

# Simplex Differential Evolution

**Musrrat Ali<sup>1</sup>, Millie Pant<sup>1</sup> and Ajith Abraham<sup>2</sup>**

<sup>1</sup> Department of Paper Technology, Indian Institute of Technology Roorkee, Saharanpur campus, Saharanpur -247001, India

<sup>2</sup> Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, P.O. Box 2259, Auburn, Washington-98071-2259, USA

musrrat.iitr@gmail.com, millidma@gmail.com, ajith.abraham@ieee.org

---

*Abstract: Differential evolution (DE) algorithms are commonly used metaheuristics for global optimization, but there has been very little research done on the generation of their initial population. The selection of the initial population in a population-based heuristic optimization method is important, since it affects the search for several iterations and often has an influence on the final solution. If no a priori information about the optima is available, the initial population is often selected randomly using pseudorandom numbers. In this paper, we have investigated the effect of generating the initial population without using the conventional methods like computer generated random numbers or quasi random sequences. We have applied non linear simplex method in conjugation of pseudorandom numbers to generate initial population for DE. Proposed algorithm is named as NSDE (using non linear simplex method), is tested on a set of 20 benchmark problems with box constraints, taken from literature and the numerical results are compared with results obtained by traditional DE and opposition based DE (ODE). Numerical results show that the proposed scheme considered by us for generating the random numbers significantly improves the performance of DE in terms of convergence rate and average CPU time.*

*Keywords: Stochastic optimization, differential evolution, crossover, initial population, random numbers*

---

## 1 Introduction

DE is comparatively a recent addition to class of population based search heuristics. Nevertheless, it has emerged as one of the techniques most favored by engineers for solving continuous optimization problems. DE [1, 2] has several attractive features. Besides being an exceptionally simple evolutionary strategy, it is significantly faster and robust for solving numerical optimization problems and is more likely to find the function's true global optimum. Also, it is worth mentioning that DE has a compact structure with a small computer code and has

fewer control parameters in comparison to other evolutionary algorithms. Originally Price and Storn proposed a single strategy for DE, which they later extended to ten different strategies [3].

DE has been successfully applied to a wide range of problems including Batch Fermentation Process [4], Optimal design of heat exchangers [5], synthesis and optimization of heat integrated distillation system [6], optimization of non-linear chemical process [7], optimization of process synthesis and design problems [8], optimization of thermal cracker operation [9], optimization of water pumping system [10], dynamic optimization of a continuous polymer reactor [11], optimization of low pressure chemical vapor deposition reactors [12], etc.

Despite having several striking features and successful applications to various fields DE is sometimes criticized for its slow convergence rate for computationally expensive functions. By varying the control parameters the convergence rate of DE may be increased but it should be noted that it do not affect the quality of solution. Generally, in population based search techniques like DE an acceptable trade-off should be maintained between convergence and type of solution, which even if not a global optimal solution should be satisfactory rather than converging to a suboptimal solution which may not even be a local solution. Several attempts have been made in this direction to fortify DE with suitable mechanisms to improve its performance. Most of the studies involve the tuning or controlling of the parameters of algorithm and improving the mutation, crossover and selection mechanism, some interesting modifications that helped in enhancing the performance of DE include introduction of greedy random strategy for selection of mutant vector [13], modifications in mutation and localization in acceptance rule [14], DE with preferential crossover [15], crossover based local search method for DE [16], self adaptive differential evolution algorithm [17], new donor schemes proposed for the mutation operation of DE [18], parent centric DE [28]. All the modified versions have shown that a slight change in the structure of DE can help in improving its performance. However, the role of the initial population, which is the topic of this paper, is widely ignored. Often, the whole area of research is set aside by a statement “generate an initial population,” without implying how it should be done. There is only few literature is available on this topic [23-27]. An interesting method for generating the initial population was suggested by Rahnamayan et al ([19], [20]) in which the initial population was generated using opposition based rule. To further continue the research in this direction, in this paper we propose two modified versions of DE to improve its performance in terms of convergence rate without compromising with the quality of solution. The modified version presented in this paper is named non linear simplex method called NSDE. In the present study our aim is to investigate the effect of initial population on payoff between convergence rate and solution quality. Our motivation is to encourage discussions on methods of initial population construction. Performances of the proposed algorithms are compared with Basic DE and differential evolution initialized by opposition based learning

(ODE), which is a recently modified version of differential evolution [19], on a set of twenty unconstrained benchmark problems.

Remaining of the paper is organized in following manner; in Section 2, we give a brief description of DE. In Section 3, the proposed algorithms are explained. Section 4 deals with experimental settings and parameter selection. Benchmark problems considered for the present study are given in Section 5. The performances of the proposed algorithms are compared with basic DE and ODE in Section 6. The conclusions based on the present study are finally drawn in Section 7.

## 2 Differential Evolution (DE)

DE starts with a population of NP candidate solutions which may be represented as  $X_{i,G}$ ,  $i = 1, \dots, NP$ , where  $i$  index denotes the population and G denotes the generation to which the population belongs. The working of DE depends on the manipulation and efficiency of three main operators; mutation, reproduction and selection which briefly described in this section.

*Mutation:* Mutation operator is the prime operator of DE and it is the implementation of this operation that makes DE different from other Evolutionary algorithms. The mutation operation of DE applies the vector differentials between the existing population members for determining both the degree and direction of perturbation applied to the individual subject of the mutation operation. The mutation process at each generation begins by randomly selecting three individuals in the population. The most often used mutation strategies implemented in the DE codes are listed below.

$$\text{DE/rand/1: } V_{i,g} = X_{r_1,g} + F * (X_{r_2,g} - X_{r_3,g}) \quad (1a)$$

$$\text{DE/rand/2: } V_{i,g} = X_{r_1,g} + F * (X_{r_2,g} - X_{r_3,g}) + F * (X_{r_4,g} - X_{r_5,g}) \quad (1b)$$

$$\text{DE/best/1: } V_{i,g} = X_{best,g} + F * (X_{r_1,g} - X_{r_2,g}) \quad (1c)$$

$$\text{DE/best/2: } V_{i,g} = X_{best,g} + F * (X_{r_1,g} - X_{r_2,g}) + F * (X_{r_3,g} - X_{r_4,g}) \quad (1d)$$

DE/rand-to-best/1:

$$V_{i,g} = X_{r_1,g} + F * (X_{best,g} - X_{r_2,g}) + F * (X_{r_3,g} - X_{r_4,g}) \quad (1e)$$

Where,  $i = 1, \dots, NP$ ,  $r_1, r_2, r_3 \in \{1, \dots, NP\}$  are randomly selected and satisfy:

$r_1 \neq r_2 \neq r_3 \neq i$ ,  $F \in [0, 1]$ , F is the control parameter proposed by Storn and Price [1].

Throughout the paper we shall refer to the strategy (1a) which is apparently the most commonly used version and shall refer to it as basic version.

*Crossover:* once the mutation phase is complete, the crossover process is activated. The perturbed individual,  $V_{i,G+1} = (v_{1,i,G+1}, \dots, v_{n,i,G+1})$ , and the current population member,  $X_{i,G} = (x_{1,i,G}, \dots, x_{n,i,G})$ , are subject to the crossover operation, that finally generates the population of candidates, or “trial” vectors,  $U_{i,G+1} = (u_{1,i,G+1}, \dots, u_{n,i,G+1})$ , as follows:

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{if } \text{rand}_j \leq Cr, \vee j = k \\ x_{j,i,G} & \text{otherwise} \end{cases} \quad (2)$$

Where,  $j = 1 \dots n$ ,  $k \in \{1, \dots, n\}$  is a random parameter’s index, chosen once for each  $i$ , and the crossover rate,  $Cr \in [0, 1]$ , the other control parameter of DE, is set by the user.

*Selection:* The selection scheme of DE also differs from that of other EAs. The population for the next generation is selected from the individual in current population and its corresponding trial vector according to the following rule:

$$X_{i,G+1} = \begin{cases} U_{i,G+1} & \text{if } f(U_{i,G+1}) \leq f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \quad (3)$$

Thus, each individual of the temporary (trial) population is compared with its counterpart in the current population. The one with the lower objective function value will survive from the tournament selection to the population of the next generation. As a result, all the individuals of the next generation are as good or better than their counterparts in the current generation. In DE trial vector is not compared against all the individuals in the current generation, but only against one individual, its counterpart, in the current generation. The pseudo code of algorithm is given here.

#### **DE pseudo code:**

- Step 1:* The first step is the random initialization of the parent population. Randomly generate a population of (say) NP vectors, each of n dimensions:  $x_{i,j} = x_{\min,j} + \text{rand}(0, 1)(x_{\max,j} - x_{\min,j})$ , where  $x_{\min,j}$  and  $x_{\max,j}$  are lower and upper bounds for  $j^{\text{th}}$  component respectively,  $\text{rand}(0,1)$  is a uniform random number between 0 and 1.
- Step 2:* Calculate the objective function value  $f(X_i)$  for all  $X_i$ .
- Step 3:* Select three points from population and generate perturbed individual  $V_i$  using equation (1a).
- Step 4:* Recombine the each target vector  $x_i$  with perturbed individual generated in step 3 to generate a trial vector  $U_i$  using equation (2).

- Step 5:* Check whether each variable of the trial vector is within range. If yes, then go to step 6 else make it within range using  $u_{i,j} = 2 * x_{\min,j} - u_{i,j}$ , if  $u_{i,j} < x_{\min,j}$  and  $u_{i,j} = 2 * x_{\max,j} - u_{i,j}$ , if  $u_{i,j} > x_{\max,j}$ , and go to step 6.
- Step 6:* Calculate the objective function value for vector  $U_i$ .
- Step 7:* Choose better of the two (function value at target and trial point) using equation (3) for next generation.
- Step 8:* Check whether convergence criterion is met if yes then stop; otherwise go to step 3.

### 3 Proposed NSDE Algorithm

In this section we describe the proposed NSDE algorithm and discuss the effect of embedding the proposed scheme in basic DE (as given in Section 2) on two simple benchmark examples taken from literature.

#### 3.1 Differential Evolution with Non linear Simplex Method (NSDE)

The NSDE uses nonlinear simplex method (NSM) developed by J. A. Nelder and R. Mead [22], in conjugation with uniform random number to construct the initial population. The procedure of NSDE is outlined as follows:

- 1 Generate a population set P of size NP uniformly as in step 1 of DE and set  $k=1$ .
- 2 Generate a point by NSM Method, which has the following main operations.
  - 2.1 Select  $n+1$  point from population P randomly and evaluate function at these points.
  - 2.2 Calculate the centroid of these points excluding the worst point, say  $X_{\max}$  at which function is maximum.
  - 2.3 *Reflection:* Reflect  $X_{\max}$  through the centroid to a new point  $X_1$ . And calculate the function value at this point.
  - 2.4 *Expansion:* If  $f(X_1) \leq f(X_{\min})$  then perform expansion to generate a new point  $X_2$  in the expanded region otherwise go to step 2.5. If  $f(X_2) < f(X_{\min})$  then include point  $X_2$  to the population Q otherwise point  $X_1$  is included to population Q and go to step 3.
  - 2.5 *Contraction:* If  $f(X_1) < f(X_{\max})$  then produce a new point  $X_3$  by contraction otherwise go to step 2.6. If  $f(X_3) < f(X_{\max})$  then add point  $X_3$  to the population otherwise include point  $X_1$  to the population Q and go to step 3.

2.6 *Reduction*: Reduction is performed when either of the conditions mentioned above (from step 2.3 – step 2.5) are not satisfied. In this step we have replaced the original reduction method of Nelder Mead, by generating uniformly distributed random point say  $X_{rand}$  within the specified range and include it in the population Q and go to step 3.

3 If  $k < NP$  go to step 2.1 with  $k = k+1$  else stop.

It can be seen from the above steps that in NSDE, initially a population set P of size NP is generated uniformly to which NSM is applied NP times to generate another population set Q so as to get a total population of size  $2*NP$ . Finally, the initial population is constructed by selecting the NP fittest points from the union of P and Q. After initialization step, NSDE algorithm works like the basic DE.

With the use of NSM method, the initial population is provided with the information of the good regions that possess each particle as a vertex of the NSM simplex in each step. The algorithm is not computationally expensive, since for each particle of the initial population one function evaluation is done, which is inevitable even if we use a randomly distributed initial population.

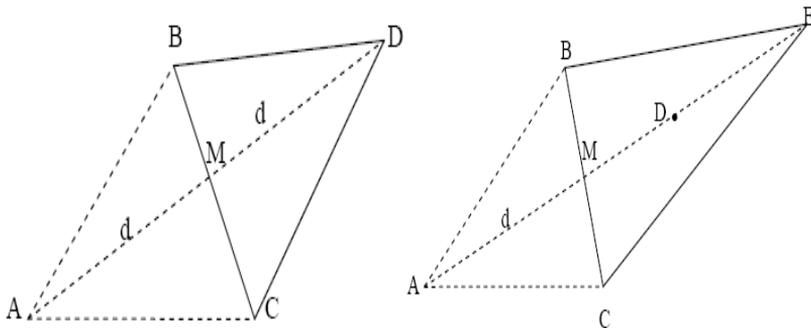


Figure 2  
Reflection of A to D

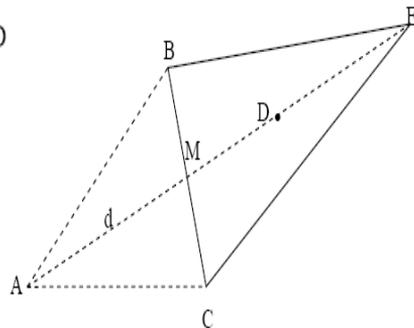


Figure 3  
Expansion of D to E

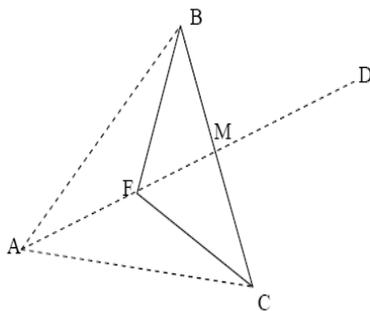


Figure 4  
Contraction of D to F

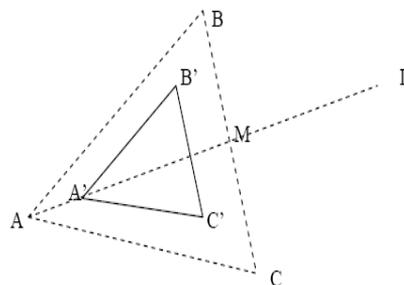


Figure 5  
Reduction of ABC to A'B'C'

### 3.2 Effects of Using the Proposed NSDE to Generate Initial Population

The initial generation of population by nonlinear simplex method makes use of the function value to determine a candidate point for the additional population. As a result in the initial step itself we get a collection of fitter individuals which may help in increasing the efficiency of the algorithm. Consequently, the probability of obtaining the optimum in fewer NFEs increases considerably or in other words the convergence rate of the algorithm becomes faster. The initial generation of 100 points within the range  $[-2, 2]$  for Rosenbrock function and within the range  $[-600, 600]$  for Griewank function using basic DE, ODE and the proposed NSDE are depicted in Figs. 6(a)-6(c) and Figs. 7(a)-7(c) respectively. From these illustrations we can observe that the search space gets concentrated around the global optima which lies at  $(1, 1)$  with objective function value zero, for two dimensional Rosenbrock function and which lies at  $(0, 0)$  with objective function value 0, for Griewank function when the initial population is constructed using NSDE. The large search domain,  $[-600, 600]$ , of Griewank function is contracted to the range of around  $[-400, 400]$  while using NSDE.

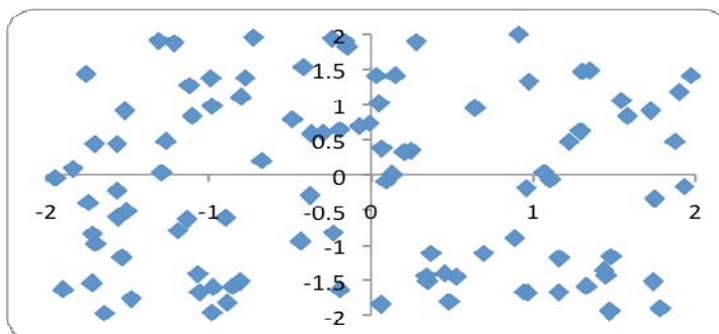


Figure 6 (a)

Initial population consisting of 100 points in the range  $[-2, 2]$  for Rosenbrock function using basic DE

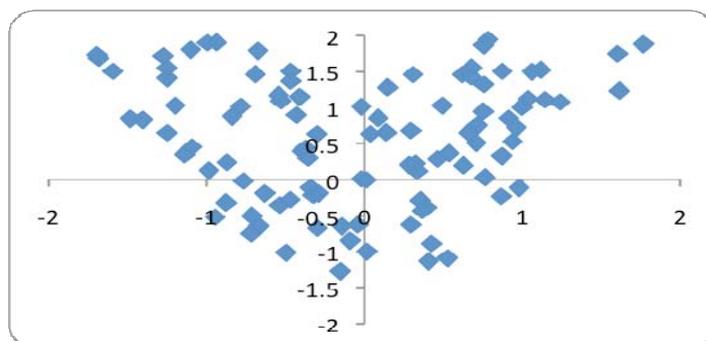


Figure 6(b)

Initial population consisting of 100 points in the range  $[-2, 2]$  for Rosenbrock function using ODE

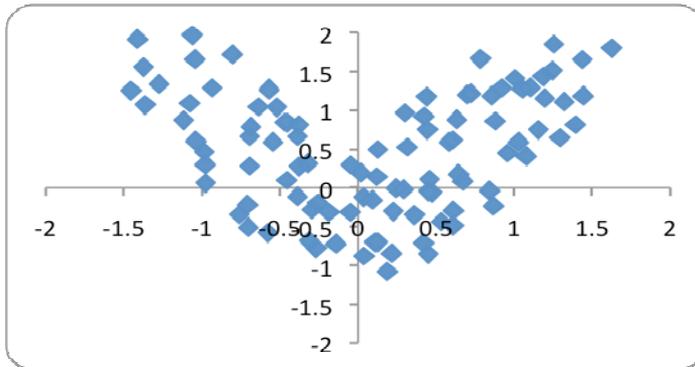


Figure 6 (c)

Initial population consisting of 100 points in the range [-2, 2] for Rosenbrock function using NSDE

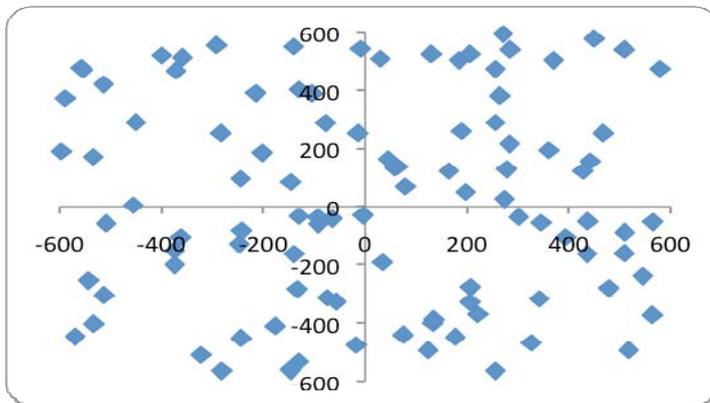


Figure 7 (a)

Initial population consisting of 100 points in the range [-600, 600] for Griewank function using DE

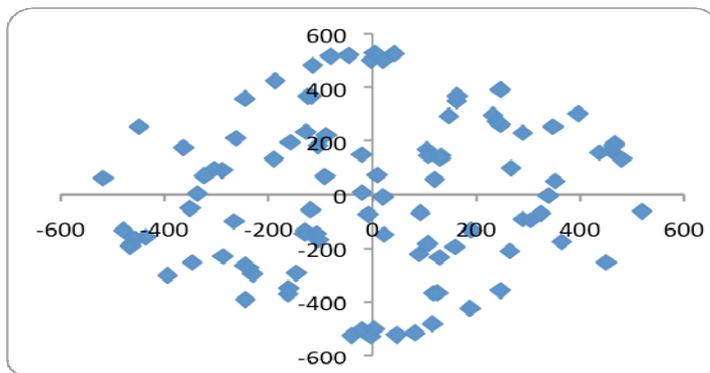


Figure 7 (b)

Initial population consisting of 100 points in the range [-600, 600] for Griewank function using ODE

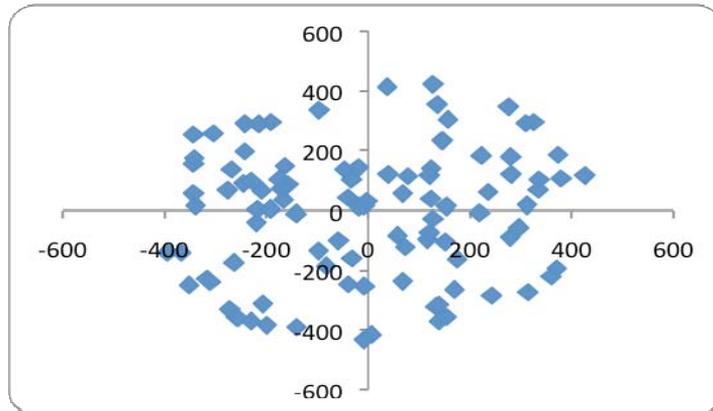


Figure 7 (c)

Initial population consisting of 100 points in the range [-600, 600] for Griewank function using NSDE

## 4 Experimental Setup

With DE, the lower limit for population size, NP, is 4 since the mutation process requires at least three other chromosomes for each parent. While testing the algorithms, we began by using the optimized control settings of DE. Population size, NP can always be increased to help maintain population diversity. As a general rule, an effective NP is between  $3 * n$  and  $5 * n$ , but can often be modified depending on the complexity of the problem. For the present study we performed several experiments with the population size as well as with the crossover rate and mutation probability rate and observed that for problems up to dimension 30 a population size of  $3*n$  is sufficient. But here we have taken fixed population size NP=100, which is slightly large than  $3*n$ . Values of scale F, outside the range of 0.4 to 1.2 are rarely effective, so F=0.5 is usually a good initial choice. In general higher value of  $C_r$  help in speeding up the convergence rate therefore in the present study we have taken  $C_r=0.9$ . All the algorithms are executed on a PIV PC, using DEV C++, thirty times for each problem. In order to be fair we have kept the same parameter settings for all the algorithms. Random numbers are generated using the inbuilt random number generator *rand ( ) function* available in DEV C++.

Over all acceleration rate AR, which is taken for the purpose of comparison is defined as [19]

$$AR = \left( 1 - \frac{\sum_{j=1}^{\mu} NFE(\text{by one algo})_j}{\sum_{j=1}^{\mu} NFE(\text{by other algo})_j} \right) \times 100$$

Where  $\mu$  is number of functions. In every case, a run was terminated when the best function value obtained is less than a threshold for the given function or when the maximum number of function evaluation (NFE= $10^6$ ) was reached. In order to have a fair comparison, these settings are kept the same for all algorithms over all benchmark functions during the simulations.

## 5 Benchmark Problems

The performance of proposed NSDE is evaluated on a test bed of twenty standard, benchmark problems with box constraints, taken from the literature [20]. Mathematical models of the benchmark problems along with the true optimum value are given in Appendix.

## 6 Numerical Results and Comparisons

### 6.1 Performance Comparison of Proposed NSDE with Basic DE and ODE

We have compared the proposed algorithms with the basic DE and ODE. Here we would like to mention that we have used ODE version given in [19] instead of [20] because in [20], the authors have used the additional features like opposition based generation jumping, etc. while in the present study we just focusing on the effect of initial population generation on differential evolution algorithm. Comparisons of the algorithms is done in terms of average fitness function value, standard deviation and the corresponding t-test value; average number of function evaluations and the average time taken by every algorithm to solve a particular problem. In every case, a run was terminated when the best function value obtained is less than a threshold for the given function [19] or when the maximum number of function evaluation (NFE= $10^6$ ) was reached.

From Table 1, which gives the average fitness function value, standard deviation and t-values it can be observed that for the 20 benchmark problems taken in the present study all the algorithms gave more or less similar results in terms of average fitness function value, with marginal difference, which are comparable to true optimum. For the function  $f_{14}$  (Step function) all the algorithms gives same results. The best and worst fitness function values obtained, in 30 runs, by all the algorithms for benchmark problems are given in Table 3.

However when we do the comparison in terms of average time taken and average number of function evaluations then the proposed NSDE emerges as a clear

winner. It converges to the optimum at a faster rate in comparison to all other algorithms. Only for functions  $f_{13}$  (Schwefel) NSDE took more NFE than ODE, whereas for the remaining 19 problems NSDE converged faster than the basic DE and ODE. For solving 20 problems the average NFE taken by NSDE are 1334200 while ODE took 1794738 NFE and DE took 1962733 NFE. This implies that NSDE has an acceleration rate of around 35% in comparison to basic DE and an acceleration rate of 26% in comparison to ODE. ODE on the other hand reduces the NFE only up to 8.56%, in comparison to basic DE. A similar trend of performance can be observed from the average computational time. For solving 20 problems NSDE took least CPU time in comparison to the other two algorithms.

Performance curves of selected benchmark problems are illustrated in Figs. 8(a)-8(h).

Table 1  
Mean fitness, standard deviation of functions in 30 runs and t-value

Function	Dim.	Mean fitness, (Standard deviation) and t-value		
		DE	ODE	NSDE
$f_1$	30	<b>0.0546854</b> (0.0131867) --	0.0901626 (0.0077778) 12.48	0.0916686 (0.00721253) 13.25
$f_2$	30	<b>0.0560517</b> (0.0116127) --	0.0918435 (0.00565233) 14.92	0.0866163 (0.00666531) 12.29
$f_3$	30	0.0957513 (0.00293408) --	0.0952397 (0.00499586) 0.48	<b>0.0951172</b> (0.00405255) 0.68
$f_4$	10	0.0931511 (0.0145175) --	0.0874112 (0.00699322) 1.92	<b>0.0851945</b> (0.0121355) 2.26
$f_5$	30	0.0915561 (0.012111) --	<b>0.0885065</b> (0.00711877) 1.17	0.0916412 (0.00860403) 0.03
$f_6$	30	0.0942648 (0.00478545) --	0.0933845 (0.00620528) 0.60	<b>0.0926704</b> (0.00735851) 0.98
$f_7$	2	<b>4.26112e-008</b> (2.5783e-008) --	6.23824e-008 (2.75612e-008) 2.82	4.9999e-008 (2.95279e-008) 1.01
$f_8$	4	0.0620131 (0.0239495) --	<b>0.0528597</b> (0.0276657) 1.35	0.0591064 (0.0123711) 0.58
$f_9$	30	0.088998 (0.00880246) --	0.092875 (0.00487147) 2.08	<b>0.0882776</b> (0.0103789) 0.29

$f_{10}$	10	-7.91444 (3.40729) --	-9.61563 (0.024986) 2.69	<b>-9.62952</b> (0.0238362) 2.71
$f_{11}$	30	<b>0.0842833</b> (0.00897659) --	0.0890837 (0.00961583) 1.97	0.0901177 (0.00969009) 2.38
$f_{12}$	30	0.0940407 (0.00501821) --	<b>0.0931232</b> (0.00502023) 0.69	0.0951981 (0.00373364) 0.99
$f_{13}$	30	0.0956696 (0.00352899) --	<b>0.0935369</b> (0.00397665) 2.16	0.0955274 (0.00495933) 0.13
$f_{14}$	30	0.0 (0.0) --	0.0 (0.0) --	0.0 (0.0) --
$f_{15}$	30	<b>0.0730003</b> (0.0169434) --	0.0880257 (0.0115251) 3.95	0.0890936 (0.00986588) 4.42
$f_{16}$	2	0.0645903 (0.0231492) --	0.0545825 (0.0263629) 1.54	<b>0.0539806</b> (0.0226797) 1.76
$f_{17}$	30	0.0910662 (0.00428958) --	<b>0.0845474</b> (0.0118228) 2.79	0.0923214 (0.00514694) 1.01
$f_{18}$	2	4.75455e-008 (2.9688e-008) --	3.63292e-008 (3.10335e-008) 1.41	<b>2.657e-008</b> (2.657e-008) 1.49
$f_{19}$	5	0.067335 (0.025448) --	<b>0.0738969</b> (0.0209749) 1.07	0.0769911 (0.0160823) 1.73
$f_{20}$	5	-3.99239 (0.00164918) --	<b>-3.99398</b> (0.00235545) 2.98	-3.99297 (0.00184151) 1.27

Table 2

Average CPU time (in sec) taken by the algorithms, mean number of function evaluation of 30 runs and over all acceleration rates

Fu n.	Dim.	Average Time (Sec)			Mean function		
		DE	ODE	NSDE	DE	ODE	NSDE
$f_1$	30	0.60	0.54	<b>0.51</b>	28020	26912	<b>26220</b>
$f_2$	30	0.60	0.57	<b>0.55</b>	37312	36639	<b>35500</b>
$f_3$	30	11.30	11.10	<b>10.40</b>	295232	295112	<b>232860</b>
$f_4$	10	2.41	2.34	<b>2.13</b>	382454	361234	<b>265420</b>
$f_5$	30	1.80	1.79	<b>1.70</b>	54503	53305	<b>52240</b>
$f_6$	30	1.50	1.45	<b>1.41</b>	52476	51589	<b>49170</b>
$f_7$	2	0.31	0.29	<b>0.23</b>	3845	3740	<b>3520</b>
$f_8$	4	0.10	0.11	<b>0.09</b>	7902	7934	<b>6780</b>
$f_9$	30	1.20	1.11	1.11	44034	41455	<b>35330</b>
$f_{10}$	10	3.02	2.93	<b>2.67</b>	220356	196871	<b>172200</b>
$f_{11}$	30	2.91	2.37	<b>1.81</b>	200924	196617	<b>44960</b>
$f_{12}$	30	0.59	0.52	<b>0.50</b>	66154	63760	<b>57800</b>
$f_{13}$	30	1.83	<b>1.24</b>	1.42	197069	<b>148742</b>	155970
$f_{14}$	30	0.71	0.65	<b>0.53</b>	42423	41578	<b>32300</b>
$f_{15}$	30	0.47	0.45	<b>0.43</b>	25903	24236	<b>22620</b>
$f_{16}$	2	0.13	0.11	<b>.10</b>	3913	3832	<b>3600</b>
$f_{17}$	30	0.92	0.81	<b>0.76</b>	55029	52455	<b>47760</b>
$f_{18}$	2	0.23	.21	<b>0.16</b>	7367	7249	<b>5150</b>
$f_{19}$	5	1.66	1.12	<b>0.61</b>	205398	150173	<b>57540</b>
$f_{20}$	5	0.39	0.37	<b>0.31</b>	32419	31305	<b>27260</b>
<b>Total</b>		32.68	30.08	27.43	1962733	1794738	<b>1434200</b>
<b>AR</b>			7.955%	16.064%		8.5592 %	26.928 %

Table 3  
Best and worst fitness function values obtained by all the algorithms

Function	Dim.	Best and Worst function values		
		DE	ODE	NSDE
$f_1$	30	0.0533706 0.0920816	0.0710478 0.0980475	0.0801175 0.0989173
$f_2$	30	0.0469366 0.0852506	0.0834493 0.0994759	0.0708503 0.0971761
$f_3$	30	0.0912359 0.099449	0.0812952 0.0991723	0.085954 0.0987729
$f_4$	10	0.0555946 0.0973456	0.0782872 0.0990834	0.0586798 0.0986525
$f_5$	30	0.0550155 0.0985525	0.0765341 0.0976009	0.0730851 0.0988916
$f_6$	30	0.0811647 0.0995538	0.0799383 0.0992613	0.0777488 0.0979521
$f_7$	2	3.03242e-009 8.24678e-008	1.9059e-008 9.47894e-008	5.64424e-009 8.50966e-008
$f_8$	4	0.0139037 0.0974824	0.00826573 0.0912189	0.0333435 0.0790444
$f_9$	30	0.0746445 0.0995713	0.0849655 0.098311	0.064171 0.0992847
$f_{10}$	10	-9.64801 -1.02642	-9.65114 -9.59249	-9.65368 -9.57805
$f_{11}$	30	0.0627431 0.0944119	0.0636232 0.0989899	0.0683468 0.0994605
$f_{12}$	30	0.0849009 0.0991914	0.0819181 0.0999306	0.0887407 0.0997806
$f_{13}$	30	0.0902771 0.0996024	0.0866648 0.0988438	0.0854635 0.0998667
$f_{14}$	30	0.0 0.0	0.0 0.0	0.0 0.0
$f_{15}$	30	0.0441712 0.0945989	0.0593778 0.0991945	0.067818 0.0992816
$f_{16}$	2	0.0277478 0.0969767	0.0129757 0.0984134	0.0297125 0.0918671
$f_{17}$	30	0.0844465 0.0975161	0.0593988 0.0997203	0.0836182 0.0996487
$f_{18}$	2	2.3063e-009 9.91416e-008	4.64862e-009 9.55725e-008	1.11418e-008 8.51751e-008
$f_{19}$	5	0.00746793 0.099394	0.0291998 0.0995076	0.0483536 0.0999149
$f_{20}$	5	-3.99626 -3.99019	-3.99737 -3.99035	-3.99644 -3.99027

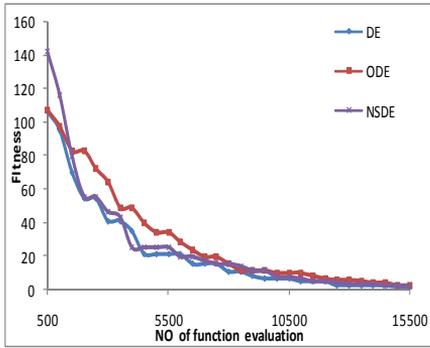


Figure 8(a)  
Sphere ( $f_1$ ) function

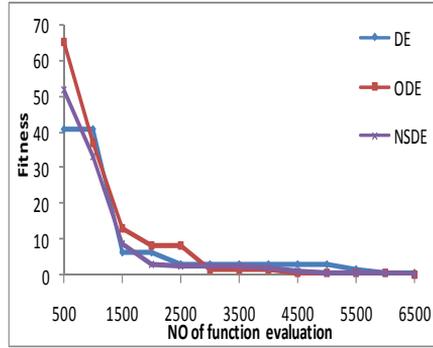


Figure 8(b)  
Colville ( $f_8$ ) function

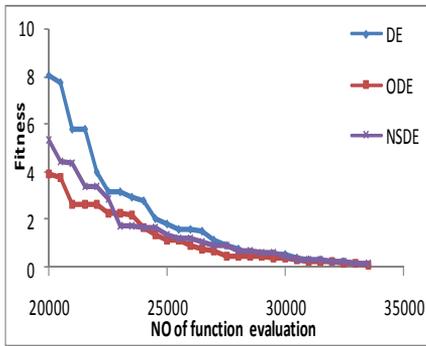


Figure 8(c)  
Axis parallel ( $f_2$ ) function

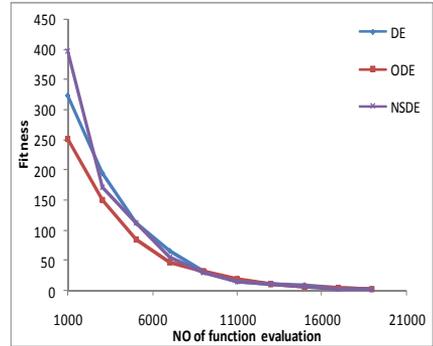


Figure 8(d)  
Griewenk ( $f_3$ ) function

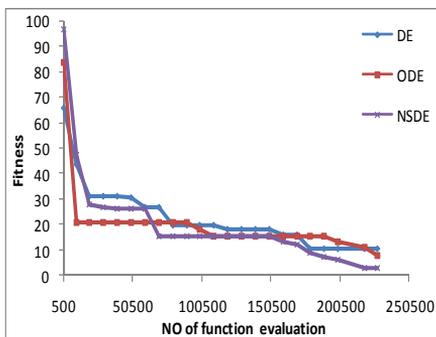


Figure 8(e)  
Restigin ( $f_4$ ) function

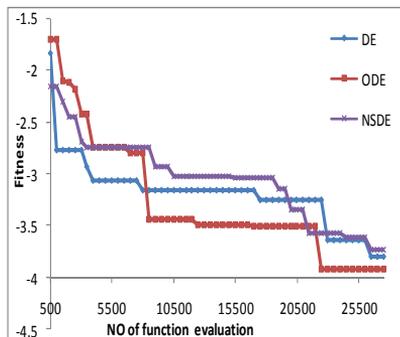


Figure 8(f)  
Inverted ( $f_{20}$ ) cosine

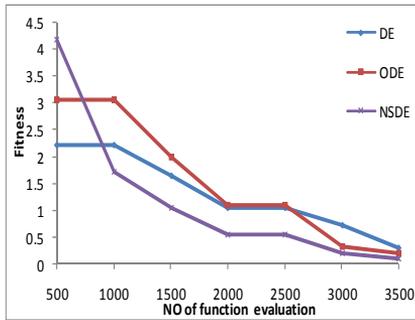


Figure 8(g)  
Tripod ( $f_{16}$ ) function

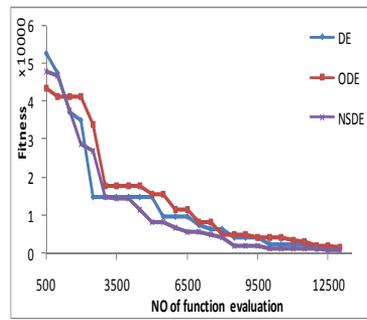


Figure 8(h)  
Step ( $f_{14}$ ) function

Figures 8(a)-8(h)  
Performance curves of few elected benchmark problems

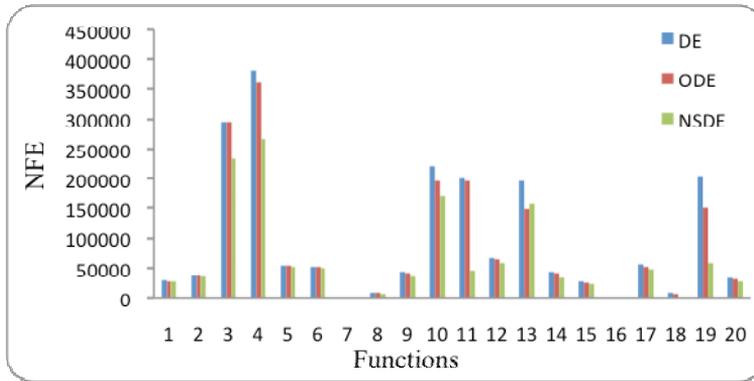


Figure 9

Average number of function evaluations taken by DE, ODE and NSDE for the 20 benchmark problems

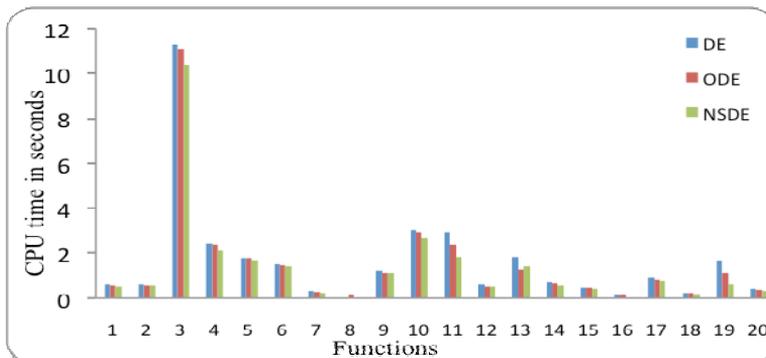


Figure 10

Average CPU time (in sec) taken by DE, ODE and NSDE for the 20 benchmark problems

## Discussion and Conclusions

In the present paper we have proposed a simple and modified variant of DE namely NSDE. The only structural difference between the proposed NSDE and the basic DE lies in the generation of initial population. NSDE applies nonlinear simplex method to generate the initial population. From the empirical studies and graphic illustrations we can say that the proposed schemes enhance the working of basic DE in terms of average CPU time and NFEs without compromising with the quality of solution. Also we would like to mention that other than the process of initial population construction, we have not made use of any other additional feature/parameter in the basic structure of DE. Though we have applied the proposed schemes in basic DE, they can be applied in any evolutionary algorithm which makes use of randomly generated initial points. As a concluding statement it can be said that providing the initial population with some extra information of search space is an important help for the DE algorithm, since it may lead to faster convergence and better quality of the solutions provided by the algorithm.

## References

- [1] R. Storn, K. Price, "DE-a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Space", *Technical Report TR-95-012, ICSI, March 1995. Available via the Internet: ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.ps.Z*, 1995
- [2] R. Storn, K. Price, "DE-a Simple and Efficient Heuristic for Global Optimization over Continuous Space", *Journal of Global Optimization* 11(4), 1997, pp. 41-359
- [3] K. Price, "An Introduction to DE", In: *Corne, D., Marco, D., Glover, F. (eds.), New Ideas in Optimization, McGraw-Hill, London (UK)*, 1999, pp. 78-108
- [4] F. S. Wang, W. M. Cheng, "Simultaneous Optimization of Feeding Rate and Operation Parameters for Fed-Batch Fermentation Processes", *Biotechnology Progress* 15, 1999, pp. 949-952
- [5] B. V. Babu, S. A. Munawar, "Differential Evolution for the Optimal Design of Heat Exchangers", In: *Proceedings of All-India seminar on Chemical Engineering Progress on Resource Development: A Vision 2010 and Beyond, Bhuvaneshwar*, 2000
- [6] B. V. Babu, R. P. Singh, "Synthesis & Optimization of Heat Integrated Distillation Systems Using Differential Evolution", In: *Proceedings of All-India seminar on Chemical Engineering Progress on Resource Development: A Vision 2010 and Beyond, Bhuvaneshwar*, 2000
- [7] R. Angira, B. V. Babu, "Optimization of Non-Linear Chemical Processes Using Modified Differential Evolution (MDE)", In: *Proceedings of the 2<sup>nd</sup> Indian International Conference on Artificial Intelligence, Pune, India, 2005a*, pp. 911-923

- 
- [8] R. Angira, B. V. Babu, "Optimization of Process Synthesis and Design Problems: A Modified Differential Evolution Approach", *Chemical Engineering Science* 61, 2006a, pp. 4707-4721
- [9] B. V. Babu, R. Angira, "Optimization of Thermal Cracker Operation using Differential Evolution", *In: Proceedings of International Symposium & 54<sup>th</sup> Annual Session of IChE, Chennai, 2001b*
- [10] B. V. Babu, R. Angira, "Optimization of Water Pumping System Using Differential Evolution Strategies", *In: Proceedings of The Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, Singapore, 2003*
- [11] M. H. Lee, C. Han, K. S. Chang, "Dynamic Optimization of a Continuous Polymer Reactor Using a Modified Differential Evolution Algorithm", *Industrial and Engineering Chemistry Research* 38, 1999, pp. 4825-4831
- [12] J. C. Lu, F. S. Wang, "Optimization of Low Pressure Chemical Vapor Deposition Reactors Using Hybrid Differential Evolution", *Canadian Journal of Chemical Engineering* 79, 2001, pp. 246-254
- [13] P. K. Bergey, C. Ragsdale, "Modified Differential Evolution: a Greedy Random Strategy for Genetic Recombination", *omega* 33, 2005, pp. 255-265
- [14] P. Kaleo, M. M. Ali, "A Numerical Study of Some Modified Differential Evolution Algorithms", *European journal of operational research* 169, 2006, pp. 1176-1184
- [15] M. M. Ali, "Differential Evolution with Preferential Crossover", *European Journal of Operational Research* 181, 2007, pp. 1137-1147
- [16] N. Noman, H. Iba, "Enhancing Differential Evolution Performance with Local Search for High Dimensional Function Optimization", *GECCO'05 Washington, DC, USA, 2005*, pp. 967-974
- [17] A. K. Qin, P. N. Suganthan, "Self Adaptive Differential Evolution Algorithm for Numerical Optimization", *IEEE*, 2005, pp. 1785-1791
- [18] H. Y. Fan, J. Lampinen, G. S. Dulikravich, "Improvements to Mutation Donor Formulation of Differential Evolution", *International Congress on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, EUROGEN, 2003, pp. 1-12
- [19] Shahryar Rahnamayan, H. R. Tizhoosh, M. M. A. Salama, "A Novel Population Initialization Method for Accelerating Evolutionary Algorithms", *computer and applied mathematics with application* 53, 2007, pp. 1605-1614
- [20] S. Rahnamayan, H. R. Tizhoosh, M. A. Salman, "Opposition-based Differential Evolution", *IEEE transaction on evolutionary computation* 12(1), 2008, pp. 64-79
-

- [21] M. M. Ali, A. Torn, "Population Set-based Global Optimization Algorithms: Some Modifications and Numerical Studies", [www.ima.umn.edu/preprints/](http://www.ima.umn.edu/preprints/), 2003
- [22] J. A. Nelder, R. Mead, "A Simplex Method for Function Minimization", *computer journal*, Vol. 7, 1965, pp. 308-313
- [23] S. Kimura, K. Matsumura, "Genetic Algorithms using low discrepancy sequences", in *proc. of GECCO 2005*, pp. 1341-1346
- [24] Nguyen X. H., Nguyen Q. Uy., R. I. McKay, P. M. Tuan, "Initializing PSO with Randomized Low-Discrepancy Sequences: The Comparative Results", In *Proc. of IEEE Congress on Evolutionary Algorithms*, 2007, pp. 1985-1992
- [25] S. Kimura, K. Matsumura, "Genetic Algorithms Using Low Discrepancy Sequences", in *proceedings of the GECCO, 2005*, pp. 1341-1346
- [26] KE Parsopoulos, MN Vrahatis, "Initializing the Particle Swarm Optimization Using Nonlinear Simplex Method", in *advances in intelligent systems, fuzzy systems, evolutionary computation*, WSEAS press, 216-221, 2002
- [27] Millie Pant, Radha Thangaraj, Crina Grosan, Ajith Abraham "Improved Particle Swarm Optimization with Low-Discrepancy Sequences", In *Proc. of IEEE Congress on Evolutionary Algorithms*, 2008, pp.
- [28] Millie Pant, Musrrat Ali, V. P. Singh "Differential Evolution with Parent Centric Crossover", In *Proc. of IEEE Congress on Evolutionary Algorithms*, 2008, pp. 141-146

## Appendix

1 Sphere function:

$$f_1(x) = \sum_{i=1}^n x_i^2, \text{ With } -5.12 \leq x_i \leq 5.12, \min f_1(0, \dots, 0) = 0$$

2 Axis parallel hyper-ellipsoid:

$$f_2(x) = \sum_{i=1}^n ix_i^2, \text{ with } -5.12 \leq x_i \leq 5.12, \min f_2(0, \dots, 0) = 0$$

3 Rosenbrock's valley:

$$f_3(x) = \sum_{i=1}^{n-1} [100(x_{i+1}^2 - x_i^2) + (1 - x_i^2)] \text{ With } -2 \leq x_i \leq 2, \min f_3(1, \dots, 1) = 0$$

4 Restigin's function:

$$f_4(x) = 10n + \sum_{i=1}^n (x_i^2 - 10 \cos(2\pi x_i)) \text{ With } -5.12 \leq x_i \leq 5.12, \min f_4(0, \dots, 0) = 0$$

5 Griewenk function:

$$f_5(x) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad \text{With } -600 \leq x_i \leq 600, \min$$

$$f_5(0, \dots, 0) = 0$$

6 Ackley's function:

$$f_6(X) = -20 * \exp\left(-.2 \sqrt{1/n \sum_{i=1}^n x_i^2}\right) - \exp\left(1/n \sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e,$$

$$\text{With } -32 \leq x_i \leq 32, \quad \min f_6(0, \dots, 0) = 0$$

7 Beale function:

$$f_7(x) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$$

$$\text{With } -4.5 \leq x_i \leq 4.5, \min f_7(3, 0.5) = 0$$

8 Colville function:

$$f_8(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2)$$

$$19.8(x_2 - 1)(x_4 - 1)$$

$$\text{,With } -10 \leq x_i \leq 10, \min f_8(1, 1, 1, 1) = 0$$

9 Levy function:

$$f_9(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n))$$

$$\text{With } -10 \leq x_i \leq 10, \min f_9(1, \dots, 1) = 0$$

10 Michalewicz function:

$$f_{10}(x) = - \sum_{i=1}^n \sin(x_i) (\sin(ix_i^2 / \pi))^{2m} \quad \text{With } 0 \leq x_i \leq \pi, m=10, \min$$

$$f_{10(n=10)} = -9.66015$$

11 Zakharov function:

$$f_{11}(x) = \sum_{i=1}^n x_i^2 + \left(\sum_{i=1}^n 0.5ix_i\right)^2 + \left(\sum_{i=1}^n 0.5ix_i\right)^4 \quad \text{With } -5 \leq x_i \leq 10, \min$$

$$f_{11}(0, \dots, 0) = 0$$

12 Schwefel's problem 2.22:

$$f_{12}(x) = \sum_{i=1}^n |x_i| + \prod_{i=1}^n |x_i| \quad \text{With } -10 \leq x_i \leq 10, \min f_{12}(0, \dots, 0) = 0$$

13 Schwefel's problem 2.21:

$$f_{13}(x) = \max_i \{|x_i|, 1 \leq i \leq n\} \text{ With } -100 \leq x_i \leq 100, \min f_{13}(0, \dots, 0) = 0$$

14 Step function:

$$f_{14}(x) = \sum_{i=1}^n (\lfloor x_i + 0.5 \rfloor)^2 \text{ With } -100 \leq x_i \leq 100, \min$$

$$f_{14}(-0.5 \leq x_i \leq 0.5) = 0$$

15 Quartic function:

$$f_{15}(x) = \sum_{i=1}^n ix_i^4 + \text{random}[0, 1] \text{ With } -1.28 \leq x_i \leq 1.28, \min$$

$$f_{15}(0, \dots, 0) = 0$$

16 Tripod function:

$$f_{16}(x) = p(x_2)(1 + p(x_1)) + |(x_1 + 50p(x_2)(1 - 2p(x_1)))| + |(x_2 + 50(1 - 2p(x_2)))|$$

$$\text{With } -100 \leq x_i \leq 100, \min$$

$$f_{16}(0, -50) = 0 \text{ where } p(x) = 1 \text{ for } x > 0 \text{ otherwise } p(x) = 0$$

17 Alpine function:

$$f_{17}(x) = \sum_{i=1}^n |x_i \sin(x_i) + 0.1x_i| \text{ With } -10 \leq x_i \leq 10, \min f_{17}(0, \dots, 0) = 0$$

18 Cshaffér's function 6:

$$f_{18}(x) = 0.5 + \frac{\sin^2 \sqrt{(x_1^2 + x_2^2)} - 0.5}{1 + 0.01(x_1^2 + x_2^2)^2} \text{ With } -10 \leq x_i \leq 10, \min$$

$$f_{18}(0, 0) = 0$$

19 Pathological function:

$$f_{19}(x) = \sum_{i=1}^{n-1} \left( 0.5 + \frac{\sin^2 \sqrt{(100x_i^2 + x_{i+1}^2)} - 0.5}{1 + 0.001(x_i^2 + x_{i+1}^2 - 2x_i x_{i+1})^2} \right)$$

$$\text{With } -100 \leq x_i \leq 100, \min f_{19}(0, \dots, 0) = 0$$

20 Inverted cosine wave function:

$$f_{20}(x) = -\sum_{i=1}^{n-1} \left( \exp\left(\frac{-(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1})}{8}\right) \cos(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}}) \right)$$

$$\text{With } -5 \leq x_i \leq 5, \min f_{20}(0, \dots, 0) = -n + 1$$