Iterative optimization RCO: A "Ruler & Compass" deterministic method

Maurice Clerc

Iterative optimization — RCO: A "Ruler & Compass" deterministic method

Maurice Clerc*

2024-09-27 DRAFT

Abstract

We present here the basic version of an iterative optimization algorithm that is deterministic, needs just one parameter, and often capable of finding a good solution after very few evaluations of the fitness function. We explain its principles using a multimodal one-dimensional problem. For such problems, it could be applied using nothing more than a ruler and a compass. We provide classical examples illustrating the algorithm's properties, as well examples where it fails. As this version does not address possible stagnation, it is really only useful for low-dimensional problems (typically at most ten), where each evaluation of a position in the search space is very costly in terms of computational resources.

1 Introduction

Many optimization problems, especially in engineering, have the following characteristics: they can only be solved by stochastic iterative methods; the number of variables is not very high; each iteration requires very costly computer resources. In addition, the user often has to adjust several parameters and many tests are required. This is why we propose a deterministic algorithm that can provide a good solution in very few iterations even on problems that are usually seen as difficult, and for which just one parameter is needed (and moreover often easy to define).

Several definitions of the theoretical *a priori* difficulty of an optimization problem (minimization here) have been proposed ([3]). For example, one can consider the number of local minima and the relative size of the subspace of acceptable solutions. If this number is high and the size small, then any iterative algorithm functioning as a black box would require numerous samples of positions in the search space and, therefore, many evaluations of positions.

Consider the function with the landscape shown in Figure 1 where the minimum is 0.999507 at position 2.412616. Although it is one-dimensional, its theoretical difficulty is high because it has many local minima.

For the algorithms tested below, we say that a solution is acceptable if its value is less than 0.9997.

For stochastic algorithms, we estimate the probability of success by running them a thousand times.

Table 1 1 then shows that, for these, many evaluations are needed to have a good chance of finding a satisfactory solution and, therefore, the actual difficulty (large number of evaluations) is in good agreement with the theoretical difficulty.

However, it is possible to define a deterministic (in the non-stochastic sense) algorithm, for which this agreement is challenged. On this problem, the Ruler & Compass Optimizer (RCO) is extremely efficient (see figure 2). More generally, it is also effective on other supposedly difficult problems. And conversely, it can be highly ineffective on problems that are supposed to be easy.

*Maurice.Clerc@WriteMe.com

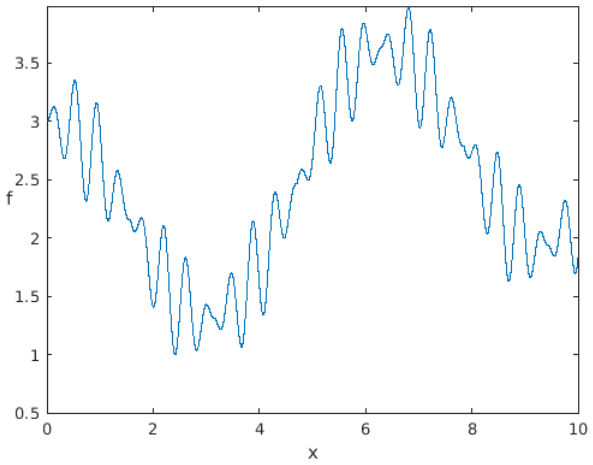


Figure 1: Test landscape sinCos15. Many local minima, very small set of acceptable solutions.

Thus, the mere existence of such an algorithm largely calls into question the relevance of classical difficulty estimates, which will have to be revised, at least by specifying their field of application in the space of optimizers.

Informally, the No Free Lunch Theorem ([6]) allows us to say that there is no such thing as an efficient optimizer for all problems. If "efficient" is to be interpreted as "finding a solution without too much difficulty", then what RCO suggests is a dual proposition: there is no *a priori* measure of difficulty valid for all optimizers.

Analyzing this conjecture is not the purpose of this study. In what follows, we'll look at the detailed operation of RCO and show on other examples how its behavior is atypical. This will allow us to sketch a typology of problems for which it is interesting to use it.

2 RCO principle

On our Test landscape RCO finds an acceptable solution for sure after 20 evaluations (see the figure 2).

However to detail the iterative construction it is better to consider a simpler example, defined by

$$f(x) = (x-3)^2 \ for\, x \in [0, 10] \tag{1}$$

Its minimum is obviously 0 on $x = 3$.

For this basic version an user-defined parameter is needed: a fixed lower bound. Note that in practice we often know one. Moreover RCO is in fact able to use an adaptive one, automatically estimated during the process (see Appendix 8.5), but the figures are then more difficult to understand.

Here we set this lower bound to -5, to more clearly see what happens, although -0.1 would be sufficient.

Tables 2 to 6 illustrate the process. Of course, in higher dimension D the lines become (hyper-)planes but the method is the same. At each time step we consider D planes defined by $D+1$ points and their intersection point [1] with the lower bound plane. There are two cases:

[1] It may happen that there is more than an unique intersection point. If so we consider we are in the case 1.

Table 1: On the problem in figure 1, metaheuristics, as expected, sometimes struggle to find an acceptable solution with a high probability, estimated over 1000 executions. Since RCO is deterministic, only one run is required.

Algorithm	Stochastic	Evaluations/run	Success rate
CMA-ES	*	10 000	0.12
DE	*	10 000	0.73
ACO	*	410 4010	0.80 0.94
Jaya	*	1000	0.985
SPSO	*	180 190	0 1
GA-MPC	*	80 90	0 1
RCO		*20*	*1*

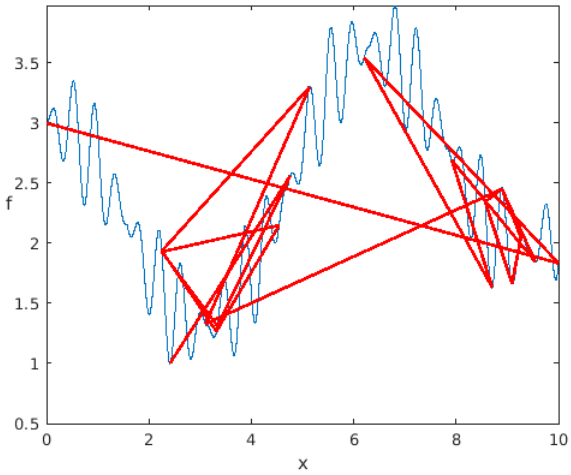


Figure 2: RCO on the test landscape sinCos15. Path followed (the first position is 0). An acceptable solution is found after 20 evaluations.

1. If its position lies outside the search space (possibly at infinity), then the new position is determined by a barycenter of the 2^D previous positions. For the calculation of the barycenter, the weight of each position is inversely proportional to the value of the function at that position. See the detail in Appendix 8.2.

2. If its position is within the search space, then it is kept as the new position.

Then, in the list of positions, the first one is removed and the new one is added.

Table 2: Principle. Step 1

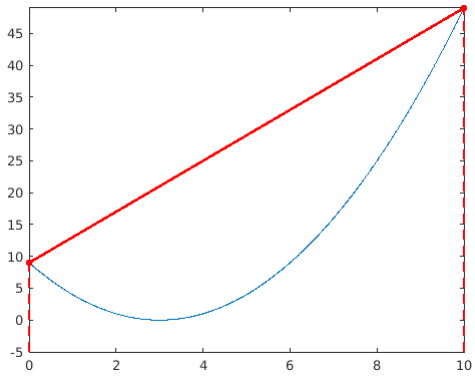
An horizontal lower bound is defined. The "corners" are evaluated. The line connecting them intersects the lower bound beyond the search space, this intersection point is rejected and will be replaced with another one (step 2).


Table 3: Principle. Step 2

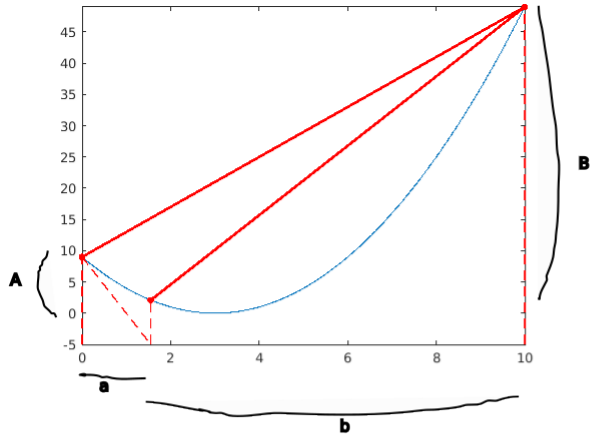
The new position is defined so that $a/b = A/B$. As the dimension is 1 it can be done thanks to a "Ruler & Compass" construction. Then the position is evaluated to define the third point.


Table 4: Principle. Step 3

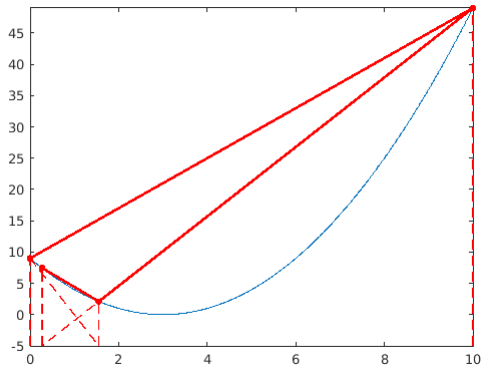
Now the line joining the two last points does intersect the lower bound on a point whose position is inside the search space. This position is then maintained and evaluated. The same process is repeated, again ...


Table 5: Principle. Step 9, zoom

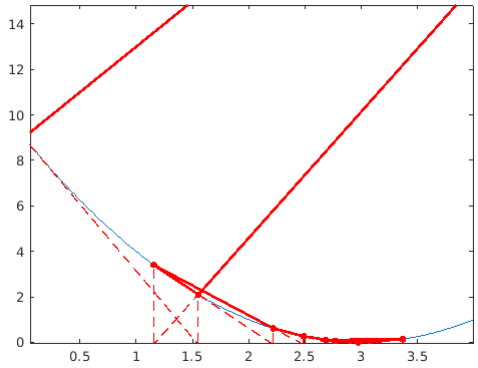
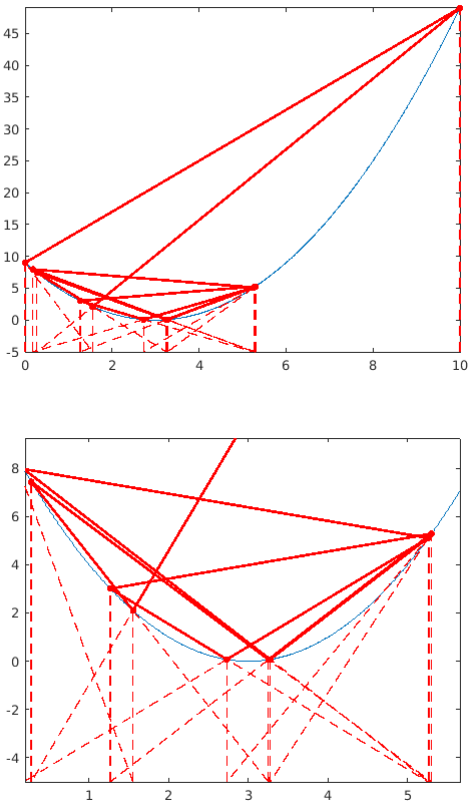
... and again.


Table 6: Principle. Step 10, and zoom

After Step 9, the line connecting the last two points is nearly horizontal and intersects the lower boundary outside of the search space. Consequently, we generate a new position using the same method as in Step 2. This occurrence becomes increasingly frequent as the positions approach the solution. Hence, the new position often approximates the mean of the last two positions, which is a favorable behavior for better approaching the solution.


3 Examples

The definitions of the problems are given in Appendix 8.7. Some of them have been shifted in order to be not too easy.

3.1 Six Hump Camel Back

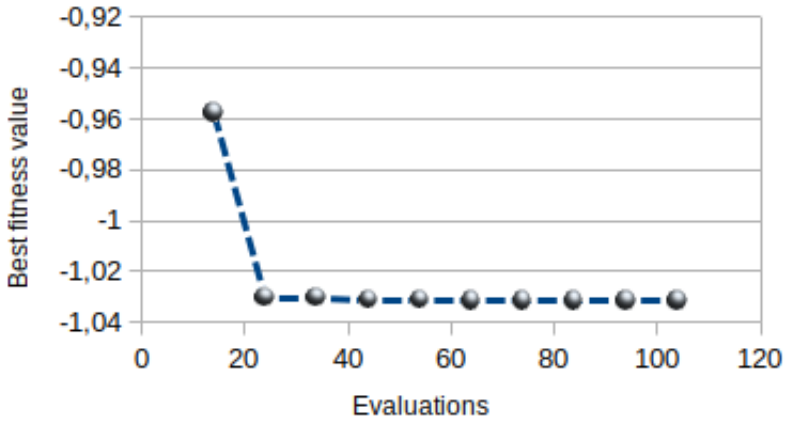
The dimension of the problem is two, so the landscape has four corners. The search space is $[-2, 2] \times [-1, 1]$.

The minimum is -1.031628453489877. A sure lower bound is -1.1. Table 7 shows the results for increasing

number of evaluations and Figure 3 the construction for 14 evaluations.

Table 7: Six Hump Camel Back

Constructed points (evaluations)	Best fitness
14	-0,957541
24	-1,030227
34	-1,030227
44	-1,031227
54	-1,031227
64	-1,031473
74	-1,031473
84	-1,031473
94	-1,031473
104	-1,031473



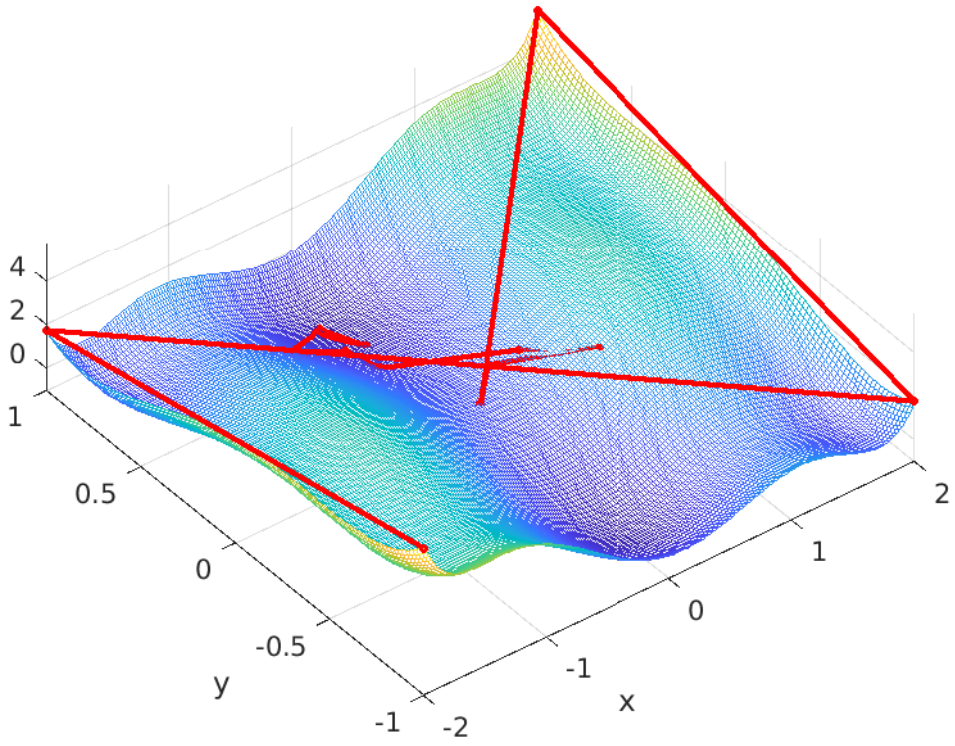


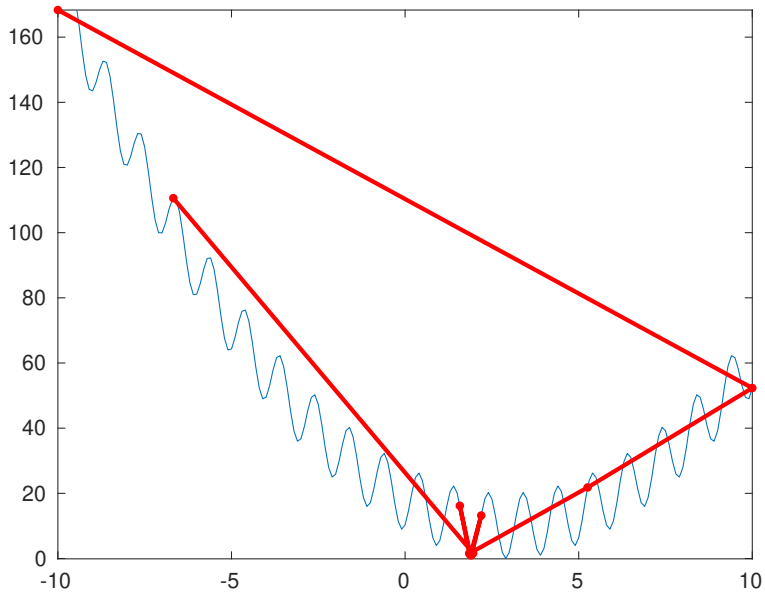
Figure 3: Six Hump Camel Back. Lower bound -1.1. After the construction of 4+10 points the result is -0.9575.

3.2 Shifted Rastrigin

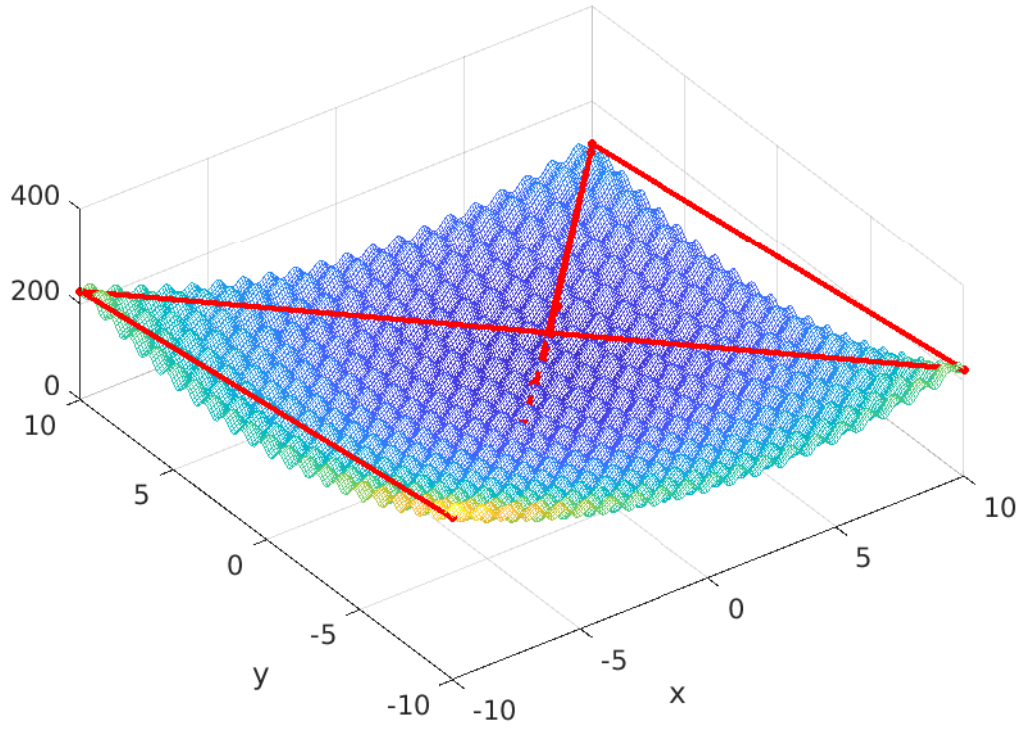
The minimum is zero, so a sure lower bound is -0.1. Table 8 shows the results for different dimensions and different numbers of evaluations and Figure 4 the constructions for 1D and 2D landscapes after 12 and 14 evaluations respectively.

Table 8: Shifted Rastrigin

Dimension	Constructed points (evaluations)	Best fitness
1	52	0.000557
2	1004	0.001511
3	1508	0.002548
4	50016	0.001735
5	250032	0.004518



(a) 1D, 12 points



10

(b) 2D, 14 points

Figure 4: Shifted Rastrigin

3.3 Rosenbrock

The minimum is zero, so a sure lower bound is -0.1. Table 9 shows the results for dimensions two to five with an increasing number of evaluations.

Table 9: Rosenbrock

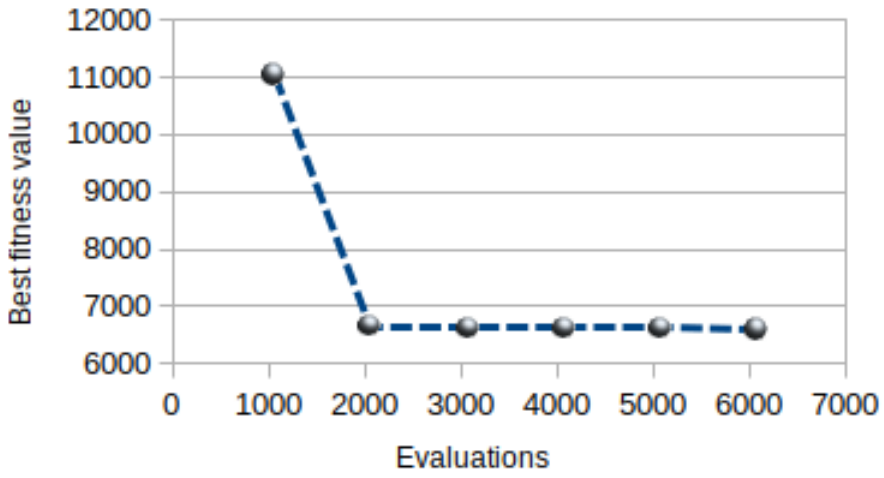
Dimension	Evaluations	Best fitness
2	252	0.002788
3	1508	0.001003
4	10016	0.003756
5	15032	0.005719

3.4 Pressure Vessel

There are four values to find, the two first ones are discrete. The minimum value is 6059.714335048436, as proved in [7].

The lower bound used here is 6000. Table 10 show the results for an increasing number of evaluations. Due to the discrete variables, the algorithm encounters difficulties in converging.

Table 10: Pressure Vessel



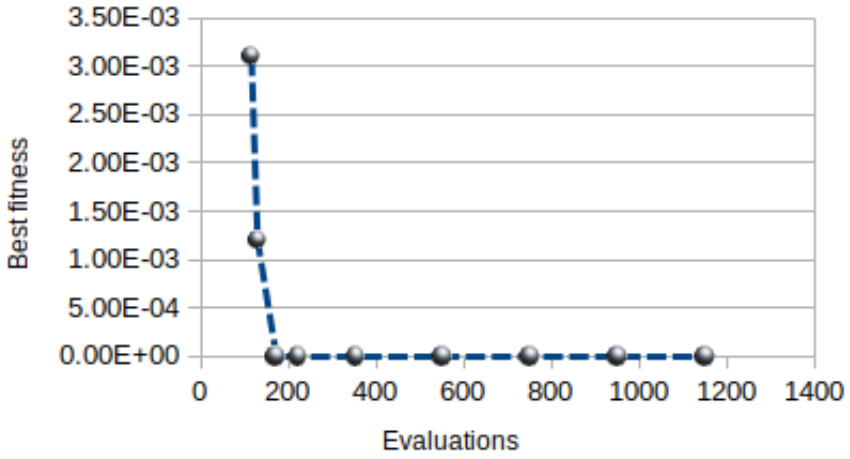
Evaluations	Best fitness
1052	11056.81
2049	6646.71
3074	6615.977
4074	6615.977
5074	6615.977
6076	6592.011

3.5 Gear Train

There are four values to find, all discrete. The minimum is 2.700857×10^{-12}.

Here the lower bound is simply set to 0. A we can see on Table 11 the algorithm quickly proposes a solution that is far from optimal, and then stagnates without further improvement.

Table 11: Gear Train. Discretization is applied only at the very end to all values, so more search effort does not always imply a better result.



Evaluations	Best fitness
116	3.10E-03
130	1.20E-03
173	6.65E-09
223	6.65E-09
1753	6.65E-09

4 Comparison

Let's compare RCO with a slightly improved version of Standard PSO (SPSO-Dicho on my technical website ([4]). To favor PSO, I've selected the best run out of five. The results are presented in 12. Also, refer to the figure 5 for Shifted Griewank, where the two methods appear to be equivalent, particularly for dimensions 1 to 5 and the number of evaluations respectively set to 100, 200, 5000, 10000, and 20000.

In lower dimensions (less than 10), RCO performs better and sometimes significantly so. However, for higher dimensions, RCO stagnates while PSO continues to find better solutions, as evidenced by Shifted Rastrigin and Rosenbrock.

Table 12: RCO vs PSO

Problem	Dimension	Evaluations	RCO	PSO
Six Hump Camel Back	2	54	**-1.031227**	-1.011030
Shifted Rastrigin	1	52	**0.000557**	0.510724
	2	1004	**0.001511**	0.4397499
	5	250032	**0.004578**	0.994959
	10	251024	35.246	**1.9899**
Rosenbrock	2	252	**0.002788**	266.94
	5	15032	**0.005719**	0.0864
	10	101024	8.996	**5.19e-08**
Pressure Vessel	4	3074	**6615.977**	6890.347
		5074	**6615.977**	6821.931
		20074	**6583.75**	6820.410
Gear Train	4	173	6.65e-09	**2.35e-09**
		1154	6.65e-09	**2.35e-09**
		199752	**1.18e-09**	1.26e-09

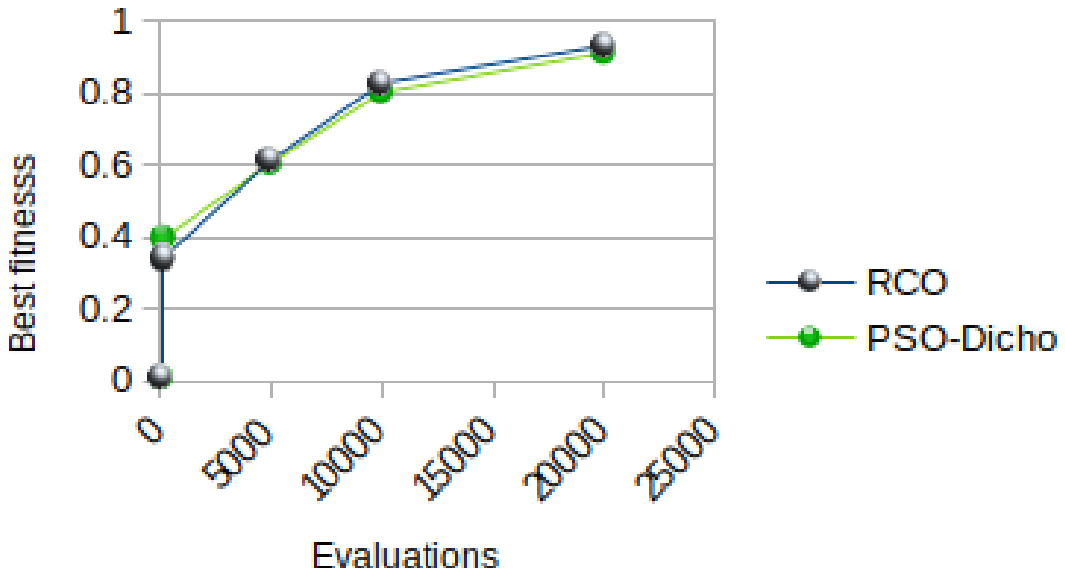


Figure 5: Shifted Griewank. RCO and PSO are equivalent.

5 Complexities

To assess an algorithm, it's common practice to conduct a theoretical analysis of both its space complexity and time complexity. However, as explained in [2], this approach isn't always appropriate, and a more reliable method is to estimate the actual memory usage and computational time.

In this context, there are two possible scenarios at each time step:

1. Solving a linear system of D equations to determine a new position with D coordinates. The required space is $D^2 + D$, with a time complexity of $O(D^3)$.

2. Computing a linear combination of 2^D positions, each with D coordinates. The required space is $O(D \times 2^D)$, and the number of multiplications is $O(2^D)$.

However such a classical reasoning does not correspond to the behavior of the algorithm in practice. The point is that the second situation usually does not occur very often except at the end of the process. Moreover it depends on the landscape of the problem and also on the "budget" (the maximum acceptable number of evaluations).

Let's consider, for example, the computing time per evaluation for two problems: Planes and Shifted Rastrigin. We have already seen the Shifted Rastrigin and Planes is defined by

$$f(x) = \sum_{d=1}^{D} (x(d) - 3) \tag{2}$$

Figure 6 depicts its landscape in two dimensions.

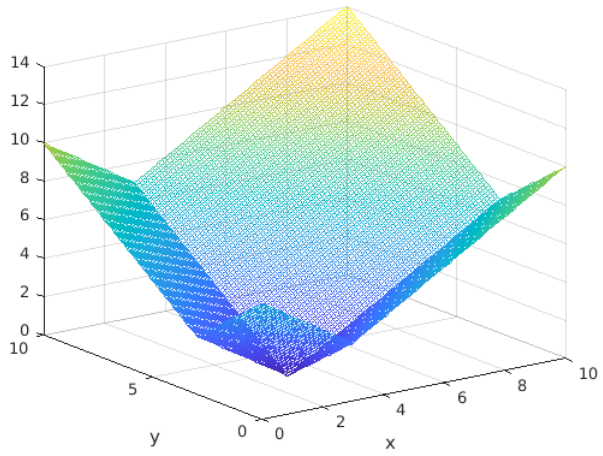
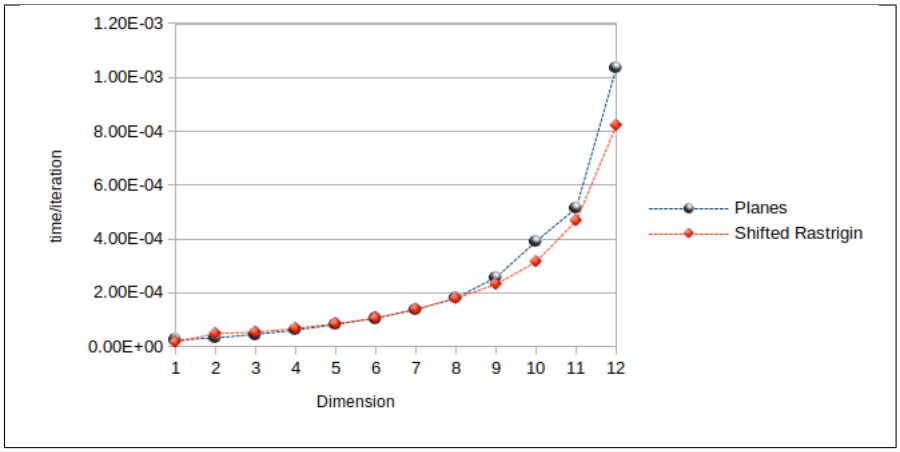


Figure 6: 2D Planes problem landscape.

As we can see on table 13 the real computing time indeed grows exponentially, but more like 1.35^D than like 2^D.

Table 13: Computing time per iteration

Dimension	Planes	Shifted Rastrigin
1	2.40E-05	1.60E-05
2	3.20E-05	4.80E-05
3	4.50E-05	5.30E-05
4	6.09E-05	6.69E-05
5	8.17E-05	8.47E-05
6	1.05E-04	1.05E-04
7	1.37E-04	1.38E-04
8	1.80E-04	1.78E-04
9	2.57E-04	2.31E-04
10	3.91E-04	3.15E-04
11	5.15E-04	4.67E-04
12	1.04E-03	1.09E-03



6 When it doesn't work

In some instances, even when problems are continuous and of low dimension, RCO doesn't perform well. For instance, let's examine the Frequency Modulated Sound Waves from the CEC 2011 competition benchmark ([5]). The search space is $[-6.4, 6.35]^6$ and the minimum value is zero. Despite 50064 evaluations, the best final value achieved is 24.07. This is primarily due to a considerable number of intersection positions (21238) lying outside the search space, rendering them unacceptable.

It's worth noting that in such cases, expanding the search space might provide some benefit if feasible. For example, if the search space is expanded to $[-500, 500]^6$, there are 17322 unacceptable intersections, resulting in a final best value of 17.42. However, many classical stochastic algorithms tend to discover significantly better solutions. This is because stochasticity can effectively navigate a highly chaotic landscape, as is the one of FMSW (see Figure 7). For example the slightly improved PSO already mentioned needs just two runs to find 2.218602×10^{-27}.

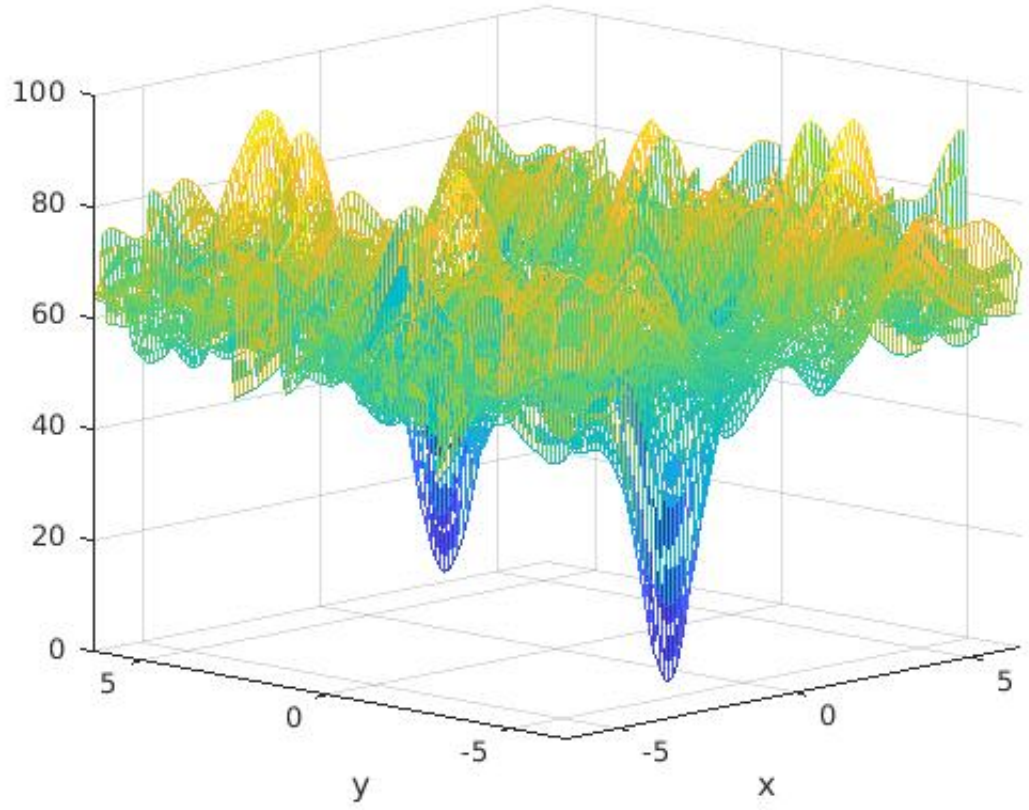


Figure 7: Frequency Modulated Sound Waves problem. Cross section on dimensions 5 and 6.

7 Conclusions and future works

Even in low dimensions, evaluating the objective function can sometimes be very costly. In such cases, RCO may be beneficial as it may propose a good solution after only a few evaluations. However, there are some disadvantages:

- It requires a reasonably good lower bound. That said, compared to most other methods, this is the only user-defined parameter. In practice, for real-world problems, such a bound is often known. Moreover RCO is able to automatically define an adaptive lower bound (see Appendix 8.5).
- It performs poorly on some problems, even in low dimensions, when the landscape is highly chaotic.
- It does not perform well on discrete problems, particularly when all variables are discrete. However, it still appears to be usable when only some of the variables are discrete.

- Its computation time per iteration increases exponentially with the dimension of the search space. Although it does not grow as quickly as theoretically predicted, in practice, it can still be challenging to use on a laptop for high-dimensional problems.
- As with many initial presentations of new optimization algorithms, such as Ant Colony Optimization (1991), Particle Swarm Optimization (1995), and Differential Evolution (1995), a formal convergence analysis has not yet been provided. Specifically, while it is clear that there is no "explosion" effect, as seen in the original PSO version ([1]), it would be beneficial to more precisely identify in which cases stagnation occurs and how it can be avoided. There is, in fact, an experimental RCO version that attempts to address the issue of stagnation, though it is not particularly convincing. For instance, in the case of the 10D Rosenbrock function (see Table 12), it finds a value of 2.3 instead of 8.996, which is still significantly worse than PSO (5.19×10^{-8}).

8 Appendix

8.1 A bit of geometry

To perform the "Ruler & Compass" construction in one dimension, as depicted in Figure 3 , it is necessary to define two segments, denoted as **a** and **b**, such that $\mathbf{a}/\mathbf{b} = \mathbf{A}/\mathbf{B}$. This method, dating back over 2600 years to Thales of Miletus, may have slipped from your memory over time...

Figure 8 illustrates the procedure.

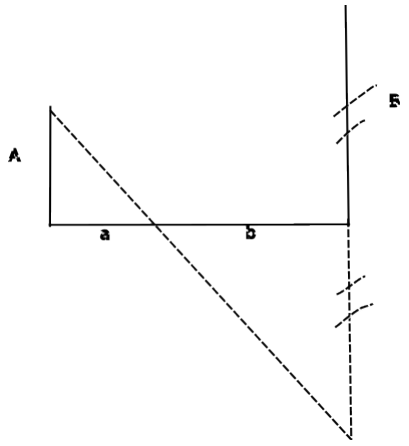


Figure 8: Ruler & Compass method to define proportional segments.

8.2 Barycenter

More generally we define the barycenter $xNew$ of the 2^D current positions x thanks to the values of the function f on these positions. Here is the Octave/Matlab© code of this calculation:

```
% Barycenter
minf=min(f);
if minf>=0 w=sum(f)-f; else w=sum(f-minf)-(f-minf); end
if w>0  xNew=sum(w'.*x)/sum(w);
else xNew=mean(x); end % Particular case, w may be null
```

Note: Of course, other formulas are possible—particularly those that use the differences between the $f-$values and the lower bound—but this one seems to produce good results.

8.3 Number of minima

In black box optimization of a non-discrete function it is theoretically impossible to know the number of (local) minima, because the set of positions contains an infinity of points. On a digital computer it is theoretically possible by exhaustive search for, because the ε-machine, the number of positions is finite. However it is huge (typically $2^{53D}$for a definition space of dimension D).

So, we have to make a comprise: an exhaustive search on a grid whose spacing is not too large (it may generate many fake minima) and not too small (too much computing time, numerical instability).

Then the following algorithm for a function f may be useful:

Define a grid with N positions along each dimension
For each position x evaluate $f(x)$, and also the values $f_i =$
$f(x_i)$ of the neighbors x_i (usually $2D$ except on the frontier).
If $f(x) < f_i$ for all i, then assume that x is a (local) minimum.

This algorithm requires about 2^{N+1} evaluations but does not need a large memory size. We could evaluate just once each position and save the result, but then the memory size would be big. And also the computing time anyway: for each position we would have to find if it has been already evaluated.

When two neighbors on the grid have the same value the algorithm may assume they are both "minima", as we can see on the figure 9.

By applying this algorithm we find that the landscape of sinCos15 has 28 minima in dimension one (and 818 in dimension two) including just one global minimum (see Figure 1). Six Hump Camel Back has six minima including two global ones. And of course Parabola is unimodal. For the Frequency-modulated Sound Waves, we find that there are at least 99,325 minima. Further refinement would be too time consuming.

8.4 Difficulty measures

Detailed descriptions, examples and references can be found in [3]. As explained in it the $\delta_{\neg NisB}$ measure is interesting, for it implicitly takes into account not only the number of local minima but also the sizes of their attraction basins. Using a tolerance threshold is usually less discriminant and the normalized roughness is too rudimentary.

As we can see, when considering the sizes of the attraction basins, the difficulty of FMSW is estimated to be less than that of Six Hump Camel Back. This is because, although there are many local minima, each basin is very small. Therefore, many stochastic algorithms can easily escape these minima.

8.5 About the lower bound

As mentioned the basic RCO needs an user-defined lower bound. Therefore, it is interesting to understand the algorithm's sensitivity to this parameter. As depicted in the figures 10, we observe that efficiency is indeed

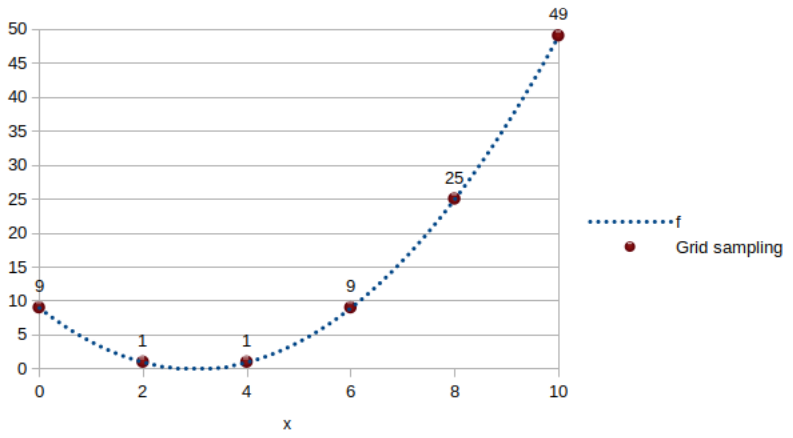
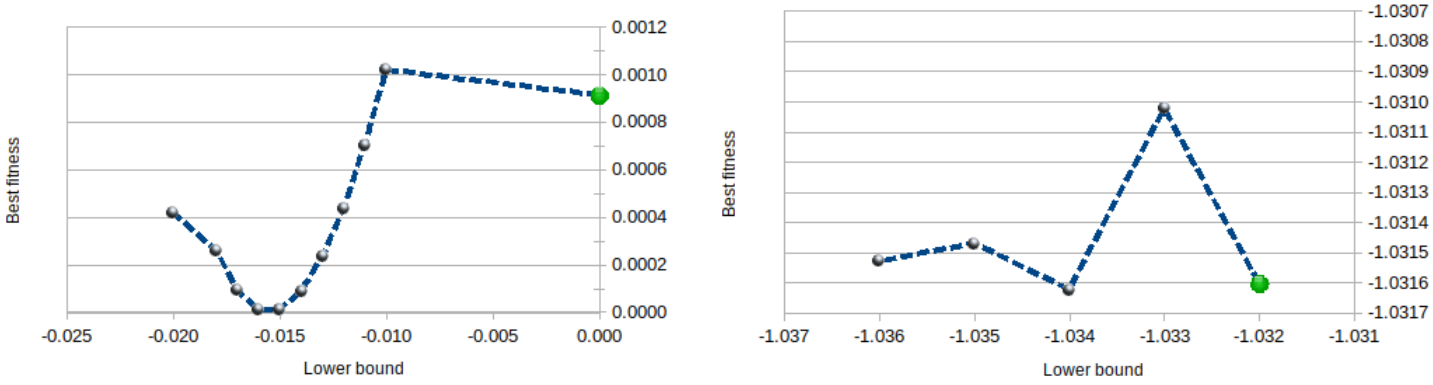


Figure 9: Looking for minima. The algorithm may find fake minima when two neighbors (here 2 and 4) on the grid have the same value..

Table 14: Difficulty measures in [0,1]. The ones of Parabola Six Hump Camel Back are given for comparison. The tolerance thresholds are relative to the global minimum value. For sinCos15 it corresponds to a value at most equal to 0.9997, as used in the Introduction.

	sinCos15	Parabola	Six Hump Camel Back	Frequency Modulated Sound Waves
Normalized roughness	0.9643	0	0.3333	> 0.9999
Tolerance threshold 1.93×10^{-4}	0.9996	0.9972	0.99997	1.0
$\delta_{\neg NisB}$	0.4706	0.0711	0.6258	0.5263

sensitive to changes in the lower bound. Though the sensitivity is not extreme, it could still be beneficial to experiment with various values. It's worth noting that setting the lower bound to the exact minimum value, if known, may be not the optimal choice.



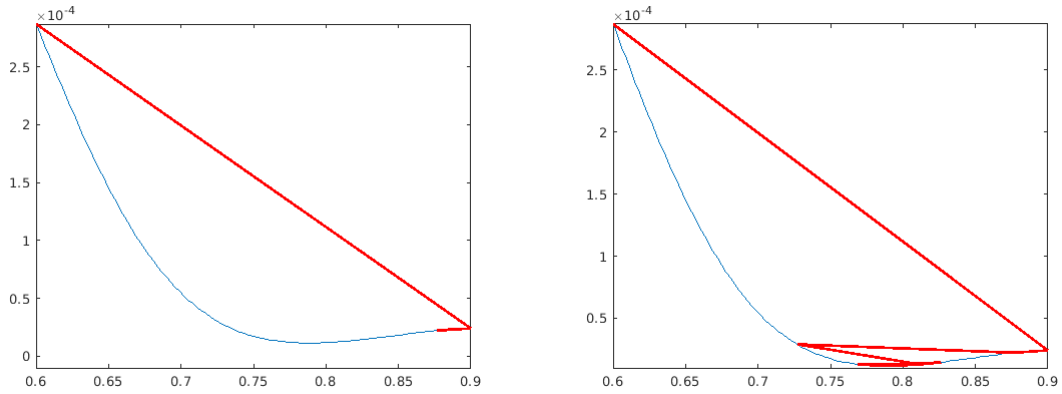
(a) Parabola

(b) Six Hump Camel Back

Figure 10: Efficiency is somewhat sensitive to the user-defined lower bound, but not excessively so.

A clear example of sensitivity can be observed when attempting to solve the optimal control problem of the Bifunctional Catalyst Blend in the CEC 2011 competition ([5]).

As depicted in Figure 11, even a slight modification to the lower bound leads to significantly different behavior due to the very small slope. Conversely, due to the same reason, a position far from the optimal one already exhibits a value close to the minimum.



(a) Lower bound -0.00001

(b) Lower bound 0.00001

Figure 11: CEC 2011 Bifunctional Catalyst Blend optimal control problem. 12 evaluations.

If you are unsure about the lower bound, you can utilize an adaptive one. It is evaluated as follows after initialization and at each iteration.

```
if min_f>0 lower_bound=min_f*coeff; else lower_bound=min_f*(2-coeff);
```

Where `min_f` is the minimum function value found so far and `coeff` an user-defined parameter in $]0, 1[$.

It appears that the results can occasionally be quite satisfactory, as evidenced by Table 15. Thus, it may be worthwhile to try different methods: user-defined and adaptive ones.

Table 15: Adaptive lower bound may improve the efficiency ... or the contrary!

	Dimension	Evaluations	Lower bound	Non adaptive	Adaptive coeff. 0.5	Adaptive coeff. 0.1
Six Hump Camel Back	2	104	-1.1	-1.031473	**-1.03157**	-0.947
Shifted Rastrigin	3	1508	-0.1	0.002548	0.06729	**3.27E-05**
	4	50016	-0.1	0.00173	**1.78E-15**	**0**
Rosenbrock	3	1508	-0.1	0.001	0.1866	0.238
	4	10016	-0.1	0.003756	**7.52E-22**	**8.94E-24**
Pressure Vessel	4	6076	6000	6592	6955	1.29E+05
Gear Train	4	1753	0	6.65E-09	1.38E-06	**6.60E-10**

8.6 More examples

For amusement, here are a few 1D and 2D figures illustrating how rapidly the algorithm approaches the solution, thanks to steps defined purely by geometry. Initially, they are quite large and gradually diminish in size as they approach the solution's position.

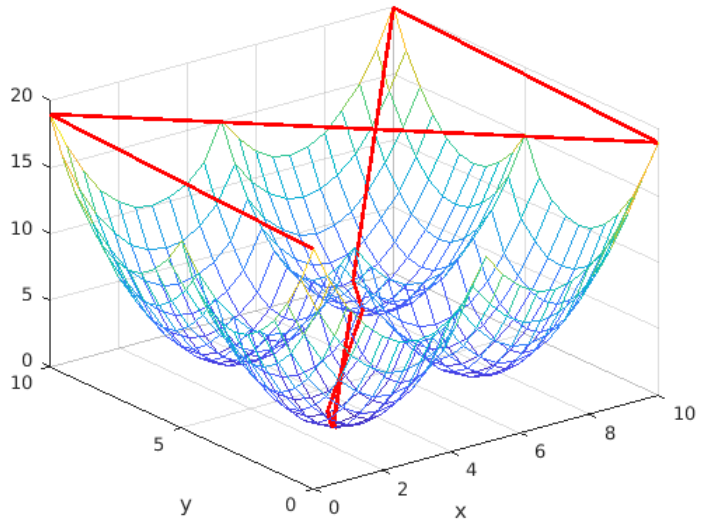


Figure 12: Multiparaboloid. 14 evaluations, final value 0.38 (real minimum 0).

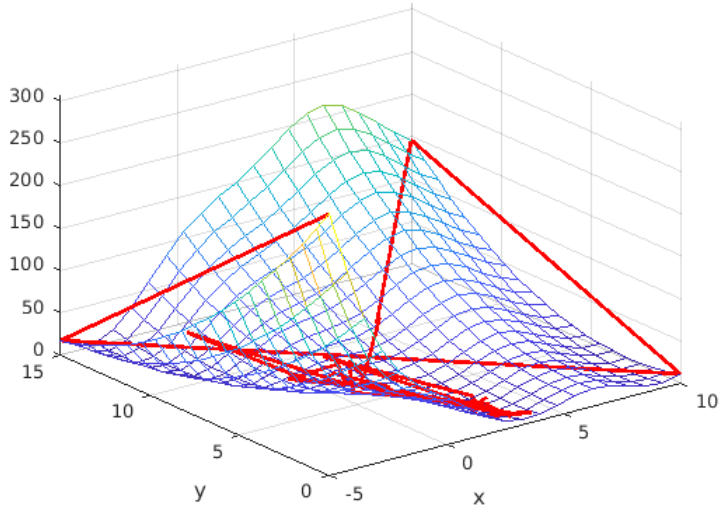
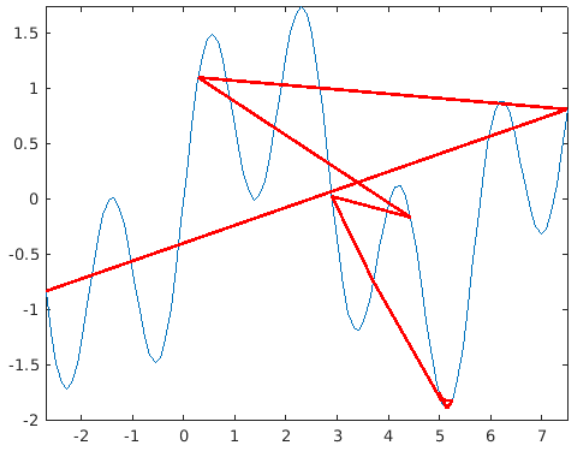
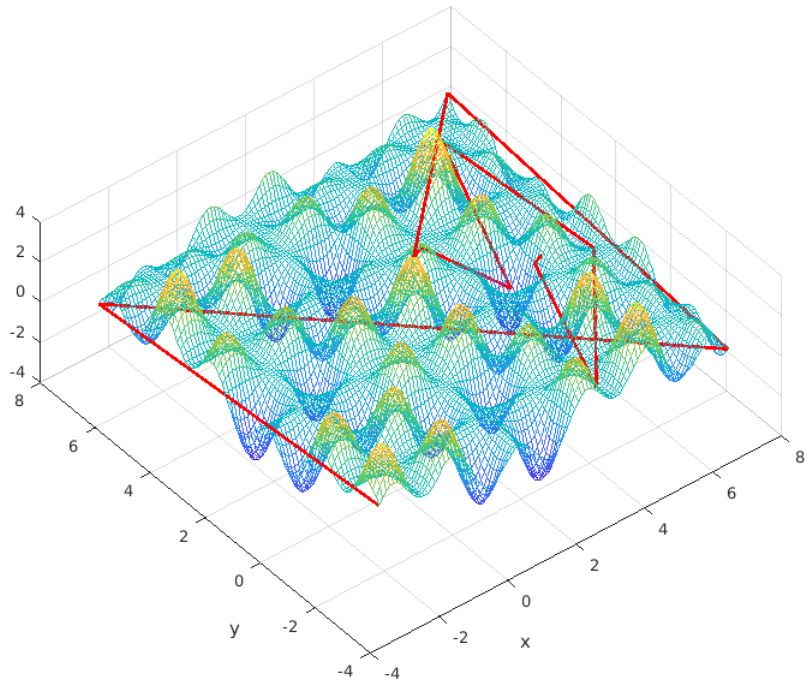


Figure 13: Branin. 44 evaluations, final value 0.3988 (real minimum 0.397887).



(a) 12 evaluations, final value -1.899584 (real minimum -1.8996).



(b) 16 evaluations, final value -1.732.

Figure 14: Combination of sinus.

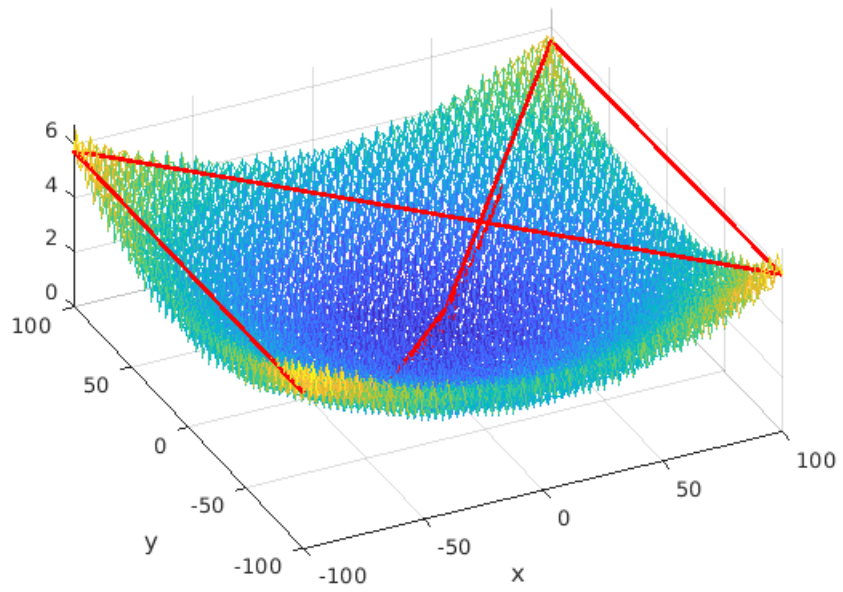


Figure 15: Shifted Griewank. 14 evaluations, final value 0.34 (real minimum 0).

8.7 Problem definitions

Name	Definition	Search space	Minimum
sinCos15	$f = 2 + \prod_{d=1}^{D} (\cos(x_d) + \lvert\sin(x_d)\cos(x_d)\rvert \sin(15x_d) + x_d/10)$	$[0, 10]^D$	0.999507 for $D = 1$
Six Hump Camel Back	$\left(4 - 2.1x_1^2 + x_1^4/3\right) x_1^2 + x_1x_2 + 4\left(x_2^2 - 1\right) x_2^2$	$[-2, 1]^2$	-1.031628453489877
Shifted Rastrigin	$\begin{cases} u_d = x_d - 10d/(D+1) \\ f = 10D + \sum_{d=1}^{D} u_d^2 - 10\cos(2\pi u_d) \end{cases}$	$[-10, 10]^D$	0
Rosenbrock	$f = \sum_{d=1}^{D-1} \left(100\left(x_{d+1} - x_d^2\right)^2 + \left(1 - x_{d+1}^2\right)\right)$	$[-100, 100]^D$	0
Pressure Vessel	$\begin{cases} 0.0193x_3 - x_1 \le 0 \\ 0.00954x_3 - x_2 \le 0 \\ 1296000 - \pi x_3^2 (x_4 + 4x_3/3) \le 0 \\ f = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 \\ \quad + x_1^2 (3.1661x_4 + 19.84x_3) \end{cases}$	$\begin{cases} q = 0.0625 \\ \{q, 2q, \ldots, 99\}^2 \\ \times [10, 200]^2 \end{cases}$	6059.714335048436
Gear Train	$(f = 1.0/6.931 - x_1x_2/(x_3x_4))^2$	$\{12, 60\}^4$	2.700857×10^{-12}
Frequency Modulated Sound Waves	$\begin{cases} \theta = \pi/50 \\ y_t = \sum_{t=1}^{100} x_1 \sin(x_2t\theta + x_3 \sin(x_4t\vartheta + x_5 \sin(x_6t\theta))) \\ y_0 = \sum_{t=1}^{100} \sin(5t\theta - 1.5\sin(4.8t\vartheta + 2\sin(4.9t\theta))) \\ f = (y_t - y_0)^2 \end{cases}$	$[-6.4, 6.35]^6$	0

References

[1] M. Clerc and J. Kennedy. "The particle swarm - explosion, stability, and convergence in a multidimensional complex space." In: *IEEE Transactions on Evolutionary Computation* 6.1 (Feb. 2002), pp. 58–73.

[2] Maurice Clerc. "Iterative optimization -Complexity and Efficiency are not antinomic." Mar. 2024. DOI: 10.13140/RG.2.2.26318.43847. URL: https://hal.science/hal-04487869 (visited on 03/11/2024).

[3] Maurice Clerc. *Iterative Optimizers - Difficulty Measures and Benchmarks*. Wiley, 2019. URL: http://www.iste.co.uk/book.php?id=1477 (visited on 04/29/2019).

[4] Maurice Clerc. *PSO technical site*. URL: http://clerc.maurice.free.fr/pso/ (visited on 03/27/2024).

[5] Swagatam Das and P. N. Suganthan. *Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems*. Tech. rep. 2011.

[6] David H. Wolpert and William G. Macready. "No Free Lunch Theorems for Optimization." In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82.

[7] Xin-She Yang et al. "True global optimality of the pressure vessel design problem: a benchmark for bio-inspired optimisation algorithms." In: *International Journal of Bio-Inspired Computation* 5.6 (Jan. 2013), pp. 329–335. ISSN: 1758-0366. DOI: 10.1504/IJBIC.2013.058910. URL: http://www.inderscienceonline.com/doi/abs/10.1504/IJBIC.2013.058910 (visited on 11/05/2017).