Journal of Computational Mathematics
Vol.43, No.1, 2025, 143–173.

http://www.global-sci.org/jcm
doi:10.4208/jcm.2307-m2023-0016

A PERTURBED QUASI-NEWTON ALGORITHM FOR BOUND-CONSTRAINED GLOBAL OPTIMIZATION*

Raouf Ziadi[1)] and Abdelatif Bencherif-Madani
Laboratory of Fundamental and Numerical Mathematics (LMFN), Department of Mathematics, Faculty of Sciences, University Ferhat Abbas Setif 1, Setif, Algeria
Emails: ziadi.raouf@gmail.com, raouf.ziadi@univ-setif.dz, lotfi_madani@yahoo.fr

Abstract

This paper presents a stochastic modification of a limited memory BFGS method to solve bound-constrained global minimization problems with a differentiable cost function with no further smoothness. The approach is a stochastic descent method where the deterministic sequence, generated by a limited memory BFGS method, is replaced by a sequence of random variables. To enhance the performance of the proposed algorithm and make sure the perturbations lie within the feasible domain, we have developed a novel perturbation technique based on truncating a multivariate double exponential distribution to deal with bound-constrained problems; the theoretical study and the simulation of the developed truncated distribution are also presented. Theoretical results ensure that the proposed method converges almost surely to the global minimum. The performance of the algorithm is demonstrated through numerical experiments on some typical test functions as well as on some further engineering problems. The numerical comparisons with stochastic and meta-heuristic methods indicate that the suggested algorithm is promising.

Mathematics subject classification: 90C26, 90C30.
Key words: Global optimization, Limited memory BFGS method, Stochastic perturbation, Truncated multivariate double exponential distribution.

1. Introduction

In this paper we consider the following bound-constrained global optimization problem:

$$\min_{x\in D} f(x), \tag{P}$$

where D is the hyper-rectangle $\prod_{i=1}^{n} D^{(i)}, D^{(i)} = [a^{(i)}, b^{(i)}]$, $n \geq 2$ and the objective function $f(x) : \mathbb{R}^n \to \mathbb{R}$ is differentiable but not necessarily convex. The problem (P) is of interest in many real-world problems involving objective functions which are only differentiable. Numerous algorithms, depending on the regularity of f, have been already proposed, see [7,8,16,27,29]. We are concerned here with differentiable objective functions and no additional smoothness on ∇f is required. As is well known the deterministic methods [10, 16, 26, 27] guarantee theoretically their convergence to the global minimum in a finite number of iterations. However, most of them suffer from computing challenges as the problem's size is relatively high. On the other hand, the stochastic population-based algorithms [3, 9, 24] are practically the most used. Unfortunately, these methods are not based on theoretical results that guarantee their convergence to the

* Received January 20, 2023 / Revised version received April 18, 2023 / Accepted July 4, 2023 / Published online December 25, 2023 /
1) Corresponding author

global minimum and most of them are computationally expensive. Their effectiveness, due to the lack of guidance by a gradient during the searching process, is relatively inferior in terms of convergence speed for this class of problems.

In differentiable optimization, the methods that are currently in active investigation include the conjugate gradient and quasi-Newton's methods [2], which generate a sequence of points $\{x_j\}_{j\in\mathbb{N}} \subset \mathbb{R}^n$ starting from an initial point $x_0 \in \mathbb{R}^n$ following the procedure

$$x_{j+1} = x_j + t_j d_j, \tag{1.1}$$

where d_j is a descent direction for f at x_j and $t_j \in \mathbb{R}^+$ is a step-length which ensures that x_{j+1} is a feasible point with $f(x_{j+1}) \leq f(x_j)$. Without convexity, these methods are limited in applications since often only local minima are obtained. In order to escape from these local minima, several modifications of the procedure (1.1) have been proposed. Pogu *et al.* [20] have proposed, in the case where the search space is a ball, a random perturbation of the gradient method with a fixed step-length and El Mouatasim *et al.* [7,8] have proposed respectively a random perturbation of the reduced and the conditioned gradient methods for constrained global optimization where the objective function is continuously differentiable; in these works, the perturbations are governed by the standard normal distribution $\mathcal{N}(0, I_n)$. Ziadi *et al.* [30] have introduced a competitive conjugate gradient algorithm by adjusting a Gaussian perturbation strategy to a variant of the Polak-Ribière conjugate gradient method to solve bound-constrained and unconstrained global optimization problems where the function's gradient is supposed to be fully Lipschitz. However, the drawback of the aforementioned methods, especially when the dimension of the problem is relatively high, is that an important number of the generated points lie outside the feasible domain, thereby being discarded slow down the algorithm.

To tackle this problem with only a differentiable cost function, we suggest here a direct simulation of a truncated multivariate double exponential law on the hype-rectangle D. The latter actually lends itself to truncation in D more efficiently than a multivariate Gaussian law $\mathcal{N}(\mu, \sigma I_n)$. To the best of our knowledge, the use of a truncated double exponential law is new and in view of the comments in Morgan [14, pp. 100-103], it turns out to be relatively efficient in our case. The rigorous simulation procedure is carried out in Section 4 below.

Moreover, we adjust our new truncated perturbation strategy by giving a new representation of the quasi-Newton methods. We show how to use it efficiently to deal with bound-constrained global optimization problems by combining our developed perturbation strategy with a variant of L-BFGS-B (limited memory Broyden-Fletcher-Goldfarb-Shanno with boundaries) algorithm proposed by Byrd *et al.* [4]. Recall that currently the so called L-BFGS-B algorithm is one of the most efficient quasi-Newton methods for solving large-scale bound-constrained problems due to features of rapid convergence and moderate memory requirement, but is still inadequate for non-convex global optimization. Our proposed method will be called P-LBFGSB (Perturbed L-BFGS-B algorithm). Starting from a point X_0 in D, the new sequence $\{X_k\}_{k\in\mathbb{N}}$ is given by

$$X_{k+1} \in \arg\min\left\{f\big(\mathcal{G}(X_k)\big), f\big(\mathcal{P}_k^1\big), f\big(\mathcal{P}_k^2\big), \cdots, f\big(\mathcal{P}_k^r\big)\right\} \tag{1.2}$$

with

$$\mathcal{P}_k^l = \mathcal{P}_l\big(\mathcal{G}(X_k)\big), \quad l = 1, 2, \ldots, r,$$

where $\mathcal{G}(X_k)$ is the last point obtained by a few iterations using the L-BFGS-B algorithm starting from X_k and $\mathcal{P}_l(\mathcal{G}(X_k))$, for $l = 1, 2, \ldots, r$, are the stochastic perturbations of the point $\mathcal{G}(X_k)$ that are renewed independently at each iteration k, having the following truncated density function:

$$\Gamma^D_{\mu,\sigma}(x) = \frac{1_D(x)}{\displaystyle\int_D N_{\mu,\sigma}(y)dy} \cdot N_{\mu,\sigma}(x), \tag{1.3}$$

where 1_D is the indicator function of D and

$$N_{\mu,\sigma}(x) = (2\,\sigma)^{-n} \exp\left(-\sum_{i=1}^{n} \left|x^{(i)} - \mu^{(i)}\right|/\sigma \right)$$

is the multivariate double exponential density function with σ is a positive scale referred to as the diversity and $\mu = (\mu^{(1)}, \mu^{(2)}, \cdots, \mu^{(n)})$ is a location vector. The perturbations $\mathcal{P}^1_k, \mathcal{P}^2_k, \cdots, \mathcal{P}^r_k$ are i.i.d (independent and identically distributed) random vectors having the common law $\Gamma^D_{\mu,\sigma}$ with the same diversity parameter that decreases to zero slowly enough to prevent the sequence $\{X_k\}_{k\in\mathbb{N}}$ from converging to a local minimum, see Section 3 below.

In what follows, after giving some notations, we briefly describe the quasi-Newton methods in Section 2 and also the L-BFGS-B algorithm. Next we introduce our proposed truncated perturbation strategy in Section 3 to be followed by its simulation in Section 4. In Section 5, a detailed description of the P-LBFGSB algorithm and its convergence properties are given in Section 6. In the last part, numerical results are reported and some conclusions are drawn.

1.1. Notation

We denote by f^* the global minimum of $f(\cdot)$ in D, i.e. $\min_D f = f^* \in \mathbb{R}$ and x^* is a global minimiser of problem (P). $\nabla f(x)$ designates the gradient of f at the point x. For $x \in \mathbb{R}^n$, $x^{(i)}$ is the i-th component of x and x^T stands for the transpose of x. I_n is the $n \times n$ identity matrix and 1_D is the indicator function of D. $\|x\|$ is the Euclidean norm of x and $\|x\|_\infty = \max_{i=1,\dots,n} |x^{(i)}|$. $\Lambda(D) = \|b-a\|$ is the diameter of D, where

$$a = \big(a^{(1)}, a^{(2)}, \cdots, a^{(n)}\big), \quad b = \big(b^{(1)}, b^{(2)}, \cdots, b^{(n)}\big).$$

The projection of x onto the feasible space D is $P_D(x)$, where

$$P^{(i)}_D(x) = \min\big\{ \max\big(a^{(i)}, x^{(i)}\big), b^{(i)}\big\}, \quad i = 1, \dots, n.$$

For the sets A and B, we denote by $A - B$ the set of points that are in A but not in B. The symbol $a \simeq b$ means that a is nearly equal to b, meas$(\cdot)$ stands for the Lebesgue measure on $\mathbb{R}^n$, r.v. for random variable or vector, a.s. for almost surely, and i.i.d. for independent and identically distributed. $\mathbb{P}(A|B) = \mathbb{P}(A \cap B)/\mathbb{P}(B)$ is the conditioned probability of A given B, with $\mathbb{P}(B) > 0$.

For notational simplicity, for $l = 1, \dots, r$ we set $\mathcal{P}^l_k$ instead of $\mathcal{P}_l(\mathcal{G}(X_k))$, and for $k = 0, 1, \dots,$ we set $\mathcal{G}_k$ instead of $\mathcal{G}(X_k)$, where $\mathcal{G}(X_k)$ designates the last point obtained by the L-BFGS-B algorithm after a few iterations, starting from X_k.

2. Quasi-Newton Methods

Quasi-Newton methods are widely used in solving large-scale unconstrained local optimization problems. They use the updating formulas for the approximation of the Hessian. Among the most successful methods are the BFGS and especially L-BFGS (limited memory BFGS).

They start from an initial point $x_0 \in \mathbb{R}^n$ and generate a sequence of points $\{x_j\}_{j\in\mathbb{N}}$ by the process (1.1), where $t_j > 0$ is a step-length which is determined by a line search procedure (usually chosen so that it satisfies the Wolfe line search conditions) to ensure a sufficient decrease of f and d_j (the descent direction) is of the form

$$d_j = -H_j \nabla f(x_j), \tag{2.1}$$

where H_j is the inverse Hessian approximation matrix updated by the following formula:

$$H_{j+1} = V_j^T H_j V_j + \rho_k s_j s_j^T \tag{2.2}$$

with

$$y_j = \nabla f(x_{j+1}) - \nabla f(x_j), \quad s_j = x_{j+1} - x_j, \quad \rho_j = \frac{1}{y_j^T s_j}, \quad V_j = I - \rho_j y_j s_j^T.$$

In the standard BFGS method, the update formula of H_j in (2.2) needs to stock a full sized n^2-matrix, whereby for large n, the induced cost in terms of memory space and calculations is too important. To reduce the cost, the limited memory BFGS method (L-BFGS) was introduced by Nocedal [13, 15] and is an adaptation of the BFGS method for large scale problems. The only difference is in the matrix update: instead of storing the matrices H_j, the L-BFGS stores the last $\tilde{m}+1$ couples $\{s_i, y_i\}_{i\in j-\tilde{m}}^{j}$, which provides a fast rate of convergence and requires minimal storage, where $\tilde{m} = \min\{j, m-1\}$ and $m \geq 1$ is a given parameter (typically $m = 5$). If the initial approximation matrix H_0 is the identity matrix, then the first m iterations of the BFGS and L-BFGS methods produce exactly the same directions. In the L-BFGS method, the matrix H_j is obtained by updating m times the basic matrix H_0 using BFGS formula with the last m iterations. From (2.2) we see that H_j can be written as

$$\begin{aligned} H_{j+1} = {} & \left[V_j^T \cdots V_{j-\tilde{m}}^T\right] H_0 \left[V_{j-\tilde{m}} \cdots V_j\right] \\ & + \rho_{j-\tilde{m}} \left[V_j^T \cdots V_{j-\tilde{m}+1}^T\right] s_{j-\tilde{m}} s_{j-\tilde{m}}^T \left[V_{j-\tilde{m}+1} \cdots V_j\right] \\ & + \rho_{j-\tilde{m}+1} \left[V_j^T \cdots V_{j-\tilde{m}+2}^T\right] s_{j-\tilde{m}+1} s_{j-\tilde{m}+1}^T \left[V_{j-\tilde{m}+2} \cdots V_j\right] \\ & \cdots\cdots\cdots\cdots \\ & + \rho_j s_j s_j^T. \end{aligned} \tag{2.3}$$

For bound-constrained problems, the L-BFGS-B algorithm (limited memory BFGS for bound-constrained optimization) introduced by Byrd *et al.* [4] is an extension of the L-BFGS algorithm to handle simple bounds. It is based on the gradient projection method and uses a limited memory BFGS matrix H to approximate the inverse Hessian of the objective function. Due to its ability to deal with bounds on the variables, it is considered as one of the most successful large-scale bound-constrained optimization methods. For a given iteration j, the matrix H_j that approximates the inverse Hessian at a point x_j is calculated using the L-BFGS update formula (2.3) and the objective function is approximated by a quadratic model as

$$q_j(x) = f(x_j) + \nabla f(x_j)^T (x - x_j) + \frac{1}{2}(x - x_j)^T H_j (x - x_j).$$

At each iteration, the L-BFGS-B algorithm minimizes $q_j(x)$ subject to D, using the gradient projection strategy to determine a set of active constraints, followed by a minimization of $q_j(x)$ regarding the active bounds as equality constraints. The computation for the generalized Cauchy point and the subspace minimization are the most crucial phases at each iteration j

(for more details see [4]). The objective of the Cauchy point computation is to minimize the quadratic approximation of the objective function $q_j(x)$, starting from the current point x_j, on the path defined by the projection of the steepest descent direction on the feasible domain. After the Cauchy point x^c is obtained, the quadratic function $q_j(x)$ is minimized over the free variables subject to their lower and upper bounds, i.e. the variables that are identified as inside the feasible design space, and then backtracked into the feasible design space to obtain $\tilde{x}$. The new search direction is computed as $d_j = \tilde{x}_j - x_j$ and a step-length t_j is determined in such a way that it satisfies the strong-Wolfe conditions (SW) to compute the new design variable x_{j+1}

$$\begin{aligned} &f(x_j + t_j d_j) - f(x_j) \leq c_1 t_j \nabla f(x_j)^T d_j, \\ &\left|\nabla f(x_j + t_j d_j)^T d_j\right| \leq c_2 \left|\nabla f(x_j)^T d_j\right|, \end{aligned} \tag{SW}$$

where $0 < c_1 < 1/2$ and $c_1 < c_2 < 1$. The matrix H_{j+1} is then computed based on the new point x_{j+1} using the L-BFGS update formula (2.3) and a new iteration is started. The algorithm stops when, for a point x_j, the norm of the projected gradient (in the sup-norm sense) onto the feasible design space is small, i.e. $\|P_D(x_j - \nabla f(x_j)) - x_j\|_\infty \simeq 0$.

3. The Truncated Multivariate Double Exponential Perturbation

As mentioned above, the convergence of the quasi-Newton methods to the global minimum cannot be guaranteed without convexity. We shall overcome this difficulty by using appropriate perturbations. The idea is to perturb the point obtained by the procedure $\mathcal{G}(\cdot)$ provided by a L-BFGS-B variant. The new sequence of iterates, denoted by $\{X_k\}_{k\in\mathbb{N}}$, is obtained using formula (1.2) where the term X_{k+1} is the record point at the step $k+1$. It is the result of a series of convenient perturbations $\mathcal{P}_k^1, \mathcal{P}_k^2, \cdots, \mathcal{P}_k^r$ of the term $\mathcal{G}(X_k)$ (see Algorithm 5.1 below). The main difficulty here is that a significant loss of information, thereby slowing down the algorithm, occurs when an important number of perturbation points are generated outside the feasible domain, especially in large scale problems. Inspired by [30], our idea is to perturb the last point thus obtained by the procedure $\mathcal{G}(\cdot)$ using the truncated multivariate double exponential law whose diversity parameter σ decreases slowly to zero. The perturbations $\mathcal{P}_k^1, \mathcal{P}_k^2, \cdots, \mathcal{P}_k^r$ are renewed independently at each iteration k and the new record point X_{k+1} is chosen according to formula (1.2). The r.v. $\mathcal{P}_k^1, \mathcal{P}_k^2, \cdots, \mathcal{P}_k^r$ are i.i.d. with density $\Gamma^D_{\mathcal{G}_k,\sigma_k}(x)$ obtained from (1.3) by letting $\mu = \mathcal{G}_k$ and $\sigma = \sigma_k$,

$$\Gamma^D_{\mathcal{G}_k,\sigma_k}(x) = \frac{1_D(x)}{\displaystyle\int_D \exp\left(-\sum_{i=1}^n \left|y^{(i)} - \mathcal{G}_k^{(i)}\right|/\sigma_k\right) dy} \exp\left(-\sum_{i=1}^n \left|x^{(i)} - \mathcal{G}_k^{(i)}\right|/\sigma_k\right), \tag{3.1}$$

where the sequence $\{\sigma_k\}_{k\in\mathbb{N}}$ is chosen as follows:

$$\sigma_k = \frac{\Lambda(D)}{\ln(k+n)^\alpha}, \tag{3.2}$$

where $\Lambda(D) = \|b - a\|, n \geq 2$ and $\alpha > 0$ is a parameter to be defined below. From (3.1), for all $k \geq 0$ and $l \in \{1, \ldots, r\}$, $\mathcal{P}_k^l \in D$ a.s., i.e. $\mathbb{P}(\mathcal{P}_k^l \in D) = 1$.

There are three parameters in the definition of $\{\sigma_k\}_{k\in\mathbb{N}}$: $\Lambda(D)$ the diameter of the feasible domain, α the rate of decrease of the sequence $\{\sigma_k\}_{k\in\mathbb{N}}$ and n the dimension of space. In our choice, $\Lambda(D)$ and α have leading roles. The explicit appearance of $\Lambda(D)$ in the numerator allows the possibility to make explorations in all regions of the feasible domain; however, given such a generality and through numerical experiments, it is plain that some extra care is needed to somehow dampen the influence of $\Lambda(D)$ (especially for large-size domains), whence the appearance of n in the denominator. Moreover, concerning the asymptotic behaviour of the sequence $\{\sigma_k\}_{k\in\mathbb{N}}$, it is important, for σ_k to decrease slowly to zero: a logarithm is a first-hand candidate slowly varying function. In this way, the first few terms of the sequence $\{\sigma_k\}_{k\in\mathbb{N}}$ will fit in with $\Lambda(D)$, that is if the search space D is large, the perturbation points will be sufficiently spread out to cover D, diversify the search and avoid stagnation in local minima zones, see Figs. 5.1-5.4.

Note that along the iterations k, the generated perturbations $\mathcal{P}_k^1, \mathcal{P}_k^2, \cdots, \mathcal{P}_k^r$ will increasingly tend to concentrate and after a certain threshold, the perturbations cluster around one point only (which is a global minimum with high probability). The rate of decrease of the sequence $\{\sigma_k\}_{k\in\mathbb{N}}$ is strongly linked to the choice of the parameter α, see the comments of Table 7.3 for its precise influence.

4. Simulation of the Truncated Double Exponential Law $\Gamma_{\mu,\sigma}^D(x)$

We suggest here a proper, though elementary, simulation of $\Gamma_{\mu,\sigma}^D$ which, as pointed out above, is of some efficiency against the truncated Gaussian law, even in a spherical domain. Recall that

$$\Gamma_{\mu,\sigma}^D(x) = \frac{1_D(x)}{\displaystyle\int_D N_{\mu,\sigma}(y)dy} \cdot N_{\mu,\sigma}(x),$$

where

$$N_{\mu,\sigma}(x) = \frac{1}{(2\sigma)^n} \exp\Bigg(- \sum_{i=1}^n |x^{(i)} - \mu^{(i)}|/\sigma \Bigg).$$

Our choice of the law $\Gamma_{\mu,\sigma}^D(x)$ is tailored to fit the situation of our paper. Since

$$D = \prod_{i=1}^n D^{(i)}, \quad D^{(i)} = [a^{(i)}, b^{(i)}], \quad i = 1, 2, \ldots, n,$$

$\Gamma_{\mu,\sigma}^D(x)$ appears as the density of the vector whose independent components are the one-dimensional truncated double exponential laws

$$\Gamma_{\mu^{(i)},\sigma}^{D^{(i)}}(s) = \frac{1_{D^{(i)}}(s)}{\displaystyle\int_{a^{(i)}}^{b^{(i)}} e^{-\frac{|s-\mu^{(i)}|}{\sigma}} ds} e^{-\frac{|s-\mu^{(i)}|}{\sigma}}, \quad i = 1, 2, \ldots, n.$$

The simulation of such a vector is then carried out by simple one-dimensional inversions. By straightforward calculations, for $i = 1, 2, \ldots, n$, we have

$$\begin{aligned}\int_{a^{(i)}}^{b^{(i)}} e^{-\frac{|s-\mu^{(i)}|}{\sigma}} ds &= \int_{a^{(i)}}^{\mu^{(i)}} e^{-\frac{\mu^{(i)}-s}{\sigma}} ds + \int_{\mu^{(i)}}^{b^{(i)}} e^{-\frac{s-\mu^{(i)}}{\sigma}} ds \\ &= \sigma\Big(2 - e^{-\frac{\mu^{(i)}-a^{(i)}}{\sigma}} - e^{-\frac{b^{(i)}-\mu^{(i)}}{\sigma}}\Big).\end{aligned}$$

The cumulative distribution function $F^{(i)}, i = 1, 2, \ldots, n$ of the density $\Gamma_{\mu^{(i)},\sigma}^{D^{(i)}}$ has two forms. When $s \in [a^{(i)}, \mu^{(i)}]$, we have

$$F^{(i)}(s) = \lambda^{(i)}\Big(e^{\frac{s}{\sigma}} - e^{\frac{a^{(i)}}{\sigma}}\Big),$$

where

$$\lambda^{(i)} = \frac{\sigma e^{-\frac{\mu^{(i)}}{\sigma}}}{\displaystyle\int_{a^{(i)}}^{b^{(i)}} e^{-\frac{|s-\mu^{(i)}|}{\sigma}} ds},$$

and when $s \in (\mu^{(i)}, b^{(i)}]$,

$$F^{(i)}(s) = F^{(i)}\left(\mu^{(i)}\right) + \beta^{(i)}\left(1 - e^{-\frac{s-\mu^{(i)}}{\sigma}}\right),$$

where $\beta^{(i)} = \lambda^{(i)} e^{\mu^{(i)}/\sigma}$.

Therefore, for a component $\mathcal{P}^{(i)}, i = 1, 2, \ldots, n$, we draw a uniform r.v. $U^{(i)}$ in $[0, 1]$; we take $\mathcal{P}^{(i)}$ to be $\sigma \log(e^{a^{(i)}/\sigma} + U^{(i)}/\lambda^{(i)})$ or $\mu^{(i)} + \sigma \log((1 - (U^{(i)} - C^{(i)})/\beta^{(i)})^{-1})$ according to the cases $U^{(i)} < C^{(i)}$ and $U^{(i)} \geq C^{(i)}$, where

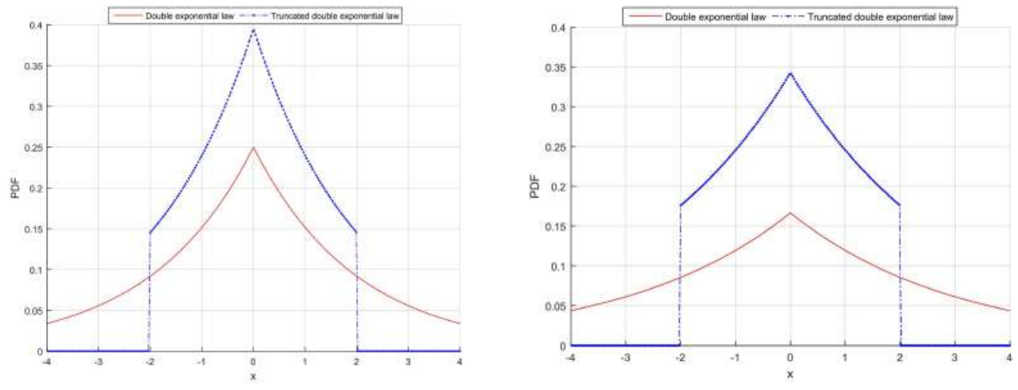
$$C^{(i)} = \lambda^{(i)}\left(e^{\frac{\mu^{(i)}}{\sigma}} - e^{\frac{a^{(i)}}{\sigma}}\right).$$

Algorithm 4.1 below simulates the generation of such a random vector $\mathcal{P}$, i.e. our proposed truncated multivariate double exponential law $\Gamma^D_{\mu,\sigma}$ for given parameters μ and σ.

Algorithm 4.1: Pseudo-Code of the Simulation of the Truncated Double Exponential Law $\Gamma^D_{\mu,\sigma}$.

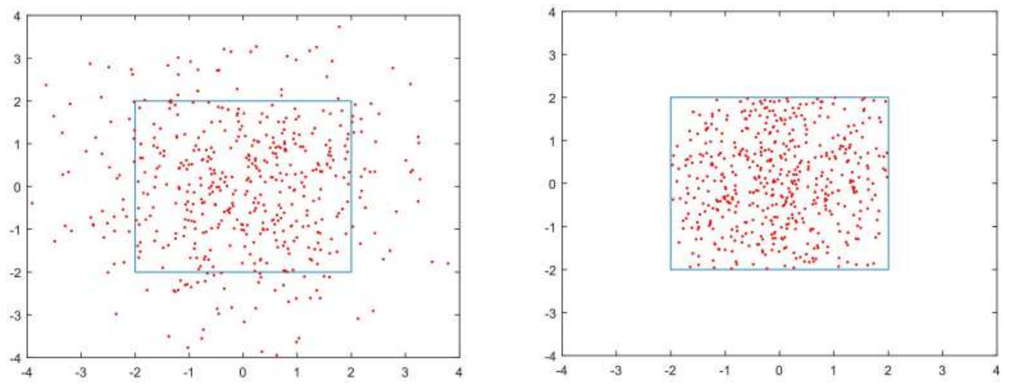
```
   Input  : $n$, lower and upper bounds $a = (a^{(1)}, a^{(2)}, \cdots, a^{(n)}), b = (b^{(1)}, b^{(2)}, \cdots, b^{(n)})$,
            the expected value $\mu = (\mu^{(1)}, \mu^{(2)}, \cdots, \mu^{(n)})$ and diversity parameters $\sigma$.
   Output: $\mathcal{P} = (\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \cdots, \mathcal{P}^{(n)})$.
 1 for $i = 1, \ldots, n$ do
       // Calculate $M^{(i)}, \lambda^{(i)}, \beta^{(i)}$ and $C^{(i)}$ for each coordinate $i$
 2     $M^{(i)} = \int_{a^{(i)}}^{b^{(i)}} e^{-\frac{|s-\mu^{(i)}|}{\sigma}} ds = \sigma\left(2 - e^{-\frac{\mu^{(i)}-a^{(i)}}{\sigma}} - e^{-\frac{b^{(i)}-\mu^{(i)}}{\sigma}}\right)$.
 3     $\lambda^{(i)} = \frac{\sigma}{M^{(i)}} e^{-\frac{\mu^{(i)}}{\sigma}}$.
 4     $\beta^{(i)} = \lambda^{(i)} e^{\frac{\mu^{(i)}}{\sigma}}$.
 5     $C^{(i)} = \lambda^{(i)}\left(e^{\frac{\mu^{(i)}}{\sigma}} - e^{\frac{a^{(i)}}{\sigma}}\right)$.
       // Generate a uniform r.v. $U^{(i)}$ on $[0, 1]$ for each coordinate $i$
 6     $U^{(i)} = \mathcal{U}_{[0,1]}$.
 7     if $U^{(i)} < C^{(i)}$ then Set $\mathcal{P}^{(i)} = \sigma \log\left(e^{\frac{a^{(i)}}{\sigma}} + \frac{U^{(i)}}{\lambda^{(i)}}\right)$.
 8     else Set $\mathcal{P}^{(i)} = \mu^{(i)} + \sigma \log\left(\left(1 - \frac{U^{(i)} - C^{(i)}}{\beta^{(i)}}\right)^{-1}\right)$.
 9 end
10 Return: $\mathcal{P}$.
```

In order to further describe our modification, Fig. 4.1 illustrates the difference between the (univariate) double exponential density and its truncation. Moreover, Figs. 4.2 and 4.3 represent the points generated by the bivariate double exponential and the bivariate truncated double exponential laws on the squares $[-2, 2]^2$ and $[-0.5, 0.5]^2$ with the same diversity parameter.



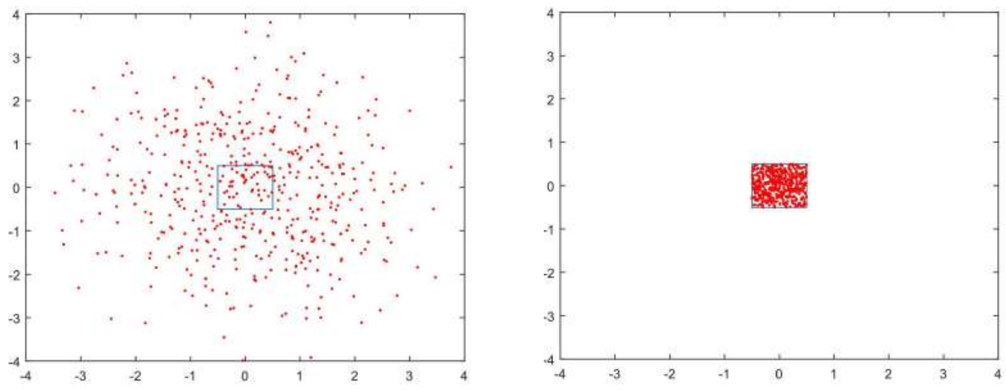
(a) $\sigma = 2$, $\mu = 0$ (b) $\sigma = 3$, $\mu = 0$

Fig. 4.1. Probability density functions on $[-2, 2]$, with different diversity parameters.



(a) (b)

Fig. 4.2. Illustration of 500 points generated by double exponential (a) and our truncated double exponential (b) distributions with $\sigma = 2$ and $\mu = (0, 0)$ in the box $[-2, 2]^2$.



(a) (b)

Fig. 4.3. Illustration of 500 points generated by double exponential (a) and our truncated double exponential (b) distributions with $\sigma = 2$ and $\mu = (0, 0)$ in the box $[-0.5, 0.5]^2$.

5. The P-LBFGSB Algorithm

5.1. Algorithm description

For each step $k \geq 0$ of the P-LBFGSB algorithm, let x_j be the j-th iterated point of the L-BFGS-B local search algorithm starting from $x_0 = X_k$, where the step-length t_j is calculated for each descent direction d_j. At the step k, in each iteration j of the L-BFGS-B algorithm, we update the initial (identity) matrix H_0 for m times using L-BFGS formula to get H_{j+1} according to the Eq. (2.3). Also, as we need to store the last m values of the pairs $\{y_i, s_i\}_{i\in j-\tilde{m}}^{j}$, we can not repeatedly perturb the main iteration (1.1) (since for $j \geq 1$ the matrix H_j is used to determine a descent direction following L-BFGS-B method). In order to exploit H_j for an effective descent while avoiding a premature convergence towards a local minimum, we shall run a few iterations of the L-BFGS-B local search method with $J_{\max} \geq m$, then the last record point thus obtained $x_{J_{\max}}$ is perturbed following the formula (1.2) and the new record point will be noted X_{k+1}. However, if for a certain $j \leq J_{\max}$ we have

$$\left\| P_D\big(x_j - \nabla f(x_j)\big) - x_j \right\|_{\infty} < 10^{-5},$$

then x_j is a local minimizer; in this case, we call upon the stochastic perturbation procedure since the L-BFGS-B method can not be used. In both cases, the new record point will be noted X_{k+1} and is determined following scheme (1.2). We continue this process until the stopping criterion is fulfilled (see Remark 5.1 below). Algorithm 5.1 below summarizes the main steps of the proposed method.

Remark 5.1 (The Stopping Conditions of the Algorithm). As can be seen in Theorem 6.1, the algorithm converges asymptotically to the global minimum with probability one; for this reason, there are two types of stop criteria that can be used to terminate the searching process. Firstly, the searching process can stop when a maximal number of iterations $K_{\max}$ has been reached. Alternatively, one can stop the algorithm if the diversity parameter σ_k reaches a value less than $\sigma_{\min}$ (small enough), since in this case there is a too poor bet seeking for a lower promising region.

Figs. 5.1-5.4 illustrate the work done by the P-LBFGSB algorithm, to minimise the Drop-Wave, Langerman, Michalewicz, and Shaffer 2 functions (by setting $r = 10, \alpha = 3, J_{\max} = 5$).

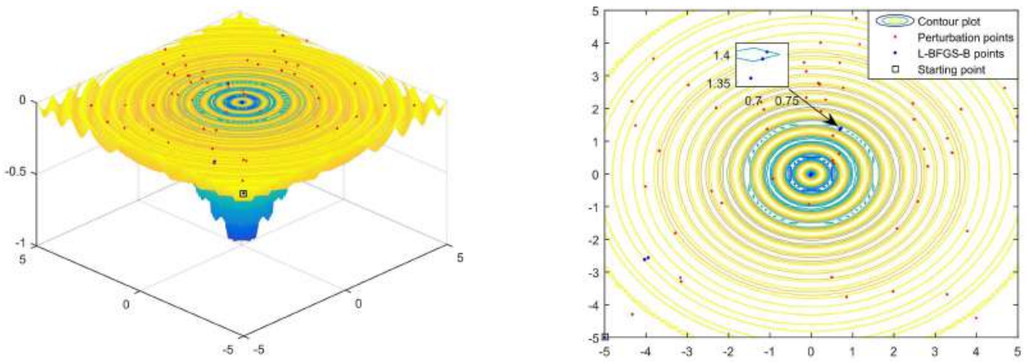


Fig. 5.1. Minimization process of the Drop-Wave function: Global minimizer $x^* = (0, 0)$ with global minimum $f(x^*) = -1$.

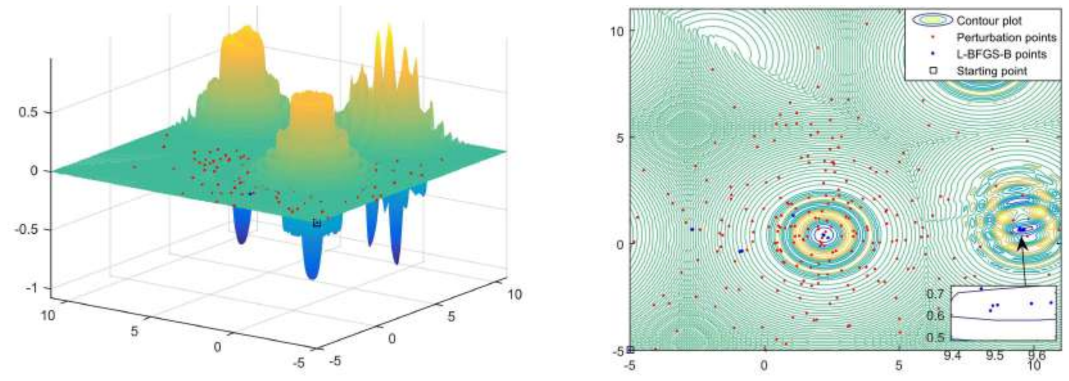


Fig. 5.2. Minimization process of the Modified Langerman function: Global minimizer $x^* = (9.68107, 0.66665)$ with global minimum $f(x^*) = -0.96500$.

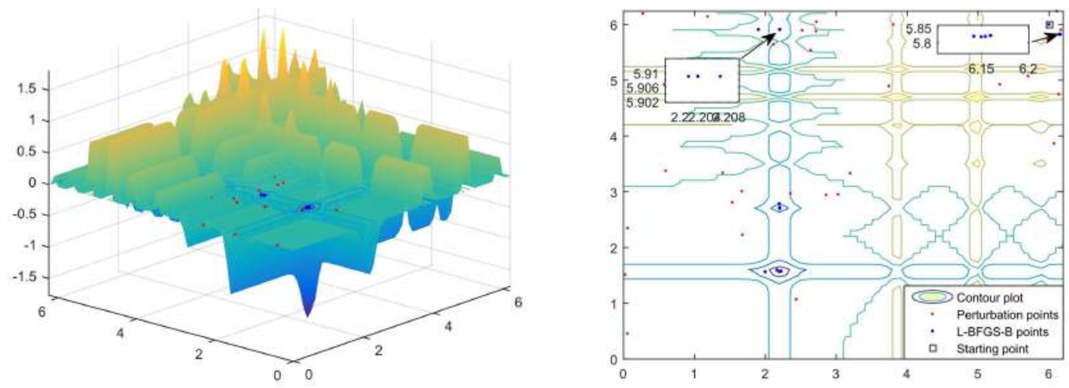


Fig. 5.3. Minimization process of the Michalewicz's function: Global minimizer $x^* = (2.2029, 1.5707)$ with global minimum $f(x^*) = -1.8013$.

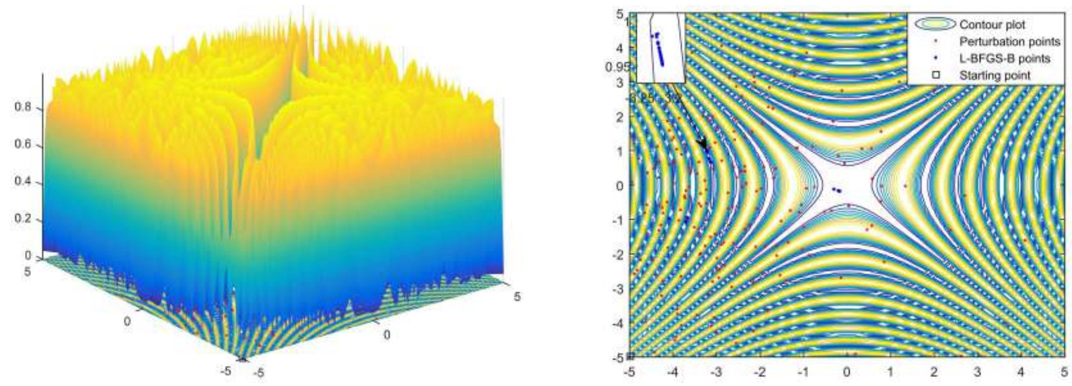


Fig. 5.4. Minimization process of the Schaffer 2 function: Global minimizer $x^* = (0, 0)$ with global minimum $f(x^*) = 0$.

Algorithm 5.1: Pseudo-Code of the P-LBFGSB Algorithm.

```
   Input  : Objective function $f$, search-space Limits: $(a, b)$, dimension of problem $n$,
            define the diversity sequence $\{\sigma_k\}_{k\in\mathbb{N}}$ with parameter $\alpha > 0$, $J_{\max}$ the
            maximum number of iterations in the L-BFGS-B descent procedure, $r$ the
            number of perturbations during a step $k$.
   Output: Global minimum $f^*$ and global minimizer $x^*$.
   // Initialization
 1 Set $k = 0$ and choose randomly $X_0$ in $D$.
 2 Set $f^* = f(X_0)$ and $x^* = X_0$.
   // Main loop
 3 while the stopping criterion not fulfilled do
 4     Set $\sigma_k = \Lambda(D)/(\ln(k+n)^\alpha)$.
       // L-BFGS-B local search phase
 5     Set $j = 0$, $x_0 = X_k$ and set $H_0 = I_n$.
 6     while $j \le J_{\max}$ and $\|P_D(x_j - \nabla f(x_j)) - x_j\|_\infty \ge 10^{-5}$ do
           // Apply the L-BFGS-B local search algorithm, starting from $x_0 = X_k$
              and let $\mathcal{G}(X_k)$ the last point obtained.
 7         Update $H_j$ using the L-BFGS update formula (2.3).
 8         Calculate $d_j$ and $t_j$.
 9         Set $x_{j+1} = x_j + t_j d_j$.
10         $j = j + 1$.
11     end
12     Set $\mathcal{G}(X_k) = x_j$    // $\mathcal{G}(X_k)$ is the last point obtained after $j$ iterations
        starting from $X_k$.
       // Perturbation phase
13     Generate $r$ random vectors i.i.d. $\mathcal{P}_k^1, \mathcal{P}_k^2, \cdots, \mathcal{P}_k^r$ following (3.1) using
        Algorithm 4.1.
14     Select $X_{k+1} \in \arg\min\{f(\mathcal{G}(X_k)), f(\mathcal{P}_k^1), f(\mathcal{P}_k^2), \cdots, f(\mathcal{P}_k^r)\}$
15     $k = k + 1$.
16 end
17 Set $x^* = X_k$, $x^*$ is a prescribed solution and $f^* = f(X_k)$ is the approximated global
    minimum.
   // Get the solution
18 Return $(x^*, f^*)$.
```

6. The Convergence of the P-LBFGSB Algorithm

Let X_0 be a starting point with $f(X_0) > f^*$. The sequence of r.v. $\{f(X_k)\}_{k\in\mathbb{N}}$ where X_k is defined in (1.2), is clearly decreasing and bounded below by f^* a.s. so that it does converge a.s. to a limit greater than or equal to f^*. It only remains to show that the limit is actually f^*. For a level $\theta > f^*$, by monotonicity, it suffices to show that the event $\{f(X_k) \le \theta,\ \text{infinitely often}\}$ holds with probability one, for all real $\theta \in (f^*, f(X_0))$. We are inspired by [23] where the following version of the celebrated Borel-Cantelli lemma is given.

Lemma 6.1. *Let $\{\mathcal{T}_k\}_{k\in\mathbb{N}}$ be a decreasing sequence of r.v. which is bounded below by $\mathcal{T}^* \in \mathbb{R}$. Suppose there exists $\gamma > 0$ such that for all $\theta \in (\mathcal{T}^*, \mathcal{T}^* + \gamma)$ there exists a sequence of positive reals $\{\delta_k(\theta)\}_{k\in\mathbb{N}}$ with*

$$\mathbb{P}\left\{\mathcal{T}_{k+1} \le \theta \,|\, \mathcal{T}_k > \theta\right\} \ge \delta_k(\theta) > 0, \quad \sum_{k=0}^{+\infty} \delta_k(\theta) = +\infty, \tag{6.1}$$

then the sequence of r.v. $\{\mathcal{T}_k\}_{k\ge 0}$ converges to $\mathcal{T}^*$ a.s.

We can make explicit calculations to exhibit the diverging series $\{\delta_k(\theta)\}_{k\in\mathbb{N}}$. Indeed we have the lemma.

Lemma 6.2. *Let $\gamma = f(X_0) - f^* > 0$ and $\theta \in (f^*, f^* + \gamma)$, set $D_\theta = \{x \in D, f(x) < \theta\}$. For all $k \in \mathbb{N}$ we have*

$$\mathbb{P}\left\{f(X_{k+1}) < \theta \,|\, f(X_k) \ge \theta\right\} \ge \big(\sigma_k \pi\, \Lambda(D)\big)^{-\frac{n}{2}} \operatorname{meas}(D_\theta) \exp\big(-n\Lambda(D)/\sigma_k\big). \tag{6.2}$$

Proof. Note that D_θ is compact with non empty interior. Since X_{k+1} is the record up to iteration k, then from (1.2), for $k \in \mathbb{N}$ and $l \in \{1, \ldots, r\}$, we have

$$f(X_{k+1}) \le f(\mathcal{P}_k^l).$$

Since $\mathcal{P}_k^1, \mathcal{P}_k^2, \cdots, \mathcal{P}_k^r$ are i.i.d. random variables, for all $\theta \in (f^*, f^* + \gamma)$ and $l \in \{1, \ldots, r\}$, we have

$$\mathbb{P}\left\{f(X_{k+1}) < \theta, f(X_k) \ge \theta\right\} \ge \mathbb{P}\left\{f\big(\mathcal{P}_k^l\big) < \theta, f(X_k) \ge \theta\right\} = \mathbb{P}\left\{f\big(\mathcal{P}_k^1\big) < \theta, f(X_k) \ge \theta\right\}.$$

It follows that

$$\begin{aligned}
&\mathbb{P}\left\{f(X_{k+1}) < \theta, f(X_k) \ge \theta\right\} \\
&\ge \mathbb{P}\left\{f(\mathcal{P}_k^1) < \theta, f(X_k) \ge \theta\right\} \\
&= \mathbb{P}\left\{\mathcal{P}_k^1 \in D_\theta, X_k \notin D_\theta\right\} \\
&= \mathbb{P}\left\{\mathcal{P}_k^1 \in D_\theta, X_k \in D - D_\theta\right\} \\
&= \int_{D-D_\theta} \frac{1}{\displaystyle\int_D \exp\left(-\sum_{i=1}^n \left|y^{(i)} - \mathcal{G}^{(i)}(X_k)\right|/\sigma_k\right) dy} \\
&\qquad \times \left\{\int_{D_\theta} \exp\left(-\sum_{i=1}^n \left|y^{(i)} - \mathcal{G}^{(i)}(x)\right|/\sigma_k\right) dy\right\} \mathbb{P}\left\{X_k \in dx\right\} \\
&\ge \frac{1}{\displaystyle\int_D \exp\left(-\sum_{i=1}^n \left|y^{(i)} - \mathcal{G}^{(i)}(X_k)\right|/\sigma_k\right) dy} \\
&\qquad \times \inf_{x\in D-D_\theta} \left\{\int_{D_\theta} \exp\left(-\sum_{i=1}^n \left|y^{(i)} - \mathcal{G}^{(i)}(x)\right|/\sigma_k\right) dy\right\} \int_{D-D_\theta} \mathbb{P}\left\{X_k \in dx\right\}.
\end{aligned}$$

Since

$$\sum_{i=1}^n \left|y^{(i)} - \mathcal{G}^{(i)}(X_k)\right|/\sigma_k \ge \|y - \mathcal{G}(X_k)\|^2/\big(\sigma_k\, \Lambda(D)\big),$$

then

$$\begin{aligned}
&\mathbb{P}\left\{f(X_{k+1}) < \theta, f(X_k) \geq \theta\right\} \\
&\geq \frac{1}{\displaystyle\int_D \exp\Big(- \|y - \mathcal{G}(X_k)\|^2 / \big(\sigma_k\, \Lambda(D)\big)\Big) dy} \\
&\quad \times \inf_{x \in D - D_\theta} \left\{ \int_{D_\theta} \exp\left(- \sum_{i=1}^{n} \big|y^{(i)} - \mathcal{G}^{(i)}(x)\big| / \sigma_k \right) dy \right\} \int_{D - D_\theta} \mathbb{P}\left\{X_k \in dx\right\} \\
&\geq \frac{1}{\displaystyle\int_{\mathbb{R}^n} \exp\Big(- \|y - \mathcal{G}(X_k)\|^2 / \big(\sigma_k\, \Lambda(D)\big)\Big) dy} \\
&\quad \times \inf_{x \in D - D_\theta} \left\{ \int_{D_\theta} \exp\left(- \sum_{i=1}^{n} \big|y^{(i)} - \mathcal{G}^{(i)}(x)\big| / \sigma_k \right) dy \right\} \int_{D - D_\theta} \mathbb{P}\left\{X_k \in dx\right\}.
\end{aligned}$$

Since $\operatorname{meas}(D - D_\theta) > 0$, then

$$\int_{D - D_\theta} \mathbb{P}\left\{X_k \in dx\right\} = \mathbb{P}\left\{X_k \notin D_\theta\right\} > 0.$$

Therefore, by the elementary Bayes formula

$$\begin{aligned}
&\mathbb{P}\big\{f(X_{k+1}) < \theta \,|\, f(X_k) \geq \theta\big\} \\
&\geq \frac{1}{\displaystyle\int_{\mathbb{R}^n} \exp\left(- \|y - \mathcal{G}(X_k)\|^2 / \big(\sigma_k\, \Lambda(D)\big)\right) dy} \inf_{x \in D - D_\theta} \left\{ \int_{D_\theta} \exp\left(- \sum_{i=1}^{n} \big|y^{(i)} - \mathcal{G}^{(i)}(x)\big| / \sigma_k \right) dy \right\}.
\end{aligned}$$

By an elementary Gaussian integral, we have

$$\int_{\mathbb{R}^n} \exp\Big(- \|y - \mathcal{G}(X_k)\|^2 / \big(\sigma_k\, \Lambda(D)\big)\Big) dy = \big(\sigma_k \pi\, \Lambda(D)\big)^{\frac{n}{2}}.$$

On the other hand, since we have

$$\sum_{i=1}^{n} \big|y^{(i)} - \mathcal{G}^{(i)}(x)\big| / \sigma_k \leq n \Lambda(D) / \sigma_k,$$

then

$$\mathbb{P}\big\{f(X_{k+1}) < \theta \,|\, f(X_k) \geq \theta\big\} \geq \operatorname{meas}(D_\theta) / \big(\sigma_k \pi\, \Lambda(D)\big)^{\frac{n}{2}} \exp\left(-n\, \Lambda(D)/\sigma_k\right).$$

The proof is complete. $\square$

We are now in the position to state the theorem.

Theorem 6.1. *Let us assume that $f(X_0) > f^*$, then the proposed algorithm converges to the global minimum almost surely.*

Proof. Since the sequence $\{\sigma_k\}_{k \in \mathbb{N}}$ is decreasing, then from Lemma 6.2, for all $\theta \in (f^*, f^* + \gamma)$ and $k \in \mathbb{N}$, we have

$$\begin{aligned}
&\mathbb{P}\big\{f(X_{k+1}) < \theta \,|\, f(X_k) \geq \theta\big\} \\
&\geq \operatorname{meas}(D_\theta) / \big(\sigma_k \pi\, \Lambda(D)\big)^{\frac{n}{2}} \exp\big(- n\, \Lambda(D)/\sigma_k\big) \\
&\geq \operatorname{meas}(D_\theta) / \big(\sigma_0 \pi\, \Lambda(D)\big)^{\frac{n}{2}} \exp\big(- n\, \Lambda(D)/\sigma_k\big).
\end{aligned}$$

On the other hand, by replacing $\{\sigma_k\}_{k \in \mathbb{N}}$ by its value in the Eq. (3.2) and since $\operatorname{meas}(D_\theta) > 0$, then for all $\theta \in (f^*, f^* + \gamma)$ and $k \in \mathbb{N}$, we have

$$\mathbb{P}\{f(X_{k+1}) < \theta \,|\, f(X_k) \geq \theta\} \geq \Big(\text{meas}(D_\theta)/\big(\sigma_0 \pi\, \Lambda(D)\big)^{\frac{n}{2}}\Big)(k+n)^{-n\alpha} > 0.$$

Now, for $\alpha \leq 1/n$ the series with the general term

$$\delta_k(\theta) = \Big(\text{meas}(D_\theta)/\big(\sigma_0 \pi\, \Lambda(D)\big)^{\frac{n}{2}}\Big)(k+n)^{-n\alpha},$$

is diverging, whence the result by Lemma 6.1. □

Inspired by [21], the next theorem deals with the case where f has a unique global minimiser x^* over D. In this situation, the sequence of best solutions $\{X_k\}_{k\in\mathbb{N}}$ converges to x^* almost surely.

Theorem 6.2. *If x^* is the unique global minimiser of f over D, then the sequence of best solutions $\{X_k\}_{k\in\mathbb{N}}$ generated by the P-LBFGSB algorithm converges to x^* almost surely.*

Proof. For all $\delta > 0$, set $S_\delta = B(x^*, \delta)$ and $\tilde{f}_\delta = \min_{x\in D-S_\delta} f(x)$; note that $\tilde{f}_\delta > f^*$. Since the sequence $\{f(X_k)\}_{k\in\mathbb{N}}$ converges to f^* a.s. (from Theorem 6.1), it follows that there exists a set $\mathcal{A} \subset \Omega$ with $\mathbb{P}(\mathcal{A}) = 0$ so for all $\omega \in \mathcal{A}^c$ we have $f(X_k(\omega)) \to f^*$. Then, for each $\omega \in \mathcal{A}^c$, there is an integer $\tilde{k}$ such that for all $k \geq \tilde{k}$ we have

$$f\big(X_k(\omega)\big) - f^* < \tilde{f}_\delta - f^*,$$

so that $f(X_k(\omega)) < \tilde{f}_\delta$. If $X_k(\omega) \in D - S_\delta$, then $\|X_k(\omega) - x^*\| \geq \delta$ and therefore $f(X_k(\omega)) \geq \tilde{f}_\delta > f^*$, which is a contradiction. Therefore, we must have $X_k(\omega) \in S_\delta$ for all $k \geq \tilde{k}$ and the sequence $\{X_k\}_{k\geq 0}$ converges to x^* a.s. □

7. Numerical Experiments

We present here a series of numerical results concerning the P-LBFGSB algorithm applied to a diverse set of test problems found in [19, 29], as well as on three engineering applications. All numerical experiments are implemented in the scientific software MATLAB version R2015a. In all experiments, the P-LBFGSB algorithm is implemented with the setting parameters $\alpha = 3$, $r = 10n$ for the perturbation phase, whereas for the local search algorithm L-BFGS-B, we have used the L-BFGS-B MATLAB code, downloadable from [31], with the setting parameters $J_{\max} = 10, m = 5$ and $H_0 = I_n$, the other parameters are set to their standard values. These parameters are considered standard except for the results in Table 7.3 where the method's response is examined when the tuning parameters assume different values. The number of evaluations involved in the P-LBFGSB algorithm is given by the following formula:

$$Nfg = Nf + n \times Ng, \tag{7.1}$$

where n is the dimension of the problem, Nf and Ng are respectively the number of evaluations of the objective and gradient functions $f(\cdot)$ and $\nabla f(\cdot)$.

To begin with, in order to illustrate the fact that the stochastic perturbation of the L-BFGS-B method does lead to a global minimiser, it is compared in Table 7.2 with three BFGS variants: standard BFGS [2], HANSO [17] and L-BFGS-B [4] using certain non-convex functions taken from Table 7.1. The standard BFGS and HANSO are unconstrained quasi-Newton algorithms, their MATLAB implementations are freely downloadable from [32] and used for

Table 7.1: Test problems.

Problem number	Dimension n	Problem name	Search region	Global minimum
1	2	Schaffer 2 function	$[-10,10]^2$	0
2	2	Drop-Wave function	$[-10,10]^2$	-1
3	2	Shubert function	$[-10,10]^2$	-186.7309
4	2	Bird function	$[-2\pi,2\pi]^2$	-106.764537
5	4	Wood function	$[-30,30]^4$	0
6	4	Colville function	$[-10,10]^4$	0
7, 8, 9, 10, 11, 12	5, 10, 20, 30, 40, 50	Dixon and Price function	$[-10,10]^n$	0
13	10			0.096103
14	20			0.192206
15	30	Cosine Mixture function	$[-30,30]^n$	0.288309
16	40			0.384412
17, 18, 19, 20, 21, 22	5, 10, 20, 30, 40, 50	Exponential function	$[-30,30]^n$	0
23, 24, 25, 26, 27, 28	4, 10, 20, 30, 40, 50	Griewank function	$[-30,30]^n$	0
29, 30, 31, 32, 33, 34	5, 10, 20, 30, 40, 50	Levy and Montalvo 1 function	$[-10,10]^n$	0
35, 36, 37	2, 8, 10	Easom function	$[-20,20]^n$	-1
38, 39, 40, 41	3, 5, 7, 10	Salomon function	$[-10,10]^n$	0
42	2			-1.80130
43	5	Michalewicz function	$[0,\pi]^n$	-4.68765
44	8			-7.66375
45	10			-9.66015
46	2			-1.08093
47	5	Modified Langerman function	$[0,10]^n$	-0.96500
48	7			-0.51700
49	10			-0.96500
50	3		$[0,1]^3$	-3.86278
51	6	Hartamann function	$[0,1]^6$	-3.32237
52, 53, 54	2, 5, 7	Modified Xin-She Yang 3 function	$[-10,10]^n$	0
55, 56, 57	2, 5, 7	Expanded Schaffer function	$[-5,5]^n$	0
58, 59, 60, 61, 62, 63	4, 10, 20, 30, 40, 50	Rosenbrock function	$[-5,5]^n$	0
64, 65, 66, 67, 68, 69	5, 10, 20, 30, 40, 50	Rastrigin function	$[-5.12,5.12]^n$	0
70, 71, 72, 73, 74, 75	4, 16, 20, 24, 40, 50	Powell function	$[-10,10]^n$	0
76, 77, 78, 79, 80, 81	5, 10, 20, 30, 40, 50	Perm's function $(P)_{n,0.5}$	$[-n,n]^n$	0
82, 83, 84, 85, 86, 87	5, 10, 20, 30, 40, 50	Ackley function	$[-30,30]^n$	0
88	5			-195.8299
89	10	Styblinski-Tang function	$[-5,5]^n$	-391.6599
90	20			-783.3195
91	30			-1174.9797
92	4			$m=5,-10.1532$
93	4	Shekel function	$[0,10]^4$	$m=7,-10.4029$
94	4			$m=10,-10.5364$
95	2			4.98151
96	5	Paviani function	$[2.001,9.99]^n$	9.73052
97	10			-45.77847
98	2			-1.4914
99	8	Sine envelope function	$[-10,10]^n$	-10.44047
100	10			-13.41403

Table 7.2: Performance comparison of P-LBFGSB with BFGS, L-BFGS-B and HANSO methods. Part I.

Problem number	x_0	BFGS
		$Nf/Ng/Nfg/CPU(s)$
24	$x_0 = (-2, \cdots, -2)$	84/84/924/0.1077 $x^*_{loc} = (-3.1401, -13.3154, \cdots, 0)$ $f^*_{loc} = 0.0468$
25	$x_0 = (10, 10, \cdots, 10)$	73/73/8036/0.1063 $x^*_{loc} = (9.42, 8.8768, \cdots, 9.8851)$ $f^*_{loc} = 0.1599$
34	$x_0 = (10, 10, \cdots, 10)$	1315/1315/67065/3.1461 $x^*_{loc} = (-1.0422, -1.0346, \cdots, 10.8345)$ $f^*_{loc} = 630.0132$
42	$x_0 = (7, 9)$	44/44/132/0.0843 $x^*_{loc} = (7, 9.0226)$ $f^*_{loc} = -0.3910$
43	$x_0 = (1, \pi, 3, \pi, 2)$	42/42/252/0.1030 $x^*_{loc} = (1, 3.1416, 2.8620, 3.1416, 2)$ $f^*_{loc} = -0.2731$
44	$x_0 = (1, \pi, \pi, 0, 0, \pi, 3, 2)$	58/58/522/0.0906 $x^*_{loc} = (1, 3.1416, 3.1416, \cdots, 1.7560)$ $f^*_{loc} = -1.9806$
45	$x_0 = (0, \pi, \cdots, 0, \pi)$	1/1/11/0.0394 $x^*_{loc} = (0, 3.1416, \cdots, 0, 3.1416)$ $f^*_{loc} = -2.6787\text{e-}309$
46	$x_0 = (3, 10)$	61/61/183/0.0875 $x^*_{loc} = (3.2601, 10.9656)$ $f^*_{loc} = -1.5611\text{e-}04$
47	$x_0 = (9, 2, 1, 5, 1)$	59/59/354/0.1050 $x^*_{loc} = (8.9967, 2.1633, 1.0093, 4.9910, 1.0311)$ $f^*_{loc} = -8.9552\text{e-}04$
65	$x_0 = (-0.3944, 1.2071, \cdots, -0.3944, 1.2071)$	51/51/561/0.1843 $x^*_{loc} = (1.9899, -2.9849, \cdots, -2.9849)$ $f^*_{loc} = 64.6722$
66	$x_0 = (-3.2700, 3.3090, \cdots, -3.2700, 3.3090)$	51/51/1071/0.11119 $x^*_{loc} = (0.9949, -0.995, \cdots, -0.995)$ $f^*_{loc} = 19.8992$
67	$x_0 = (-1, -1, -8, \cdots, -1, -1, -8)$	60/60/1860/0.1402 $x^*_{loc} = (0.9949, 0.9949, \cdots, 7.9592)$ $f^*_{loc} = 656.6558$
68	$x_0 = (1.5, 2.5, 4.2, 3.2, \cdots, 1.5, 2.5, 4.2, 3.2)$	62/62/2542/0.159858 $x^*_{loc} = (0.9949, 1.9899, \cdots, -4.9747)$ $f^*_{loc} = 1.9648\text{e-}03$
69	$x_0 = (-8.5, 4, -2.2, -3, -1, \cdots, -8.5, 4, -2.2, -3, -1)$	58/58/2958/0.1715 $x^*_{loc} = (0, -0.9950, \cdots, -2.9849)$ $f^*_{loc} = 895.4446$
82	$x_0 = (10, 10, 10, 10, 10)$	40/40/240/0.0840 $x^*_{loc} = (9.9949, 9.9949, 10, 10, 10)$ $f^*_{loc} = 17.2919$
84	$x_0 = (10, 10, 10, \cdots, 10)$	40/40/840/ 0.1097 $x^*_{loc} = (9.9949, 9.9949, \cdots, 10)$ $f^*_{loc} = 17.2919$
87	$x_0 = (10, 10, \cdots, 10)$	40/40/2040/ 0.1112 $x^*_{loc} = (9.9949, 9.9949, \cdots, 10)$ $f^*_{loc} = 17.2919$

Table 7.2: Performance comparison of P-LBFGSB with BFGS, L-BFGS-B and HANSO methods. Part II.

Problem number	x_0	L-BFGS-B
		$Nf/Ng/Nfg/CPU(s)$
24	$x_0 = (-2, \cdots, -2)$	57/11/167/0.1173 $x^*_{loc} = (-3.1346, -4.4232, \cdots, 0.0001, -0.0163)$ $f^*_{loc} = 0.0076$
25	$x_0 = (10, 10, \cdots, 10)$	109/21/529/0.1756 $x^*_{loc} = (3.1400, 4.4384, \cdots, -0.0001, -0.0001)$ $f^*_{loc} = 0.0296$
34	$x_0 = (10, 10, \cdots, 10)$	422/52/3022/1.2485 $x^*_{loc} = (-0.9541, -0.9423, \cdots, 10.8340)$ $f^*_{loc} = 2.0883\text{e}{+}03$
42	$x_0 = (7, 9)$	15/3/21/0.0906 $x^*_{loc} = (7, 9.0187)$ $f^*_{loc} = -0.3829$
43	$x_0 = (1, \pi, 3, \pi, 2)$	42/5/67/0.1031 $x^*_{loc} = (1, 3.1416, 2.8620, 3.1416)$ $f^*_{loc} = -0.2731$
44	$x_0 = (1, \pi, \pi, 0, 0, \pi, 3, 2)$	66/8/130/0.1164 $x^*_{loc} = (1, 3.1416, 3.1416, 0, 0, 3.1416, 3.0250, 2.0777)$ $f^*_{loc} = -0.9894$
45	$x_0 = (0, \pi, \cdots, 0, \pi)$	21/1/31/0.0678 $x^*_{loc} = (0, 3.1416, \cdots, 0, 3.1416)$ $f^*_{loc} = -2.6787\text{e-}309$
46	$x_0 = (3, 10)$	12/3/18/0.0915 $x^*_{loc} = (2.9362, 10.0158)$ $f^*_{loc} = -1.2081\text{e-}04$
47	$x_0 = (9, 2, 1, 5, 1)$	27/6/57/0.3511 $x^*_{loc} = (9.0010, 2.0006, 1.0067, 5.0070)$ $f^*_{loc} = -8.9331\text{e-}04$
65	$x_0 = (-0.3944, 1.2071, \cdots, -0.3944, 1.2071)$	50/12/170/0.1151 $x^*_{loc} = (-0.5\text{e-}4, -0.4\text{e-}4, \cdots, -0.5\text{e-}4, -0.4\text{e-}4)$ $f^*_{loc} = 100$
66	$x_0 = (-3.2700, 3.3090, \cdots, -3.2700, 3.3090)$	63/12/303/0.1417 $x^*_{loc} = (-0.9950, 0.9950, \cdots, -0.9950, 0.9950)$ $f^*_{loc} = 19.8992$
67	$x_0 = (-1, -1, -8, \cdots, -1, -1, -8)$	55/8/295/0.1520 $x^*_{loc} = (-0.9950, -0.9950, \cdots, -0.9950, -5.9696)$ $f^*_{loc} = 378.0791$
68	$x_0 = (1.5, 2.5, 4.2, 3.2, \cdots, 1.5, 2.5, 4.2, 3.2)$	35/3/155/0.1389 $x^*_{loc} = (-4.9747, 3.9797, \cdots, 1.9899, -2.9849)$ $f^*_{loc} = 537.2740$
69	$x_0 = (-8.5, 4, -2.2, -3, -1, \cdots,$ $-8.5, 4, -2.2, -3, -1)$	186/22/1286/1.0681 $x^*_{loc} = (-4.9747, 3.9797, \cdots, -2.9849, -0.9950)$ $f^*_{loc} = 417.8805$
82	$x_0 = (10, 10, 10, 10, 10)$	40/4/60/0.0989 $x^*_{loc} = (0.9685, 0.9685, 10, 10, 10)$ $f^*_{loc} = 3.5745$
84	$x_0 = (10, 10, 10, \cdots, 10)$	40/4/120/0.1095 $x^*_{loc} = (0.9685, 0.9685, \cdots, 10)$ $f^*_{loc} = 3.5745$
87	$x_0 = (10, 10, \cdots, 10)$	40/4/240/0.1984 $x^*_{loc} = (0.9685, 0.9685, \cdots, 10)$ $f^*_{loc} = 3.5745$

Table 7.2: Performance comparison of P-LBFGSB with BFGS, L-BFGS-B and HANSO methods. Part III.

Problem number	x_0	HANSO
		$Nf/Ng/Nfg/CPU(s)$
24	$x_0 = (-2, \cdots, -2)$	49/49/539/0.1107 $x^*_{loc} = (-3.1401, -13.3154, \cdots, 0)$ $f^* = 0.0468$
25	$x_0 = (10, 10, \cdots, 10)$	180/180/3780/0.1918 $x^*_{loc} = (-0.0001, -4.4385, \cdots, -0.0001)$ $f^*_{loc} = 0.0172$
34	$x_0 = (10, 10, \cdots, 10)$	1315/1315/67065/3.0851 $x^* = (-1.0422, -1.0346, \cdots, 10.8345)$ $f^*_{loc} = 630.0132$
42	$x_0 = (7, 9)$	13/13/39/0.0999 $x^*_{loc} = (7, 9.0226)$ $f^*_{loc} = -0.3910$
43	$x_0 = (1, \pi, 3, \pi, 2)$	42/42/252/0.1124 $x^*_{loc} = (1, 3.1416, 2.8620, 3.1416, 2.0000)$ $f^*_{loc} = -0.2731$
44	$x_0 = (1, \pi, \pi, 0, 0, \pi, 3, 2)$	58/58/522/0.1114 $x^*_{loc} = (1, 3.1416, 3.1416, \cdots, 1.7560)$ $f^*_{loc} = -1.9806$
45	$x_0 = (0, \pi, \cdots, 0, \pi)$	1/1/11/0.0734 $x^*_{loc} = (0, 3.1416, \cdots, 0, 3.1416)$ $f^*_{loc} = -2.6787\text{e-}309$
46	$x_0 = (3, 10)$	8/8/24/0.0753 $x^*_{loc} = (2.9369, 10.0163)$ $f = -1.2085\text{e-}04$
47	$x_0 = (9, 2, 1, 5, 1)$	4/4/24/0.0755 $x^*_{loc} = (9.001, 2.0002, 1.0068, 5.0071, 1.0046)$ $f^*_{loc} = -8.9330\text{e-}04$
65	$x_0 = (-0.3944, 1.2071, \cdots, -0.3944, 1.2071)$	51/51/561/0.1189 $x^*_{loc} = (1.9899, -2.9849, \cdots, -2.9849)$ $f^*_{loc} = 64.6722$
66	$x_0 = (-3.2700, 3.3090, \cdots, -3.2700, 3.3090)$	51/51/1071/0.1301 $x^*_{loc} = (0.9949, -0.995, \cdots, -0.995)$ $f^*_{loc} = 19.8992$
67	$x_0 = (-1, -1, -8, \cdots, -1, -1, -8)$	60/60/1860/0.1596 $x^*_{loc} = (0.9949, 0.9949, \cdots, 7.9592)$ $f^*_{loc} = 656.6558$
68	$x_0 = (1.5, 2.5, 4.2, 3.2, \cdots, 1.5, 2.5, 4.2, 3.2)$	62/62/2542/0.1926 $x^*_{loc} = (0.9949, 1.9899, \cdots, -4.9747)$ $f^*_{loc} = 457.6776$
69	$x_0 = (-8.5, 4, -2.2, -3, -1, \cdots, -8.5, 4, -2.2, -3, -1)$	58/58/2958/0.1906 $x^*_{loc} = (0, -0.9950, \cdots, -2.9849)$ $f^*_{loc} = 895.4446$
82	$x_0 = (10, 10, 10, 10, 10)$	40/40/240/0.1034 $x^*_{loc} = (9.9949, 9.9949, 10, 10, 10)$ $f^*_{loc} = 17.2919$
84	$x_0 = (10, 10, 10, \cdots, 10)$	40/40/840/0.1191 $x^*_{loc} = (9.9949, 9.9949, \cdots, 10)$ $f^*_{loc} = 17.2919$
87	$x_0 = (10, 10, \cdots, 10)$	40/40/2040/ 0.1393 $x^*_{loc} = (9.9949, 9.9949, \cdots, 10)$ $f^*_{loc} = 17.2919$

Table 7.2: Performance comparison of P-LBFGSB with BFGS, L-BFGS-B and HANSO methods. Part IV.

Problem number	x_0	P-LBFGSB
		$Nf/Ng/Nfg/CPU(s)$
24	$x_0 = (-2, \cdots, -2)$	14637/224/4.3335 $x^* = (-0.0018, -2.7390, \cdots, -0.0045)$ $f^* =$ 9.4607e-05
25	$x_0 = (10, 10, \cdots, 10)$	34479/309/40659/7.0916 $x^* = (-0.0047, -0.0009, \cdots, 0.056)$ $f^* =$ 8.6333e-05
34	$x_0 = (10, 10, \cdots, 10)$	4501/471/28051/12.6333 $x^* = (-1.0001, -1, \cdots, -1)$ $f^* =$ 3.9062e-08
42	$x_0 = (7, 9)$	412/31/474/0.1780 $x^* = (2.2018, 1.5679)$ $f^* = -1.8009$
43	$x_0 = (1, \pi, 3, \pi, 2)$	10560/252/11820/3.3544 $x^* = (2.1941, 1.5743, 1.2865, 1.9241, 1.7178)$ $f^* = -4.68765$
44	$x_0 = (1, \pi, \pi, 0, 0, \pi, 3, 2)$	14666/242/16602/4.7398 $x^* = (2.2020, 1.5712, \cdots, 1.7561)$ $f^* = -7.6637$
45	$x_0 = (0, \pi, \cdots, 0, \pi)$	35954/510/41054/7.3787 $x^* = (2.2029, 1.5708, \cdots, 1.5708)$ $f^* = -9.6601$
46	$x_0 = (3, 10)$	2892/116/3124/0.9723 $x^* = (9.7184, 0.6819)$ $f^* = -1.0809$
47	$x_0 = (9, 2, 1, 5, 1)$	919/52/1179/0.4795 $x^* = (8.0739, 8.7770, 3.4670, 1.8670, 6.7080)$ $f^* = -0.96495$
65	$x_0 = (-0.3944, 1.2071, \cdots, -0.3944, 1.2071)$	51/11/161/0.1314 $x^* = (-0.5049\text{e-}3, -0.4950\text{e-}3, \cdots, -0.495\text{e-}3)$ $f^* =$ 4.9593e-06
66	$x_0 = (-3.2700, 3.3090, \cdots, -3.2700, 3.3090)$	16861/150/19861/5.9594 $x^* = (-0.5148\text{e-}03, 0.1746\text{e-}03, \cdots, 0.1889\text{e-}03)$ $f^* =$ 8.8052e-05
67	$x_0 = (-1, -1, -8, \cdots, -1, -1, -8)$	29446/179/34816/8.2777 $x^* = (0.0580\text{e-}3, 0.0538\text{e-}3, \cdots, 0.1650\text{e-}3)$ $f^* =$ 9.922e-05
68	$x_0 = (1.5, 2.5, 4.2, 3.2, \cdots, 1.5, 2.5, 4.2, 3.2)$	48559/189/56119/12.6417 $x^* = (0.2995\text{e-}3, -0.3979\text{e-}3, \cdots, 0.0775\text{e-}3)$ $f^* =$ 9.6297e-05
69	$x_0 = (-8.5, 4, -2.2, -3, -1, \cdots,$ $-8.5, 4, -2.2, -3, -1)$	107918/409/128368/26.5683 $x^* = (-0.4914\text{e-}4, -0.4912\text{e-}4, \cdots, -0.5011\text{e-}4)$ $f^* =$ 2.4667e-05
82	$x_0 = (10, 10, 10, 10, 10)$	428/44/648/0.1988 $x^* = (0.588\text{e-}04, -0.004\text{e-}04, -0.0326\text{e-}04,$ $0.8711\text{e-}04, 0.2354\text{e-}04)$ $f^* =$ 7.1168e-05
84	$x_0 = (10, 10, 10, \cdots, 10)$	1166/55/1716/0.3418 $x^* = (0.0223\text{e-}04, 0.4126\text{e-}04, \cdots, 0.5494\text{e-}04)$ $f^* =$ 9.5431e-05
87	$x_0 = (10, 10, \cdots, 10)$	41262/636/47622/6.0528 $x^* = (-0.1665\text{e-}04, 0.1029\text{e-}04, \cdots, -0.3017\text{e-}04)$ $f^* =$ 9.8294e-05

Table 7.3: Numerical study of the influential parameters of the P-LBFGSB algorithm. Part I.

Problem number		8		12	
		$Average(Std.dev.)$	$Nf/Ng/CPU(s)$	$Average(Std.dev.)$	$Nf/Ng/CPU(s)$
α	2	3.5050E-06(3.3301E-05)	45123/1185/7.5286	2.9388E+00(4.3123E+00)	53395/7705/53.6261
	2.5	3.9264E-08(5.6691E-07)	40509/889/5.5597	6.9485E-03(3.3325E-01)	49908/10551/75.0566
	3*	1.2146E-04(1.2751E-04)	44085/928/7.2014	3.1451E-01(3.1704E-01)	59430/12355/81.6257
	4	5.7768E-02(2.3940E+00)	39996/799/6.3408	9.9581E+00(4.9824E-01)	45335/7224/53.8717
r	50	3.33241E-02(3.5288E+00)	24584/1352/5.04965	6.2145E+00(1.5088E+00)	38519/12751/84.5804
	100*	1.2146E-04(1.2751E-04)	44085/928/7.2014	3.1451E-01(3.1704E-01)	59430/12355/81.6257
	300	6.4123E-09(2.1482E-10)	98584/824/13.5080	7.2111E-05(7.2541E-05)	102251/11827/78.041
	500	0(0)	156365/636/19.5507	4.4445E-09(2.0828E-11)	161025/11152/82.7466
$j_{\max}$	5	3.2142E-06(2.2145E-04)	34292/525/5.6363	2.2145E-05(2.5241E-05)	42631/5295/38.9917
	10*	1.2146E-04(1.2751E-04)	44085/928/7.2014	3.1401E-01(3.1704E-01)	59430/12355/81.6207
	20	0(0)	53854/2156/9.2480	3.0520E-01(2.5401E-01)	69570/22514/132.0509
$K_{\max}$	100	2.6635E+00(1.3794E+01)	14402/333/2.2189	6.1377E+01(3.1341E+01)	58571/4822/46.8258
	300*	1.2146E-04(1.2051E-04)	44085/928/7.2014	3.1451E-01(3.1714E-01)	59430/12355/81.6257
	500	0(0)	65541/11089/21.7916	1.2247E-06(7.8170E-04)	99420/22514/133.2417
Problem number		26		28	
		$Average(Std.dev.)$	$Nf/Ng/CPU(s)$	$Average(Std.dev.)$	$Nf/Ng/CPU(s)$
α	2	2.9225E-01(1.9228E+00)	48996/2358/4.2943	3.8407E-01(1.38547E+00)	48328/1599/5.0907
	2.5	4.4487E-07(3.3384E-09)	56675/3527/6.5157	2.9580E-03(4.9274E-03)	52330/1908/4.2200
	3*	7.7157E-06(3.6175E-08)	53069/2854/5.4641	1.9145E-03(1.3337E-02)	54825/2224/7.6815
	4	1.9341E-01(5.336E-01)	45883/2238/4.8843	8.9676E-02(6.6186E-01)	51285/2127/6.8120
r	50	3.33243E+00(3.2001E-01)	29021/3624/3.632	4.2877E-01(2.1009E-01)	43535/2452/4.8015
	100*	7.7157E-06(3.6175E-08)	53069/2854/5.4641	1.9145E-03(1.3337E-02)	54825/2224/7.6815
	300	0(0)	96525/2557/6.02147	1.5252E-08(4.4111E-10)	100305/2295/8.9078
	500	0(0)	158095/2105/8.5504	0(0)	159182/1899/9.7621
$j_{\max}$	5	4.21451E-07(2.00014E-06)	39522/1319/2.3691	3.5004E-04(2.4250E-04)	41412/1430/3.9885
	10*	7.7157E-06(3.6175E-08)	53069/2854/5.4641	1.9145E-03(1.3337E-02)	54825/2224/7.6815
	20	0(0)	66907/5584/8.3975	2.0042E-02(8.2142E-04)	86900/4504/10.4414
$K_{\max}$	100	3.0292E-01(3.0140E+01)	15238/1268/2.0391	5.9204E-01(6.2114E-01)	22368/1135/3.8257
	300*	7.7157E-06(3.6175E-08)	53069/2854/5.4641	1.9145E-03(1.3337E-02)	54825/2224/7.6815
	500	0(0)	98628/4331/10.9897	6.8141E-10(6.3224E-12)	78496/4686/11.3768
Problem number		61		67	
		$Average(Std.dev.)$	$Nf/Ng/CPU(s)$	$Average(Std.dev.)$	$Nf/Ng/CPU(s)$
α	2	7.7308E-01(3.0347E-01)	49385/2152/3.7069	6.3528E-05(3.5204E-03)	35245/215/5.2383
	2.5	6.2304E-02(1.7240E-01)	50235/2295/3.4120	3.5240E-07(2.0047E-08)	38387/322/7.3291
	3*	2.0057E-02(4.9915E-01)	48338/2075/3.1083	6.4475E-06(3.9366E-08)	39605/378/9.9254
	4	6.3052E-01(2.2123E-01)	43520/1725/2.2145	1.8604E-02(1.3047E-01)	35027/226/7.3506
r	50	1.0042E-01(1.3387E-01)	27354/2604/2.4801	4.6144E-07(5.9918E-09)	19009/199/6.3197
	100*	2.0057E-02(4.9915E-01)	48338/2075/3.1083	6.4475E-06(3.9366E-08)	39605/378/9.9254
	300	3.1524E-06(8.4417E-08)	108338/2148/4.0530	0(0)	95819/315/22.4487
	500	4.8834E-09(8.3142E-12)	163523/1866/4.4371	0(0)	154330/282/29.9407
$j_{\max}$	5	2.3959E-07(4.3643E-07)	37159/928/1.8307	3.5884E-04(3.5121E-02)	36321/108/7.9952
	10*	2.0057E-02(4.9915E-01)	48338/2075/3.1083	6.4475E-06(3.9366E-08)	39605/378/9.9254
	20	6.2201E-05(2.3500E-03)	68689/5159/5.2545	0(0)	47289/1038/17.5073
$K_{\max}$	100	6.0958E-01(3.9500E+00)	16504/955/1.4498	3.0040E-02(2.9949E-01)	21309/198/6.1132
	300*	2.0057E-02(4.9915E-01)	48338/2075/3.1083	6.4475E-06(3.9366E-08)	39605/378/9.9254
	500	2.0911E-07(3.0133E-09)	115304/4322/9.6950	0(0)	77595/772/20.0537

Table 7.3: Numerical study of the influential parameters of the P-LBFGSB algorithm. Part II.

Problem		79		87	
number		*Average*(*Std.dev.*)	$Nf/Ng/CPU(s)$	*Average*(*Std.dev.*)	$Nf/Ng/CPU(s)$
	2	3.2014E-07(3.3351E-09)	33049/1347/17.8936	6.0242E-01(2.3847E-01)	44284/968/3.3017
α	2.5	0(0)	31118/997/14.1915	4.8387E-03(2.8350E-03)	46057/1127/3.2908
	3*	0(0)	31589/1102/15.5070	5.5978E-03(5.9117E-01)	43143/885/3.0459
	4	2.2518E-03(1.2521E-01)	31210/975/13.6310	2.9284E-03(6.8507E-02)	44342/989/3.3541
	50	4.4084E-06(4.80174E-09)	26508/1189/14.4572	2.2250E-01(1.3312E+00)	28320/827/2.2450
r	100*	0(0)	31589/1102/15.5070	5.5978E-03(5.9117E-01)	43143/885/3.0459
	300	0(0)	91298/872/23.4867	3.3379E-07(9.7738E-09)	108243/1086/4.8570
	500	0(0)	151133/796/34.0575	0(0)	167917/989/4.8379
	5	8.9281E-05(1.2812E-06)	30535/728/8.7986	2.3514E-01(2.2141E-01)	36235/324/1.43253
$j_{\max}$	10*	0(0)	31589/1102/15.5070	5.5978E-03(5.9117E-01)	43143/885/3.0459
	20	0(0)	50232/3067/27.1711	1.6603E-02(1.3811E-03)	59385/2014/5.2083
	100	7.4554E-04(3.9004E-02)	16833/777/8.8896	1.4030E-01(1.4899E-01)	18202/399/0.8597
$K_{\max}$	300*	0(0)	31589/1102/15.5070	5.5978E-03(5.9117E-01)	43143/885/3.0459
	500	0(0)	61735/3233/34.9535	3.9499E-09(5.7342E-12)	90555/2093/3.5009

all the experiments of Table 7.2 with default parameters setting (using weak Wolfe line search conditions). The same initial point is used for all methods for each test problem. For the BFGS and HANSO algorithms, each implementation has been stopped when a point x_k such that $\|\nabla f(x_k)\| < 10^{-6}$ is found, whereas for the L-BFGS-B algorithm, each implementation has been stopped when a point x_k such that $\|P_D(x_k - \nabla f(x_k)) - x_k\|_\infty < 10^{-5}$ is found.

As can be seen from these results, the proposed method converges to a global minimiser in a finite number of evaluation points whereas L-BFGS-B, BFGS and its modified version HANSO all converge to a local minimum only.

Before making comparisons (with other methods), we analyse in Table 7.3 the influence of various parameters by considering the basic set of values: $\alpha = 3, r = 100, K_{\max} = 300$ and $j_{\max} = 10$. The performance of the method can be observed under the action of one parameter, the other parameters are kept to their standard values. This will give us a clue on the influence of each parameter in the algorithm. For this study, the proposed algorithm is executed independently 10 times per test problem and we record the average and standard deviation (*Std.dev.*) of the solution error $|f(X_k) - f^*|$. The mean $CPU(s)$-time (in seconds) and the average number of evaluations of the objective and gradient functions are also recorded.

From the numerical results, we see that the convergence of the P-LBFGSB algorithm depends on the choice of the parameters $\alpha, r, J_{\max}$ and $K_{\max}$: α (the rate of decrease of the sequence σ_k), r (the number of perturbations accomplished in each step k), $J_{\max}$ (the maximal number of iterations executed by the L-BFGS-B in each step k) and the maximal number of steps to be performed $K_{\max}$. We can then see that the best choices for α are between 2.5 and 3. For α below 2.5, the values of σ_k decrease slowly with far-off perturbations which increases the possibility of spotting new remote record points; however, in this case, the generated perturbations become important and the algorithm slows down, especially when $n \geq 10$. Contrariwise, when $\alpha \geq 3$ the sequence σ_k decreases more rapidly so that, after a certain number of iterations, most of the generated points cluster around the perturbed point, making it hard to escape from an eventual local minimizer, whence a premature convergence towards a local minimum is possible. As far as $J_{\max}$ and r are concerned, an increment in $J_{\max}$ improves quickly the value of f, but we also risk to stagnate for a while in a local minimizer, so we must increase the parameter r for the stochastic perturbation to generates sufficient points to hope for an escape from stagnation, whence the drawback of having to generate a relatively important number of points to spot the global minimum with the desired accuracy. To sum up, in order to avoid too many points while searching for the global minimum with the desired accuracy, we suggest standard values for these parameters. Of course, these values have to be moderately changed (by increasing $r, J_{\max}$ and decreasing α) when D or n become larger or when the accuracy is more stringent.

In order to show the efficiency of the P-LBFGSB method, it is compared with two stochastic algorithms: G-CARTopt (classification and regression trees) [22] and AESLS (Alienor-Evtushenko-stochastic-local-search) [28], and six well known evolutionary algorithms in global optimization: DE (differential evolution) [24], CMA-ES (covariance matrix adaptation evolution strategy) [12], SPSO (standard particle swarm optimization) [25], HS (harmony search) [11], EO (equilibrium optimizer) [9] and COA (coyote optimization algorithm) [18], by observing the number of function evaluations and the time elapsed by each algorithm to obtain an approximate solution. The solutions of the test problems whose objective functions have diverse analytical expressions and structures are observed and the results of the numerical experiments are reported in Table 7.4. The MATLAB implementations of the methods DE, CMA-ES, SPSO, HS, G-CARTopt, EO and COA are downloadable from [33–39] (with default setting parameters). In all the experiments every computation was terminated as successful when a recorded solution with error satisfying

$$err = |f(x_k) - f^*| \leq 10^{-5}, \tag{7.2}$$

was reached within 5×10^5 function evaluations and whose calculation time does not exceed 100 seconds, otherwise, the computation was considered as failure. The stopping criterion of the P-LBFGSB algorithm (see Algorithm P-LBFGSB step 4) is replaced by the stopping condition (7.2) and the number of function evaluations is calculated by the formula (7.1).

In these comparisons, all test problems have been independently run twenty times by the algorithm under consideration; the mean number of the evaluations and the mean calculation time for each algorithm have been reported; the average number of evaluations of the gradient functions is moreover recorded for the P-LBFGSB. If during the trials, a method has failed at least once, the number of failures was reported. For a given problem, the average number has not been calculated for an algorithm with at least 5 failures in 20 executions; for failures less than 5, the average of the twenty trials is calculated and, for each failure, the maximal number of evaluation points or the maximal CPU-time (depending on the failure case) is associated. The mean value is then displayed with an indication of the number of failures between parentheses. For P-LBFGSB algorithm the average number of the evaluations is calculated by the following formula:

$$Nfg_{mean} = Nf_{mean} + n \times Ng_{mean}.$$

From Table 7.4, in Figs. 7.1 and 7.2, the global performances of the seven algorithms are compared with P-LBFGSB (using their respective performance profiles relative to the number of function evaluations and CPU-time needed to reach the global minimum) under the logarithmic performance profile of Dolan and Moré [6]. For each method, we plot the fraction p of problems for which the method has a number of function evaluations (respectively CPU-time) that is within a factor τ. The top curve in the plot corresponds to the method that solves most problems within a factor τ, for more details see [6].

Table 7.4: Number of function evaluations required by P-LBFGSB and the other methods to reach the global minimum. Part I.

Problem	P-LBFGSB	DE	SPSO	HS	G-CARTopt
number	$Nfg_{mean}/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$
1	3047/0.1365	2640/0.3412	700/0.0716	71232/4.7149	322/0.6959
2	4845/0.4244	6288/0.8093	2996/0.2991	fail(9)	4063/8.1148
3	6677/0.2009	6264/0.8124	4932/0.4895	7436/0.4616	960/1.0887
4	2835/0.0256	2224/0.2202	2855/0.3663	14444/1.2814	645/0.4473
5	2917/0.1437	91160/12.4980	21272/3.0186	fail(20)	5225/15.7623
6	35837/1.8829	14040/1.3116	4965/0.7962	fail(12)	2128/3.9982
7	24225/3.5922	12576/2.0571	5040/1.7474	90375/10.3851(1)	2736/9.2372
8	172649/14.4369	fail(10)	fail(16)	270541/48.5828(4)	fail(20)
9	244494/37.8796(2)	fail(15)	fail(20)	fail(18)	fail(20)
10	fail(8)	fail(11)	fail(20)	fail(20)	fail(20)
11	fail(8)	fail(06)	fail(20)	fail(20)	fail(20)
12	fail(7)	fail(20)	fail(20)	fail(20)	fail(20)
13	28780/6.8223	9648/3.730	13200/2.1121	35044/4.9306	fail(12)
14	38653/7.4508	42616/6.9917	58484/20.6384	341824/82.7057(4)	fail(19)
15	351863/53.3252(1)	107958/18.0329	fail(18)	fail(12)	fail(20)
16	449487/68.5691(2)	219076/38.0083	fail(20)	fail(8)	fail(20)
17	348/0.0637	2012/0.3433	1596/0.2180	4516/0.4808	1251/3.9725
18	1282/0.1048	4492/0.8052	2628/0.5274	12912/1.8332	3138/14.5657
19	95981/1.0748	9536/1.7367	4800/1.6028	41080/9.5732	7712/33.0923
20	144740/1.4437	14588/2.6591	205168/118.1265	358260/127.6444	fail(15)
21	263661/4.5537	19936/3.6051	fail(6)	457972/214.0011(4)	fail(20)
22	fail(9)	25804/4.7529	fail(20)	fail(9)	fail(20)
23	740/0.0515	11664/1.9128	267708/41.5324	fail(12)	fail(6)
24	245006/5.2910(1)	17620/2.8547	350000/70.7129(3)	fail(11)	fail(18)
25	145645/2.0760(1)	12640/2.1517	fail(5)	168542/45.3009(3)	fail(20)
26	208141/4.1927(1)	18380/3.1482	308953/65.3254(4)	fail(20)	fail(20)
27	272296/4.6197(1)	23144/3.9049	114496/78.2541(2)	fail(20)	fail(20)
28	275804/6.3620(1)	27964/4.9080	485968/72.3255(4)	fail(20)	fail(20)
29	1271/0.2054	2344/0.4420	2680/0.4717	3784/0.4402	2144/6.8838
30	2949/0.4055	5348/0.9580	3876/0.9780	9092/1.5253	4606/19.2140
31	9284/1.2377	11176/2.0511	6980/7.0903	28860/7.9062	8934/34.3338
32	16704/2.4027	17756/3.2792	10140/5.6123	484988/184.2204(4)	fail(12)
33	22140/3.4692	24716/4.5897	10808/6.2293	fail(6)	fail(20)
34	20636/3.4644	32636/6.3958	20640/18.6353	fail(15)	fail(20)
35	1456/0.0560	1414/0.1310	1780/0.2452	0.3903/9572	571/0.3755
36	10252/0.8248	8964/0.8048	fail(20)	72604/6.0541(1)	4114/13.0553
37	14241/0.9585	16964/1.49242	fail(20)	124720/12.6486(2)	5293/17.6602
38	215227/7.1459(2)	93663/9.1399	21920/3.1466	fail(20)	251456/82.1066(4)
39	222269/8.4945(4)	fail(17)	fail(20)	fail(20)	fail(12)
40	fail(20)	fail(17)	fail(17)	fail(20)	fail(17)
41	fail(20)	fail(14)	fail(19)	fail(20)	fail(20)
42	1162/0.0919	708/0.1127	1216/0.1512	1240/0.0871	275/0.3019
43	19808/5.6737	3148/0.5283	303504(4)/57.7091	5216/0.5466	fail(5)
44	44551/6.5385	55780/8.7762(1)	56528/10.0644	26464/3.6214	fail(8)
45	36785/6.4542	59580/9.6003	fail(14)	64152/10.1642	fail(15)
46	110/0.1335	3916/0.7080	64040/11.0479	71216/6.9471(1)	968/3.758
47	11050/3.5153	139576/27.6813	54220/11.0580	323496/43.7390(4)	1186/46.4116
48	192518/45.9213(3)	416516/81.3752(4)	103372(1)/29.2819	393362/71.1899(4)	fail(9)
49	fail(10)	fail(20)	52772(1)/17.3358	fail(9)	fail(20)
50	3385/0.2208	1184/0.1219	2025/0.3030	2056/0.3317	614/0.5701

Table 7.4: Number of function evaluations required by P-LBFGSB and the other methods to reach the global minimum. Part II.

Problem	P-LBFGSB	DE	SPSO	HS	G-CARTopt
number	$Nfg_{mean}/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$
51	fail(12)	fail(20)	fail(20)	fail(20)	fail(20)
52	120964/5.3163	fail(16)	3000/0.4500	32432/1.5006	1091/0.8137
53	404649/16.1077(3)	fail(16)	52830/9.7001	fail(20)	10654/22.5418
54	fail(20)	fail(20)	fail(15)	fail(20)	fail(20)
55	38457/5.8930	15900/2.9337	4815/1.0029	372348/35.0889(4)	12842/85.5033(3)
56	20145/8.2041	119412/24.5623	fail(15)	fail(9)	1897/5.9501
57	fail(12)	429826/98.7955	fail(20)	fail(14)	fail(10)
58	12384/2.8475	20820/3.3032	37148/5.4190	fail(12)	fail(12)
59	28947/10.8417	181320/28.5223	276584(2)/67.5560	fail(20)	fail(20)
60	477501/60.0164(3)	fail(8)	fail(20)	fail(20)	fail(20)
61	fail(9)	fail(6)	fail(20)	fail(20)	fail(20)
62	fail(11)	fail(11)	fail (20)	fail(20)	fail(20)
63	fail(9)	fail(20)	fail(20)	fail(20)	fail(20)
64	10102/3.5955	5320/0.8668	176788/32.1008(4)	10364/0.9935	fail(5)
65	12826/2.8450	14116/2.3429	fail(6)	23636/3.3730	fail(20)
66	51524/11.3096	50600/8.4894	fail(20)	455048/108.1896(4)	fail(20)
67	55299/11.7770	142708/23.7796	fail(20)	fail(12)	fail(20)
68	191223/42.6672	360420/60.3150	fail(20)	fail(20)	fail(20)
69	333193/76.5341	fail(20)	fail(20)	fail(20)	fail(20)
70	2285/0.1108	8064/1.2501	4004/0.5319	37844/3.2007	1516/4.7361
71	12723/0.5055	158152/25.4620	140676/39.1219	fail(17)	fail(10)
72	17914/0.7267	317080/52.3128	230492/79.7261	fail(20)	fail(12)
73	20968/0.7664	491152/80.9260	353980/136.0873(3)	fail(20)	fail(20)
74	37523/1.2182	fail(20)	fail(8)	fail(20)	fail(20)
75	47474/1.3446	fail(17)	fail(14)	fail(20)	fail(20)
76	3314/0.1130	3036/0.5258	3292/0.4883	7096/0.8679	1932/6.9557
77	9566/0.5688	7456/1.3319	7324/1.7166	25920/4.8751	6208/32.2524
78	17817/1.0594	18388/5.8843	42476/20.2240	297420/175.3301(2)	fail(10)
79	29909/15.6359	102456/18.3595	132320/106.1886(3)	fail(5)	fail(12)
80	45147/28.2387	49820/39.6211	fail(8)	fail(11)	fail(20)
81	70525/36.8422	72772/85.9157	fail(6)	fail(20)	fail(20)
82	2059/0.3899	5044/0.6717	5336/0.7293	24600/2.3562	6114/26.4751
83	6025/0.0872	9908/1.3178	7344/1.4622	34952/4.6243	6176/29.2344
84	10912/0.1479	19260/2.6075	12308/3.9865	fail(20)	fail(11)
85	24045/0.3049	28344/4.0764	307084/142.9028(2)	fail(20)	fail(20)
86	83679/1.0334	38264/5.4343	fail(20)	fail(20)	fail(20)
87	109618/5.4801	48432/6.9138	fail(20)	fail(20)	fail(20)
88	253840/15.87555(2)	fail(10)	5860/1.0060	252736/16.6449(4)	3475/8.3538
98	fail(8)	fail(10)	302560/84.9217(4)	159228/16.8822(3)	fail(14)
90	fail(11)	fail(15)	fail(11)	360825/66.7224(4)	fail(14)
91	fail(20)	fail(20)	fail(20)	fail(17)	fail(20)
92	25381/2.16665	17220/1.6057	53640/6.8815(1)	158996/9.4665	1803/3.2096
93	29868/1.8686	29304/2.7425	3930/0.6217	fail(14)	fail(9)
94	125304/3.5240(2)	100016/9.6665	4175/0.6783	fail(20)	17808/64.8135
95	404/0.0362	508/0.0705	875/0.1247	1720/0.0939	291/0.2005
96	666/0.0382	2096/0.2438	2705/0.4778	8632/0.6452	1596/3.3528
97	1857/0.0805	5604/0.6293	5305/1.2836	22644/2.5355	4495/13.5779
98	984/0.0481	592/0.0724	332/0.0454	fail(9)	279/0.3894
99	fail(7)	422200/50.5455(4)	170472/36.1353	50096/5.0071	fail(13)
100	fail(7)	415376/47.0313	227362/43.0603	fail(20)	fail(20)

Table 7.4: Number of function evaluations required by P-LBFGSB and the other methods to reach the global minimum. Part III.

Problem	AESLS	EO	COA	CMA-ES
number	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$
1	1188/0.2597	1228/0.0911	1422/0.1405	5037/0.2567
2	27668/2.9180	1968/0.1484	12220/0.6202	fail(13)
3	1102/0.0862	4300/0.4979	8986/0.4670	15923/4.6999
4	5063/0.2086	1410/0.0475	4720/0.1925	18350/2.4207
5	16764/0.9173	188597/17.8046	46454/3.1169	7360/1.6778
6	427713/37.1732(4)	21037/4.2304(2)	21418/0.8759	51704/3.1174
7	26733/1.4802	fail(20)	30117/1.7171	2700/0.7549
8	fail(20)	fail(20)	106814/6.0952	fail(8)
9	fail(20)	fail(20)	458842/26.1955(4)	fail(8)
10	fail(20)	fail(20)	fail(12)	fail(12)
11	fail(20)	fail(20)	fail(20)	fail(20)
12	fail(20)	fail(20)	fail(20)	fail(20)
13	304040/43.8516(4)	12255/1.3798	86950/11.2123	11924/2.0914
14	fail(5)	20843/2.0874	198805/25.8456	31356/8.0637
15	fail(12)	24123/2.5333	312475/38.9567(1)	30225/7.5262
16	fail(20)	360804/25.3255(2)	361900/47.4209	46312/12.0480
17	424/0.0307	2487/0.2635	30475/1.1908	9536/1.7367
18	3816/0.4067	3877/0.4201	77890/3.0394	8811/1.9070
19	333116/83.3204	4432/0.4444	169765/6.7281	27742/3.9948
20	437387/87.0294(3)	5245/0.5929	251260/10.0687	44954/7.2015
21	fail(9)	5542/0.5397	330025/13.4297	61185/11.6997
22	fail(17)	5510/0.6366	401305/16.8382	77840/24.7163
23	15764/1.9403	7695/0.5040	27082/1.8662	fail(9)
24	317967/48.3792	4360/0.4119	122345/8.4421	5040/1.2911
25	464949/77.8232(2)	2145/0.1981	126065/9.0915	15184/4.0120
26	fail(6)	2455/0.2848	167945/12.3440	24360/7.1496
27	fail(20)	2605/0.2596	244351/18.3591(1)	32550/6.9250
28	fail(20)	2845/0.3497	239414/18.4144(4)	42688/7.1677
29	9495/1.5515	1954/0.2676	18070/1.1773	1602/0.4125
30	228991/40.2568(2)	7850/0.9805	55045/3.6133	4818/1.2667
31	135520/18.5295	15122/2.2080	119500/7.9855	14248/3.8863
32	486757/76.1811(3)	22400/2.3058	176785/11.9776	24444/6.8215
33	fail(6)	29120/4.3084	233980/16.1046	31830/5.3533
34	fail(12)	37965/3.8966	289540/20.2766	41504/7.3690
35	167/0.0114	980/0.0292	5476/0.2031	378/0.0504
36	250092/21.5296	4520/0.1408	61600/2.5162	88681/4.2804
37	fail(13)	5705/0.1783	85250/3.3549	fail(6)
38	fail(5)	7590/0.2812	fail(10)	fail(17)
39	fail(9)	162790/5.9122(4)	fail(20)	fail(20)
40	fail(19)	fail(20)	fail(20)	fail(20)
41	fail(19)	fail(20)	fail(20)	fail(20)
42	1052/0.0964	862/0.0954	2935/0.3607	570/0.1324
43	137047/47.0626(1)	fail(9)	25525/3.0734	97111/22.5431
44	221113/42.0041	fail(16)	59530/7.7435	fail(16)
45	fail(6)	fail(20)	82990/10.8918	fail(20)
46	4719/1.1473	5823/0.9283	fail(5)	1320/0.4541
47	336951/93.9013(3)	400560/42.3869(4)	218963/41.8466(4)	57432/21.5914(2)
48	fail(9)	fail(7)	fail(7)	fail(17)
49	fail(19)	fail(20)	265123/51.0071(4)	fail(20)
50	635/0.0404	1057/0.0370	5248/0.2278	282/0.0414

Table 7.4: Number of function evaluations required by P-LBFGSB and the other methods to reach the global minimum. Part IV.

Problem	AESLS	EO	COA	CMA-ES
number	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$	$feval/CPU(s)$
51	fail(20)	fail(20)	fail(12)	fail(13)
52	33511/3.1316	920/0.0348	14656/0.6568	22414/2.9638
53	fail(7)	128328/4.7584(2)	fail(9)	201420/18.2177(3)
54	fail(11)	fail(17)	fail(20)	fail(17)
55	9669/3.0752	2730/0.3259	46108/5.6346	51404/8.2004
56	fail(9)	303035/43.1558(4)	103852/14.8987(2)	fail(8)
57	fail(10)	fail(18)	318567/36.4441(4)	fail(11)
58	48680/4.3381	fail(20)	fail(20)	27600/6.2213
59	467463/41.4330(3)	fail(6)	281465/19.0103	498300/90.1053(4)
60	fail(12)	fail(11)	fail(20)	fail(8)
61	fail(20)	fail(11)	fail(20)	fail(13)
62	fail(20)	fail(20)	fail(20)	fail(20)
63	fail(20)	fail(20)	fail(20)	fail(8)
64	149261/42.0386(4)	2110/0.2490	42237/2.4866	156240/29.9365(3)
65	fail(17)	3270/0.3835	101705/5.7970	419265/80.3112(4)
66	fail(17)	5113/0.6139	212825/12.2736	fail(11)
67	fail(17)	5053/0.6253	308071/18.0668	fail(20)
68	fail(17)	11343/4.5432	412420/24.6229	fail(20)
69	fail(17)	fail(5)	486871/29.5983	fail(20)
70	1172/0.1018	2111/0.2346	16762/1.1745	1784/0.4488
71	60015/5.6749	5158/0.4171	314482/22.6776	22560/4.9689
72	107655/10.6568	5197/0.4372	465442/34.1212(4)	111773/27.2109
73	198530/19.7912	3815/0.3131	fail(12)	210980/26.1855
74	fail(20)	5607/0.7565	fail(20)	fail(10)
75	fail(20)	5453/0.7089	fail(20)	fail(20)
76	867/0.0905	5115/0.8070	24408/3.0958	3096/0.7583
77	6520/1.3023	11330/3.9301	93117/13.3153	10296/3.0052
78	77664/45.1479	18572/14.1780	258768/87.5680	40560/20.9688
79	326997/393.0954	26070/35.5304	fail(12)	89320/84.5556
80	fail(7)	5380/69.6549	fail(17)	fail(9)
81	fail(13)	fail(5)	fail(20)	89848/86.9093
82	102244/12.2848	1753/0.2079	42550/2.2356	2808/0.6940
83	212055/27.3097(2)	3442/0.4205	107440/5.6991	8492/1.9061
84	fail(6)	4052/0.4592	229720/12.4291	24492/3.7145
85	fail(17)	4452/0.4845	327355/18.0816	38388/10.8115
86	fail(14)	4887/0.6036	412285/23.1954	51150/10.1175
87	fail(20)	5194/0.6911	491995/28.2281(2)	64032/15.3524
88	109230/9.0916	300450/10.0423(3)	fail(12)	400475/20.0451(4)
98	fail(5)	300309/10.5323(4)	fail(20)	fail(14)
90	fail(11)	fail(8)	fail(20)	fail(20)
91	fail(20)	fail(14)	fail(20)	fail(20)
92	283691/25.4181(4)	5625/0.1834	18820/0.8066	11857/0.7408
93	236329/20.7825	19070/0.6691	90880/3.5973	22041/2.5122
94	22785/0.8825	106310/3.4598	37324/1.4100	98557/5.0017
95	122/0.0316	650/0.0315	2272/0.1147	315/0.0241
96	669/0.0375	2228/0.1002	15904/0.7973	5274/0.3414
97	3687/0.1938	6710/0.2953	54388/2.6292	33174/2.2414
98	1036/0.0685	325/0.0173	fail(10)	700/0.1028
99	fail(7)	11190/0.5444	85696/4.9491	86924/5.4201
100	fail(7)	fail(14)	fail(11)	fail(17)

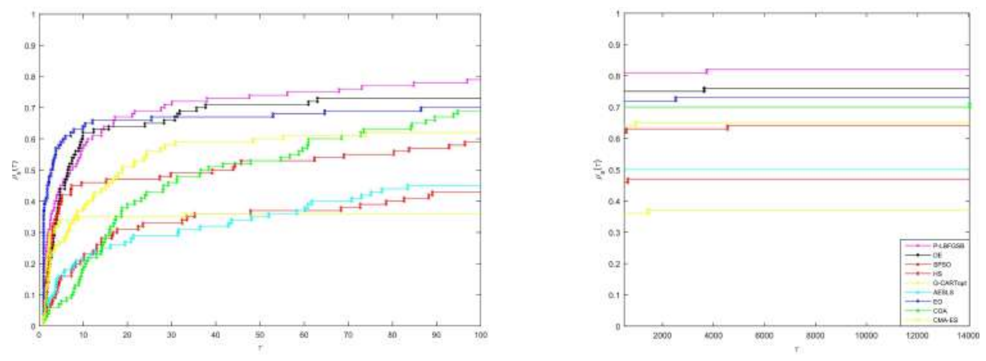


Fig. 7.1. Performance profiles plot based on the number of function evaluations.

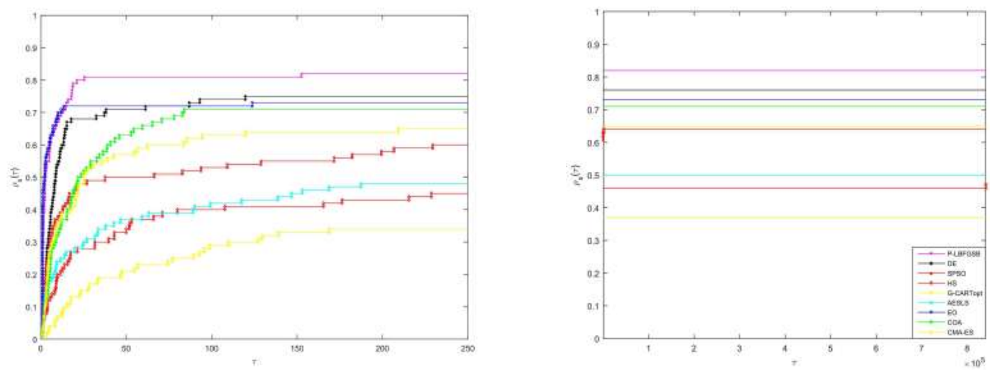


Fig. 7.2. Performance profiles plot based on CPU-time.

Figs. 7.1 and 7.2 show that the P-LBFGSB curve dominates the other curves, in particular, it is fastest for about 29% of the test problems and it solves about 82% of the test problems successfully, followed by DE and EO with respectively 76% and 74%. COA and CMA-ES have respectively the fourth and the fifth best performance by solving about 71% and 65% of the tests problems, whereas SPSO, AESLS, HS and G-CART score respectively about $64\%, 50\%, 47\%$ and 37%.

These outcomes indicate that, for this class of problems, the P-LBFGSB algorithm works rather well. One of the main reasons for this is the fact that the actual exploitation of the gradient leads towards lower regions with a moderate number of trial points. However, the suggested algorithm generates a somewhat large number of points in the case where the global minimizer attraction region is very narrow, in which case we would need more regularity on the gradient. On the whole, the numerical comparisons show that the suggested algorithm appears promising and competitive in practice.

7.1. Further applications to some engineering problems

In this section, we provide examples of common engineering problems that are modelled as global optimization problems and solved by the proposed algorithm. In these experiments, the P-LBFGSB algorithm was executed thirty independent runs for each problem to determine the best-obtained solution x^* and each run was terminated when the diversity parameter σ_k reaches a value less than $\sigma_{\min} = 10^{-2}$.

Lennard-Jones atomic cluster problem [5]

The Lennard-Jones potential is a mathematical model used to describe the interaction between two neutral atoms or molecules. It is commonly used in simulations of intermolecular forces, particularly for noble gases and non-polar molecules. Determining the most stable configuration of a cluster with N-atoms amounts to find the relative positions of the atoms that minimize the potential energy. The potential energy minimum of the cluster has the following form:

$$\min E = \sum_{i \neq j}^{N} 4\varepsilon \left(\left(\frac{\sigma}{\|r_{ij}\|} \right)^{12} - \left(\frac{\sigma}{\|r_{ij}\|} \right)^{6} \right),$$

where N designates the atomic cluster size, $r_{ij} = \|x_i - x_j\|$ is the distance between atoms i and j ($x_i \in \mathbb{R}^3$ is the position of the i-th atom), ε is the depth of the potential well and σ is the distance at which the potential energy is zero. The best obtained solutions for this problem with $N = 2, 3, \ldots, 7$ (using reduced units $\sigma = \varepsilon = 1$) are represented in Table 7.5, where the variables are bounded as

$$-2 \leq x_i \leq 2, \quad i = 1, 2, \ldots, n, \quad n = 3N.$$

Table 7.5: Obtained solutions for the Lennard-Jones cluster problem.

N	x_i	Energy/E
2	$1.2889, 1.5285, -1.5020, 1.6200, 0.5900, -1.5999$	-1
3	$1.3713, -0.8436, -1.6044, 1.5023, -0.8614, -0.6131,$ $0.6265, -0.5682, -0.9966$	-3
4	$1.1414, -0.8995, 0.1269, 1.5330, -1.4839, -0.5839,$ $1.9759, -0.6190, -0.3475, 1.9994, -1.3848, 0.2952$	-6
5	$0.3927, -0.4090, 0.4639, 0.0608, -0.1168, -0.4346,$ $0.4762, -1.0173, -0.3228, -0.4233, -0.8345, 0.0689,$ 4-0.4560,0.1104,0.38834	-9.1039
6	$-0.7887, -0.8172, -1.1727, -1.3771, -0.7708, -0.3555,$ $-0.8696, -1.6045, -0.5645, -0.3866, -0.7827, -0.2634,$ $-1.6595, -0.3375, -1.2053, -1.6415, -1.3345, -1.1356$	-12.3029
7	$-1.6595, -0.3375, -1.2053, -1.6415, -1.3345, -1.1356$ $-1.1380, 0.5423, -0.1939, -1.1930, -0.3560, 0.2301,$ $-0.3032, 0.0664, 0.0980$	-15.5331

Gas transmission compressor design problem [1]

The main goal of this problem is to estimate the gas transmission parameters l_c, λ, D for a gas pipe line transmission system to minimize the total cost of delivering 100 million cubic ft of gas per a day, where D is the length of the inside diameter of the gas pipe, l_c is the distance between the two compressors and λ is the compression ratio of the first and the second compressor. The mathematical model of this application is given by

$$\begin{aligned}\min T_c(l_c, \lambda, D) = 8.61 \times 10^5 l_c^{\frac{1}{2}} D^{-\frac{2}{3}} \lambda (\lambda^2 - 1)^{-\frac{1}{2}} + 3.69 \times 10^4 D \\ + 7.72 \times 10^8 l_c^{-1} \lambda^{0.219} - 765.43 \times 10^6 l_c^{-1},\end{aligned}$$

where the variables are bounded as

$$10 \le l_c \le 55, \quad 1.1 \le \lambda \le 2, \quad 10 \le D \le 40.$$

The best obtained solution for this problem is $(l_c^*, \lambda^*, D^*) = (53.446709, 1.190100, 24.718578)$ for which the objective value is $T_c^* = 2.964375\text{e+}06$.

Optimal capacity of gas production facilities problem [1]

The main goal of this problem is to obtain the pressure of the compressor p_c and optimal production of oxygen v with minimum cost. The total production cost of gas is given by

$$\begin{aligned}\min T_c(v, p_c) = 61.8 + 5.72v + 0.2623 &\left((40 - v) \ln\left(\frac{p_c}{200}\right)\right)^{-0.85} \\ &+ 0.087(40 - v) \ln\left(\frac{p_c}{200}\right) + 700.23 p_c^{-0.75},\end{aligned}$$

where the variables are bounded as

$$17.5 \le v \le 40, \quad 300 \le p_c \le 600.$$

The best obtained solution for this problem is $(v^*, p_c^*) = (17.5000, 600.0000)$ for which the objective value is $T_c^* = 1.698437\text{e+}02$.

8. Conclusion

In this paper we present a method for solving bound-constrained, non-convex global optimization problems where the objective function f is differentiable and without specific smoothness. The proposed method is a combination of two procedures. The main procedure is a limited memory quasi-Newton method. This procedure is used to try to spot at each iteration a point that improves upon the value of f. Since this may lead prematurely to a local minimum, such a point is avoided thanks to a stochastic perturbation procedure which diversifies the search and guides the algorithm to regions of the feasible domain not yet explored. In order to quicken the combined algorithm and make sure the perturbations lie within the feasible domain, we have developed a novel perturbation technique by truncating a multivariate double exponential distribution to deal with bound-constrained problems. The theoretical study and the simulation of the developed truncated distribution are also presented. Mathematical results concerning the convergence to the global minimum are established. Preliminary numerical experiments indicate that the algorithm is promising and competitive in practice.

References

[1] N. Andrei, *Nonlinear Optimization Applications Using the GAMS Technology*, Springer, 2013.

[2] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Wiley-Interscience, 2006.

[3] J. Brownlee, *Clever Algorithms, Nature-Inspired Programming Recipes*, LuLu.com, 2011.

[4] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu, A limited memory algorithm for bound constrained optimization, *SIAM J. Sci. Comput.*, **16**:5 (1995), 1190–1208.

[5] S. Das and P.N. Suganthan, *Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems*, Technical Report, Jadavpur University, Nanyang Technological University, 2010.

[6] E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles, *Math. Program.*, **91** (2002), 201–213.

[7] A. El Mouatasim, R. Ellaia, and E.S. de Cursi, Stochastic perturbation of reduced gradient & GRG methods for nonconvex programming problems, *Appl. Math. Comput.*, **226** (2014), 198–211.

[8] A. El Mouatasim and A. Ettahiri, Conditional gradient and bisection algorithms for non-convex optimization problem with random perturbation, *Appl. Math. E-Notes*, **22** (2022), 142–159.

[9] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, Equilibrium optimizer: A novel optimization algorithm, *Knowl. Based Syst.*, **191** (2020), 105190.

[10] C.A. Floudas, *Deterministic Global Optimization: Theory, Methods and Applications*, Springer, 2013.

[11] Z.W. Geem, J.H. Kim, and G.V. Loganathan, A new heuristic optimization algorithm: Harmony search, *Simulation*, **76**:2 (2001), 60–68.

[12] N. Hansen and A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation, in: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (1996), 312–317.

[13] D.C. Liu, and J. Nocedal, On the limited memory BFGS method for large scale optimization, *Math. Program.*, **45**:1 (1989), 503–528.

[14] B.J.T. Morgan, *Elements of Simulation*, Chapman-Hall, 1984.

[15] J. Nocedal, Updating quasi-Newton matrices with limited storage, *Math. Comput.*, **35** (1980), 773–782.

[16] M. Ouanes, H.A. Le Thi, T.P. Nguyen, and A. Zidna, New quadratic lower bound for multivariate functions in global optimization, *Math. Comput. Simulation*, **109** (2015), 197–211.

[17] M. Overton, *HANSO: Hybrid Algorithm for Non-Smooth Optimization 2.2*, 2019.

[18] J. Pierezan and L.S. Coelho, Coyote Optimization Algorithm: A new metaheuristic for global optimization problems, in: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, (2018), 2633–2640.

[19] V. Plevris and G. Solorzano, A collection of 30 multidimensional functions for global optimization benchmarking, *Data*, **7**:4 (2022), 46.

[20] M. Pogu and J.S. De Cursi, Global optimization by random perturbation of the gradient method with a fixed parameter, *J. Global Optim.*, **5**:2 (1994), 159–180.

[21] R.G. Regis, Convergence guarantees for generalized adaptive stochastic search methods for continuous global optimization, *European J. Oper. Res.*, **207**:3 (2010), 1187–1202.

[22] B.L. Robertson, C.J. Price, and M. Reale, A CARTopt method for bound-constrained global optimization, *ANZIAM J.*, **55**:2 (2013), 109–128.

[23] R.Y. Rubinstein and D.P. Kroese, *Simulation and the Monte Carlo Method*, Wiley, 1981.

[24] R. Storn and K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, *J. Global Optim.*, **11**:4 (1997), 341–359.

[25] M. Zambrano-Bigiarini, M. Clerc and R. Rojas, Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements, in: *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, (2013), 2337–2344.

[26] R. Ziadi and A. Bencherif-Madani, A covering method for continuous global optimization, *Int. J. Comput. Sci. Math.*, **13**:4 (2021), 369–390.

[27] R. Ziadi and A. Bencherif-Madani, A mixed algorithm for smooth global optimization, *J. Math. Model.*, **11**:2 (2023), 207–228.

[28] R. Ziadi, A. Bencherif-Madani, and R. Ellaia, Continuous global optimization through the generation of parametric curves, *Appl. Math. Comput.*, **282** (2016), 65–83.

[29] R. Ziadi, A. Bencherif-Madani, and R. Ellaia, A deterministic method for continuous global optimization using a dense curve, *Math. Comput. Simulation*, **178** (2020), 62–91.

[30] R. Ziadi, R. Ellaia, and A. Bencherif-Madani, Global optimization through a stochastic perturbation of the Polak-Ribière conjugate gradient method, *J. Comput. Appl. Math.*, **317** (2017), 672–684.

[31] https://github.com/bgranzow/L-BFGS-B
[32] https://cs.nyu.edu/ overton/software/hanso/
[33] http://www.particleswarm.info/Programs
[34] http://www.math.canterbury.ac.nz/ b.robertson/research.html
[35] https://yarpiz.com/231/ypea107-differential-evolution
[36] https://www.mathworks.com/matlabcentral/fileexchange/28850-harmony-search-algorithm
[37] https://github.com/afshinfaramarzi/Equilibrium-Optimizer
[38] https://github.com/jkpir/COA
[39] https://yarpiz.com/235/ypea108-cma-es