presents experimental results and comparative studies to evaluate the performance of real-coded GAs on benchmark functions.

In our project we used two different algorithms

1 – **Genetic Algorithm:** Trying to finds Local Optima in the search space

(Peaks or Valleys)

2 – **Differential Evolution:** Trying to finds Global Optima in the search space (Highest point of this Peaks or Valleys)

Experiments & Results:

Just one iteration of the generation in both GA & DE on the 5 functions

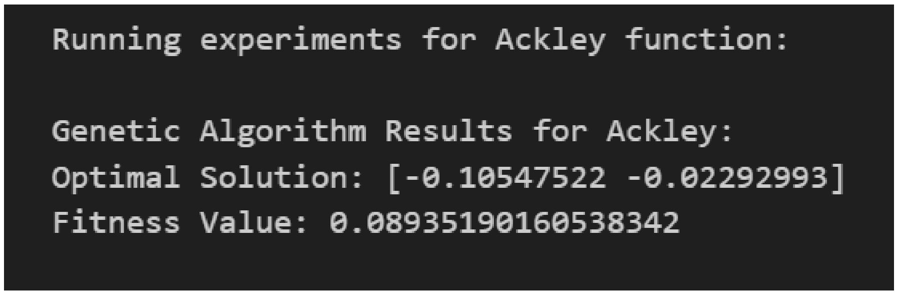
Using:

Parent Selection: Tournament Selection With Elitism

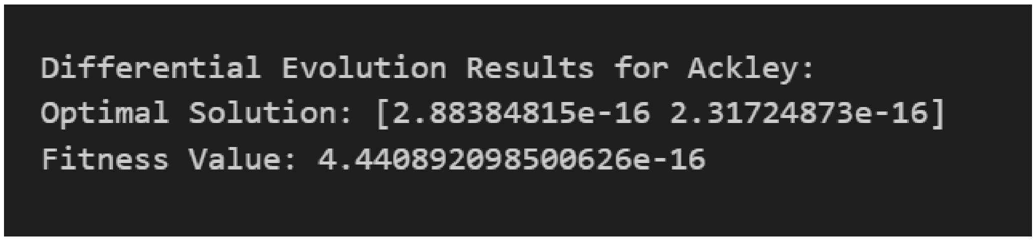
Crossover: Single Point Crossover

Mutation: Uniform Mutation

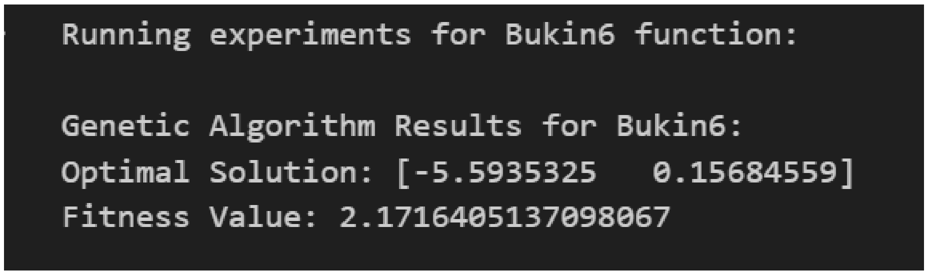
Survival Selection: Generational replacement



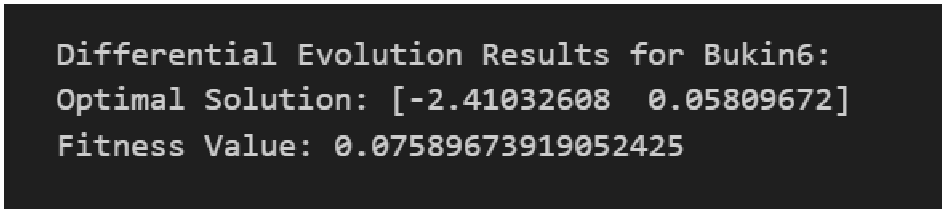
```
Running experiments for Ackley function:

Genetic Algorithm Results for Ackley:
Optimal Solution: [-0.10547522 -0.02292993]
Fitness Value: 0.08935190160538342
```



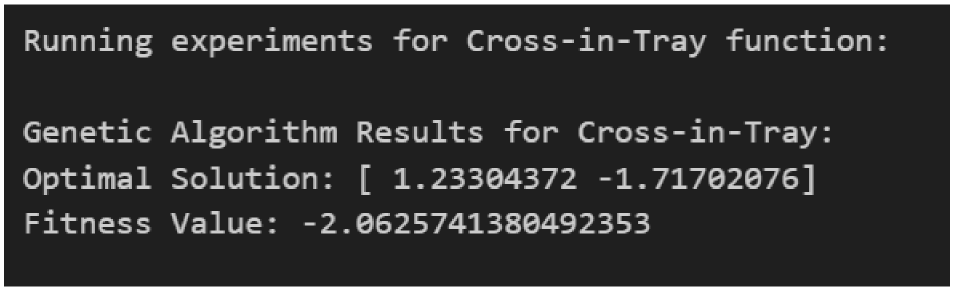
```
Differential Evolution Results for Ackley:
Optimal Solution: [2.88384815e-16 2.31724873e-16]
Fitness Value: 4.440892098500626e-16
```



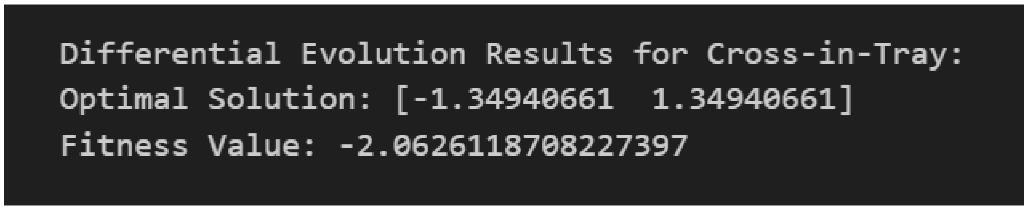
```
Running experiments for Bukin6 function:

Genetic Algorithm Results for Bukin6:
Optimal Solution: [-5.5935325   0.15684559]
Fitness Value: 2.1716405137098067
```



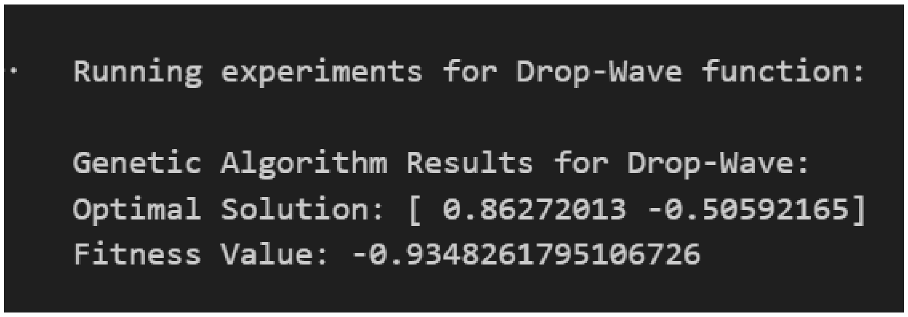
```
Differential Evolution Results for Bukin6:
Optimal Solution: [-2.41032608  0.05809672]
Fitness Value: 0.07589673919052425
```



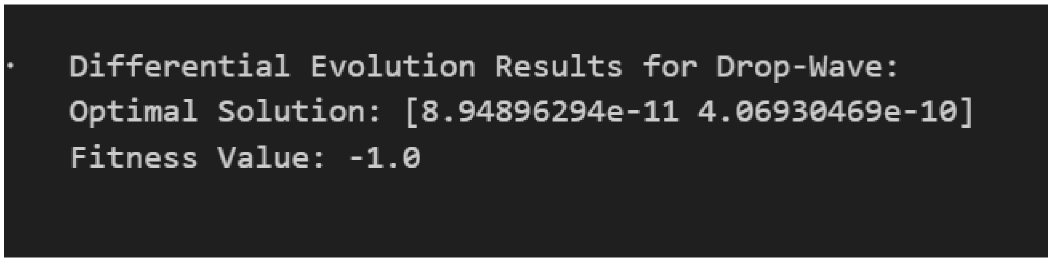
```
Running experiments for Cross-in-Tray function:

Genetic Algorithm Results for Cross-in-Tray:
Optimal Solution: [ 1.23304372 -1.71702076]
Fitness Value: -2.0625741380492353
```



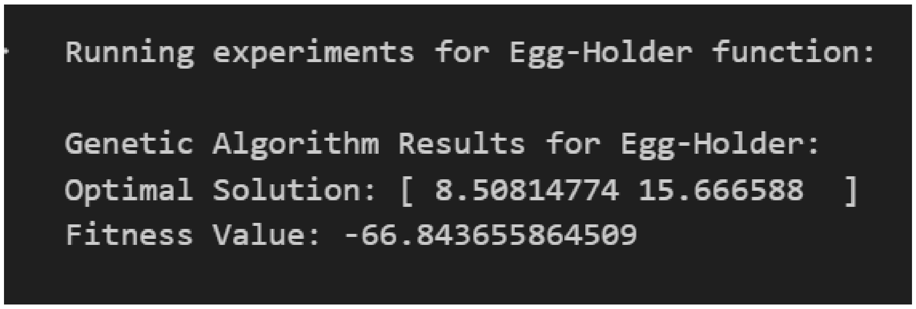
```
Differential Evolution Results for Cross-in-Tray:
Optimal Solution: [-1.34940661  1.34940661]
Fitness Value: -2.0626118708227397
```



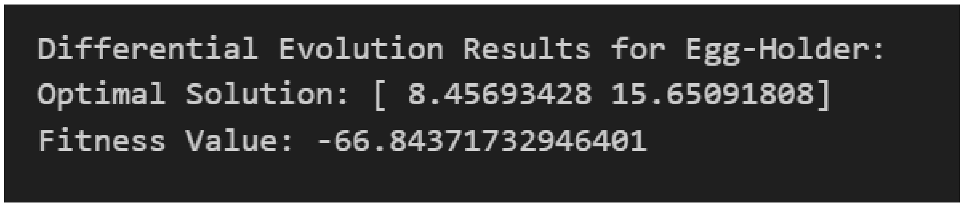
```
Running experiments for Drop-Wave function:

Genetic Algorithm Results for Drop-Wave:
Optimal Solution: [ 0.86272013 -0.50592165]
Fitness Value: -0.9348261795106726
```



```
Differential Evolution Results for Drop-Wave:
Optimal Solution: [8.94896294e-11 4.06930469e-10]
Fitness Value: -1.0
```



```
Running experiments for Egg-Holder function:

Genetic Algorithm Results for Egg-Holder:
Optimal Solution: [ 8.50814774 15.666588  ]
Fitness Value: -66.843655864509
```



```
Differential Evolution Results for Egg-Holder:
Optimal Solution: [ 8.45693428 15.65091808]
Fitness Value: -66.84371732946401
```

Just one iteration of the generation in both GA & DE on the 5 functions

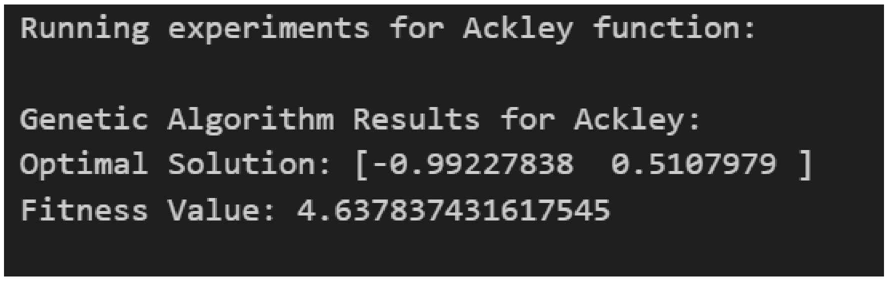
Using :

Parent Selection: Roulette Wheel Selection

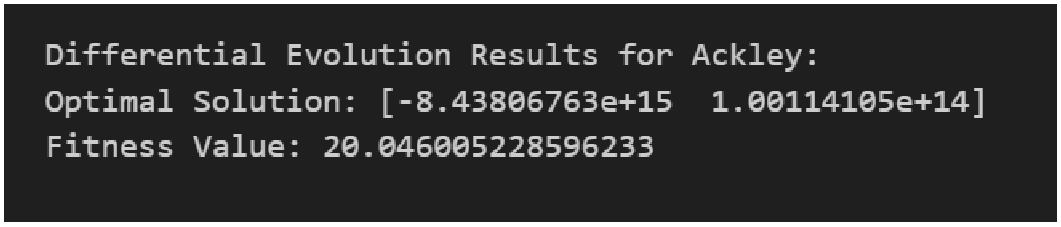
Crossover: Multiple Point Crossover

Mutation: Scramble Mutation

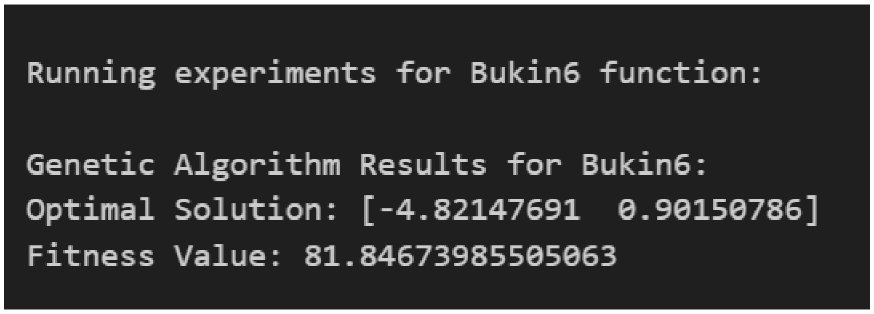
Survival Selection: Steady state replacement



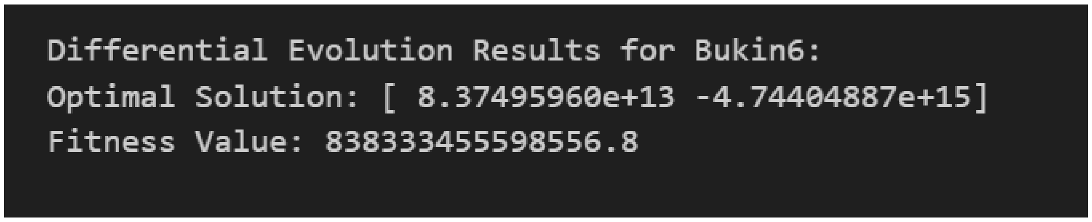
```
Running experiments for Ackley function:

Genetic Algorithm Results for Ackley:
Optimal Solution: [-0.99227838  0.5107979 ]
Fitness Value: 4.637837431617545
```



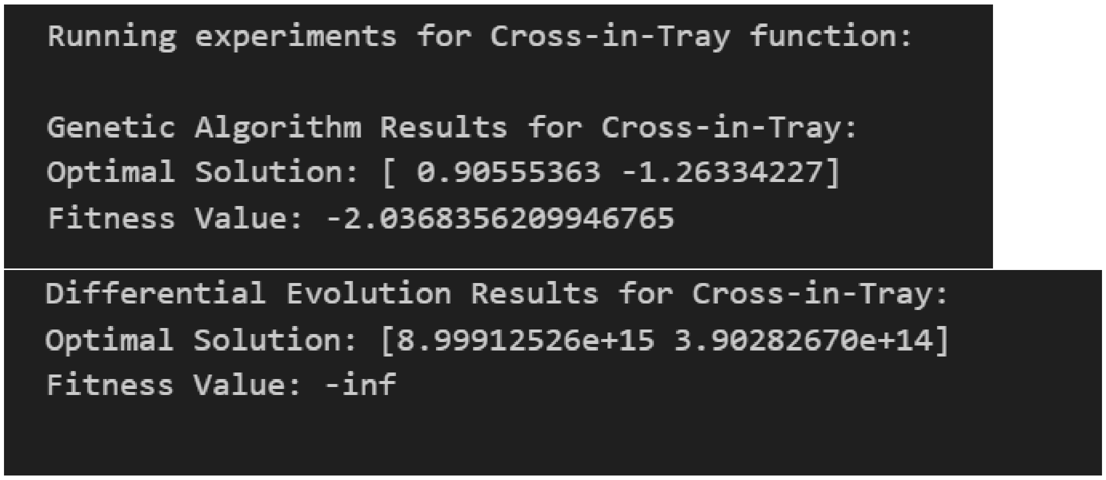
```
 Differential Evolution Results for Ackley:
 Optimal Solution: [-8.43806763e+15  1.00114105e+14]
 Fitness Value: 20.046005228596233
```

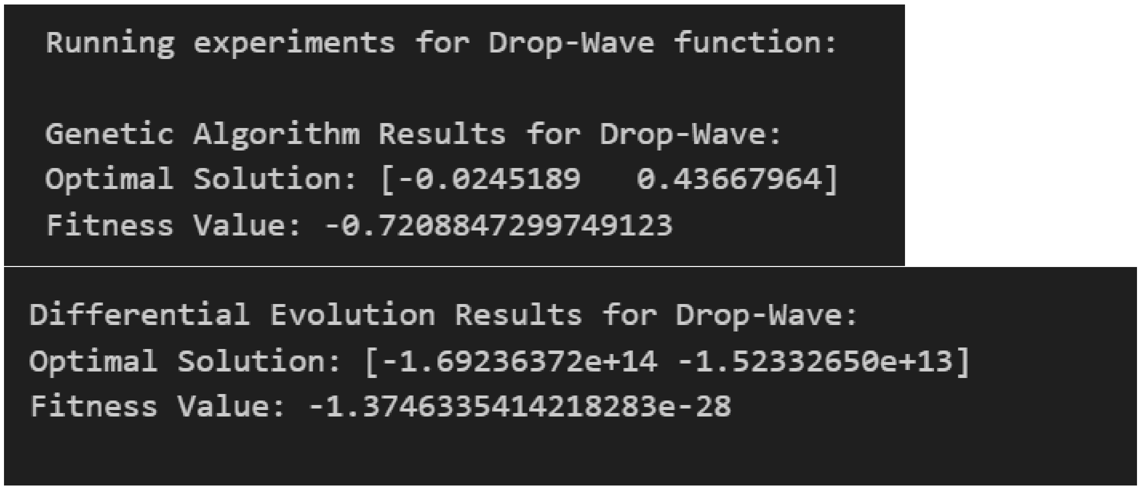


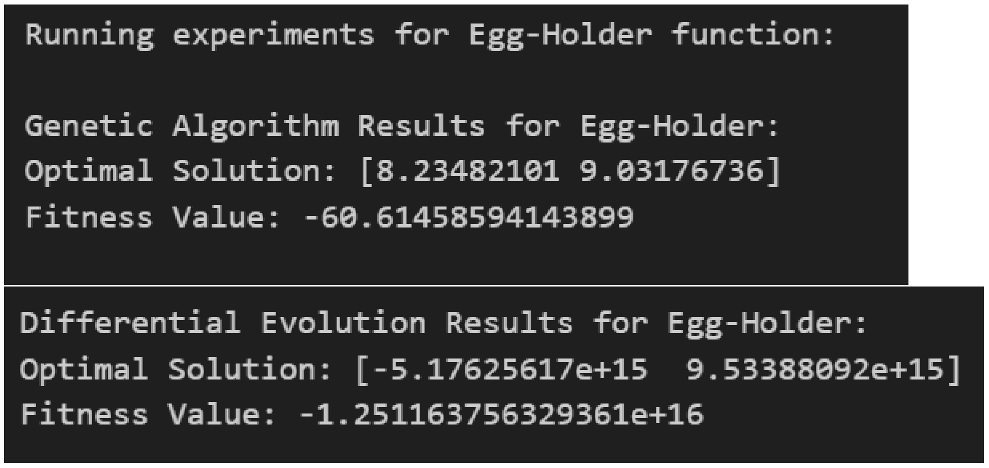
```
Running experiments for Bukin6 function:

Genetic Algorithm Results for Bukin6:
Optimal Solution: [-4.82147691  0.90150786]
Fitness Value: 81.84673985505063
```



```
 Differential Evolution Results for Bukin6:
 Optimal Solution: [ 8.37495960e+13 -4.74404887e+15]
 Fitness Value: 838333455598556.8
```


```
Running experiments for Cross-in-Tray function:

Genetic Algorithm Results for Cross-in-Tray:
Optimal Solution: [ 0.90555363 -1.26334227]
Fitness Value: -2.0368356209946765
```
```
Differential Evolution Results for Cross-in-Tray:
Optimal Solution: [8.99912526e+15 3.90282670e+14]
Fitness Value: -inf
```


```
Running experiments for Drop-Wave function:

Genetic Algorithm Results for Drop-Wave:
Optimal Solution: [-0.0245189   0.43667964]
Fitness Value: -0.7208847299749123
```
```
Differential Evolution Results for Drop-Wave:
Optimal Solution: [-1.69236372e+14 -1.52332650e+13]
Fitness Value: -1.3746335414218283e-28
```


```
Running experiments for Egg-Holder function:

Genetic Algorithm Results for Egg-Holder:
Optimal Solution: [8.23482101 9.03176736]
Fitness Value: -60.61458594143899
```
```
Differential Evolution Results for Egg-Holder:
Optimal Solution: [-5.17625617e+15  9.53388092e+15]
Fitness Value: -1.251163756329361e+16
```

- **The algorithm should outputs the best solution found, which represents the global minimum of the benchmark function.**

Analysis, Discussion:

- Analysis of the results, what are the insights?

Overall the Differential Evolution deal with this type of problems better than the Genetic Algorithm because it shows many better results

The Combinations of

(1)

Parent Selection: Tournament Selection With Elitism

Crossover: Single Point Crossover

Mutation: Uniform Mutation

Survival Selection: Generational replacement

Gives better results than

(2)

Parent Selection: Roulette Wheel Selection

Crossover: Multiple Point Crossover

Mutation: Scramble Mutation

Survival Selection: Steady state replacement

- **Why did the algorithm behave in such a way? What might be the future modifications you'd like to try when solving this problem** ?

Tring different Crossover rates , Mutation rates , Crossover methods , Mutation methods , Selection mechanism , and Survival selection methods , etc…