Opposition-Based Differential Evolution (ODE) with Variable Jumping Rate

S. Rahnamayan¹, H.R. Tizhoosh¹, M.M.A. Salama²
Faculty of Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
¹Pattern Analysis and Machine Intelligence (PAMI) Research Group
²Medical Instrument Analysis and Machine Intelligence (MIAMI) Research Group
shahryar@pami.uwaterloo.ca, tizhoosh@uwaterloo.ca, m.salama@ece.uwaterloo.ca

Abstract—In this paper, a time varying jumping rate (TVJR) model for Opposition-Based Differential Evolution (ODE) has been proposed. According to this model, the jumping rate changes linearly during the evolution based on the number of function evaluations. A test suite with 15 well-known benchmark functions has been employed to compare performance of the DE and ODE with variable jumping rate settings. Results show that a higher jumping rate is more desirable during the exploration than during the exploitation. Details for the proposed approach and the conducted experiments are provided.

I. INTRODUCTION

Generally speaking, parameter control in Evolutionary Algorithms (EAs) can be performed by following three ways [1]: Deterministic, adaptive, and self-adaptive. The first one uses a predefined rule to modify the parameter value without gaining any feedback from the evolution process while the second one changes the parameter value based on the information which receives from search process. The third one utilizes the same evolutionary approach not only to solve the problem but also to optimize own control parameters by encoding some strategic parameters inside the population.

The idea proposed in this paper is similar to Das et al. work [2]. They utilized time varying approach for setting of the scale factor $F$ in Differential Evolution (DE), which can be considered as a deterministic approach according to the mentioned categorization.

The concept of opposition-based learning (OBL) was introduced by Tizhoosh [3] and has thus far been applied to accelerate reinforcement learning [4]–[6], backpropagation learning [7], and differential evolution [8]–[10]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate (i.e., guess and opposite guess) in order to achieve a better approximation of the current candidate solution. Opposition-based differential evolution (ODE) [8] uses opposite numbers during population initialization and also for generating new populations during the evolutionary process. ODE introduces a new parameter, called jumping rate. In this paper, a time varying policy to set this parameter in order to achieve higher convergence velocity will be proposed.

Organization of this paper is as follows: Differential Evolution, the parent algorithm, is briefly reviewed in section II. In section III, the concept of opposition-based learning is explained. The opposition-based DE (ODE) is reviewed and also the proposed jumping rate setting policies are discussed in section IV. Experimental verifications are given in section V. Finally, the work is concluded in section VI.

II. DIFFERENTIAL EVOLUTION

Differential Evolution (DE) is a population-based and directed search method [11], [12]. Like many other evolutionary algorithms, it starts with an initial population vector, which is randomly generated when no a priori knowledge about the solution space is available.

Let us assume that $X_{i,G}(i = 1, 2, ..., N_p)$ are candidate solution vectors in the generation $G$ ($N_p$: population size). Successive populations are generated by adding the weighted difference of two randomly selected vectors to a third randomly selected vector.

For classical DE (DE/rand/1/bin), the mutation, crossover, and selection operators are straightforwardly defined as follows:

**Mutation** - For each vector $X_{i,G}$ in generation $G$ a mutant vector $V_{i,G}$ is defined by

$$V_{i,G} = X_{a,G} + F(X_{b,G} - X_{c,G}),$$  \(1\)

where $i = \{1, 2, ..., N_p\}$ and $a$, $b$, and $c$ are mutually different random integer indices selected from $\{1, 2, ..., N_p\}$. Further, $i$, $a$, $b$, and $c$ are different so that $N_p \geq 4$ is required. $F \in [0, 2]$ is a real constant which determines the amplification of the added differential variation of $(X_{b,G} - X_{c,G})$. Larger values for $F$ result in higher diversity in the generated population and lower
values cause faster convergence.

**Crossover** - DE utilizes the crossover operation to increase the diversity of the population. It defines the following trial vector:

\[ U_{i,G} = (U_{1i,G}, U_{2i,G}, ..., U_{Di,G}), \]

where \( D \) is the problem dimension and

\[ U_{ji,G} = \begin{cases} V_{ji,G} & \text{if } \text{rand}_{ji}((0, 1) \leq C_r, \\ X_{ji,G} & \text{otherwise.} \end{cases} \]

\( C_r \in (0, 1) \) is the predefined crossover rate constant, and \( \text{rand}_{ji}((0, 1) \) is the \( j^{th} \) evaluation of a uniform random number generator. Most popular values for \( C_r \) are in the range of \((0.4, 1) \) [2].

**Selection** - The approach that must decide which vector \((U_{i,G} \) or \( X_{i,G} \)) should be a member of next (new) generation, \( G + 1 \). For a maximization problem, the vector with the higher fitness value is chosen. There are other variants based on different mutation strategies [13].

III. OPPOSITION-BASED LEARNING

Generally speaking, evolutionary optimization methods start with some initial solutions (initial population) and try to improve them toward optimal solution(s). The process of searching terminates when some predefined criteria are satisfied. In the absence of any a priori information about the solution, we usually start with random guesses. The computation time, among others, is related to the distance of these initial guesses from the optimal solution. We can improve our chance of starting with a closer (fitter) solution by simultaneously checking the opposite solution. By doing this, the fitter one (guess or opposite guess) can be chosen as an initial solution. In fact, according to probability theory, 50% of the time a guess is further from the solution than its opposite. So, starting with the closer of the two guesses (as judged by their fitness) has the potential to accelerate convergence. The same approach can be applied not only to initial solutions but also continuously to each solution in the current population. However, before concentrating on opposition-based learning, we need to define the concept of opposite numbers [3]:

**Definition (Opposite Number)** - Let \( x \in [a, b] \) be a real number. The opposite number \( \bar{x} \) is defined by

\[ \bar{x} = a + b - x. \]  

Similarly, this definition can be extended to higher dimensions as follows [3]:

\[ \bar{x}_i = a_i + b_i - x_i. \]  

**Opposition-Based Optimization** - Let \( P = (x_1, x_2, ..., x_n) \) be a point in an \( n \)-dimensional space (i.e. a candidate solution). Assume \( f(\cdot) \) is a fitness function which is used to measure the candidate’s fitness. According to the definition of the opposite point, \( \bar{P} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) \) is the opposite of \( P = (x_1, x_2, ..., x_n) \). Now, if \( f(\bar{P}) \geq f(P) \), then point \( P \) can be replaced with \( \bar{P} \); otherwise we continue with \( P \).

Hence, the point and its opposite point are evaluated simultaneously in order to continue with the fitter one.

IV. REVISITING ODE AND PROPOSING VARIABLE JUMPING RATES

Similar to all population-based optimization algorithms, two main steps are distinguishable for DE, namely population initialization and producing new generations by evolutionary operations such as mutation, crossover, and selection. We will enhance these two steps using the opposition-based learning scheme. The original DE is chosen as a parent algorithm and the proposed opposition-based ideas are embedded in DE to accelerate its convergence velocity. Corresponding pseudo-code for the ODE is given in Table I. Newly added/extended code segments will be explained in the following subsections.

A. Opposition-Based Population Initialization

Random number generation, in absence of a priori knowledge, is a widely used choice to create an initial population. But as mentioned in section III, by utilizing opposition-based learning we can obtain fitter starting candidate solutions even when there is no knowledge about the solution(s). Steps 1-5 from Table I show the implementation of opposition-based initialization for the ODE. Following steps show that procedure:

1) Initialize the population \( P(N_P) \) randomly,
2) Calculate opposite population by

\[ \text{OP}_{i,j} = a_j + b_j - P_{i,j}, \]

\[ i = 1, 2, ..., N_p; \quad j = 1, 2, ..., D, \]

where \( P_{i,j} \) and \( \text{OP}_{i,j} \) denote \( j^{th} \) variable of the \( i^{th} \) vector of the population and the opposite-population, respectively.
3) Select the \( N_p \) fittest individuals from \( \{ P \cup OP \} \) as initial population.

**B. Opposition-Based Generation Jumping**

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a new solution candidate, which ideally is fitter than the current one. Based on a jumping rate \( J_r \) (i.e., jumping probability), after generating new populations by mutation, crossover, and selection, the opposite population is calculated and the \( N_p \) fittest individuals are selected from the union of the current population and the opposite population. As a difference to opposition-based initialization, it should be noted here that in order to calculate the opposite population for generation jumping, the opposite of each variable is calculated dynamically. That is, the maximum and minimum values of each variable in the current population \( ([\text{MIN}_j^p, \text{MAX}_j^p]) \) are used to calculate opposite points instead of using variables’ predefined interval boundaries \( ([a_j, b_j]) \):

\[
\text{OP}_{i,j} = \text{MIN}_j^p + \text{MAX}_j^p - P_{i,j}, \quad (7)
\]

\( i = 1, 2, \ldots, N_p; \quad j = 1, 2, \ldots, D. \)

By staying within variables’ interval static boundaries, we would jump outside of the already shrunken search space and the knowledge of the current reduced space (converged population) would be lost. Hence, we calculate opposite points by using variables’ current interval in the population \( ([\text{MIN}_j^p, \text{MAX}_j^p]) \) which is, as the search does progress, increasingly smaller than the corresponding initial range \( [a_j, b_j] \). Steps 26-32 from Table I show the implementation of opposition-based generation jumping for ODE.

**C. Proposed jumping rate settings**

In the first version of the opposition-based differential evolution (ODE) [8–10], a constant value for jumping rate was used \( (J_r = 0.3) \). In this paper, two types of time varying jumping rate are investigated (linearly increasing and decreasing functions). Three proposed settings for the current investigation are as follows:

- \( J_r \) (constant) = \( J_{r_{ave}} \),
- \( J_r \) (TVJR1) = \( (J_{r_{max}} - J_{r_{min}}) \times (\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}) \),
- \( J_r \) (TVJR2) = \( (J_{r_{max}} - J_{r_{min}}) - (J_{r_{max}} - J_{r_{min}}) \times (\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}) \),

where \( J_{r_{ave}} \), \( J_{r_{max}} \), and \( J_{r_{min}} \) are the average, maximum, and minimum jumping rates, respectively. \( \text{MAX}_{\text{NFC}} \) and \( \text{NFC} \) are the maximum number of function calls and the current number of function calls, respectively.

In order to support as fair as possible comparison between these three different jumping rate settings, the average jumping rate should be the same for all of them. So, obviously we should have \( J_{r_{ave}} = \frac{(J_{r_{max}} + J_{r_{min}})}{2} \). Following values for these parameters are selected: \( J_{r_{ave}} = 0.3 \) [8–10] and \( J_{r_{min}} = 0 \) (no jumping), so \( J_{r_{max}} = 0.6 \) is resulted. Figure 1 shows the corresponding diagrams (jumping rate vs. NFCs) for four following settings:

- \( J_r \) (constant) = 0.3,
- \( J_r \) (TVJR1) = 0.6 × \( (\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}) \),
- \( J_r \) (TVJR2) = 0.6 – 0.6 × \( (\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}) \),
- \( J_r \) (constant) = 0.6.

\( J_r \) (TVJR1) represents higher jumping rate during exploration and lower jumping rate during exploitation (tuning); \( J_r \) (TVJR2) performs exactly in reverse manner. By these settings, we can investigate effects of generation jumping during optimization process.

**V. EXPERIMENTAL VERIFICATION**

In this section we describe the benchmark functions, comparison strategies, algorithm settings, and present the results.

**A. Benchmark Functions**

A set of 15 well-known benchmark functions has been used for performance verification of the proposed approach. The classical differential evolution (DE) and the opposition-based DE (ODE) with different jumping
TABLE I


Steps 1-5 and 26-32 are implementations of opposition-based initialization and opposition-based generation jumping, respectively.

/* Opposition-Based Population Initialization */
1. Generate uniformly distributed random population $P_0$;
2. for ($i = 0 : i < N_p : i += 1$)
3. for ($j = 0 : j < D : j += 1$)
4. OP$_{0,i,j} = a_j + b_j - P_{0,i,j}$;
5. Select $N_p$ fittest individuals from set the $\{P_0, OP_0\}$ as initial population $P_0$;
/* End of Opposition-Based Population Initialization */
6. while (BFV > VTR and NFC < MAXNFC )
7. { for ($i = 0 : i < N_p : i += 1$)
8. Select three parents $P_a$, $P_b$, and $P_c$ randomly from current population where $i \neq a \neq b \neq c$;
9. for ($j = 0 : j < D : j += 1$)
10. $V_{i,j} = P_a + F \times (P_b - P_c)$;
11. if ($\text{rand}(0, 1) < C_r$)
12. $U_{i,j} = V_{i,j}$;
13. else
14. $U_{i,j} = P_{i,j}$;
15. Evaluate $U_i$;
16. if ($f(U_i) \leq f(P_i)$)
17. $P'_i = U_i$;
18. else
19. $P'_i = P_i$;
20. $P = P'$;
/* Opposition-Based Generation Jumping */
21. if ($\text{rand}(0, 1) < J_r$)
22. { for ($i = 0 : i < N_p : i += 1$)
23. for ($j = 0 : j < D : j += 1$)
24. OP$_{i,j} = \min^p_j + \max^p_j - P_{i,j}$;
25. Select $N_p$ fittest individuals from set the $\{P, OP\}$ as current population $P$;
/* End of Opposition-Based Generation Jumping */
26. }
27. }
28. }
/* End of Opposition-Based Population Initialization */
29. }
30. }
31. }
32. }
33. }
rate setting policies (constant values and time varyings) are compared. The definition of the benchmark functions and their global optimum(s) are listed in Appendix A.

B. Comparison Strategies and Metrics

In this study, three metrics, namely, number of function calls (NFC), success rate (SR), and success performance (SP) [14] have been utilized to compare the algorithms. We compare the convergence velocity by measuring the number of function calls which is the most commonly used metric in the literature [8]–[10], [14]. A smaller NFC means higher convergence velocity. The termination criterion is to find a value smaller than the value-to-reach (VTR) before reaching the maximum number of function calls MAX$_{\text{NFC}}$. In order to minimize the effect of the stochastic nature of the algorithms on the metric, the reported number of function calls for each function is the average over 50 trials.

The number of times, for which the algorithm succeeds to reach the VTR for each test function is measured as the success rate SR:

\[
\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{8}
\]

The average success rate (SR$_{\text{ave}}$) over $n$ test functions are calculated as follows:

\[
\text{SR}_{\text{ave}} = \frac{1}{n} \sum_{i=1}^{n} \text{SR}_i. \tag{9}
\]

Both of NFC and SR are important measures in an optimization process. So, two individual objectives should be considered simultaneously to compare competitors. In order to combine these two metrics, a new measure, called success performance (SP), has been introduced as follows [14]:

\[
\text{SP} = \frac{\text{mean (NFC for successful runs)}}{\text{SR}}. \tag{10}
\]

By this definition, the two following algorithms have equal performances (SP=100):

Algorithm A: mean (NFC for successful runs)=50 and SR=0.5,

Algorithm B: mean (NFC for successful runs)=100 and SR=1.

SP is our the main measure to judge which algorithm performs better.

C. Setting Control Parameters

Parameter settings for all conducted experiments are as follows:

- Population size, $N_p = 100$ [15]–[17]
- Differential amplification factor, $F = 0.5$ [13], [15], [18]–[20]
- Crossover probability constant, $C_r = 0.9$ [13], [15], [18]–[20]
- Maximum number of function calls, MAX$_{\text{NFC}}$:
  - $2 \times 10^5$ for $f_1, f_2, f_3, f_{5}, f_7, f_8, f_{11}$,
  - $5 \times 10^5$ for $f_4, f_9, f_{10}, f_{13}$,
- Jumping rates (see Figure 1):
  - $J_r (TVJR1)$ still keeps the first position in both rankings.
  - $J_r (TVJR2)$ show the highest one (for NFCs and SPs) and the average success rates. We can rank these algorithms as ODE (TVJR1) (best), ODE ($J_r = 0.3$), ODE (TVJR2), and DE with the respect to the total success performance to solve 15 problems. As we mentioned before, the success performance is a measure which considers the number of function calls and the success rate simultaneously and so it can be utilized for a more reliable comparison of the optimization algorithms. ODE ($J_r = 0.6$) presents the lowest average success rate (0.89); while DE and ODE (TVJR2) show the highest one (0.97).

Pair comparison of these algorithms are presented in Table III. Given number in each cell shows on how many functions the algorithm in the table’s row outperforms the corresponding algorithm in the table’s column. The last column of the table shows the total numbers (number of cases which the algorithm can outperform other competitors); by comparing these numbers we obtain the following ranking result: ODE (TVJR1) (best), ODE ($J_r = 0.6$), ODE ($J_r = 0.3$), ODE (TVJR2), and DE. ODE ($J_r = 0.6$) and ODE ($J_r = 0.3$) have changed their positions compared to the previous ranking. But, ODE (TVJR1) still keeps the first position in both rankings. Results for $J_r = 0.3$ and $J_r = 0.6$ confirm that the constant higher jumping rate reduces the overall success rate.

D. Results

Results of applying DE, ODE ($J_r = 0.3$), ODE ($J_r = 0.6$), ODE (TVJR1), and ODE (TVJR2) to solve 15 test problems are given in Table II. The best success performance for each function is highlighted in boldface. The last rows of the table present the sum (for NFCs and SPs) and the average success rates. We can rank these algorithms as ODE (TVJR1) (best), ODE ($J_r = 0.3$), ODE ($J_r = 0.6$), ODE (TVJR2), and DE with the respect to the total success performance to solve 15 problems. As we mentioned before, the success performance is a measure which considers the number of function calls and the success rate simultaneously and so it can be utilized for a more reliable comparison of the optimization algorithms. ODE ($J_r = 0.6$) presents the lowest average success rate (0.89); while DE and ODE (TVJR2) show the highest one (0.97).

Pair comparison of these algorithms are presented in Table III. Given number in each cell shows on how many functions the algorithm in the table’s row outperforms the corresponding algorithm in the table’s column. The last column of the table shows the total numbers (number of cases which the algorithm can outperform other competitors); by comparing these numbers we obtain the following ranking result: ODE (TVJR1) (best), ODE ($J_r = 0.6$), ODE ($J_r = 0.3$), ODE (TVJR2), and DE. ODE ($J_r = 0.6$) and ODE ($J_r = 0.3$) have changed their positions compared to the previous ranking. But, ODE (TVJR1) still keeps the first position in both rankings. Results for $J_r = 0.3$ and $J_r = 0.6$ confirm that the constant higher jumping rate reduces the overall success rate.

VI. Conclusion

In this paper, the time varying jumping rate for opposition-based differential evolution was proposed and two behaviorally reverse versions of them (linearly decreasing and increasing functions) were compared with the constant settings ($J_{r_{\text{ave}}}$ and $J_{r_{\text{max}}}$). The results
# Table II

Comparison of DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2). D: Dimension, NFC: Number of function calls (average over 50 trials), SR: Success rate, SP: Success performance. The last rows of the table present the sum (for NFCs and SPs) and the average success rates. The best success performance for each case is highlighted in boldface.

<table>
<thead>
<tr>
<th>$F$</th>
<th>$D$</th>
<th>DE</th>
<th>$J_r = 0.3$</th>
<th>ODE (TVJR1)</th>
<th>ODE (TVJR2)</th>
<th>ODE ($J_r = 0.6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NFC</td>
<td>SR</td>
<td>SP</td>
<td>NFC</td>
<td>SR</td>
</tr>
<tr>
<td>$f_1$</td>
<td>30</td>
<td>87748</td>
<td>1</td>
<td>87748</td>
<td>47716</td>
<td>1</td>
</tr>
<tr>
<td>$f_2$</td>
<td>30</td>
<td>96488</td>
<td>1</td>
<td>96488</td>
<td>53304</td>
<td>1</td>
</tr>
<tr>
<td>$f_3$</td>
<td>20</td>
<td>177880</td>
<td>1</td>
<td>177880</td>
<td>168680</td>
<td>1</td>
</tr>
<tr>
<td>$f_4$</td>
<td>10</td>
<td>328844</td>
<td>1</td>
<td>328844</td>
<td>65056</td>
<td>0.64</td>
</tr>
<tr>
<td>$f_5$</td>
<td>30</td>
<td>113428</td>
<td>1</td>
<td>113428</td>
<td>64920</td>
<td>1</td>
</tr>
<tr>
<td>$f_6$</td>
<td>30</td>
<td>25140</td>
<td>1</td>
<td>25140</td>
<td>8328</td>
<td>1</td>
</tr>
<tr>
<td>$f_7$</td>
<td>30</td>
<td>149512</td>
<td>1</td>
<td>149512</td>
<td>98296</td>
<td>1</td>
</tr>
<tr>
<td>$f_8$</td>
<td>30</td>
<td>101460</td>
<td>1</td>
<td>101460</td>
<td>70408</td>
<td>1</td>
</tr>
<tr>
<td>$f_9$</td>
<td>10</td>
<td>215260</td>
<td>0.56</td>
<td>384393</td>
<td>168470</td>
<td>0.76</td>
</tr>
<tr>
<td>$f_{10}$</td>
<td>30</td>
<td>385192</td>
<td>1</td>
<td>385192</td>
<td>369104</td>
<td>1</td>
</tr>
<tr>
<td>$f_{11}$</td>
<td>30</td>
<td>187300</td>
<td>1</td>
<td>187300</td>
<td>155636</td>
<td>1</td>
</tr>
<tr>
<td>$f_{12}$</td>
<td>30</td>
<td>41588</td>
<td>1</td>
<td>41588</td>
<td>23124</td>
<td>1</td>
</tr>
<tr>
<td>$f_{13}$</td>
<td>30</td>
<td>411164</td>
<td>1</td>
<td>411164</td>
<td>337532</td>
<td>1</td>
</tr>
<tr>
<td>$f_{14}$</td>
<td>10</td>
<td>19528</td>
<td>1</td>
<td>19528</td>
<td>15704</td>
<td>1</td>
</tr>
<tr>
<td>$f_{15}$</td>
<td>10</td>
<td>37824</td>
<td>1</td>
<td>37824</td>
<td>24260</td>
<td>1</td>
</tr>
<tr>
<td>$\sum_{F=1}^{15}$</td>
<td></td>
<td>2397996</td>
<td>2567129</td>
<td>1670538</td>
<td>1781973</td>
<td>1637634</td>
</tr>
<tr>
<td>$\bar{SR}_{ave}$</td>
<td></td>
<td>0.97</td>
<td>0.94</td>
<td>0.95</td>
<td>0.97</td>
<td>0.89</td>
</tr>
</tbody>
</table>
confirm that the linearly decreasing jumping rate performs better than constant settings and also than linearly increasing policy. This means generation jumping in the exploration time is more desirable than during exploitation. Because during the fine-tuning, we are faced with shrunkened search space and the jumping of the individuals may not be advantageous. We know that there is no exact border between exploration and exploitation time. Hence, the gradual behavior for the decreasing and increasing functions are proposed.

The proposed jumping rate function utilizes the maximum number of function calls (MAXSEC) which may not be exactly known for some black-box optimization problem; this can be regarded as a disadvantage for the proposed method. Adaptive setting of the jumping rate can be a desirable solution which will be a focus of our research in future.

REFERENCES


APPENDIX A. LIST OF BENCHMARK FUNCTIONS

1st De Jong

\[ f_1(X) = \sum_{i=1}^{n} x_i^2, \]

with \(-5.12 \leq x_i \leq 5.12, \]

\[ \min(f_1) = f_1(0, \ldots, 0) = 0. \]
- **Axis Parallel Hyper-Ellipsoid**
  \[ f_2(X) = \sum_{i=1}^{n} ix_i^2, \]
  with \(-5.12 \leq x_i \leq 5.12, \min(f_2) = f_2(0, ..., 0) = 0.\)

- **Schwefel's Problem 1.2**
  \[ f_3(X) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2, \]
  with \(-65 \leq x_i \leq 65, \min(f_3) = f_3(0, ..., 0) = 0.\)

- **Rastrigin's Function**
  \[ f_4(X) = 10n + \sum_{i=1}^{n} (x_i^2 - 10 \cos(2\pi x_i)), \]
  with \(-5.12 \leq x_i \leq 5.12, \min(f_4) = f_4(0, ..., 0) = 0.\)

- **Griewangk's Function**
  \[ f_5(X) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \]
  with \(-600 \leq x_i \leq 600, \min(f_5) = f_5(0, ..., 0) = 0.\)

- **Sum of Different Power**
  \[ f_6(X) = \sum_{i=1}^{n} |x_i|^{(i+1)}, \]
  with \(-1 \leq x_i \leq 1, \min(f_6) = f_6(0, ..., 0) = 0.\)

- **Ackley's Problem**
  \[ f_7(X) = -20 \exp(-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e, \]
  with \(-32 \leq x_i \leq 32, \min(f_7) = f_7(0, ..., 0) = 0.\)

- **Levy Function**
  \[ f_8(X) = \sin^2(3\pi x_1) + \sum_{i=2}^{n-1} (x_i - 1)^2 \]
  \[(1 + \sin^2(3\pi x_{i+1}) + (x_n - 1)(1 + \sin^2(2\pi x_n)), \]
  with \(-10 \leq x_i \leq 10, \min(f_8) = f_8(1, ..., 1) = 0.\)

- **Michalewicz Function**
  \[ f_9(X) = -\sum_{i=1}^{n} \sin(x_i) \sin(\frac{ix_i^2}{\pi})^{2m}, \]
  with \(0 \leq x_i \leq \pi, m = 10, \min(f_9_{(m=10)}) = -9.66015.\)

- **Zakharov Function**
  \[ f_{10}(X) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^{2} + \left(\sum_{i=1}^{n} 0.5ix_i\right)^{4}, \]
  with \(-5 \leq x_i \leq 10, \min(f_{10}) = f_{10}(0, ..., 0) = 0.\)

- **Schwefel's Problem 2.22**
  \[ f_{11}(X) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|, \]
  with \(-10 \leq x_i \leq 10, \min(f_{11}) = f_{11}(0, ..., 0) = 0.\)

- **Step Function**
  \[ f_{12}(X) = \sum_{i=1}^{n} (|x_i + 0.5|)^{2}, \]
  with \(-100 \leq x_i \leq 100, \min(f_{12}) = f_{12}(-0.5 \leq x_i < 0.5) = 0.\)

- **Alpine Function**
  \[ f_{13}(X) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|, \]
  with \(-10 \leq x_i \leq 10, \min(f_{13}) = f_{13}(0, ..., 0) = 0.\)

- **Exponential Problem**
  \[ f_{14}(X) = \exp(-0.5 \sum_{i=1}^{n} x_i^2), \]
  with \(-1 \leq x_i \leq 1, \min(f_{14}) = f_{14}(0, ..., 0) = 1.\)

- **Salomon Problem**
  \[ f_{15}(X) = 1 - \cos(2\pi \| x \|) + 0.1 \| x \|, \]
  where \(\| x \| = \sqrt{\sum_{i=1}^{n} x_i^2}, \)
  with \(-100 \leq x_i \leq 100, \min(f_{15}) = f_{15}(0, 0, ..., 0) = 0.\)