STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **9**, September 2021, pp 630–664.
Published online in International Academic Press (www.IAPress.org)

A New Hybrid Optimizer for Global Optimization Based on a Comparative Study Remarks of Classical Gradient Descent Variants

Mouad Touarsi [1], Driss Gretete [2], Abdelmajid Elouadi [3,*]

[1] *ILM departement , Engineering Science Laboratory (ENSA Kenitra) , Ibn Tofail University , Morocco*
[2] *Professor of Mathematics (Probabilities) , Head of ILM departement , Ibn Tofail University , Morocco*
[3] *Management Professor , Ibn Tofail University , Morocco*

Abstract In this paper, we present an empirical comparison of some Gradient Descent variants used to solve global optimization problems for large search domains. The aim is to identify which one of them is more suitable for solving an optimization problem regardless of the features of the used test function. Five variants of Gradient Descent were implemented in the R language and tested on a benchmark of five test functions. We proved the dependence between the choice of the variant and the obtained performances using the khi-2 test in a sample of 120 experiments. Those test functions vary on convexity, the number of local minima, and are classified according to some criteria. We had chosen a range of values for each algorithm parameter. Results are compared in terms of accuracy and convergence speed. Based on the obtained results, we defined the priority of usage for those variants and we contributed by a new hybrid optimizer. The new optimizer is tested in a benchmark of well-known test functions and two real applications were proposed. Except for the classical gradient descent algorithm, only stochastic versions of those variants are considered in this paper.

Keywords global numerical optimization, mono-objective, descent gradient variants, analytic hierarchy process, hybrid optimization, random search.

DOI: 10.19139/soic-2310-5070-1005

1. Introduction

Optimization techniques have common applications in fields such as differential calculus, regression models for prediction, shapes optimization, topological optimization, and other applications in logistic and graph theory [1]. The optimization is mono-objective when it consists of finding the best solution that optimizes a given objective [2]. On the other hand, multi-objective optimization concerns multiple contradictory criteria for making a decision [3]. Commonly, Numerical methods can provide practical and adaptable solutions in both cases. Although finding exact analytical solutions is a challenging task because of dimensions or because of the nature of the objective function, algorithms such as gradient descent are considered to find acceptable solutions with an error margin [4]. One of the main issues with gradient descent variants is how to select the appropriate algorithm according to the problems features. When it comes to applying gradient descent variants on a real application, a practitioner will prefer to use some criteria for making a quick decision. Because not all variants have the same performance. The use of a decision technique will help in saving time, especially while performing a simulation. For this purpose, we will compare the performance of gradient descent variants based on a panel of test functions. After that, we will apply a khi-2 test to help in deploying suitable decisions that match the researchers goals or understanding of a problem.

The paper is organized as follows: In Section 2, we provide a review of the related work. In section 3, we present briefly the mono-objective optimization. Afterward, we describe the used five variants in Section 4. The used test functions as well as the obtained performance results are presented in section 5. The statistical khi-2 and ahp technique are deployed to our

*Correspondence to: Abdelmajid Elouadi (Email: abdelmajid.elouadi@uit.ac.ma) , Ibn Tofail University , Morocco .

ISSN 2310-5070 (online) ISSN 2311-004X (print)
Copyright © 2021 International Academic Press

case study. In section 6, we will suggest and test our hybrid optimizer in a benchmark of 11 test functions then we will report the performances. Two real applications are proposed in section 7. Last of all, we discuss the results then we conclude.

2. Related works

Gradient descent method was first introduced by Louis Augustin Cauchy in his Compte rendu lacadmie des Sciences of October 18, 1847, [5]. Gradient descent is based on the observation that if a function is continuous and non-negative, the function should decrease in the direction of the negative gradient. The difficulty of this method is how to choose the suitable learning rate. The method is often not suitable for non-convex problems, an effect of fluctuations around local minimum could be observed. For that reason, the method isn't widely used in real applications [6][7]. To surpass those limits, Robbins and Monro in their paper, A Stochastic Approximation Method proves that for a given equation when the solution is unique, the stochastic method converges in probability to the desired solution [8]. Even though the results of this algorithm are not deterministic, they are good in the sense of reducing the dependence with the starting points. This dependence may be problematic for large search space [1]. Later, Kiefer and Wolfowitz subsequently published their paper, Stochastic Estimation of the Maximum of a Regression Function, which is more recognizable to be the first application of stochastic approximation in optimization [9]. Momentum version appaired in Rumelhart, Hinton, and Williamss article about learning by error propagation [10]. In 1997, Qian proved that the momentum term is equivalent to the mass of a Newtonian particle that moves through a viscous medium under a conservative force field [11]. In 2011, John Duchi, Elad Hazan, and Yoram Singer presented a new adaptive method that incorporates the geometry of the observed data. This new algorithm adaptively modifies the step learning depending on the data dispersion [12]. In the next year, Tieleman, Tijmen, Hinton, and Geoffrey suggested dividing the learning rate by an exponentially decaying average of squared gradients. This optimizer reduces oscillations in the vertical direction and increases the learning step in the horizontal direction to converge quickly [13]. Concerning hybrid optimization, it consists of combining several optimization techniques at once . Hybrid optimizers are known to be quite efficient in optimization.[14][15]

3. The problem

In the case of optimizing a single criterion f, the term optimum means either its maximum or minimum depending on the decision we are looking for. For example, if we own an industrial plant and we need to deliver products to customers in a way that minimizes the transportation costs then the optimum will be the minimum of the transportation cost function. The unconstrained optimization problem consists of minimizing a real-valued function f in their definition domain $D \neq \varnothing$:

$$\begin{aligned} f &: D \to \mathbb{R} \\ x &: \longmapsto f(x) \end{aligned}$$

Where $D \subset \mathbb{R}^N$ and $N \in \mathbb{N}$.

We search a vector x^* of n-components that verify for all $x \in D$:

$$f(x^*) \leq f(x)$$

This vector represents the global optimum of f in the domain D. A local optimum is an optimum of only a subset of the domain D. The problem is unconstrained because we we don't impose any conditions on the N variables and we assume that f its defined in their definition domain D [2]. Only this type of optimization problems is discussed in this research.

4. Gradient descent variants literature review

Multiple gradient descent variants exist. They vary according to how much data we use to evaluate the objective function gradient. We make a trade-off (based on the amount of data) between the parameter update precision versus the necessary time of performing an update.

4.1. Vanilla Gradient descent

If a multi-variables function is defined and differentiated on the neighborhood of its minimum x^* then the function f decrease in the opposite direction of the gradient. The algorithm starts by choosing an initial guess solution $x_0 \in D$.

The simplest form of update is to change the parameters along the negative gradient direction . The standard gradient descent algorithm is formulated as follows :

$$x\,(\,j\,,\ \ t+1) = x(\,j\,,\,t\,) - \ \alpha.\nabla f(x\,(\,j,t)\,) \tag{1}$$

Where :

- $x\,(\,j\,,\ \ t+1)$ Represents the j-ith component of the current solution $x(t)$ and $t+1$ represents the actual iteration.
- $\nabla f(x\ (j,t))$ Represents the j-ith component of the gradient vector for the function $f(x)$ in the last iteration t .
- α is the learning rate.

For a small positive, if the last estimation of the gradient is negative then:

$$f(\,x(\,j\,,\,t+1\,)\,) \geq f(\,x(\,j\,,\,t\,)\,) \tag{2}$$

Because we need to calculate the gradients for the whole dataset to perform just one update, traditional vanilla gradient descent performs slowly close to the minimum and is hard to control for large data sets that do not fit in memory. The other problem is the way we choose the convergence step α. Step α must be chosen carefully because it determines both the convergence speed and the accuracy of the estimated solution x^*. This parameter determines how big of an update we perform. [5] [16] .

This version of the main discussed algorithm is also called Batch gradient descent. It's based on the naive / full computation of the gradient and uses the entire available dataset. Note that state-of-the-art deep learning libraries provide automatic gradient computation like the well-known Rootsolve R package.

When evaluated on the full dataset, and when the learning rate is low enough, this is guaranteed to make non-negative progress on the loss function. Batch gradient descent is guaranteed to converge to the global minimum for convex error shapes and to local minimum for non-convex surfaces.

The following R code is an example of how batch gradient descent could be implemented :

```
1  library("rootSolve")
2
3  grad_descent<-function(objFun ,iter = 100, alpha = 0.001 , start_init ){
4
5    # define the objective function f(x)
6    # iter is the number of iterations to try
7    # alpha is the step parameter
8    # define the gradient of f(x)
9    # Note we don't split up the gradient
10   init = start_init    #initial point search
11
12   gradient_1 <- function(init , objFun) {
13   result <- gradient(objFun, init,pert = 1e-8)     # vector of gradient / partial derivatives
14   return(result)
15   }
16
17   x <- init
18
19   # create a vector to contain all xs for all steps
20   x.All = numeric(iter)
21   # gradient descent method to find the minimum
22
23   for(i in seq_len(iter)){
24     # Guard against NaNs
25     tmp <- c(x) - alpha * gradient_1(x , objFun)
26     if ( !is.nan(suppressWarnings(objFun(tmp))) ) {
27       x <- tmp
```

```
28     }
29 
30     print(c(i, x,objFun(x)))    # we print the current iteration with corresponding
       objective function value
31   }
32 
33   # print result and plot all xs for every iteration
34     print(paste("The minimum of f(x) is ", objFun(x), " at position x = ", x, sep = ""))
35     plot(x.All, type = "l")
36 }
37 }
```

Listing 1: Example of used R Vanilla GD code

4.2. Stochastic gradient descent

Batch gradient descent performs redundant computations for large datasets , as it recomputes gradients for similar examples before each parameter update. If the training set is very large (big data) and no simple formula is available for computing gradients, calculating the sum of the gradients can quickly become excessive. [8] [9] [17] [18]

It is in order to improve the computational cost of each step that the stochastic gradient descent method had been developed. Indeed, at each step, this method draws a random sample from the set of functions f_i constituting the sum. This trick becomes very effective in the case of large or very large learning problem [19] [8] [9] .

The stochastic gradient is a descent method used to minimize an objective function formulated as a sum of M-differentiated functions f_i.

The objective function could be formulated as a sum of M terms :

$$f(x) = \frac{1}{M}\sum_{i=1}^{M} f_i \tag{3}$$

The estimators that minimize a sum are called the M-estimators, they are used in the estimation of maximum-likelihood or even the empirical risk minimization for supervised learning problems. In supervised learning, each function f_i is associated with an observation belonging to the data set. Evaluating the gradient for the entire data set may require a lot of computational resources [8] [9].

To resolve the problem, the stochastic gradient samples a subset of the sum pieces (we choose at uniform K functions f_i from the sum where $K \leq M$ then we evaluate the gradient vector) [8] [9].

The algorithm starts by choosing an initial vector of parameters x_0 and a learning rate α. First, we evaluate the gradient for only a sum of K function f_i that are chosen in uniform. This means that the chosen functions could differ from an iteration to the other one. After that, we execute the update equation (4) until an approximation of the minimum is obtained [8] [9] [1] [20] :

$$x(\,j\,,\,t+1\,) = x(\,j\,,\,t\,) - \alpha.\frac{1}{M}.\nabla(f_i(\,x(\,j\,,\,t\,))) \tag{4}$$

Note that we can shuffle a single function f_i at each iteration because of the gradient operator linearity. If we shuffle more than one function f_i simultaneously, the version of the algorithm is then called the mini-batch stochastic descent gradient. The problem with the stochastic gradient method is the frequent updates and fluctuations. This complicates the convergence even if we choose a small step α.

It has been shown that even though we slowly decrease the learning rate, SGD shows the same convergence behavior as batch gradient descent. The mini-batch version could be a solution for the fluctuations around the optimum. For this, it's preferable at each iteration to evaluate the full gradient value uniformly for only some K-random dimensions and to keep the other dimensions unchanged until the next iteration [21].

An example of R implementation could be something like :

```
library("rootSolve")

stoc_grad<-function(objFun,iter = 50000, alpha = 0.00001, start_init ){

  init = start_init

  gradient_1 <- function(init , objFun ) {

    p=ceiling(runif(1,min=0,max =length(init) ))
    result <- gradient(objFun, init,pert = 1e-8)     # vector of gradient / partial d??
    rivatives
    #replace(result, sample(length(init),2), 0) # we shuffle three compenenets

    # print("gradient")
    # print(replace(result, sample(length(init),2), 0))

    return(replace(result,p, 0))
  }

  x <- init

  # create a vector to contain all xs for all steps
  x.All = numeric(iter)

  # gradient descent method to find the minimum

  for(i in seq_len(iter)){

    tmp = c(x) - (alpha)*gradient_1(x,objFun)

    alpha=exp(-i)

    if (!is.nan(suppressWarnings(objFun(tmp) ))) {
      x <- tmp
    }

    x.All[i] = x

  if(!iter>10000){
   print(c(i, x,objFun(x)))
  }

  }

  # print result and plot all xs for every iteration
   print(paste("The minimum of f(x) is ", objFun(x), " at position x = ", x, sep = ""))
   plot(x.All, type = "l")

}
```

Listing 2: Example of used R SGD code

The following figure shows the oscillation problem for the ackley test function :

4.3. *Momentum*

Among the other variants we find the momentum method . It appears in an article by Rumelhart, Hinton and Williams about the back propagation learning. This update can be motivated from a physical perspective of the optimization problem. The optimization process could be seen as equivalent to the process of simulating the parameter vector of a particle rolling on the landscape.[11] [22] , [23] [24]

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum . Momentum is a method that helps accelerate

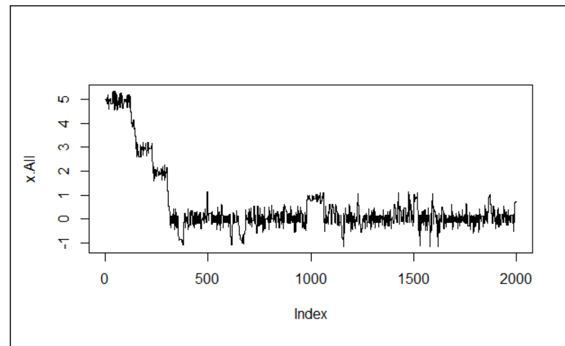


Figure 1. Example of SGD running - case of the ackley function

SGD in the relevant direction and dampens oscillations [11] [17] [25] .

Since the force on particle with a mass m is related to the gradient of potential energy (i.e.F =?∇U), the force felt by the particle is precisely the (negative) gradient of the loss function. Moreover, $F = ma$ so the (negative) gradient is in this view proportional to the acceleration of the particle. Note that this is different from the SGD update shown above, where the gradient directly integrates the position. Instead, the physics view suggests an update in which the gradient only directly influences the velocity, which in turn has an effect on the position. In particular, the loss can be interpreted as the height of a hilly terrain (and therefore also to the potential energy since $U = mgh$ and therefore $U \propto h$).[26][24]

The SGD method with moment keeps in memory the update at each step $\Delta x(t)$, and calculates the following update as a convex combination of the current gradient and the previous change:

$$\Delta x(\ j\ ,\ t+1\ ) = \alpha.\nabla f(\ x(\ j,\ t\ )) +\ \beta.\Delta x(\ j\ ,\ t\ ) \tag{5}$$

The term momentum is used in physics when a particle is subject to a rotation movement applied by a force. The moment measures the ability of a force to rotate an object around an axis or a reference point.The name moment comes from an analogy with the moment in physics: the vector of parameters $x(t)$, considered as a particle which travels through the space of parameters (often in large dimension), undergoes an acceleration via the gradient (which acts as a "force"). Unlike the classic SGD method, this variant tends to keep traveling in the same direction, preventing oscillations. [11][17] [27]

The previous variants of the gradient descent algorithms have troubles when the convexity is irregular and variations are more severe in one dimension than others. In this case, the momentum approach tries to keep the vector we want to estimate in the direction of the gradient. By this, the referent vector can be seen as a particle subject to a force that pulls the vector to the local minimum. The force in our situation can be compared to the gradient of the objective function [11][17][27][24].

Unlike classical gradient descent, the momentum approach keeps the referent vector x traveling in the same direction to which prevent oscillations of the loss function. The new update Δx is a convex linear combination of the last update and the gradient value [28][24] [29] [30]:

The term $\beta.\Delta x(\ j\ ,\ t\ )$ represents the previous update of the field $x(\ j\ ,\ t\ )$ multiplied by a weight β.

The new $\Delta x(\ j\ ,\ t+1\ )$ is a moving average of the gradient and the last update [10]. Commonly, we choose a weight β such that: $0.8 \leq \beta \leq 0.999$

By default, we choose a value of $\beta = 0.9.$. The term $\alpha.$ represents the convergence step. We use the same equation of the descent gradient using the new update as follow:

$$x(\ j\ ,\ t+1\ ) = x(\ j\ ,\ t\ ) - \alpha.\Delta x(\ j\ ,\ t+1) \tag{6}$$

Essentially, we drive a ball down a hill by using momentum. As it moves downhill, the ball accumulates energy, getting faster and faster on the way (until it reaches its terminal velocity, if air resistance is present, i.e. $\beta \leq 1$).

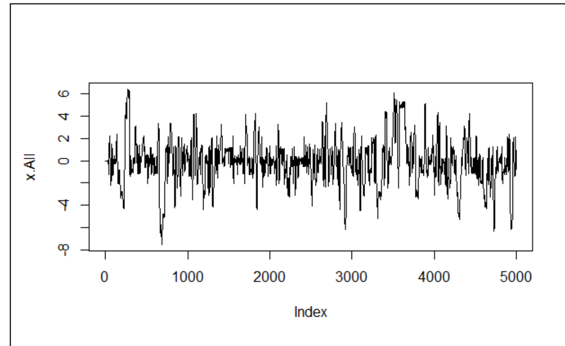


Figure 2. SGD momentum improvement - case of the ackley function

For our parameter changes, the same thing happens: For dimensions whose gradients point in the same directions, the momentum term increases, and decreases updates for dimensions whose gradients change directions. As a consequence, we achieve faster convergence and decreased oscillation.

On the other hand , a particle that rolls down a hill is extremely unsatisfying, blindly following the slope. We just want to see a clever particle , a particle that acts as a ball which has a sense of where it's going, so that before the hill ramps up again, it knows how to slow down. [17]

4.4. Adaptive gradient descent

Adagrad version adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters $x(t)$. For this reason, it is well-suited for dealing with sparse data. This version of the gradient descent algorithm takes the sparsity of parameters into account. Adagrad uses a different learning rate for every parameter $x(t)$ at every time step t [12] [31] [17] [32].

An interesting application of Adagrad was done by Dean et al. he has found that Adagrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google, which C among other things C learned to recognize cats in YouTube videos.[17]

In the current iteration $\tau + 1$, The Adagrad reduces the convergence step α for a high variation in a given point and increases the same step for a low gradient value [33] [12]. For this reason,the algorithm is suitable for sparse data.

Let $G(t)$ be the outer product matrix of the gradient vector $\nabla f(\ x(t)\ )$ at the t-ith iteration. The algorithm divides the step α by the sum of diagonal elements $G(\ j\ ,\ j\ ,\ t\ )$ of the matrix $G(t)$ until the current iteration $\tau \in \mathbb{N}$ [34]. The update formula is as follows :

$$x(\ j\ ,\ \tau+1\ ) = x(\ j\ ,\ \tau\ ) - \alpha.[\ S(\ j\ ,\ j\ ,\tau\ )]^{-1/2}.\nabla f(\ x(\ j\ ,\tau)\ ) \tag{7}$$

Where :

$$S(\ j\ ,\ j\ ,\tau\ ) = \sum_{k=1}^{\tau} G(\ j\ ,\ j\ ,\ k\ ) \tag{8}$$

And : for all t in $\{1, ..., \tau\}$

$$G(\ j\ ,\ j\ ,t\ ) = [\ \nabla f(\ x(\ j,t\ )\ ]^2. \tag{9}$$

Besides, $G(i\ ,\ j\ ,\ t)$ is the element within the row i and column j of the outer matrix $G(t)$ and τ is the number of iterations.

We have usually $(i, j) \in [\![1, N]\!] \times [\![1, N]\!]$. Note that Adagrad performs also on non-convex problems [12] [35].

In Adagrad update rule, we modify the general learning rate α at each time step t for every parameter $x(\ j,\ t)$ based on the past gradients that have been computed for $x(\ j,\ t-1)$. j is the j-ith dimension of the parameters vector.

One of the key advantages for Adagrad is removing the manually calibration need for the learning rate. The denominator's square gradients accumulation's could be also a drawback: As any added term is positive, during training the cumulative amount continues to increase[17][34][33] .

At a consequence, this leads the learning rate to diminish and inevitably become infinitesimally small, at which point extra information can no longer be obtained by the algorithm. The RMSProp algorithm is aimed at fixing this weakness [17][34][33] .

4.5. Root Mean Square Propagation

RMSProp is a similar gradient descent version compared to Adagrad, the difference is that RMSProp considers the last value and the actual value of the gradient while dividing the learning rate by $S(j\ j, \tau)$ [13] [36] [31].

In the j-ith iteration. RMSProp avoids radically the diminish of the learning rate α [13] [37] [38].

The RMSProp update formula is formulated as follow:

$$v(x(j,t+1)) = \gamma.v(x(j,t)) + (1-\gamma).(\nabla f(x(j,t))^2 \tag{10}$$

Where $x(j,t+1)$ is the j-ith field of the N-dimensional solution x that we want to estimate and t represents the last iteration. The coefficient γ is called the memory factor, we choose a value of γ such as $0.1 \leq \gamma \leq 0.9$.

Using this formula of updates, the formula of descent gradient is as follow [38][1]:

$$x(\ j\ ,\ t+1) = x(\ j\ ,\ t\ ) - \alpha.\frac{\nabla f(\ x(\ j\ ,\ t\ )\ )}{\sqrt{v(x(\ j\ ,\ t+1\ )}} \tag{11}$$

$v(\ x(\ j\ ,\ t\ ))$ Is ensured to be positive when computing the square. For this, we could modify equation (10) by initially adding a small positive number $\epsilon 0$ to its right side [13] [17].

```
1  library("rootSolve")
2  RMSprop<-function(objFun ,iter = 50000, alpha = 0.00001,lambda=0.2 ,start_init ){
3
4    init = start_init
5    gradient_1 <- function(init , objFun ) {
6
7      p=ceiling(runif(1,min=0,max =length(init) ))
8      result <- gradient(objFun, init,pert = 1e-8)     # vector of gradient / partial d??
       rivatives
9      # print("gradient")
10     # print(replace(result, sample(length(init),2), 0))
11     # return(replace(result,p, 0))
12     return(result)
13   }
14
15   x <- init # create a vector to contain all xs for all steps
16   x.All = numeric(iter)
17   tmp<-rep(0,length(init))
18   V=(1-lambda)*(gradient_1(x,objFun))^2
19   # gradient descent method to find the minimum
20   for(i in seq_len(iter)){
21
22     V=lambda*V+(1-lambda)*(gradient_1(x,objFun))^2
23     tmp = x - alpha*gradient_1(x,objFun)/sqrt(V)
24
25     if ( !is.nan(suppressWarnings(objFun(tmp))) ) {
26       x <- tmp
27     }
28
29     x.All[i] = x
```

```
30      print(c(i,x,objFun(x)))
31    }
32
33    # print result and plot all xs for every iteration
34    print(paste("The minimum of f(x) is ", objFun(x), " at position x = ", x, sep = ""))
35    plot(x.All, type = "l")
36  }
```

Listing 3: Example of used RMSPROP R code

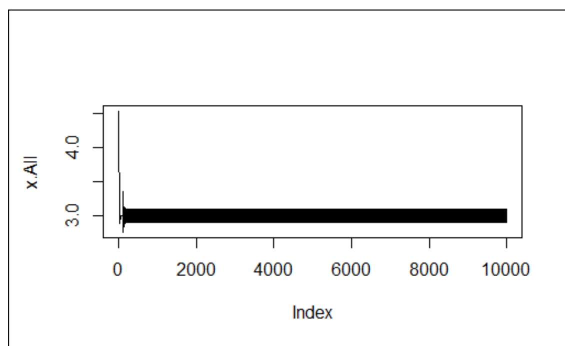


Figure 3. Example of RMSProp runing - Himmelblau's function

5. Comparing performances

After presenting different variants of the descent gradient algorithm, we will try in this section to measure the performances of the algorithms already presented. Firstly, we will define the convergence rate. After that, we give the results of those algorithms applied in a benchmark of five test functions [39][40]. The used test functions are:

F1: Himmelblau's function
F2: Rastrigin function
F3: Ackley function
F4: Sphere function
F5: Beale function

The formula of those functions could be found in the appendix. The following table classifies the used test functions:

Table 1. Classification of the used test functions

	F1	F2	F3	F4	F5
Convexity	No	No	No	Yes	No
Continuity	Yes	Yes	Yes	Yes	Yes
Modality	Multi	Multi	Multi	Mono	Multi
Separability	No	Yes	No	Yes	No

The convergence rate of a sequence is the speed at which their terms converge to a certain unique value called the sequence limit [41]. In our case, we consider that an algorithm is fast if the related number of iterations is small. The accuracy is measured as the difference between the exact value of the minimum and the quantity we got using a variant of descent gradient. In practice, some sequences converge quickly to the limit value but they lack in accuracy for a given algorithm.

To conduct our experiments, the algorithms, and the used test functions are implemented under R language, we had chosen $[-6, 6] \times [-6, 6]$ as a study domain and we executed 8×10^5 iterations for each experiment. The initial point is usually

$(x, y) = (5, 5)$. Except for the classical gradient descent, only the stochastic versions of the previously presented algorithms are tested. First, we deduct that the more α values are small, there is more chance that the error value will diminish for the classical gradient descent and the algorithm will give accurate results.

On the other hand, the convergence speed will be slow for small values of α as shown in Table 2. The gradient descent gives good results for both F1 & F4 but fails for F2 & F3. This is because F2 & F3 are characterized by a large number of local minima regularly distributed and with and a large search space.

Table 2. Experiments results of the classical gradient descent

Test Function	Classical gradient descent algorithm				
	α *value*	Number ofiterations	Value of X	Value of Y	f(x,y)
F1	0.1	10^2	$1,37 \times E37$	$-2,92 \times E117$	INF
	0.001	10^2	2,99	1,99	$8,35 \times E-15$
	0.00001	10^4	2,98	2,044	$3,05 \times E-02$
F2	0.1	$1,0127 \times 10^4$	$-5,26 \times E-03$	$-5,26 \times E-03$	$1,10 \times E-02$
	0.01	2×10^4	2,72	2,72	38,52
	0.00001	2×10^4	4,97	4,97	49,74
F3	0.1	10^3	4.95	4.95	12,67
	0.01	10^4	4.98	4.98	12,63
	0.00001	10^5	4.98	4.98	12,63
F4	0.1	10^2	$-3,981 \times E-09$	$-3,981 \times E-09$	$3,17 \times E-17$
	0.01	10^3	$-3,41 \times E-09$	$-3,41 \times E-09$	$2,33 \times E-17$
	0.00001	9×10^4	0,6766	0,6766	0,9157
F5	0.1	10^2	$-2,76 \times E220$	$-8,27 \times E220$	INF
	0.01	10^2	$-2,58 \times E163$	$-7,27 \times E163$	INF
	0.00001	10^4	3.31	0.43	0.487

Secondly, we conclude that the stochastic algorithm could deteriorate the results of the classical descent gradient version whether or not we consider convex objective functions, which is the case for F1 and F2. Thus, the stochastic version should be used only for large dimensions or for an objective function that is represented as a sum with a large number of terms. The results deterioration is mainly explained by the fluctuations around the minimum because we shuffle components of the gradient vector at the uniformly.

Table 3. Experiments results of the stochastic gradient descent algorithm

Test Function	Stochastic gradient descent algorithm				
	α *value*	Number ofiterations	Value of X	Value of Y	f(x,y)
F1	0.1	10^2	4.79	-INF	INF
	0.01	10^3	3.33	0.42	12.4453
	0.00001	$2,35 \times 10^3$	3.13	1.95	0.5729
F2	0.1	10^5	-7.57	4.24	104.166
	0.01	10^3	3.07	3.01	19.6097
	0.00001	10^5	3.98	3.98	31.8392
F3	0.1	10^5	-0.43	0.32	3.695
	0.01	10^3	3.98	3.98	10.998
	0.00001	10^5	3.98	3.98	10.9982
F4	0.1	10^3	3.23E-66	3.23E-66	7.317E-127
	0.01	10^3	1.14E-5	1.14E-5	1.47E-10
	0.00001	3×10^5	0.0726	0.0726	0.01078
F5	0.1	10^2	5.12 E+7	4.94 E+108	INF
	0.01	10^2	5.16 +118	- 5.01 E+18	INF
	0.00001	5×10^3	3.223	0.6011	0.0928

Third, the momentum version is suitable for F3 because a flat outer region and a large hole in the center characterize it, but a bad choice of β could extremely influence the accuracy of the results. This remark explains the convergence difference between F2 compared to F3 although the fact that both share the property of having a large number of local minima regularly distributed. To obtain good results, we should avoid small values of α.

Fourthly, the adaptive version can give close results of the solutions (X, Y) for each test function even with trying different values of parameters (which is the case of F2 in Table 5). In the case we obtain close results for the objective function, we can choose a high step as shown for F3 with $\alpha = 0.9$ (this sample will not be considered for khi-2 test to avoid bias). The results of Table 5 illustrates that if we increase the value of the parameter α, we will get more chances to reach the optimum.

Last, of all, the RMSPROP improves the performance of the adaptive version. The effect of using this version is clear for F1. For F1, RMSPROP helped to ensure a stable convergence by continuing the navigation in the relevant directions and by softening the oscillations in irrelevant directions.

For F2, this version gave close results which is the same effect of the Adagrad version. The remark illustrates the ability of this version to deal with the sparsity of data. Increasing can lead to improving the convergence speed.

On the other hand, Both Adagrad and RMSPROP provide better performance for F3 compared to F2 due to the shape of F3 where the global optimum is cavernous in the central hole. Based on our results, we notice that we should avoid small steps for this category of objective functions (F2) because iterations can stick at a local minimum. Concerning a convex problem, which is the case of F4, the results of convergence are acceptable even though the momentum provided the best performance for the F4 function.

Table 4. Experiments results of the momentum gradient descent algorithm

Test Function	Momentum gradient descent algorithm					
	α	β	Number of iterations	Value of X	Value of Y	f(x,y)
F1	0.1	0.8	10^3	-7.24 E+249	-3.426 E+125	INF
		0.9	10^3	-5.96 E+80	-1.773 E+160	INF
	0.01	0.8	10^4	-2.823	1.9907	35.4199
		0.9	10^3	1.47 E+157	-1.0297 E+104	INF
	0.00001	0.8	10^3	3.1404	2.0055	0.7798
		0.9	10^3	3.1225	2.0006	0.3821
F2	0.1	0.8	10^3	-14.84	-1.692	241.224
		0.9	10^3	-5.873	-48.76	2424.65
	0.01	0.8	10^3	5.6580	3.7445	71.8431
		0.9	10^3	2.6223	-4.130	44.322
	0.00001	0.8	10^3	3.9797	3.9797	31.8384
		0.9	10^3	3.9797	3.9797	31.8384
F3	0.1	0.8	10^3	-0.227	-0.285	2.7622
		0.9	10^3	47.003	15.425	21.6450
	0.01	0.8	10^3	3.9830	3.9830	10.9982
		0.9	10^3	-0.125	-0.703	3.4095
	0.00001	0.8	10^4	3.9836	3.9833	10.9982
		0.9	10^4	3.9834	3.9827	10.9982
F4	0.1	0.8	10^3	-1.018 E-43	1.1437 E-43	2.34524 E-86
		0.9	10^3	2.3987 E-19	-5.401 E-18	2.54610 E-35
	0.01	0.8	10^3	7.7307 E-47	-6.1495 E-46	3.84147 E-91
		0.9	10^3	-1.639 E-22	-7.2746 E-24	2.69311 E-44
	0.00001	0.8	10^3	3.7477	3.7386	28.022
		0.9	10^3	3.4911	3.5076	24.4913
F5	0.1	0.8	10^3	-1.19 E+06	3.07 E+70	INF
		0.9	10^3	1.74 E+204	-8.30 E+268	INF
	0.01	0.8	10^3	1.26 E+59	-INF	INF
		0.9	10^3	2.74 E+59	-INF	INF
	0.00001	0.8	10^3	-2.87 E+221	-3.899 E221	INF
		0.9	10^3	-8.04 E+213	1.04 E+24	INF

Table 5. Experiments results of the Adagrad gradient descent algorithm

Test Function	Adagrad gradient descent algorithm				
	α value	Number of iterations	Value of X	Value of Y	f(x,y)
F1	0.1	10^3	2.9501	2.0051	0.0858
	0.01	2×10^3	2.9087	2.2966	1.4549
	0.00001	5×10^3	3.9963	3.9963	248.565
F2	0.1	10^3	3.9797	3.9797	31.8384
	0.01	10^2	3.9797	3.9797	31.8384
	0.00001	10^5	3.9950	3.9550	31.9311
F3	0.1	10^3	3.9831	3.9829	10.9982
	0.01	10^4	3.9830	3.9830	10.9982
	0.00001	10^5	3.9951	3.9951	11.0059
	0.9	10^4	-0.007	-0.0003	0.0226
F4	0.1	2×10^3	0.1426	0.1174	0.03412
	0.01	2×10^5	0.0905	0.0907	0.01643
	0.00001	10^5	3.9948	3.9948	31.9178
F5	0.1	$2,5 \times 10^4$	1,8745	1,3606	49,47
	0.01	$2,5 \times 10^4$	2,3201	2,2900	960,13
	0.00001	$2,5 \times 10^4$	3,9918	3,9918	$6,7 \times 10^4$

Our results of using RMSPROP are shown in Table 6 :

Table 6. Experiments results of the RMSPROP gradient descent algorithm

Test Function	RMSPROP gradient descent algorithm					
	α	γ	Number of iterations	Value of X	Value of Y	f(x,y)
F1	0.1	0.2	10^3	3.0022	1.9535	0.03398
		0.5	10^2	3.0270	1.9555	0.03623
		0.9	10^2	2.9682	2.0038	0.03467
	0.01	0.2	10^3	2.9992	1.9853	0.00387
		0.5	10^3	2.9911	2.0033	2.506 E-03
		0.9	10^3	2.9953	2.0019	6.74723 E-04
	0.00001	0.2	27×10^3	2.9669	2.1321	0.26699
		0.5	25×10^4	2.9973	2.1536	0.36459
		0.9	$2,32 \times 10^5$	2.9629	2.1338	0.27431
F2	0.1	0.2	10^3	3.9336	3.9286	32.7521
		0.5	10^3	3.9342	3.9022	33.3743
		0.9	10^3	3.9179	4.0510	33.5714
	0.01	0.2	10^3	3.9847	3.9737	31.8505
		0.5	10^3	3.9713	3.9844	31.8565
		0.9	10^3	3.9876	3.9746	31.8558
	0.00001	0.2	10^4	3.9797	3.9797	31.8384
		0.5	10^4	3.9797	3.9797	31.8384
		0.9	10^4	3.9797	3.9797	31.8384
F3	0.1	0.2	10^3	0.0235	0.0438	0.2056
		0.5	10^3	0.0072	0.0967	0.50688
		0.9	10^3	-0.117	0.0570	0.7739
	0.01	0.2	$2,9 \times 10^5$	3.9574	4.0101	11.0350
		0.5	3×10^5	3.9713	3.9964	11.0067
		0.9	$2,6 \times 10^5$	3.9609	4.0044	11.0233
	0.00001	0.2	$2,5 \times 10^5$	3.9830	3.9830	10.9982
		0.5	$2,7 \times 10^5$	3.9830	3.9831	10.9982
		0.9	$2,5 \times 10^5$	3.9830	3.9830	10.9982
F4	0.1	0.2	10^3	0.0061	-0.060	0.0502
		0.5	10^3	0.0519	0.0515	0.00535
		0.9	10^3	-0.057	0.0588	0.00678
	0.01	0.2	10^3	-0.051	0.0044	4.6057 E-5
		0.5	10^3	0.0051	0.0063	6.7633 E-5
		0.9	10^3	0.0055	-0.007	8.3653 E-05
	0.00001	0.2	$4,3 \times 10^5$	1.0226	1.0293	2.09721
		0.5	4×10^5	1.0459	1.0452	2.18655
		0.9	$3,8 \times 10^5$	0.9454	0.9485	1.7937
F5	0.1	0.2	10^2	1.820	0.03	0.90
		0.5	10^2	2.120	0.1817	0.36
		0.9	10^3	2.409	0.319	0.11
	0.01	0.2	10^3	1.855	0.036	0.83
		0.5	10^3	2.103	0.162	0.39
		0.9	10^3	2.135	0.110	0.42
	0.00001	0.2	10^5	3.308	3.307	1.54 E+4
		0.5	10^5	3.262	3.262	1.38 E+4
		0.9	10^5	3.198	3.195	1.18 E+4

For now, we will define our approach to select the best variant depending on several criteria, and regardless of the problem features. The khi-2 test is a statistical test where the test statistic follows a khi-2 law under the null hypothesis. This test allows verifying the adequation of a series of observations to a probability law. We assume that the steps for applying the khi-2 test are known for the reader.

The details of the khi-2 test could be found at [42]. The null hypothesis is formulated as follow:

H0: There is ***no dependence*** *between the choice of* ***the method variant type*** *and the obtained* ***convergence speed*** *for the considered test functions*

Table 7 gives the number of choices of parameters for each method variant where the number of iterations required for convergence is strictly less than 10^4 (which means: a constant $\times 10^3$). We consider a variant very fast when this variant requires that magnitude of iterations.

Table 7. The contingency table of the convergence speed

	M_1	M_2	M_3	M_4	M_5	Total
Fast convergence speed	7	10	27	7	28	79
Low convergence speed	8	5	3	8	17	41
Total	15	15	30	15	45	120

The khi-2 score for table 7 is equal to 12.95 $>$ Chi-2(4, **0.02**) = 11.66 which means that we can reject the null hypothesis. Thus, the obtained convergence speed will depend on the choice of the gradient descent variant type. Similarly, we prove that the accuracy metric will depend on the chosen variant type. Table 8 gives the associated contingency table of the accuracy based on the conducted experiments:

Table 8. The contingency table of the accuracy

	M_1	M_2	M_3	M_4	M_5	Total
High accuracy	5	5	6	3	24	43
Low accuracy	10	10	24	12	21	77
Total	15	15	30	15	45	120

The khi-2 score for table 8 is equal to 10.98 $>$ Chi-2(4, **0.05**) = 9.48 which means that we can reject the null hypothesis and we could deduct that the accuracy performance depends on the chosen variant. This justifies the need for classifying the priority of usage for those variants. For this purpose, we had applied the AHP technique that is a multi-criteria analytical approach for decision support. The process of using AHP is explained in detail at [43] [44]. The used criteria are accuracy, convergence speed, robustness, and fluctuations. Those criteria characterize the performance of any numerical method and are denoted in order as C1, C2, and C3 & C4. The used alternatives are the five gradient descent variants denoted in order as M1, M2, M3, M4 & M5. The following table gives choices priorities based on criteria & alternatives performance:

Table 9. The priority of choices based on criteria and alternatives performance

	C1	C2	C3	C4
M1	0,136	0,046	0,054	0,063
M2	0,062	0,105	0,054	0,045
M3	0,218	0,245	0,127	0,133
M4	0,062	0,105	0,258	0,258
M5	0,520	0,497	0,504	0,499

Note that the sum of each column is equal to 1. The matrix elements sum is equal to the number of criteria which is 4. The priorities of actions for each alternative are presented in the next table:

Table 10. The priority of usage based on criteria and alternatives performance

Gradient descent variant	Priority of usage
M1	0,083853165
M2	0,076775086
M3	0,20924938
M4	0,1227359
M5	0,507386469
Total: 100%	

We understand from previous tables that AHP results are coherent with the khi-2 test results. The RMSPROP has a priority of 50.73% to be chosen for a random situation based on the elaborated judgment matrice. In the next section, we will propose and test our hydride optimizer. This optimizer combines recursive random search with the stochastic RMSPRPOP. This optimizer is built based on our understanding of the obtained experimental results for the considered gradient descent variants.

6. Suggested hybrid optimizer

Because the accuracy of the compared versions usually depends on the chosen initial position $x_0 \in D$ for large search domains and into the characteristics of the used test function. We will suggest and test our optimizer that is based on the following suggested algorithms written in pseudo-codes :

Algorithm 1: Initial_domain_search_n_dim

Inputs : N , p , $Lower_boundaries_vector$, $Upper_boundaries_vector$, $Object_fun$

$X = Uniform(N, min = Lower_boundaries_vector, max = Upper_boundaries_vector)$

// we generate N points at uniform from the n-dimensional study domain defined by Lower_boundaries_vector and Upper_boundaries_vector. Those vectors have the same length, which is equal to n.

$Y = matrix(data = X, nrow = N, ncol = n)$

// we store those N points in a matrix with n columns and N rows where the rows represent the n-dimensional points. The columns represent the dimensions of the n-dimensional space.

$Outer = Object_fun(Y)$

// we compute the objective function values for all of those N points present in the matrix Y rows. After that, we add Outer after the last column of Y with Cbind.

Y=Cbind (Y, Outer)

$P_minimum_values = Subset(Y, Sort(Outer[1 : p]))$

// we retrieve the p-points in the n-dimensional space from Y rows that are minima in term of Object_fun values. P_minimum_values is a submatrix of Y.

Lower_boundaries=apply (P_minimum_values, 1, min)

// we retrieve the minimum of each column of the matrix P_minimum_values. Those minima present the lower-boundaries of the new reduced domain study.

Upper_boundaries=apply (P_minimum_values, 1, max)

// we retrieve the maximum of each column of the matrix P_minimum_values.

New_domain= Cbind (Lower_boundaries, Upper_boundaries)

// we place the vectors Lower_boundaries and Upper_boundaries in columns of the matrix New_domain. Each row presents the boundaries of the new domain in the respective dimension.

Outputs : New_domain

Example of R outputs for a 3-d domain study $[-4.5, 4.5]^3$:

Initial_search_domain_function_pdim (8, 5, c (-4.5,-4.5,-4.5), c (4.5, 4.5, 4.5), rastrigin)

	L_boundaries	Upper_boundaries
X1:	-0.038243	-0.038243
X2:	-1.145693	1.060902
X3:	0.909735	2.057402
Obj-val:	7.900063	7.013848

Figure 4. Example of the proposed algorithm 1 outputs

The algorithm 1 starts by retrieving for each dimension the lower and the upper boundaries. First, the algorithm generates N points at uniform from the study domain defined by the two boundaries vectors then it computes the corresponding objective function values for those N points. Secondly, we subset the **p** points that are minima in term of the used objective function. After that, we identify for each objective function variable/column the minimum and the maximum values. Those values represent the boundaries of the new study domain, which is the output of algorithm 1.

Concerning the algorithm 2, we call the first algorithm recursively in a number of iterations that is given as input. The purpose is to reduce the study domain as possible to obtain an initial point. In each iteration of the algorithm 2, we adjust the boundaries by adding a very small quantity p/N to the Upper_boundaries_vector and we subtract the same quantity from the Lower_boundaries_vector. Those adjusted vectors will be the inputs for the upcoming call of algorithm 1 and the number of iterations will decrease by one. In each iteration, we retrieve the column where the objective function is minimum. This column represents the current obtained initial point and algorithm 2 continues until the stopping condition is met. The output of the algorithm 2 is the initial point vector that will be used by stochastic RMSPROP to solve the optimization problem.

The next figure gives an example of algorithm 2 result :

Example of R outputs for a 3-d domain study $[-4.5, 4.5]^3$(the obtained initial point is in green):

Initialization (3, 40, 5, c (-4.5,-4.5,-4.5), c (4.5, 4.5, 4.5), rastrigin)

```
[,1]          [,2]
X1:        -0.03516109 0.98802338
X2:        -0.07485103 0.98214379
X3:        -0.06447904 0.05900116
obj-val:    2.149234   2.714765
[1] "Init_sol"
X1:          X2:          X3:           obj-val:
-0.03516109 -0.07485103 -0.06447904  2.149234
[,1]           [,2]
X1:        -0.02814939 -0.02620201
X2:         0.01386379  0.01386379
X3:        -0.01272420  0.01855646
obj-val:    0.2270078   0.2422451
[1] "Init_sol"
X1  :        X2  :        X3  :         obj-val  :
-0.02814939 0.01386379 -0.01272420  0.2270078
[,1]            [,2]
X1:        -0.0120990900  0.014463534
X2:         0.0007967621  0.006058173
X3:        -0.0050612641 -0.004791464
Obj-val  :  0.03423591    0.05330876
[1] "init_sol"
X1  :            X2  :            X3  :             obj-val  :
-0.0120990900  0.0007967621 -0.0050612641  0.03423591
```

Figure 5. Example of the proposed algorithm 2 outputs

Algorithm 2: Initialization // recursive search

Inputs : Nbre_iterations, N, p, Lower_boundaries_vector, Upper_boundaries_vector, Object_fun

D=Call **Initial_domain_search_n_dim** (N, p, Lower_boundaries_vector, Upper_boundaries_vector, Object_fun)

// we initialize the matrix D by calling Algorithm 1

Init_sol=D [, D [nrow(D),] == min (D [nrow(D),])]

// we retrieve from D the unique column where the objective function Obj-val is minimum

D=D [- nrow (D),]

// we delete the last row of D that contains the objective function values (Obj-val).

Repeat {

MIN_t = D [, 1] - p/N

MAX_t = D [, 2] + p/N

// we substruct the quantity p/N from the first column of D (L_boundaries)

// we add the quantity p/N to the second column of D

Nbre_init_iteration=Nbre_init_iteration-1

// The remaining number of iterations is Nbre_init_iteration, we substruct 1 in each call

If ($Nbre_init_iteration == 0$) {

Return (Init_sol)

Break

// we break the function Initialization if the condition is meet and we return the initial solution $Init_sol$

}

D=Call Initialization (Nbre_init_iteration, N, p, MIN_t, MAX_t, Objfun)

// Algorithm 2 is called recursively until Nbre_init_iteration variable is equal to zero.

}

Outputs : Init_sol

The next algorithm presents the suggested hybrid optimizer :

Algorithm 3: Hybrid Stochastic RMSPROP optimizer with random search initialization for large domains

Inputs : Nbre_iterations, N, p, Lower_boundaries_vector, Upper_boundaries_vector, Object_fun, α, γ.

X_init = Call **Initialization** (Nbre_iterations, N, p, Lower_boundaries_vector, Upper_boundaries_vector, Object_fun)

X_sol= Call **Stochastic_RMSPROP** (X_init, α, γ)

// The final obtained solution of the proposed hybrid optimizer

Outputs : X_sol

Our optimizer starts by reducing recursively the study domain in a number of iterations **Nbre_iterations** (which is the role of algorithm 2). After reporting an initial solution by algorithm 2, the hybrid optimizer uses the stochastic version of RMSPROP to obtain an accurate solution to the problem for the considered objective function.

To improve the solution **X_sol** we obtained by the proposed hybrid optimizer, we suggest applying a number of operations such as : replacing all this vector components by the minimum / maximum values , integer part of components , rounding of values , etc. The purpose of algorithm 4 (presented as R code) is to compare some possible modifications of the obtained solution . The output of algorithm 4 is a solution better than **X_sol** :

```
1
2
3 Improvement<-function(x_sol, objFun ,rounds_to_try){
4 # x1 to x10 represent the possible improvements we could make .
5
6 x1=x_sol                          # The first row of M : The solution x_sol itself
7 x2=rep(min(x_sol),length(x_sol))  # The second row of M : replace all x_sol components by
     its minimum
```

```
x3=rep(max(x_sol),length(x_sol))  # The third row of M : replace all x_sol components by its
     maximum
x4=as.integer(x_sol)              # For each of x_sol components , we replace the component
    by the integer part

x5=ceiling(x_sol) # ceiling() takes a single numeric argument x and returns a numeric vector
     containing the smallest integers not less than the corresponding elements of x.

x6=rep(min(ceiling(x_sol)),length(x_sol))
x7=rep(max(ceiling(x_sol)),length(x_sol))

x8=floor(x_sol)  # floor() takes a single numeric argument x and returns a numeric vector
    containing the largest integers not greater than the corresponding elements of x.

x9=rep(max(floor(x_sol)),length(x_sol))
x10=rep(min(floor(x_sol)),length(x_sol))

# Trying to round solution
c11=round(x_sol, digits = 0)

for (i in 1:rounds_to_try){
c11=rbind(c11,round(x_sol, digits = i))  # The i-th row of c11 contains the roundig of x_sol
     until the i-th digit

}
M=rbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11)   # we rowbind the possible improvements / we
    store them in the matrix M rows .

M=cbind(M,"Objfun"=apply(M,1,objFun)) # For each row / possible improvement , we compute the
     objective value associated with the row

X_improved=M[order(M[,"Objfun"], decreasing = FALSE),][1,]
print("Decision")  # We print the best improvement we had made
print(X_improved)
print("Matrix of possible improvements") # We print the matrix M
print(M)

return(X_improved)
}
```

Listing 4: Algorithm 4 - Improving the hybrid optimizer solution by rounding operations

6.1. principles analysis of the suggested optimizer

To present how our algorithm is efficient, we should first mention that the used two-stages algorithm combines a slightly modified random search version with the accuracy features of the RMSprop. The input of the proposed optimizer is an n-dimensional volume. First of all , the algorithm 1 samples N points and retireve the p vectors with the minimum objective values. Secondly , the p vectors are stored in a matrix Y where columns represents the n-dimensions and rows represents the p vectors. Therefore , we retrieve for each column of the Y matrix its minimum and its maximum. The combining of those minima and maximas represents the new rectangluar search domain.

Given an interval $[a, b]$ In one-dimensional , consider a sample of N points drawn at uniform such that : $S_1 = \{x_1, x_2, ..., x_N\}$ then consider its graph G=$\{(x_1, f(x_1), ..., (x_N, f(x_N)\} \, where$:

f(x_1) $\geq f(x_2)... \geq f(x_N) \; and \; x_N \geq ... \geq x_{N-p} \geq ... \geq x_1$ (Without losing of generality).

Then , for the global solution x^* :

$P(x^{(1)}_{min} \leq x^* \leq x^{(1)}_{max}) = \frac{x^{(1)}_{max} - x^{(1)}_{min}}{b-a}$

Such that S_1 boundaries are : $x^{(1)}_{min} = min\{S_1\}$ and $x^{(1)}_{max} = max\{S_1\}$ (the power index (1) represents the current iteration).

As explained at [45] , consider a region of $v = 5\%$ around the minimum , the drawn sample has a chance of 5% in landing within the minimum interval. The probability that all of the N points miss the desired interval is :

$(1 - v)^N = (1 - 0.05)^N$

So the probability of the N points to miss the desired interval is : $1 - (1 - v)^N$. As we want a confidence level of 95% then we should have : $1 - (1 - v)^N > 0.95 \Rightarrow N = 60$

Now consider the points $S_2 = \{x_{N-p} ..., x_N\}$ which represents the obtained p minimum values of the first iteration. Suppose that the upper boundaries of search domain for the second iteration are : $x^{(2)}_{min} = x_{N-p}$ and $x^{(2)}_{max} = x_N$.

For the set S_2 , we are sure with 95% that at least one element is belongign to the minimum region . By adding a fluctuation term of p/N , we avoid situations where x^* isn't within S_2 boundaries $[L^{(2)} = x^{(2)}_{min}; U^{(2)} = x^{(2)}_{max}]$.

Now consider the search domain of the third iteration $[L^{(2)} - p/N; U^{(2)} + p/N]$ and sample N points again uniformly. Construct the new interval L$^{(2)}; U^{(2)}$] and correct it to be $[L^{(3)} - p/N; U^{(3)} + p/N]$.

Repeat this procedure $nbre_iterations$ times until reaching the last iteration. After that , Report the solution with the minimum objective value denoted **x_init**. This solution will be improved by the Rmsprop variant.

For proving the convergence accuracy of the stochastic RMSPROP, we refer the reader to [46]. We had tested our optimizer in several multi-dimensional test functions that vary on their complexity and some of the obtained results are given in table 11 .

6.2. *Illustration example*

To illustrate how our algorithm works, we will consider the simple 4d sphere function case (n=4).

The study domain is $[-6, 6]^4 = [-6, 6] \times [-6, 6] \times [-6, 6] \times [-6, 6]$(it represents the optimizer input) so boundaries of the search domain are : Lower_boundaries_vector = (-6,-6,-6,-6) and Upper_boundaries_vector = (6,6,6,6)

The used parameters are : N_iter=3 , N=7 , p=2 , Lower_boundaries_vector=rep(-6,4) , Upper_boundaries_vector=rep(6,4) , Objfun=Sphere_1_pdim , Rmsprop_iter=1000 , alpha = 0.1 , lambda=0.9 , rounds_to_try=10

From this domain $[-6, 6]^4$, we will sample in each iteration $N = 7$ vectors (belonging to 4-dimensional space) and we will report the $p = 2$ vectors such as the objective values represents the p lowest values of the matrix Y (The two first rows).

After obtaining a sub-matrix of Y with p rows , we will retrieve the minimum and maximum of each column. The next step is to subtract the quantity $p/N = 3/7 = 0.285$. We obtain the New_search_domain (used in the next iteration 2) then we repeat the same procedure $N_iter = 3$.

In the last iteration $N_iter = 3$, we consider the actual New_search_domain then we report the column with the minimum objective value. This columns is the initial solution X_init that will be used by the RMSPROP (We run $Rmsprop\ (X_init, \alpha, \lambda)$ $Rmsprop_iter = 1000$ times).

After running RMSPROP , we obtain X_sol (which is the algorithm 3 output). We improve this solution by applying several rounding operations (We obtain the $X_improved$ solution).

The next R output illustrates that **our conceived optimizer was able to obtain the exact solution** of the 4-dimensional sphere function :

```
1 [1] "p/N"      "p=2"    "N=7"  "n=4"       "0.285714285714286"
2 [1] " Y matrix  : We had created N=7 vectors  , The last column represents the objfun value "
3          X1          X2         X3         X4          objfun
4 [1,]   0.02317865   0.7321702   4.548808  -1.7116222   24.15791
5 [2,]   3.11638165  -3.7814377   2.502486   4.4049393   49.67703
6 [3,]  -1.48218395   3.8111884   3.514621   5.4698777   58.99415
```

```
7  [4 ,]   4.49967960  0.4711529 −3.227314 −5.5347297  61.51789
8  [5 ,]  −2.57771804 −4.7701161 −2.364299 −5.4679915  64.88748
9  [6 ,]   5.92790742 −2.9609501 −5.290385  0.6381693  72.30275
10 [7 ,]  −5.65442471  4.9268807 −5.455438  5.6672393 118.12608
11 [1] "We selected the first p−rows "
12              Y                                       objfun
13 [1 ,] 0.02317865  0.7321702 4.548808 −1.711622 24.15791
14 [2 ,] 3.11638165 −3.7814377 2.502486  4.404939 49.67703
15 [1] "iteration 1"
16 [1] "New_search_domain"
17          L_bound     U_bound
18 X1       −0.2625356  3.402096
19 X2       −4.0671520  1.017884
20 X3        2.2167714  4.834522
21 X4       −1.9973365  4.690654
22 Objfun   25.5140791 57.985182
23
24 [1] "p/N"      "p=2"    "N=7" "n=4"        "0.285714285714286"
25 [1] " Y matrix : We had created N=7 vectors , The last column represents the objfun value "
26        X1             X2          X3          X4           objfun
27 [1 ,]   1.555821678 −0.6240687 2.599677 −0.2381384   9.625071
28 [2 ,]   0.504681790 −2.6029133 3.128620  0.8054785 17.466920
29 [3 ,]   3.027363246 −0.3299426 2.959313 −1.3225050 19.780341
30 [4 ,]   2.302834340 −0.7104443 4.091197  2.3137839 27.899270
31 [5 ,]   0.467910245 −2.7245188 4.674959  2.5998943 36.256637
32 [6 ,]  −0.004128654 −0.8988267 4.084195  4.5336900 38.042899
33 [7 ,]   2.379866222 −1.4864051 4.112094  4.4163327 44.286476
34 [1] " We selected the first p−rows "
35              Y                                       objfun
36 [1 ,] 1.5558217 −0.6240687 2.599677 −0.2381384   9.625071
37 [2 ,] 0.5046818 −2.6029133 3.128620  0.8054785 17.466920
38 [1] " iteration 2"
39 [1] " New_search_domain "
40          L_bound     U_bound
41 X1        0.2189675  1.8415360
42 X2       −2.8886276 −0.3383544
43 X3        2.3139623  3.4143342
44 X4       −0.5238527  1.0911928
45 Objfun   14.0209595  16.3541182
46
47 [1] "p/N"      "p=2"    "N=7" "n=4"        "0.285714285714286"
48 [1] " Y matrix : We had created N=7 vectors , The last column represents the objfun value "
49       X1           X2          X3          X4         objfun
50 [1 ,] 1.3125472 −0.7363105 2.361524  0.5717645  8.168643
51 [2 ,] 0.5649406 −2.4966890 2.635841 −0.4315972 13.686547
52 [3 ,] 1.5607746 −1.9408374 2.801027 −0.2324693 14.102661
53 [4 ,] 1.2189323 −1.6282879 3.199503 −0.2039405 14.415526
54 [5 ,] 0.9168073 −2.6514770 2.640698  0.9584484 15.762775
55 [6 ,] 1.6821103 −2.7172811 2.423547 −0.4962821 16.332989
56 [7 ,] 0.6916646 −2.4129567 3.311227 −0.5232971 17.538824
57 [1] " iteration 3"
58 [1] " We selected the first p−rows "
59              Y                                       objfun
60  [1 ,] 1.3125472 −0.7363105 2.361524  0.5717645  8.168643
61 [2 ,] 0.5649406 −2.4966890 2.635841 −0.4315972 13.686547
62 [1] " New_search_domain "
63         L_bound     U_bound
64 X       0.2792264   1.5982614
65        −2.7824033  −0.4505962
66         2.0758096   2.9215552
67        −0.7173115   0.8574788
68 Objfun 12.6432568 12.0282310
69
70 " Algorithme 2 output ( Initialization_phase : This solution will be the start point of RMSPROP ) :"
71
72             [ ,1]              [ ,2]                  [ ,3]                 [ ,4]
73 [1 ,] "1.59826143695565" "−0.450596205313941" "2.92155516538981" "0.857478781442245"
74
75
76 #−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
77
78 " Algorithme 3 output ( Hybrid stochastic RMSPROP optimizer with random search initialization ) :"
79
80 [1] "The minimum of f(x) is 0.00926212229767579 at position x = −0.0457854473974009"
81 [1] "X_sol"
82             [ ,1]       [ ,2]          [ ,3]         [ ,4]
83 [1 ,] −0.04578545 0.0312465 7.579842e−07 0.07867319
84
85 #−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
86
87 [1]  "Decision after improvement phase ( The optional algorithm 4 ) "
88 [1]  "X_improved"
89                              Objfun
90      0       0       0       0       0
91 [1]  "Table of possible improvements we made ( algorithm 4 ) "
92                                                              Objfun
```

```
93  x1   -0.04578545  0.03124650   7.579842e-07  0.07867319  0.009262122
94  x2   -0.04578545 -0.04578545  -4.578545e-02 -0.04578545  0.008385229
95  x3    0.07867319  0.07867319   7.867319e-02  0.07867319  0.024757886
96  x4    0.00000000  0.00000000   0.000000e+00  0.00000000  0.000000000
97  x5    0.00000000  1.00000000   1.000000e+00  1.00000000  3.000000000
98  x6    0.00000000  0.00000000   0.000000e+00  0.00000000  0.000000000
99  x7    1.00000000  1.00000000   1.000000e+00  1.00000000  4.000000000
100 x8   -1.00000000  0.00000000   0.000000e+00  0.00000000  1.000000000
101 x9    0.00000000  0.00000000   0.000000e+00  0.00000000  0.000000000
102 x10  -1.00000000 -1.00000000  -1.000000e+00 -1.00000000  4.000000000
103 x11   0.00000000  0.00000000   0.000000e+00  0.00000000  0.000000000
104      -0.05000000  0.03000000   0.000000e+00  0.08000000  0.009800000
105      -0.04600000  0.03100000   0.000000e+00  0.07900000  0.009318000
106      -0.04580000  0.03120000   0.000000e+00  0.07870000  0.009264770
107      -0.04579000  0.03125000   0.000000e+00  0.07867000  0.009262256
108      -0.04578500  0.03124600   1.000000e-06  0.07867300  0.009262020
109      -0.04578540  0.03124650   8.000000e-07  0.07867320  0.009262119
110      -0.04578545  0.03124650   7.600000e-07  0.07867319  0.009262122
111      -0.04578545  0.03124650   7.580000e-07  0.07867320  0.009262122
112      -0.04578545  0.03124650   7.580000e-07  0.07867319  0.009262122
113                               Objfun
114       0          0          0          0          0
```

For this simple case , the next figure presents the evolution for each component based on the number of iterations. Each curve represents one of the 4 dimensions :

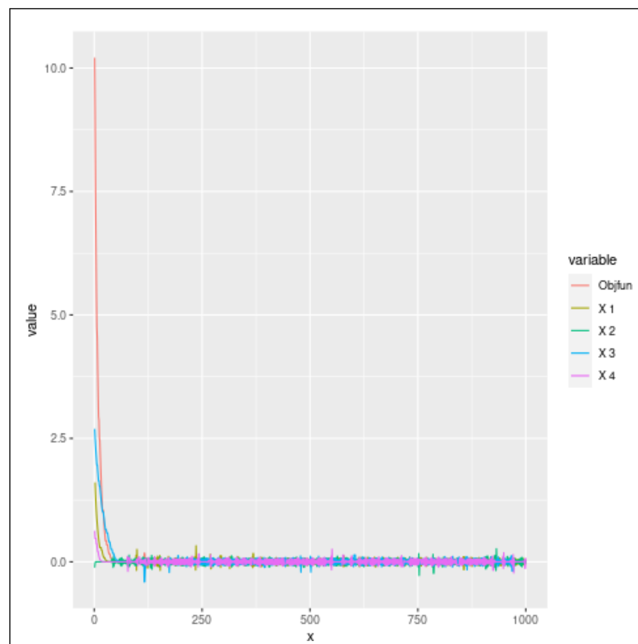


Figure 6. Convergence plot of the conceived optimizer for the 4-d shpere function based on number of iterations

Table 11. Some simulation results of the suggested hybrid optimizer

Test function	Study domain	initial obtained solution (Algorithm 2)		Obtained hybrid optimizer solution (Algorithm 3)	Objective function value of the solution
Sphere (dim = 13)	$[-6,6]^{13}$	$Nbre_iter = 5$ $N = 20$ $P = 4$ Iter = 50 $\alpha = 0.1$ $\gamma = 0.99$	$X1 = -0.553850768$ $X2 = -1.493074761$ $X3 = 0.6767506500$ $X4 = -0.602909613$ $X5 = -1.047292736$ $X6 = -0.191411525$ $X7 = -0.079137285$ $X8 = 0.2588126665$ $X9 = -1.444664397$ $X10 = -0.92061842$ $X11 = 0.424461785$ $X12 = -0.38351937$ $X13 = -0.15875219$	$X1 = -4.9999e-09$ $X2 = -5.0000e-09$ $X3 = -5e-09$ $X4 = -5e-09$ $X5 = -5e-09$ $X6 = -5e-09$ $X7 = -5.000003e-09$ $X8 = -4.307553e-09$ $X9 = -5.0000006e-09$ $X10 = -5e-09$ $X11 = -5.000723e-09$ $X12 = -5.392374e-09$ $X13 = -5.00000000053859e-09$	$3.22640e-16$
Himmelblau's (dim = 2)	$[-6,6]^2$	$N_ter = 6$ $N = 40$ $P = 5$ Iter = 100 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = 3.156640$ $X2 = 2.194130$	$X1 = 2.99876$ $X2 = 2.01136$	0.001983
Rastrigin function (n = 8)	$[-5.12,5.12]^8$	$N_ter = 13$ $N = 40$ $P = 10$ Iter = 50 $\alpha = 0.1$ $\gamma = 0.9$	$X1 = 0.2634143048$ $X2 = 0.9917922607$ $X3 = 1.2036587692$ $X4 = 1.0961852101$ $X5 = 1.1506293024$ $X6 = 0.0556164503$ $X7 = 0.03749775658$ $X8 = 0.14211747971$	$X1 = -0.007194304$ $X2 = 1.0007044373$ $X3 = 0.994805762$ $X4 = 0.9897889558$ $X5 = 0.9949586327$ $X6 = -0.004661976$ $X7 = 0.00352526731$ $X8 = -0.00242884$	4.00990032

Table 11. Some simulation results of the suggested hybrid optimizer

Ackley (dim = 2)	$[-5, 5]^2$	$N_iter = 13$ $N = 18$ $P = 5$ Iter = 50 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = -0.1406797$ $X2 = -0.1101426$	$X1 = -1.6473706210342e-08$ $X2 = -3.51273940349251e-08$	$1.097385e-07$
Beale	$[-4.5, 4.5]^2$	$N_iter = 13$ $N = 20$ $P = 5$ Iter = 50 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = 2.77737$ $X2 = 0.63832$	$X1 = 2.980414$ $X2 = 0.494759$	$6.54143085562211e-05$
Shekel (dim = 4)	$[0, 10]^4$	$N_iter = 3$ $N = 20$ $P = 5$ Iter = 10^3 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = 3.964629112$ $X2 = 3.287228306$ $X3 = 4.047133078$ $X4 = 4.159858556$	$X1 = 4.0060974456$ $X2 = 3.9907111167$ $X3 = 4.0069265117$ $X4 = 3.9947221534$	-10.51979349

Table 11. Some simulation results of the suggested hybrid optimizer

Ackley (dim = 2)	$[-5,5]^2$	$N_iter = 13$ $N = 18$ $P = 5$ Iter = 50 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = -0.1406797$ $X2 = -0.1101426$	$X1 = -1.6473706210342e-08$ $X2 = -3.51273940349251e-08$	$1.097385e-07$
Beale	$[-4.5,4.5]^2$	$N_iter = 13$ $N = 20$ $P = 5$ Iter = 50 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = 2.77737$ $X2 = 0.63832$	$X1 = 2.980414$ $X2 = 0.494759$	$6.54143085562211e-05$
Shekel (dim = 4)	$[0,10]^4$	$N_iter = 3$ $N = 20$ $P = 5$ Iter = 10^3 $\alpha = 0.01$ $\gamma = 0.9$	$X1 = 3.964629112$ $X2 = 3.287228306$ $X3 = 4.047133078$ $X4 = 4.159858556$	$X1 = 4.0060974456$ $X2 = 3.9907111167$ $X3 = 4.0069265117$ $X4 = 3.9947221534$	-10.51979349

Table 11. Some simulation results of the suggested hybrid optimizer

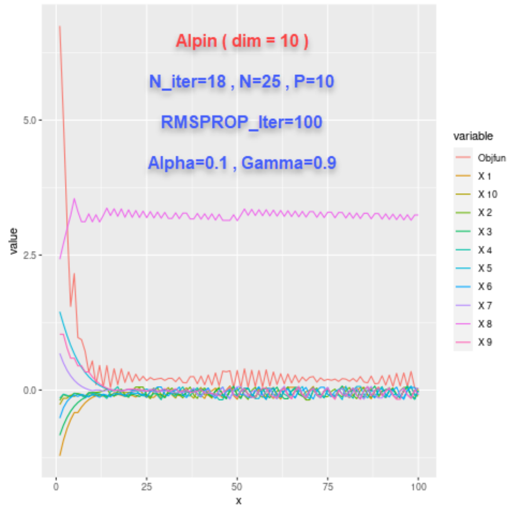
Colville (dim = 4)	$[-10,10]^4$	$N_iter = 13$ $N = 40$ $P = 10$ Iter = 100 $\alpha = 0.1$ $\gamma = 0.9$	$X1 = 1.5033964240414$ $X2 = 1.518321258364$ $X3 = 1.282880998239$ $X4 = 1.814449944069$	$X1 = 1.0495214623$ $X2 = 1.1309573138$ $X3 = 0.97205952961$ $X4 = 0.960630041902$	0.199087493
Schaffer (dim = 2)	$[-100,100]^2$	$N_iter = 17$ $N = 20$ $P = 5$ Iter = 10^3 $\alpha = 0.1$ $\gamma = 0.9$	$X1 = 0.73493131146$ $X2 = 0.8971795174$	$X1 = 1.4414812e-06$ $X2 = -1.871135e-06$	$5.551115123e-15$

The obtained results show that the number of iterations has been significantly reduced with an order of $k \times 10^2$ (where k is an integer & $1 \leq k \leq 3$) at most and the algorithm 2 is able to give initial points that are near the area where the true optimum exists. In the previous table, **N_iter** represents the number of iterations used by the conceived initialization algorithm 2 and **Iter** represents the number of iterations used by the stochastic RMSPROP (algorithm 3).

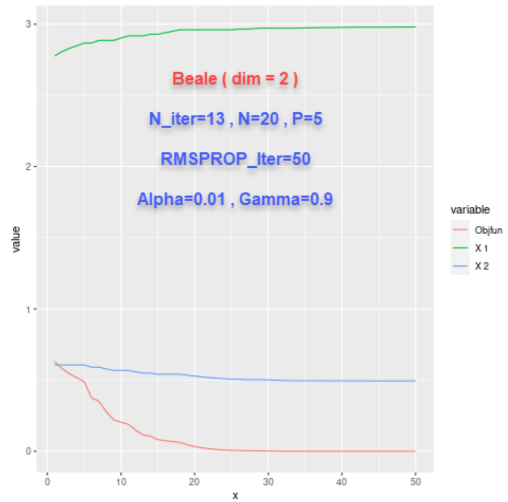
The next figures represents the profiling graphs of the conceived hybrid optimizer based on the used test functions. The figures 7 , 8 and 9 shows that the suggested hybrid optimizer reaches the area where the true optimum exists in the cost of a few number of iterations even for large search domains and regardless of the objective function dimensionality. In most cases, we noticed that a very low amplitude fluctuation is observed at that instant and an approximated solution is then returned. Using the R profiling tools such as the tictoc package , we had collected the CPU time and the accuracy measures . An example of profiling code is as follow :

```
1  library(tictoc)
2  tic("Algorithm used cpu time : ")
3  function\_call() \\
4  toc() \\ # Algorithm used cpu time : 60.026 sec elapsed \\
```

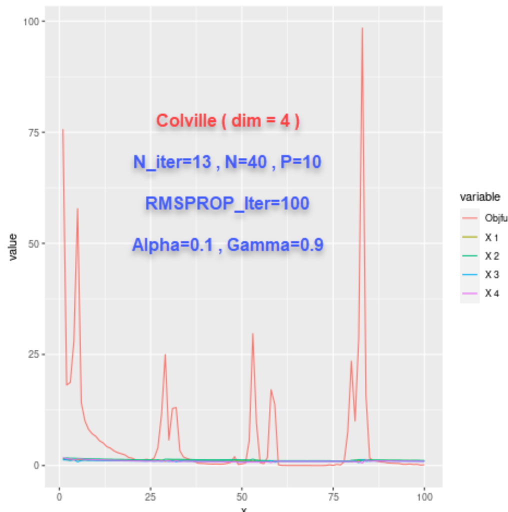
Listing 1: Example of R code profiling



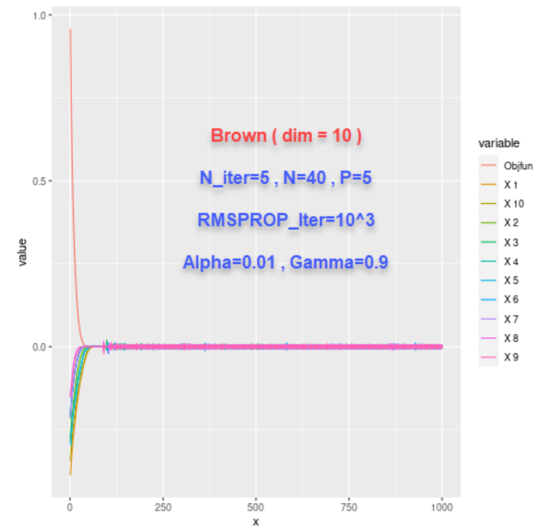
(a) Alpin (dim = 10)



(b) Beale (dim = 2)

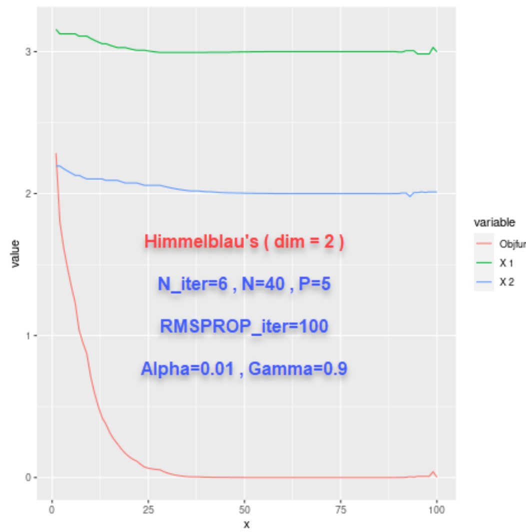


(c) Colville (dim = 4)

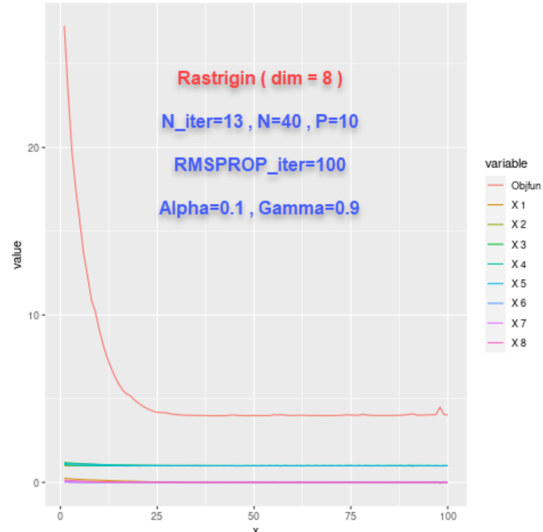


(d) Brown (dim = 10)

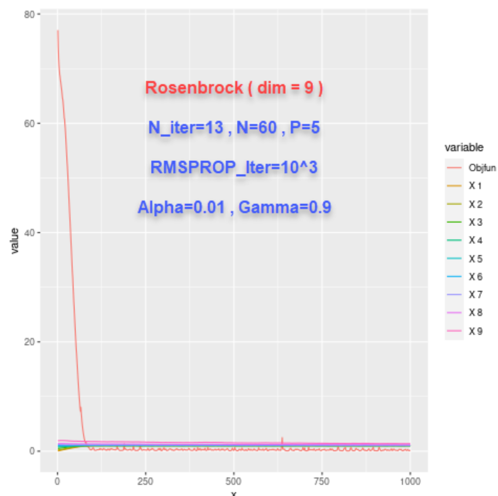
Figure 7. Convergence plots of the conceived hybrid optimizer based on iterations number : part-1



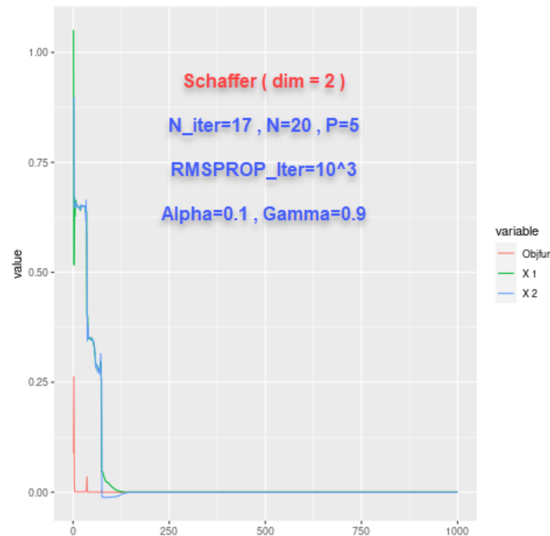
(a) Himmelblau's (dim = 2)



(b) Rastrigin (dim = 8)

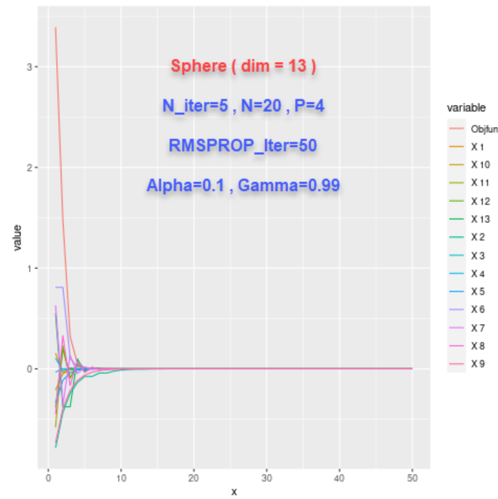


(c) Rosenbrock (dim = 9)

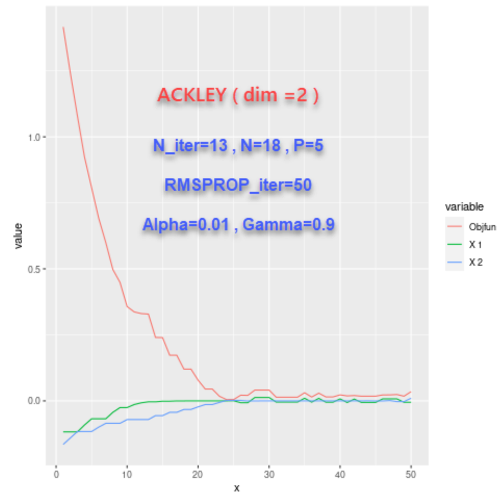


(d) Schaffer (dim = 2)

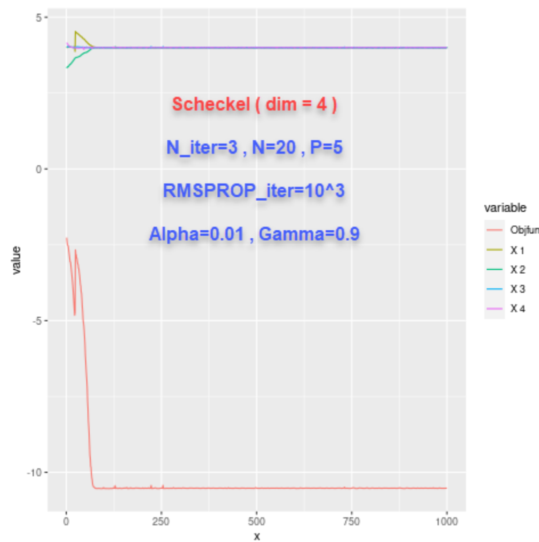
Figure 8. Convergence plots of the conceived hybrid optimizer based on iterations number : part-2



(a) Sphere (dim = 13)



(b) Ackley (dim = 2)



(c) Sheckel (dim = 4)

Figure 9. Convergence plots of the conceived hybrid optimizer based on iterations number : part-3

The figures 7 , 8 and 9 represents the convergence plots of the used test functions. Those plots were drawn using the R ggplot2 package. For each of the used test functions, the curves represents the evolution of each component/dimension based on the number of iterations. In order to measure the convergence speed and the accuracy of the conceived optimizer, we had chosen the following performance metrics in addition to the CPU time [47] :

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i^{opt} - \widehat{x_i}| \qquad (12)$$

and :

$$MPE = \frac{1}{n}\sum_{i=1}^{n}\frac{x_i^{opt} - \widehat{x_i}}{x_i^{opt}} \qquad (13)$$

where x_i^{opt} is the i-th component of the true optimum for the concerned test function and $\widehat{x_i}$ is the i-th component of the computed solution by the algorithm. **Note that the drawback of MPE measure is that it is undefined whenever a single actual value /denominator is zero as mentioned in [48]**. The next table gives the obtained performances results :

Table 12. Summary of the conceived optimizer performance measures

Test function	CPU Mean time (sec)	MAE	MPE
Himmelblaus	0.036	0.01556404	0.00143391
Rastrigin	0.10	0.25610	Undefined
Ackley	0.032	2.580055e-08	Undefined
Sphere	0.020	4.14798e-10	Undefined
Beale	0.056	0.0097928	-0.00850469
Shekel	0.028	0.002322220	-9.642327e-05
Brown	0.064	0.000611665	Undefined
Rosenbrock	0.048	0.00997344	0.0241082
Alpin	0.059	0.32422425	Undefined
Colville	0.055	0.03273932847	0.028292086948
Schaffer	0.040	9.355678e-07	Undefined

To analyze the obtained convergence speed and the accuracy results, we should first remember the reader **that the main purpose is to find an approximate solution in large search domains. Our hybrid optimizer context is for global search optimization which is totally different from the local search context.** If we go back to table 6, the classical RMSPROP alone wasn't able to converge for the rastrigin n-dimensional function and the obtained objective value was $f(x, y) = 31.85$ which is far from the optimum and this is just for a two-dimensional problem. Our suggested optimizer was able to solve the rastrigin problem in 8-dimensions with an error of $MAE = 0.25610$. This error is very small and acceptable compared to the search domain and taking into account the truncation error which means that the real error is strictly lower than the MAE value of the rastrigin problem. Concerning the Schaffer and Alpin functions, the objective value at the true optimum is equal to zero. The proposed hybrid optimizer was able to minimize the objective values for those functions to be equal respectively to 5.551115123e-15 and 0.02013423150 for the large study domains $[-100, 100]^2$ and $[0, 10]^{10}$. The proposed hybrid optimizer consumes fewer resources of CPU time/number of iterations compared to the classical stochastic RMSPOP.

The results of table 12 shows that the suggested optimizer is able to find relevant approximate solutions for the considered artificial landscape problems.

After finding the approximate solution in global search, any other local search optimizer algorithm as well as our suggested optimizer, could be used by the practitioner if he desires to improve the obtained solution accuracy. It should be mentioned that among the used test functions , there are functions that vary greatly locally (which are sensitive to small variations of the input space). This will result in rapid amplification of the objective function locally for small perturbation. **Using the suggested algorithm 4 , we were be able to improve the accuracy of the obtained solutions**. The results are presented in table 13 :

Table 13. Final obtained solutions after using the optional algorithm 4

Test function	Real solution	Final obtained solution (algorithm 4)	Error
Himmelblaus	(3,2)	(3,2)	0
Rastrigin	(0, . . . 8 times . . . ,0)	(0, . . . 8 times . . . ,0)	0
Ackley	(0,0)	(0,0)	0
Sphere	(0, . . . 13 times . . . ,0)	(0, . . . 13 times . . . ,0)	0
Beale	(3,0.5)	(2.980414,0.494759)	0.02027509
Shekel	(4,4,4,4)	(4,4,4,4)	0
Brown	(0, . . . 10 times . . . ,0)	(0, . . . 10 times . . . ,0)	0
Rosenbrock	(1, . . . 9 times . . . ,1)	(1, . . . 9 times . . . ,1)	0
Alpin	(0, . . . 10 times . . . ,0)	(0, . . . 10 times . . . ,0)	0
Colville	(1,1,1,1)	(1,1,1,1)	0
Schaffer	(0,0)	(0,0)	0

Except the case of the beale function , the algorithm 4 was able to find the exact solutions. In the next section, we will propose two real applications of the proposed hybrid optimizer. Those applications concern the mechanical and the biology fields.

7. Real applications

7.1. Mechanical application

Consider the mechanical problem of a beam embedded on one side and subjected to a concentrated load P on the other. The beam has a length l. Its material has Young's modulus E and the section has an inertia I.This structure is modeled by a beam element of the Bernoulli type of length l, whose degrees of freedom in the xOy plane are the two rotations θ_1 and θ_2 as well as the two translations V_1 and V_2. The purpose is to minimize the potential energy stored in the structure is given by the sum of the internal energy and the work of the applied loads, also called compliance. The objective function is formulated as described in [49][50] with the limit conditions : $\theta_1 = 0$ and $V_1 = 0$.

If we consider $x_1 = V_2$, $x_2 = \theta_2 l$ and $\frac{Pl^3}{EI} = 1$, then the objective is to minimize in $[-5, 5]^2$:

$$\Pi = \frac{EI}{2l^3}(12\ V_2^2 + 4\theta_2^2 l^2 - 12\ V_2 l\theta_2) + PV_2$$

which is equivalent to minimize:

$$f(x_1, x_2) = 12x_1^2 + 4x_2^2 - 12x_1x_2 + 2x_1$$

The true optimum is at $(x_1, x_2) = (\frac{-1}{3}, \frac{-1}{2})$ with $f(x_1, x_2) = -0.3333...$

Using the suggested hybrid algorithm with parameters ($N = 20$, $p = 2$, $\alpha = 0.01$, $\gamma = 0.98$) and a total of 56 iterations, we had found :

$\mathrm{x}_1 = -0.3331000$, $x_2 = -0.4996330$

with :

$\mathrm{f}(\mathrm{x}_1, x_2) = -0.3333332$, $CPU_time = 0.019s$, $MAE = 0.0003001666$ and $MPE = -0.00071699999$

7.2. biology application

In a biology experiment, we study the relationship between the concentration of the substrate [S] and the reaction rate in an enzymatic reaction from data reported in the following table [51]:

Table 14. Experimental data for an enzymatic reaction

index	1	2	3	4	5	6	7
S	0.038	0.194	0.425	0.626	1.253	2.500	3.740
rate	0.050	0.127	0.094	0.2122	0.2729	0.2665	0.3317

The purpose is to find the optimal non-linear regression parameters for the following model :

$$rate = \frac{V_{max}.[S]}{K_M + [S]}$$

The minimum least squares optimization concerns the parameters V_{max} and K_M. The true optimal parameters are :

$V_{max} = 0.362$ and $K_M = 0.556$.

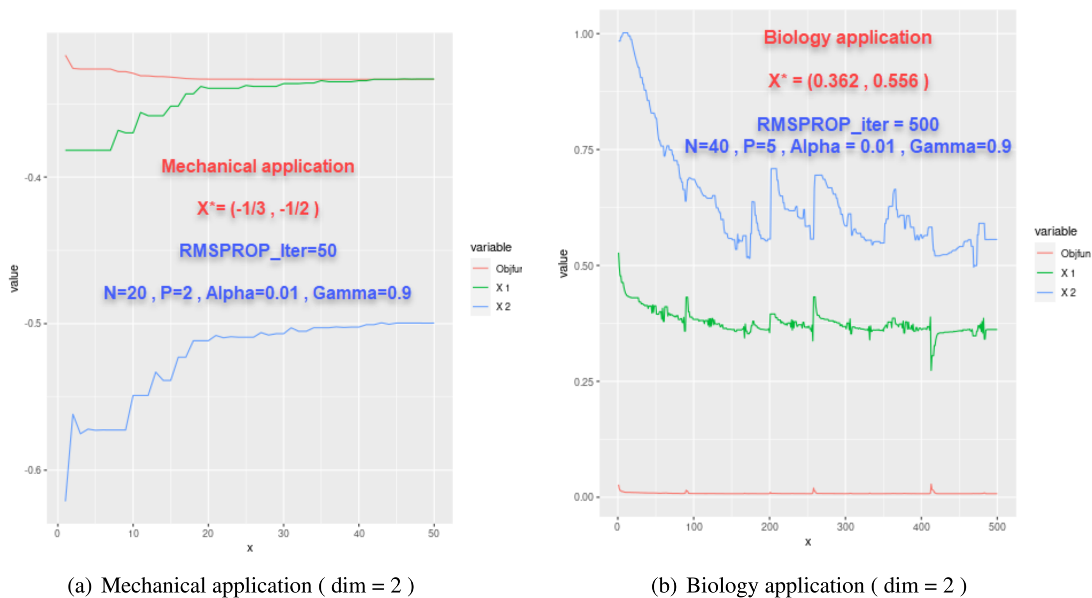
Using the suggested hybrid optimizer with the parameters ($N = 40$, $p = 5$, $\alpha = 0.01$, $\gamma = 0.9$) and a total of 55 iterations , we had found :

$x_1 = 0.3686288885411317$, $x_2 = 0.584903810979226$

with :

$MSE = 0.00787198729199282$, $CPU_time = 0.044s$, and $MAE = 0.005866082032$

In this application, we used the same study domain of the previous mechanical problem as input for the suggested hybrid optimizer. The next figures give the convergence plot for those applications :



(a) Mechanical application (dim = 2)

(b) Biology application (dim = 2)

Figure 10. Convergence plots of the conceived hybrid optimizer for the two real applications : Part-1

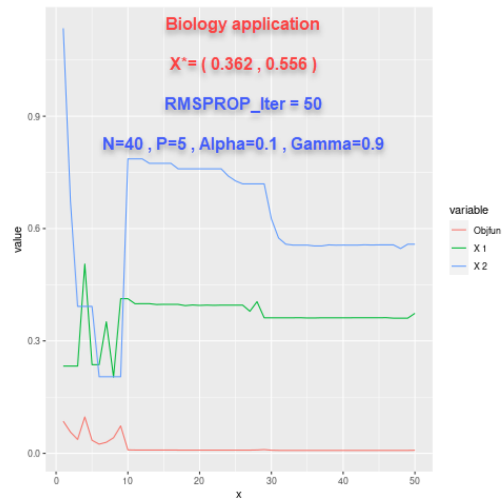


Figure 11. Convergence plots of the conceived hybrid optimizer for the two real applications : Part-2

8. Discussion and conclusion

In this paper, we had compared some variants of the well-known descent gradient algorithm in a benchmark of five test functions. We proved a statistical test using 120 experiments samples that the performance metrics depend on the chosen version of the algorithm regardless of the considered problem features. After that, we have arranged those variants based on their usage priority using the AHP decision technique in two-dimensional. We had found that RMSPROP has a priority of usage equal to 50.73%. Based on those results that are coherent with previous related works, we suggested a new hybrid optimizer that combines a recursive random search of initial points and the accuracy features of the stochastic RMSPROP. The features of the conceived algorithm consist of its ability to use the given study domain as input in addition to reaching approximate solutions for global optimization problems with the use of fewer resources. In a benchmark of 11 multi-dimensional test functions that vary on their complexity and dimensionality, we proved that our hybrid algorithm can effectively reduce useless iterations of the classical stochastic RMSPROP, which results in faster convergence speed. The simulation results of the new proposed hybrid algorithm show the obtention of accurate and significant results for the considered test functions in large search domains. Two real applications of mechanical and biology fields were proposed.

9. Acknowledgment

This work was supported by the Moroccan CNRST: National Center for Scientific and Technical Research.The authors are grateful to the referees for their valuable comments and suggestions.

Appendix :Test functions

The used test functions are from : [39, 40, 52, 53]

Test function	Study domain	Formula	Global minimum at
Rastrigin (dim = n)	$-5.12 \leq x_i \leq 5.12$	$f(x_1 \cdots x_n) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$	$f(0,\cdots,0) = 0$
Ackley (dim = 2)	$-5 \leq x_i \leq 5$	$f(x_0 \cdots x_2) = -20exp(-0.2\sqrt{\frac{1}{2}\sum_{i=1}^{2} x_i^2}) - exp(\frac{1}{2}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$	$f(0,0) = 0$
Rosenbrock (dim=n)	$\mathbb{R}^N$	$f(x_1 \cdots x_n) = \sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$	$f(1,1,\cdots,1) = 0$
Sphere (dim = n)	$\mathbb{R}^N$	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} x_i^2$	$f(0,\cdots,0) = 0$
Beale (dim = 2)	$-4.5 \leq x_i \leq 4.5$	$(1.5 - x + x.y)^2 + (2.25 - x + x.y^2)^2 + (2.625 - x + x.y^3)^2$	$f(3,0.5) = 0$
Alpine (dim = n)	$0 \leq x_i \leq 10$	$\sum_{i=1}^{n} \mid x_i.sin(x_i) + 0.1.x_i \mid$	$f(0,\cdots,0) = 0$
Brown (dim = n)	$-1 \leq x_i \leq 4$	$\sum_{i=1}^{n-1}[(x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)}]$	$f(0,\cdots,0) = 0$
Schaffer (dim = 2)	$-100 \leq x_i \leq 100$	$f(x,y) = 0.5 + \frac{sin(x^2 - y^2) - 0.5}{[1 + 0.001 \cdot (x^2 + y^2)]^2}$	$f(0,0) = 0$
Colville (dim = 4)	$-10 \leq x_i \leq 10$	$100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 19.8(x_2 - 1).(x_4 - 1) + 10.1((x_2 - 1)^2) + (x_4 - 1)^2))$	$f(1,1,1,1) = 0$
Himmelblaus	$-6 \leq x_i \leq 6$	$(x^2 + y - 11)^2 + (x + y^2 - 7)^2$	$f(3,2) = 0$ $f(-2.805...,3.131) = 0$ $f(-3.779...,-3.283) = 0$ $f(3.584...,-1.848) = 0$
Shekel (dim = 4)	$0 \leq x_i \leq 10$	$f(x) = -\sum_{i=1}^{m}\sum_{j=1}^{4}((x_j - C_{ji})^2 + \beta_i)^{-1}$	$f(4,4,4,4) = -10.5364$

Shekel function parameters :

m=10

$$C = \begin{bmatrix} 4.00 & 1.00 & 8.00 & 6.00 & 3.00 & 2.00 & 5.00 & 8.00 & 6.00 & 7.00 \\ 4.00 & 1.00 & 8.00 & 6.00 & 7.00 & 9.00 & 3.00 & 1.00 & 2.00 & 3.60 \\ 4.00 & 1.00 & 8.00 & 6.00 & 3.00 & 2.00 & 5.00 & 8.00 & 6.00 & 7.00 \\ 4.00 & 1.00 & 8.00 & 6.00 & 7.00 & 9.00 & 3.00 & 1.00 & 2.00 & 3.60 \end{bmatrix}$$

$\beta = \frac{1}{10}[1,2,2,4,4,6,3,7,5,5]^T$

REFERENCES

1. X.-S. Yang, *Optimization Techniques and Applications with Examples*. John Wiley & Sons, 2018.
2. C. T. Kelley, *Iterative methods for optimization*. SIAM, 1999.
3. M. Cavazzuti, *Optimization methods: from theory to design scientific and technological aspects in mechanics*. Springer Science & Business Media, 2012.
4. J.-P. Grivet, *Méthodes numériques appliquées pour le scientifique et lingénieur (edition 2009): Edition 2013*. EDP sciences, 2012.
5. C. Lemaréchal, "Cauchy and the gradient method," *Doc Math Extra*, vol. 251, p. 254, 2012.
6. A. Cauchy, "Méthode générale pour la résolution des systemes déquations simultanées," *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.
7. J. C. Meza, "Steepest descent," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 6, pp. 719–722, 2010.
8. H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.
9. J. Kiefer, J. Wolfowitz, *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.
10. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
11. N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.
12. J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.
13. T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.
14. R. Thangaraj, M. Pant, A. Abraham, and P. Bouvry, "Particle swarm optimization: hybridization perspectives and experimental illustrations," *Applied Mathematics and Computation*, vol. 217, no. 12, pp. 5208–5226, 2011.
15. H.-q. Li and L. Li, "A novel hybrid particle swarm optimization algorithm combined with harmony search for high dimensional optimization problems," in *The 2007 International Conference on Intelligent Pervasive Computing (IPC 2007)*, pp. 94–97, IEEE, 2007.
16. A. Antoniou and W.-S. Lu, *Practical optimization: algorithms and engineering applications*. Springer Science & Business Media, 2007.
17. S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
18. M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International Conference on Machine Learning*, pp. 1225–1234, PMLR, 2016.
19. R. Sweke, F. Wilde, J. J. Meyer, M. Schuld, P. K. Fährmann, B. Meynard-Piganeau, and J. Eisert, "Stochastic gradient descent for hybrid quantum-classical optimization," *Quantum*, vol. 4, p. 314, 2020.
20. W. E. L. Ilboudo, T. Kobayashi, and K. Sugimoto, "Tadam: A robust stochastic gradient optimizer," *arXiv preprint arXiv:2003.00179*, 2020.
21. S. Khirirat, H. R. Feyzmahdavian, and M. Johansson, "Mini-batch gradient descent: Faster convergence under data sparsity," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 2880–2887, IEEE, 2017.
22. J. Ding, L. Han, and D. Li, "An adaptive control momentum method as an optimizer in the cloud," *Future Generation Computer Systems*, vol. 89, pp. 192–200, 2018.
23. J. Duda, "Sgd momentum optimizer with step estimation by online parabola model," *arXiv preprint arXiv:1907.07063*, 2019.
24. G. Goh, "Why momentum really works," *Distill*, vol. 2, no. 4, p. e6, 2017.
25. M. Yaqub, F. Jinchao, M. S. Zia, K. Arshid, K. Jia, Z. U. Rehman, and A. Mehmood, "State-of-the-art cnn optimizer for brain tumor segmentation in magnetic resonance images," *Brain Sciences*, vol. 10, no. 7, p. 427, 2020.
26. "Cs231n convolutional neural networks for visual recognition."
27. H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum sgd for distributed non-convex optimization," in *International Conference on Machine Learning*, pp. 7184–7193, PMLR, 2019.
28. M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.
29. A. Cutkosky and H. Mehta, "Momentum improves normalized sgd," in *International Conference on Machine Learning*, pp. 2260–2268, PMLR, 2020.
30. J. Chen and A. Kyrillidis, "Decaying momentum helps neural network training," *arXiv preprint arXiv:1910.04952*, 2019.
31. S. Reddi, M. Zaheer, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *Proceeding of 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018.
32. N. Zhang, D. Lei, and J. Zhao, "An improved adagrad gradient descent optimization algorithm," in *2018 Chinese Automation Congress (CAC)*, pp. 2359–2362, IEEE, 2018.
33. M. C. Mukkamala and M. Hein, "Variants of rmsprop and adagrad with logarithmic regret bounds," in *International Conference on Machine Learning*, pp. 2545–2553, PMLR, 2017.
34. A. Lydia and S. Francis, "Adagradan optimizer for stochastic gradient descent," *Int. J. Inf. Comput. Sci.*, vol. 6, no. 5, 2019.
35. A. Défossez, L. Bottou, F. Bach, and N. Usunier, "On the convergence of adam and adagrad," *arXiv preprint arXiv:2003.02395*, 2020.
36. R. V. K. Reddy, B. S. Rao, and K. P. Raju, "Handwritten hindi digits recognition using convolutional neural network with rmsprop optimization," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 45–51, IEEE, 2018.
37. M. E. Khan, Z. Liu, V. Tangkaratt, and Y. Gal, "Vprop: Variational inference using rmsprop," *arXiv preprint arXiv:1712.01038*, 2017.
38. D. V. Babu, C. Karthikeyan, A. Kumar, *et al.*, "Performance analysis of cost and accuracy for whale swarm and rmsprop optimizer," in *IOP Conference Series: Materials Science and Engineering*, vol. 993, p. 012080, IOP Publishing, 2020.
39. M. Molga and C. Smutnicki, "Test functions for optimization needs," *Test functions for optimization needs*, vol. 101, p. 48, 2005.
40. N. Andrei, "An unconstrained optimization test functions collection," *Adv. Model. Optim*, vol. 10, no. 1, pp. 147–161, 2008.
41. J. Barzilai and M. A. Dempster, "Measuring rates of convergence of numerical algorithms," *Journal of optimization theory and applications*, vol. 78, no. 1, pp. 109–125, 1993.

42. D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, jun 2004.
43. J.-B. Rakotoarivelo, *Aide à la décision multi-critère pour la gestion des risques dans le domaine financier*. PhD thesis, 2018.
44. E. Mu and M. Pereyra-Rojas, *Practical decision making: an introduction to the Analytic Hierarchy Process (AHP) using super decisions V2*. Springer, 2016.
45. J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.
46. S. De, A. Mukherjee, and E. Ullah, "Convergence guarantees for rmsprop and adam in non-convex optimization and an empirical comparison to nesterov acceleration," *arXiv preprint arXiv:1807.06766*, 2018.
47. C. Vastrad *et al.*, "Performance analysis of neural network models for oxazolines and oxazoles derivatives descriptor dataset," *arXiv preprint arXiv:1312.2853*, 2013.
48. A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.
49. R. T. Haftka and Z. Gürdal, *Elements of structural optimization*, vol. 11. Springer Science & Business Media, 2012.
50. J.-C. Craveur, M. Bruyneel, and P. Gourmelen, *Optimisation des structures mécaniques: Méthodes numériques et éléments finis*. Dunod, 2014.
51. M. Rouaud, *Calcul dincertitudes*. Paris, France: Creative Commons, 2014.
52. M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.
53. A. A. Abusnaina, A. I. Alsalibi, *et al.*, "Modified global flower pollination algorithm and its application for optimization problems," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 11, no. 3, pp. 496–507, 2019.