Feasible Cone Algorithm for solving Linearly and Nonlinearly Constrained Problems with Derivative Free Optimization and Strategies for Dealing with Multimodal Functions.

Ubaldo M. García-Palomares¹ · Ildemaro J. García-Urrea² · Pedro S. Rodríguez-Hernández¹

Received: date / Accepted: date

Abstract The first sections of this paper describes a non-monotone derivative free optimization (DFO) algorithm to find a local minimum of a linearly constrained problem. The algorithm generates at each iteration a finite number of rays of a cone of feasible directions by way of the well known pivot transformations frequently used in the solution of linear systems. Convergence to a first-order stationary point is shown under rather standard conditions. The paper also solves non-linearly constrained problems, which are aggregated to the objective function with an exact penalty function. Under suitable conditions it is proved that stationary points of the aggregated problem are also stationary points of the original non-linearly constrained problem. In all cases, the algorithm includes strategies (heuristics) to cover the possibility of dealing with functions that possess multiple local minima. Instead of delivering the global minimum, the algorithm may provide the user with a set of local minima to choose from. Finally, the algorithm satisfactorily solves a set of small to medium size problems which have exhibited serious difficulties for their solution by other optimization techniques. A slightly modified algorithm solved a multi-batch multi-product plant that was modeled as a MINLP.

Keywords Derivative Free · Linear and Nonlinear constraints · Exact Penalty functions · MINLP · Multiple Minima

1 Introduction

This paper is concerned with the implementation of sequential derivative-free algorithms that, under suitable conditions, locate stationary points of the hard constrained

¹Universidad de Vigo
Information Technology Group, Atlantic
36310, Vigo, Spain
Tel. +34 98681 3464/2174
Fax: +34 98681 2116
E-mail: ubaldo@gti.uvigo.es, pedro@det.uvigo.es

²Universidad Simón Bolívar
Dep. Procesos y Sistemas. Cómputo Científico
1080, Caracas, Venezuela
Tel.: +58 212 906 3304/3372
Fax: +58 212 906 3303/3370
E-mail: garciap@usb.ve, ijurrea@usb.ve
problem (HCP)

\[
\text{(HCP)} \quad \min_{x \in \mathcal{F}} f(x), \quad \mathcal{F} = \{ x \in \mathbb{R}^n : x \in \mathcal{E}, x \in \mathcal{H} \}.
\]

The set \( \mathcal{E} \) is composed by all constraints for which it is relatively easy to determine a finite number of rays that define a cone of feasible directions. This is the case for unconstrained, box constraints and linear constraints. Set \( \mathcal{E} \) is relevant in this paper because Section 2.1 describes an algorithm that solves an optimization model subjected to \( x \in \mathcal{E} \). These results are then used for solving (1) by penalization techniques.

Throughout the paper we enforce

**Assumption A0:** The constraint set \( \mathcal{F} \) is non empty and neither evaluation nor approximation of derivatives are reliable.

We resort to derivative-free optimization (DFO) techniques for solving HCP. Related papers have been appearing at such a pace that it is a formidable task to cite all these results are then used for solving (1) by penalization techniques.

Given \( x' \in \mathcal{F} \), a step size \( \tau_i \in \mathbb{R}_+ \), a reference value \( \varphi_i \geq f(x') \), a set \( Q_i \) of directions that contains a subset of \( q \) linearly independent feasible directions \( C_i = \{ d_1, \ldots, d_q \} \), and a forcing function \( \sigma(\cdot) \) characterized by:

\[
\begin{align*}
\sigma(\cdot) : R_+ &\rightarrow R_+ \\
\text{non decreasing: } (\tau_2 > \tau_1) &\Rightarrow (\sigma(\tau_2) \geq \sigma(\tau_1)) \\
\text{strictly positive: } (\tau > 0) &\Rightarrow (\sigma(\tau) > 0), \\
\sigma(0) : \lim_{\tau \downarrow 0} \sigma(\tau)/\tau &= 0,
\end{align*}
\]

\( x' \) will be declared a DQMP if

\[
d \in Q_i \Rightarrow \begin{cases} 
(x' + \tau_i d) \notin \mathcal{F}, \\
\text{OR} \quad f(x' + \tau_i d) > \varphi_i - \sigma(\tau_i)
\end{cases}
\]

The rationale for a DFO algorithm is based on the following observation: Let \( K \subseteq \mathbb{N} \) be an index sequence of points satisfying (3) and let \((x, f(x), 0, f(x), Q)\) be an accumulation point of \( \{ (x', f(x'), \tau_i, \varphi_i, Q_i) \}_{i \in K} \). Since by assumption \( Q_i \) contains a set of feasible directions \( C_i \), then \( (x' + \tau_i d) \in \mathcal{F} \) for \( d \in C_i \) and large enough \( i \); therefore if \( f(\cdot) \) is strictly differentiable at \( x \) we obtain for \( d_k \in C \) converging to \( d_k, k = 1, \ldots, q \) that

\[
\begin{align*}
\langle \nabla f(x), d_k \rangle &= \lim_{i \uparrow k} \frac{f(x' + \tau_i d_k) - f(x')}{\tau_i} \\
&\geq \lim_{i \uparrow k} \frac{f(x' + \tau_i d_k) - \varphi_i}{\tau_i} \\
&\geq - \lim_{i \uparrow k} \sigma(\tau_i)/\tau_i = 0;
\end{align*}
\]

which means that \( d_k \) is not a direction of descent. Let \( C(x) = \{ d_1, \ldots, d_q \} \) and let

\[
C(x)^+ = \{ y \in \mathbb{R}^q : y = \sum_{k=1}^q \alpha_k d_k, \alpha_k \geq 0, d_k \in C(x) \},
\]

and let \( \mathcal{E}(x) \) be the cone of feasible directions at \( x \). Conditions (4), \( C(x)^+ = \mathcal{E}(x) \), differentiability and convexity
are necessary and sufficient conditions for \( \bar{x} \) to be a global solution of HCP; however these conditions are not sufficient for non smooth functions, which may exhibit descent directions in \( \mathcal{C}(\bar{x}) \). We would like to satisfy the Clarke’s necessary condition on \( \mathcal{C}(\bar{x}) \), namely

\[
(d \in \mathcal{C}(\bar{x})) \Rightarrow \limsup_{i \in K} \frac{f(x^i + \tau_id) - f(x^i)}{\tau_i} \geq 0,
\]

but convergence to a Clarke point is ensured only if \( \{Q_i\}_{i \in \mathbb{N}} \) is asymptotically dense, which might demand a large number of function evaluations. So, we focus our attention on algorithms that solve HCP by sampling \( f(\cdot) \) on direction sets \( \{Q_i\}_{i \in \mathbb{N}} \) with a bounded number of elements.

We analyze in section 2 a basic model with bound and linear constraints, mainly to elucidate an efficient way of generating the set of search directions \( Q \) and to show the algorithm convergence. Section 2 also points out how to transform an HCP to a more manageable model by the use of penalization techniques. In section 3 we write a pseudo code of an elaborate description of the algorithm, which includes strategies not studied in previous works. The ideas exposed follow the line of research laid out in [16–19]. Section 4 introduces some heuristics to find the best minimizers of a multi-modal function. Section 5 applies the algorithm and its heuristics on a sample of problems that have exhibited serious obstacles for their solution by other well known optimization techniques. The results obtained are promising. A minor modification of the algorithm successfully solves a MINLP arisen in Chemical Engineering. Finally Section 6 specifies some continuation paths that might follow this line of research.

2 Algorithmic description

This section describes the algorithm for solving problem (1). We first describe an algorithm for solving the problem with the absence of the constraint set \( \mathcal{H} \) and then we show how to incorporate these techniques for solving problem (1) by exact penalization techniques when \( \mathcal{H} \) is made up by non-linear equalities and inequalities.

2.1 Easy constraints. \( \mathcal{H} = \mathbb{R}^n \).

This section is concerned with the easy constrained problem (ECP) with linear and bound constraints. Our main objective is to describe a technique to generate a set of search directions converging to \( \bar{Q} \) that contains a finite set of feasible directions \( \mathcal{C} \), with \( \mathcal{C}^0 = \mathcal{C} \) as suggested in the previous section.

Given a real full rank \( mn \times n \) matrix \( A \) we define the ECP as:

\[
\text{(ECP)} \quad \min_{x \in \mathcal{E}} f(x), \quad \mathcal{E} = \{ x \in \mathbb{R}^n : s_y \leq y = Ax \leq t_y, s_x \leq x \leq t_x \} \neq \emptyset, \quad (6)
\]

where \( s_x, t_x \) are, respectively, the lower and upper bounds on the \( x \) variables, and \( s_y, t_y \) are the respective bounds on the \( y \) variables. We denote the variables \( s_y = (s_{y1}, \ldots, s_{yn}) \)
as independent variables and $y \triangleq x_D = (x_{n+1}, \ldots, x_{n+m})$ as the dependent variables. The bound constraints can be represented by $s_j \leq x_j \leq t_j$, with $-\infty \leq s_j \leq t_j \leq +\infty$, $j = 1, \ldots, n + m$. The initial tableau representing the set $S$ is given in Figure 1. We can exchange the roles of $x_{n+j}$ and $x_k$ by applying the well known pivot transformation shown for convenience in Figure 2. We will refer to the variable $x_j$ and to the $j$-th constraint interchangeably. We denote the distance of $x_j$ to its bounds as

$$\delta_j = \min (t_j - x_j, x_j - s_j)$$  \hspace{1cm} (7)$$

and we say that $x_j$ is active if $\delta_j = 0$.

![Initial tableau: $y \triangleq x_D = Ax \triangleq Ax_I$](Fig. 1)

![One pivot transformation](Fig. 2)

To avoid digression from the main ideas we assume non-degeneracy. Reference [2] is appropriate to deal with degenerate systems.

**Assumption A1:** The system is non-degenerate; that is, no feasible point has more than $n$ active variables.

**Remark 1** From our Linear Algebra textbooks we know that after several pivot transformations we can come up with an index set $I \subset \{1, \ldots, n+m\}$ of $n$ independent variables with the lowest $\delta$ values, an index set $D \subset \{1, \ldots, n+m\}$ of $m$ dependent variables, and a tableau with a matrix $\bar{A}$ showing a feasible point, that is,

$$\bar{x} = (\bar{x}_I, \bar{x}_D), s_I \leq \bar{x}_I \leq t_I, s_D \leq \bar{x}_D = \bar{A}\bar{x}_I \leq t_D, \text{ with } \max_{j \in I} \bar{\delta}_j \leq \min_{k \in D} \delta_k.$$  \hspace{1cm} (8)
where ties are broken arbitrarily. We should point out that when \( s_j = t_j \) the \( j \)-th variable is active and will always be independent. We could also get rid of this variable and reduce the dimensionality of the problem. We recall that only recent extensions to MADS can handle equalities [6].

In the sequel \( e_k \in R_n \) denotes the \( k \)-th column of the \( nxn \) identity matrix.

**Lemma 1** Let \( \bar{x} = (\bar{x}_I, \bar{x}_D = \tilde{A}\bar{x}_I) \) be feasible and let \( j \in I \) be the index component of any constraint active at its lower bound, that is, \( \bar{x}_j = s_j \). Let \( x'_j = \bar{x}_j + \tau e_j \) for some \( j \in I \) and let \( x'_k = \bar{x}_k \) for \( k \in I, k \neq j \).

Under assumption A1, the point \( x' = (x'_I, x'_D = \tilde{A}x'_I) \) is feasible for sufficiently small \( \tau \); formally \( \exists \tau > 0 : (\forall \tau < \bar{\tau})(x' \in \mathcal{D}) \)

**Proof** With no loss of generality we assume that the \( j \)-th variable is located on the \( p \)-th column of \( \tilde{A} \), for some \( p \in \{1, \ldots, n\} \). We read from the tableau that \( x'_D - \bar{x}_D = \tilde{A}(x'_I - \bar{x}_I) = \tau \Lambda e_p \), the \( p \)-th column of \( \tilde{A} \).

Since \( x'_I = x_I \), \( k \in I, k \neq j \), the lemma holds if \( \bar{\tau} \) is the smallest value of \( \tau \) that satisfies both \( \bar{x}_j + \tau \leq t_j \) and \( s_D \leq \bar{x}_D + \tau \Lambda e_p \leq t_D \) \( \Box \)

**Remark 2** \( \tilde{A}e_p \) is a ray of a cone of feasible directions at \( \bar{x} \).

**Lemma 2** Let \( \bar{x} = (\bar{x}_I, \bar{x}_D = \tilde{A}\bar{x}_I) \) be feasible and let \( j \in I \) be the index component of any constraint active at its upper bound, that is, \( \bar{x}_j = t_j \). Let \( x'_j = \bar{x}_j - \tau e_j \) for some \( j \in I \) and let \( x'_k = \bar{x}_k \) for \( k \in I, k \neq j \).

Under assumption A1, the point \( x' = (x'_I, x'_D = \tilde{A}x'_I) \) is feasible for sufficiently small \( \tau \); formally \( \exists \tau > 0 : (\forall \tau < \bar{\tau})(x' \in \mathcal{D}) \)

**Proof** It is omitted.

**Remark 3** \( -\tilde{A}e_p \) is a ray of a cone of feasible directions at \( \bar{x} \).

Let \( j \) be the index component of a non active independent constraint, located at the \( p \)-th column of \( \tilde{A} \). Both \( x' = \bar{x} + \tau e_j \) and \( x' = \bar{x} - \tau e_j \) are feasible for small enough \( \tau \).

**Remark 4** \( \tilde{A}e_p \) and \( -\tilde{A}e_p \) are rays of a feasible cone at \( \bar{x} \).

From the previous arguments we conclude that the set

\[
Q = \{ q : q = \tilde{A}e_k, \text{ or } q = -\tilde{A}e_k, \ k = 1, \ldots, n \}
\]

contains the rays of a feasible cone at \( \bar{x} \) on the tableau exhibiting the feasible point \( \bar{x} = (\bar{x}_I, \bar{x}_D = \tilde{A}\bar{x}_I) \) with all active constraints as independent variables.

### 2.2 Basic algorithm

Figure 3 describes the basic iteration of the algorithm with parameters \((\beta, \mu, \gamma)\) chosen arbitrarily in the given intervals.

The repeated application of the basic iteration generates a sequence \( \{(x_i', f_i, \tau_i, \varphi_i, Q_i)\}_{i \in \mathbb{N}} \) with the following properties:
Lemma 3. For all $i \in \mathbb{N}$ we obtain: (a) $\varphi_{i+1} \leq \varphi_i$, and (b) $f(x) \trianglerighteq f(x') \leq \varphi_i$

Proof. [17, Proposition 4.1].

If (3) holds, $\varphi_{i+1} = \varphi_i$; otherwise

$$\varphi_{i+1} = (1 - \beta_i) f(x') + \beta_i \varphi_i \leq \varphi_i - (1 - \beta_i) \sigma(\tau), \quad (10)$$

and (a) holds. To prove (b) we use induction hypothesis. If (3) holds, $x^{i+1} = x'$ and $\varphi_{i+1} = \varphi_i$. Hence, $f(x^{i+1}) = f(x') \leq \varphi_i = \varphi_{i+1}$. If (3) does not hold $\varphi_{i+1} = (1 - \beta_i) f(x') + \beta_i \varphi_i \geq f(x') + \beta_i \sigma(\tau)$, and the proof is complete. \square

Lemma 4. If $f(\cdot)$ is bounded below on $\mathcal{E}$, then $\liminf_{r \in \mathbb{N}} \tau_r = 0$.

Proof. If (3) fails only a finite number of times, the assertion of the lemma is obvious. If (3) fails an infinite number of times, we assume that the assertion of the lemma is false, and show a contradiction.

If the lemma is false, there exists $0 < \tau$ such that $\tau < \tau_i$ for all large enough $i$, and $0 < \sigma \trianglerighteq \sigma(\tau) \leq \sigma(\tau_i)$ by definition (2) of $\sigma(\cdot)$.

Let $\beta = 0.7$ be the upper bound of $\beta_i, i \in \mathbb{N}$, and let $J = \{i_1, i_2, \ldots, i_{k-1}, i_k, \ldots\}$, with $i_k$ large enough. From (10) and the previous lemma we obtain

$$\varphi_{i_k + 1} \leq \varphi_{i_k} - (1 - \beta) \sigma$$
$$\leq \varphi_{i_{k-1}} - 2(1 - \beta) \sigma$$
$$\vdots$$
$$\leq \varphi_{i_1} - k(1 - \beta) \sigma,$$

which shows that $\{\varphi\}_{i \in \mathbb{N}}$ is unbounded below, and a fortiori $f(\cdot)$ is unbounded below: a contradiction. \square

We are almost ready to prove convergence to a point satisfying the necessary condition. We will show that

$$(d \in C(x)) \implies (\nabla f^T d \geq 0), \quad (11)$$
where $C(\bar{x})$ is the subset of $Q(\bar{x})$ that define the rays of the feasible cone at $\bar{x}$. From (4) we immediately conclude that
\[
(d \in C(\bar{x})) \Rightarrow (\nabla f^T d \geq 0),
\]
(12)

We need the following assumptions:

A0. $(\beta \neq \emptyset)$ and derivative information is not available,
A1. the system is non-degenerate,
A2. $\{x^i\}, i \in \mathbb{N}$ remains in a compact set,
A3. $f(\cdot)$ is bounded below on $\beta$ and it is strictly differentiable at the limit points of $\{x^i\}_{i \in \mathbb{N}}$, and
A4. $\sigma(\cdot)$ satisfies (2).

We remark that a practitioner needs not verify a priori the theoretical conditions A0 - A3. We also point out that A2 is ensured if we assume compactness of $\mathcal{F}$, compactness of level sets of $f(\cdot)$, $f(\cdot)$ is concave, or any other convenient theoretical assumption.

**Theorem 1** Let $\mathbb{K} \subseteq \mathbb{N}$, and let $\{(x^i, f_i, \tau_i, \phi_i, Q_i)\}_{i \in \mathbb{K}}$ be a sequence of DQMPs, generated by the basic algorithm described in Figure 3, converging to $(\bar{x}, \bar{f}, \bar{\tau}, \bar{\phi}, \bar{Q})$, with $\bar{\tau} = 0$. Under assumptions (A0 - A4), (12) holds.

**Proof** Let $\delta_j = \min(\bar{x}_j - s_j, t_j - \bar{x}_j) = 0$ for some $j \in \{1, 2, \ldots, m+n\}$. By construction (8 and 9) and A1, we assert that the variable $x_j$ will be an independent variable for all large enough $i \in \mathbb{K}$. Consequently, with no loss of generality, we assert that the tableau matrix $A^i$ will not change for all large enough $i \in \mathbb{K}$. Formally, as the number of matrices is finite, $(\exists \mathbb{K}' \subseteq \mathbb{K})$ such that $A^i = A^j$ for $i \in \mathbb{K}'$, and $j \in \mathbb{K}'$ sufficiently large. Since the search directions are the columns of $A^i$ and by Remarks 2, 3 and 4 they contain the rays of the feasible cone, we deduce (11). Finally, as $\mu_i \to 0$, (4) implies (12).

**Corollary 1** If (A0-A4) hold and $f(\cdot)$ is pseudo convex, the accumulation point $\bar{x}$ generated solves ECP.

**Proof** It is a well known fact. □

We should point out that Theorem 1 could also be proved along the lines suggested in [23, Section 3.8], but our proof is straightforward and easier to follow.

We would like to end this Subsection with a word of advice on the choice of suitable values for the parameters $\mu_i$, $\beta_i$, and $\gamma$ required in the basic iteration described in Figure 3. Previous numerical tests [17] seem to indicate that the number of function evaluations does not change significantly when the parameters are chosen in the suggested intervals: $\mu_i \in [0.4 0.8]$, $\gamma_i \in [1 1.5]$, $\beta_i \in [0 0.7]$; however $\beta_i \simeq 1$ is more appropriate for global optimization, because it enhances hill climbing, a feature that helps the algorithm escape from a local minimum. Table 1 shows a typical behavior of the algorithm for a constant value for $\beta$ at all iterations. For the illustration we are using the well-known Rosenbrock function and the Shekel function defined by (22) and Table 3. As expected, the bigger the $\beta$ value, the bigger evals, the number of function evaluations, but at the same time a better success is reported in regards to the number of times the known global minimum is found.
Table 1 Influence of $\beta$ on the basic algorithm. Average results in 1000 runs

<table>
<thead>
<tr>
<th>VALUES OF PARAMETER $\beta$</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenbrock (10 variables)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>success(%)</td>
<td>90</td>
<td>99.1</td>
<td>100</td>
<td>99.9</td>
<td>100</td>
</tr>
<tr>
<td>evals</td>
<td>3557</td>
<td>3905</td>
<td>4720</td>
<td>7038</td>
<td>11643</td>
</tr>
<tr>
<td>Shekel (5 variables)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>success(%)</td>
<td>6.4</td>
<td>7.3</td>
<td>8.3</td>
<td>9.4</td>
<td>17.2</td>
</tr>
<tr>
<td>evals</td>
<td>1305</td>
<td>1669</td>
<td>2295</td>
<td>4196</td>
<td>7895</td>
</tr>
</tbody>
</table>

2.3 Hard constraints

To deal with the set $\mathcal{H}$ we aggregate to the objective function a non negative term $\rho c(x)$, where $\rho > 0$ and $c(x) : \mathcal{E} \rightarrow \mathbb{R}_+$ is a continuous function that acts as an estimate of the proximity of a point $x \notin \mathcal{H}$ to the set $\mathcal{H}$. This paper defines the augmented function

$$F(x) \triangleq f(x) + \rho c(x), \quad (13)$$

and uses the algorithm explained in Section 2.1 to the problem

$$\min_{x \in \mathcal{E}} \{ f(x) + \rho c(x) \}. \quad (14)$$

The algorithm generates a sequence of points that belong to the set $\mathcal{E}$ and under suitable conditions the accumulation points of its DQMPs are stationary for problem

$$\min f(x), \quad x \in (\mathcal{E} \cap \mathcal{H}). \quad (15)$$

We remark that penalization techniques that solve (14) when $\rho \rightarrow \infty$ may lead to an ill conditioned problem. Augmented Lagrangean and exact penalty functions ensure the existence of a finite $\rho$. Augmented Lagrangeans assume that the second order sufficiency conditions hold, and its implementation involves mechanisms to update the unknown Lagrange multipliers. On the other hand, exact penalty functions are in general non differentiable which implies special techniques for their solution (These remarks are well known and appear in most nonlinear programming books). In principle, we want $P1$ and $P2$ to hold:

$P1$. $c(x) \geq 0$ and $(c(x) = 0) \iff (x \in (\mathcal{E} \cap \mathcal{H}))$

$P2$. If $x \in \mathcal{E}$ and $x \notin \mathcal{H}$, there exists a point $x' \in \mathcal{E}$ with a lower functional value. In other words

$$x \in \{ x \in \mathcal{E} : c(x) > 0 \} \Rightarrow \exists x' \in \mathcal{E} \text{ satisfying:} \quad F(x') < F(x) \quad (16)$$

$P1$ ensures that $(x \in (\mathcal{E} \cap \mathcal{H})) \iff (F(x) = f(x))$, and $P2$ prevents any $x : \{ x \in \mathcal{E} \mid x \notin \mathcal{H} \}$ from being a minimum of $F(\cdot)$. This condition is reminiscent of the $H_2$ condition imposed in [8] to ensure zero duality gap in Lagrangean functions.
In general, P2 has no practical way of being verified; however, if we restrict the set \( \mathcal{H} \) to non linear equalities and inequalities constraints we can adapt the elegant results given in [11,14] to our ideas. In the sequel we will be concerned with the set

\[ \mathcal{H} = \{ x \in \mathbb{R}^n : g_j(x) \leq 0, j = 1, \ldots, r \}, \tag{17} \]

where \( g_j(x), j = 1, \ldots, r \) are strictly differentiable functions with bounded derivatives. We keep in mind that the equality \( h(x) = 0 \) is included in \( \mathcal{H} \) with the inequalities \( h(x) \leq 0, -h(x) \leq 0 \). As a proximity function \( c(\cdot) : \mathcal{E} \to \mathbb{R} \) this paper suggests

\[ c(x) = \max \{ 0, \sum_{j \in J(x)} g_j(x) \}, \quad J(x) = \{ k : g_k(x) > 0 \} \tag{18} \]

Let \( x \in \mathcal{E} \) and let \( \mathcal{C}(x) \) be the cone of feasible directions at \( x \) on the set \( \mathcal{E} \). We say that \( x \in \mathcal{E} \) satisfies the Constraint Qualification on Linear Systems (CQLS) when

\[ (c(x) > 0) \Rightarrow (\exists d \in \mathcal{C}(x))(\forall j : g_j(x) > 0) \nabla g_j(x)^T d < 0 \tag{19} \]

**Lemma 5** Let \( x \in \mathcal{E} \) and \( c(x) > 0 \) for \( c(\cdot) \) defined by (18). For big enough \( \rho > 0 \), the direction specified by (19) is a direction of descent for the augmented function \( F(x) = f(x) + \rho c(x) \). More precisely, \( F(x + \tau d) < F(x) \) for small enough \( \tau > 0 \).

**Proof** As \( c(x) > 0 \), by definition \( c(x) = \sum_{j \in J(x)} g_j(x) \), with \( J(x) \) defined in (18). For the direction \( d \) specified by (19) we have for large enough \( \rho > 0 \) that:

\[ \nabla F(x)^T d = \nabla f(x)^T d + \rho \sum_{j \in J(x)} \nabla g_j(x)^T d < 0; \]

and \( d \) is a direction of descent for \( F(\cdot) \) at \( x \). The validity of the lemma follows by standard arguments. \( \square \)

**Corollary 2** Let \( K \subseteq \mathbb{N} \), and let \( \{(x^i, f_i, \tau_i, \phi_i, Q_i)\}_{i \in K} \to (\bar{x}, \bar{f}, \bar{\tau}, \bar{\phi}, \bar{Q}) \), with \( \tau = 0 \) be a sequence of DQMPs, generated by the basic algorithm described in Figure 3 for solving (14). If assumptions (A0 - A4) hold and CQLS (19) also holds then \( \bar{x} \) is a feasible point for problem (15), that is, \( \bar{x} \in (\mathcal{E} \cap \mathcal{H}) \).

**Proof** We need to prove that \( c(\bar{x}) = 0 \). We proceed by contradiction. If we assume that \( c(\bar{x}) > 0 \), then there exists a large enough \( i \in K \) such that \( J(x_i) = J(\bar{x}) \) for all \( i \geq \bar{i} \), and \( J \) defined by (18). By assumption \( c(x) = \sum_{j \in J(x)} g_j(x) \) is strictly differentiable. A fortiori, \( F(x) = f(x) + \rho c(x) \) is strictly differentiable. Furthermore, all assumptions of theorem 1 hold for all \( i \in K, i \geq \bar{i} \); therefore \( \nabla F(\bar{x})^T d \geq 0 \) for all \( d \in \mathcal{C}(\bar{x}) \); however, lemma 5 ensures that \( \nabla F(\bar{x})^T d < 0 \) for some \( d \in \mathcal{C}(\bar{x}) \) and a contradiction arises. We conclude that \( c(\bar{x}) = 0 \). \( \square \)

Next Theorem claims that any accumulation point of a sequence of DQMPs converging to \( \bar{\tau} = 0 \) when the algorithm is applied for solving problem (14), is also stationary for problem (15).
Theorem 2 If A0 - A2, A4 hold, If the proximity function c(·) : E → R+ is given by (18), If p > 0 is large enough, If the (CQLS) given by (19) holds and If g_j(x), j = 1, ..., r are strictly differentiable functions with bounded derivatives, then any accumulation point \bar{x} of the DQMPs with \bar{\tau} = 0 generated by the algorithm for solving (14) is a stationary point for (15).

Proof Let us recall that our algorithm will generate an index sequence K of DQMPs converging to some \bar{x}, \bar{\tau} with \bar{\tau} = 0. Besides, for large i ∈ K we have for a feasible direction d_i converging to \bar{d} that F(x_i + \tau_id_i) - F(x_i) ≥ -σ(\tau_i). Since (x_i + \tau_id_i) ∈ \mathcal{F} we obtain that F(x_i + \tau_id_i) = f(x_i + \tau_id_i) and

\[ f(x_i + \tau_id_i) - f(x_i) ≥ F(x_i + \tau_id_i) - F(x_i) ≥ -σ(\tau_i) \] (20)

Taking limits on both sides we obtain \nabla f(\bar{x})^T \bar{d} ≥ 0 and by construction of the algorithm (11) holds. □

3 Algorithm

Figure 4 is a pseudo code that solves min\{x ∈ E \} F(x), where F(·) is the augmented objective function defined by (13), or by any other penalization scheme. Line numbers and comments are mixed with the pseudo code to facilitate its comprehension. We remark that this version incorporates a strategy proposed in [3, Section 5] to search for the global optimum of the unconstrained problem min\{x ∈ \mathbb{R}^n \} f(x). Given a local minimum x, the authors in [3] propose to find another local minimum y from a randomly chosen starting point. If f(y) < f(x), y is a better local minimum and replaces x, i.e. x ← y, and the process is repeated. If a better minimum is not found after a certain number of iterations, the last minimum obtained is taken as the global minimum. In our approach we do not locate a local minimum to the constrained problem (14), because theoretically this is an infinite process. We rather keep xb, the best point thus far, and its value Fb = F(xb). If there is no improvement after a finite number of iterations, a random z ∈ E is generated to restart the algorithm at xb as it will be explained in numeral 8 below, among other features of the code in Figure 4. It is intermingled with heuristics that try to better its chances of getting a global minimum.

1. The algorithm requires the input shown: a starting point x_0 and its function value F_0, an initial stepsize \tau_0 bigger than the required accuracy \epsilon, the incremental functional value \Delta F_0, generally taken as 20% of the function value, that is: \Delta F_0 = 0.2(1 + |F_0|), and MaxCount, the maximum number of times that a random point z ∈ \mathcal{F} can be generated. The larger this number, the bigger the possibilities of reaching the global minimum.

2. The algorithm stops when \tau_i is small enough (line 1). It is a common practice to include a safeguard to prevent a budget depletion. Typically the user should give an upper bound to the number of function evaluations.

3. Due to the non monotone feature of the algorithm, the best function value found is always saved (lines 5, 8, and 23).
4. The set $Q_i$ (line 13) must include the feasible cone generators for the active constraints. The set used in our experiments is constructed by (8) and (9). We also include random directions and the simplex gradient of the original objective function $f()$, because of its trivial computation. If $f \in I$ we obtain:
\[ \nabla f_j \approx \frac{f(x_I + \tau_i x_j) - f(x_I - \tau_i x_j)}{2\tau_i} \]

5. We recommend that directions in \( Q \) be taken in a random fashion one after the other (line 15). We also drew \( \beta, \mu, \gamma \) from a uniform distribution in the given intervals (lines 21, 28, 37) and recall the reader that a large \( \beta \) value is preferred for global optimization but it demands a larger number of function evaluations.

6. On line 27 we check if \( x \) is a DQMP, which occurs when the IF sentence in line 17 does not hold for all \( d \in Q_i \), that is, \( \text{move} = 0 \).

7. When the IF sentence in line 17 holds for some \( d_{ij} \in Q_i \) we could pick \( y = x + \tau_i d_{ij} \); nonetheless, line 19 allows the choice of any \( y \in \mathcal{E} \) such that \( F_y \Delta = F(y) \leq \varphi_i - \sigma(\tau_i) \).

8. Lines 29–33 and lines 2–12 incorporates a variant of the Addis-Locatelli technique [3, Section 5]. At the initial iteration we have two prospective starting points: \( x^0 \) and a random \( z \in \mathcal{E} \) (line 3). The algorithm saves the better one and applies the basic algorithm 3 to the other one, denoted as \( x' \) (lines 6 and 9) until a DQMP is identified, instead of a local minimum as proposed in [3]. If the DQMP is not better than \( Fb \), the best \( F() \) value found thus far, i.e. \( (F \geq Fb) \), and \( \tau_i \) is small enough, we go back and restart the algorithm at \( xb \) (lines 29–33). To avoid excessive computation we set \( \text{MaxCount} \) as an upper bound to the number of restarting procedures allowed (line 2).

9. Note that on line 30 we have set \( \varphi_{i+1} = Fb \); however convergence is preserved for any \( \varphi_{i+1} < F_i \). So, if you are willing to explore the space more fully, you can set \( \varphi_{i+1} = Fb + (F_i - Fb)/2 \).

10. The algorithm allows an increase in \( \tau \) (lines 37). This happens when we have moved in at least one half of the directions that belong to \( D_i \) (line 36).

4 Multiple minima

Previous papers in non monotonous DFO algorithms have been concerned with multiprocessing [16, 17, 19]. The parallel implementation with linear constraints employs the same theoretical background but its implementation deserves some analysis.

Attention has also been given to global optimization and encouraging results have been reported [18]. This paper assumes that the objective function possess an unknown number of local minima and to our knowledge it is the first paper that describes a non monotonous deterministic DFO algorithm that provides the practitioner with a list of prospective candidates as solutions of a function subjected to linear and nonlinear equalities and inequalities. Previous works that detect multiple minima are stochastically biased and mainly based on multistart technique.

This Section suggests strategies (heuristics) to address two issues regarding multiple minima:

- **Multiple global minima**
- **Multiple minima with close \( F(\cdot) \) values**
In real applications the two cases just mentioned coincide numerically due mainly to roundoff errors and imprecision of the data. In addition, some global solutions cannot be easily implemented in real problems, or they can not even be captured by the idealized model that tries to mimic the behavior of the system at hand. The recent paper [26] will help the reader understand the intrinsic difficulties that must face an algorithm that tries to find multiple minima. It also gives a good account of recent research in the area. We should stress that our work differs notably from [26]. On the first place they solve a problem constrained by bounded variables only; while ours extend the results of the previous sections of this paper and solve a more general problem subjected to linear and non linear (in)equalities. Secondly their algorithm is a stochastic process and they prove that all local minima are found almost surely, provided the number of points randomly sampled grows to infinity and that every local optimization run certainly generates a local minimum. Our algorithm will provide the practitioner with a list of local candidates whose function values slightly differ from the best minimizer found. Our numerical results show that the best minimizer reported by our algorithm commonly coincides with the global solution. We add some heuristics to the algorithm described in the previous sections, which are evaluated by numerical examples. Figure 5 describes a pseudo code. As in section 3 we show the line numbers and comments to facilitate its reading. We advise the reader to look at the code presented in Figure 5 before reading its description.

4.1 Description of the code

The heart of the algorithm to detect multiple minima is described in Figure 4. We recall that \( F(\cdot) : \mathcal{E} \rightarrow \mathbb{R} \) is the augmented function \( F(x) = f(x) + \rho c(x) \). We require an initial population \( P \) and some extra parameters: \( \text{Lips} \), \( \text{eps} \) and \( \delta \).

- The algorithm works with a finite population \( P \) and applies the code given in Fig. 4 to each \( x \in P \) (lines 7a, 7b), except when \( F(x) \) is too high with respect to \( \min_{x \in P} F(x) \), or when \( x \) is too close to another point in \( P \). In both cases \( x \) is deleted from \( P \) (line 4).
- The parameter \( \text{Lips} \) is an estimate of the Lipschitz constant of \( F(\cdot) \). If the value of \( \text{Lips} \) is accurate we can safely delete \( x \) from \( P \) in line 4. If \( \text{Lips} \) is unknown we may use a large value so that the first operand of the IF instruction in line 3 is never TRUE.
- \( x \) is also deleted from \( P \) when it is close to any other point in \( P \), in order to avoid possible repetition of the same sequence of points that would be generated by the code in figure 4.
- \( |P| \), the size of \( P \), can be preserved if we insert a random point in \( P \) between lines 4 and 5.
- Note that the precision of the algorithm goes in crescendo (line 10). This allows the deletion of additional points along the way. The value of \( \gamma \) in line 10 is taken randomly in \([5 10]\).

We should remark that the algorithm does not always return all minimizers. This is in general out of the question for functions with an unknown number of local
Pseudo Code

Input: \( P = \{y_1, \ldots, y_p\} \)
\( p \) random points in \( \mathcal{E} \)
\( \text{Lips, } \epsilon > 0, \delta > 0 \)
parameters

0. WHILe (\( \epsilon > \epsilon^* \))
1. Compute \( F_{\text{min}} = \min_{x \in P} F(x) \)
2. FOR ALL \( x \in P \)
3. IF \( F(x) > F_{\text{min}} + \text{Lips} ||x - x_{\text{min}}|| \)
OR \( \exists y \in P: ||y - x|| \leq \delta \)
4. \( P = P - \{x\} \)
delete \( x \) from \( P \)
5. CONTINUE
process next \( x \in P \)
6. END IF
7a. Starting at \( x^0 = x \), solve min \( F(x), x \in \mathcal{E} \)
7b. using the algorithm coded in Fig. 4. Terminate at \( x < \epsilon \).
8. \( P = P - \{x^0\} + \{x\} \)
Replace \( x^0 \) by \( x \) in \( P \)
9. END FOR
10. \( \epsilon = \epsilon / \gamma \)
random \( \gamma \in [5, 10] \)
11. END WHILE

Return \( P, F(P) \)

Fig. 5 Algorithm that provides Multiple Minima

minima. Besides, the algorithm only yields the best minimizers. It prunes \( P \) in line 4
to discard unnecessary minimizers with large function values. The practitioner should
limit the number of local minima of his/her interest by giving the size \( |P_0| \) of the initial
population.

5 Numerical experiments

These preliminaries tests are carried out with the purpose of evaluating the values
of the parameters \( \text{Lips, } |P|, \) and \( \delta \). We emphasize that the aim of the algorithm
is not to find all local minimizers, but only those whose function values are close to
the global minimum. Besides, the algorithm ignores the number of local minimizers,
even if this datum is known. We carried out numerical tests on a sample of multi-
modal test functions suggested for finding their global minimizers. These problems
were chosen for two reasons: On the one hand, they were not included in the vast
numerical test presented in the previous work to find a global minimizer of functions
subjected to bounds on the variables, where our parental algorithm was highly com-
petitive [17]. On the other hand, the test problems were chosen because authors have
admittedly considered they are particularly hard and present a real challenge to other
optimization techniques. In particular, we have included in our test a real non-linearly
constrained problem with mixed variables, which was solved by adapting the algo-
rithm to deal with integer variables. We admit however these results are preliminary
and future research should be pursued.

The code was written in MatLab R2010a and ran on Windows on a PC equipped
with an Intel 870 @ 2.93 GHz. As most problems have a large number of local minima
we often chose an initial population size \( |P_0| = 20 \). Ideally, this figure should be close
to the number of function minimizers the practitioner hopes or wishes to find. We
estimate the Lipschitz constant \( \text{Lips} = (f_{\text{max}} - f_{\text{min}})/||x_{\text{max}} - x_{\text{min}}|| \), where \( f_{\text{max}} \) and \( f_{\text{min}} \) are respectively the maximum and the minimum value in the initial population
\( P_0 \). We chose the initial \( \tau = 0.2 \) for problems with a large number of minima or
restricted to small boxes, otherwise $\tau = 1$. We set the accuracy $\epsilon_{ps} = 0.001$ and $\epsilon = 0.001\epsilon_{ps}$. Finally to observe the algorithm performance until convergence, we did not impose budgetary limitations, that is, the number of function evaluations was not a criterion to stop the algorithm.

We tested the algorithm with the toy problem reported in [20]. The authors claim that this problem shares features with a real hydrological model. We also tried on the Shekel problem, which has been recognized as a difficult problem [35]; the Ackley problem with 30 variables, which has been very elusive in the literature [25]. Finally we tried the Thomson problem with 96, 150, 300 and 600 variables from the Dolan et al benchmark collection [12]. They claim that some modern algorithms using first and second order derivative information have failed on these problems. Moreover, it is being argued that DFO techniques are not suitable for solving problems of these sizes [9, Section 1.3], although some progress has been recently reported [17]. We also show for the toy, Shekel and Ackley problems, the behavior of the algorithm when the practitioner is only concerned with finding a local minimum. Finally we included in the test a real problem of minimizing the cost involved in a batch production plant that must manufacture different quantities of chemical products. This model is formulated with mixed variables; nonetheless, we found excellent results with a slight modification of the algorithm.

Because of the randomness involved we run the algorithm 100 times for each case and report the number of minima and function evaluations per run showing the best, the average and the worst behavior in each case. We also add a column, denoted as $\text{succ}$, that shows the number of times the global minimum was obtained by the algorithm in 100 runs, when this datum is known.

Our first test problem was recently introduced by Gramacy et al [20]. They claimed that this toy problem with two variables, shares common features with a real-data hydrology problem. It has 3 minima and the global minimum is at $\bar{x} = [0.1954 \ 0.4044]$ with $f(\bar{x}) = 0.5998$. We ran the algorithm starting at $x^0 = (0 \ 0)$ with different values for the parameters $|P|$, $\tau$ and MaxCount. For the toy problem, where $0 \leq x \leq 1$, an initial $\tau = 1$ does not seem convenient, although the results are not severely influenced. We mostly used $\tau = 0.2$. Each row of Table 2 is the outcome of 100 runs with the values of $|P|$, $\tau$ and MaxCount as shown on the row.

Toy problem: minimize $x_1 + x_2$ subject to:

\[
\begin{align*}
0 & \leq x_1 \leq 1 \\
1.5 & \leq x_1 + 2x_2 + 0.5 \sin(2\pi(x_1^2 - 2x_2)) \\
x_1^2 + x_2^2 & \leq 1.5
\end{align*}
\] 

As expected, Table 2 shows that the bigger the population size $|P|$ and the number of random restarts MaxCount, the higher the number of function evaluations; however the success rate is high for this problem. The first row of the table shows the results when all strategies for looking for a global minimum are turned off, namely $\beta = 0$, $|P| = 1$, and MaxCount = 0. Even in this case, the algorithm found the global minimum in 60% of the runs with a reduced number of function evaluations.
Table 2 Performance of the algorithm on the toy problem (2 variables and 3 minima) with different parameters and $\delta = 0.1$ (100 runs).

| Func (#var) | #min | $|P|$ | $\tau$ | MaxCount | succ minima per run | Func.Evals |
|------------|------|------|-------|----------|---------------------|------------|
| toy (2)    | 3    | 1    | 0.2   | 0        | 60                  | 1          |
|            |      |      |       |          | 141 - 230 - 319     |            |
|            | 3    | 5    | 1     | 5        | 93                  | 1 - 1.3 - 3|
|            | 3    | 5    | 0.2   | 2        | 100                 | 1 - 1.2 - 3|
|            |      |      |       |          | 1872 - 3139 - 4957  |            |
|            | 3    | 5    | 0.2   | 5        | 100                 | 1 - 1.1 - 3|
|            | 3    | 5    | 0.2   | 10       | 100                 | 1 - 1.1 - 3|
|            |      |      |       |          | 1693 - 5259 - 12236 |            |

NOMAD’s performance: 7 67 - 137 - 907

Gramacy et al [20] report that their method found the global minimum in 99% of the runs, whereas other methods, including MADS, failed in 15% of the runs. Ours always reports the global minimum when $|P| = 5$; $\tau = 0.2$. Moreover, when MaxCount = 2, only 3139 function evaluations, in average, was needed. The authors in [20] report an average of 8240 evaluations with an algorithm designed to look only for the global minimum. Table 2 also shows NOMAD’s performance. The global was attained in 7% of the runs. We should emphasize that these results are very encouraging because our algorithm is able to generate multiple minima, while [20] and NOMAD are devoted to find a single minimum. Table 2 clearly shows that the extra effort spent by the Addis-Locatelli strategy was unnecessary.

Our second test problem was the Shekel’s function with 5 variable and 30 minima as formulated in [16].

$$
\min_x \left\{ -\sum_{k=1}^{30} \frac{1}{\sum_{j=1}^{5} ((x_j + y_j - M_{kj})^2 + c_k)} \right\}, \quad y = \begin{bmatrix} 1.024914 \\ 0.651725 \\ 9.613927 \\ -1.379139 \\ 4.564090 \end{bmatrix}, \quad (22)
$$

where $M$ and $c$ are given in Table 3.

Table 4 shows the results for Shekel’s function with a starting point randomly generated with a uniform distribution in the interval [0 1]. The first row turned off all features for global optimization. Again we observe the influence of the size of the population and the parameter MaxCount on the performance of the algorithm. We should point out that this is the common outcome for other numerical tests not included here. The practitioner must decide the priority: striving for multiple minima, or global minimum, or limiting the budget on function evaluations and accept a local minimum. We should pay attention to the rise in success from 9% to 55% when the population changes from 1 to 20, while the average of function evaluations per individual in $P$ decreases from 3798 to 2400 in average. Note that the population size of 20 underestimates the 30 minima of the Shekel’s function we are working with. The algorithm always reported between 7 and 11 minima in all runs and found the global over 55% of the runs. This means that a population of 20 points is still large to discern local minimizers with functional values close to the best one found by the algorithm. It is worth mentioning that Tsoulus and Lagaris [36] described an
Table 3  Shekel’s data.

<table>
<thead>
<tr>
<th>( M )</th>
<th>( c )</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
9.681 & 0.667 & 4.783 & 9.095 & 3.517 \\
9.400 & 2.041 & 3.788 & 7.931 & 2.882 \\
8.025 & 9.152 & 5.114 & 7.621 & 4.564 \\
2.196 & 0.415 & 5.649 & 6.979 & 9.510 \\
8.074 & 8.777 & 3.467 & 1.863 & 6.708 \\
7.650 & 5.658 & 0.720 & 2.764 & 3.278 \\
1.256 & 3.605 & 8.623 & 6.905 & 4.584 \\
8.314 & 2.261 & 4.224 & 1.781 & 4.124 \\
0.226 & 8.858 & 1.420 & 0.945 & 1.622 \\
7.305 & 2.228 & 1.242 & 5.928 & 9.133 \\
0.652 & 7.027 & 0.508 & 4.876 & 8.807 \\
2.699 & 3.516 & 5.874 & 4.119 & 4.461 \\
8.327 & 3.897 & 2.017 & 9.570 & 9.825 \\
2.132 & 7.006 & 7.136 & 2.641 & 1.882 \\
4.707 & 5.579 & 4.080 & 0.581 & 9.698 \\
3.004 & 7.559 & 8.567 & 0.322 & 7.128 \\
8.632 & 4.409 & 4.832 & 5.768 & 7.050 \\
4.887 & 9.112 & 0.170 & 8.967 & 9.693 \\
2.440 & 6.686 & 4.299 & 1.007 & 7.008 \\
0.652 & 2.343 & 1.370 & 0.821 & 1.310 \\
5.558 & 1.272 & 5.756 & 9.857 & 2.279 \\
8.798 & 0.880 & 2.370 & 0.168 & 1.701 \\
1.460 & 8.057 & 1.336 & 7.217 & 7.914 \\
0.432 & 8.645 & 8.774 & 0.249 & 8.081 \\
0.679 & 2.800 & 5.523 & 3.049 & 2.968 \\
4.263 & 1.074 & 7.286 & 5.999 & 8.291 \\
9.496 & 4.830 & 3.150 & 8.270 & 5.079 \\
4.138 & 2.562 & 2.532 & 9.661 & 5.611
\end{bmatrix}
| \[
\begin{bmatrix}
0.806 \\
0.517 \\
0.100 \\
0.908 \\
0.965 \\
0.669 \\
0.524 \\
0.902 \\
0.531 \\
0.876 \\
0.462 \\
0.491 \\
0.463 \\
0.714 \\
0.352 \\
0.869 \\
0.813 \\
0.811 \\
0.828 \\
0.964 \\
0.789 \\
0.360 \\
0.369 \\
0.992 \\
0.332 \\
0.817 \\
0.632 \\
0.883 \\
0.608 \\
0.326
\end{bmatrix}
\]

Table 4  Performance of the algorithm on the Shekel’s problem with and without linear constraints (100 runs).

| Func (\#var) | #run | \(|P|\) | \(\tau\) | MaxCount | succ | minima per run | Func.Evals |
|-------------|------|--------|--------|----------|------|----------------|------------|
|             |      |        |        |          |      | low - avg - high | low - avg - high |
| Sh (5)      |      |        |        |          |      |                |            |
| Shekel      | 30   | 1      | 0.2    | 0        | 3    | 1              | 460 - 1727 - 4351 |
|             | 30   | 1      | 0.2    | 2        | 4    | 1              | 701 - 2243 - 6921 |
|             | 30   | 1      | 0.2    | 5        | 5    | 1              | 781 - 3075 - 8794 |
|             | 30   | 1      | 0.2    | 10       | 9    | 1              | 741 - 3798 - 17170 |
|             | 30   | 20     | 0.2    | 2        | 55   | 8 - 11 - 15    | 37120 - 48001 - 65689 |
|             | 30   | 20     | 0.2    | 5        | 66   | 7 - 11 - 15    | 46760 - 73670 - 115970 |
|             | 30   | 20     | 0.2    | 10       | 69   | 7 - 10.5 - 15  | 47840 - 94240 - 156460 |
| **          | 30   | 20     | 1      | 5        | 27   | 11 - 14.9 - 18 | 187297 - 244273 - 288961 |

NOMAD’s performance:  
1  453 - 608 - 857
NOMAD’s performance: **  
0  202 - 823 - 5192

** Additional linear constraints to place the global minimum at the boundary.
For details see [16, App. A, Problem 13]
algorithm to find all function minima and solved a simpler version of the Shekel 5-D function with only 5 minima. They needed 25930 function evaluations and 25431 gradient evaluations, which shows up the difficulty of this problem. NOMAD’s behavior is acceptable provided there exists no concern in finding the global. Table 4 also shows the performance of the algorithm when 6 linear constraints and 6 bounds are added to the formulation of the problem with three active linear constraints and one active bound at the optimum solution. This rules out the possibility of dealing with an unconstrained minimization problem. The problem became hard under the presence of these constraints. The success rate fell appreciably and the number of function evaluations increased notably.

We next tried the Ackley function with $n$ variables.

$$\text{minimize } 20 + \exp(1 - 20\exp\left(-0.2\sqrt{n \sum_{j=1}^{n} x_j^2}\right) - \exp\left(\frac{1}{n} \sum_{j=1}^{n} \cos(2\pi x_j)\right)$$

subject to: $-15 \leq x \leq 30$

We tried NOMAD on the Ackley’s function with constraints (24) and failed in 68% of the runs with the message “constraint violation measure > 1”. The results

| Func (#var) | #run | $|P|$ | $\tau$ | $\delta$ | Max succ Count | Succ | minima per run | Func.Evals | low - avg - high |
|-------------|------|------|------|-------|--------------|------|----------------|------------|-----------------|
| Ackley 30   | 30   | 1    | 1    | 0     | 0            | 70   | 1              | 18601 - 20235 - 22081 |
|              | 30   | 20   | 0.2  | 0.1   | 5            | 90   | 14 - 18.2 - 20 | 114290 - 152380 - 200830 |
|              | 30   | 20   | 1    | 0.1   | 5            | 87   | 17 - 18 - 19   | 144720 - 199260 - 262918 |
| Ackley* 30  | 30   | 20   | 1    | 0.5   | 5            | 94   | 18 - 18.1 - 19 | 144259 - 196464 - 261515 |
|              | 30   | 20   | 1    | 0.1   | 5            | 64   | 15 - 18.9 - 20 | 29854 - 45686 - 93222 |
| Ackley* 5   | 30   | 20   | 1    | 0.5   | 5            | 63   | 15 - 18.3 - 20 | 30257 - 44332 - 69695 |

Ackley* incorporates the nonlinear constraints (24)

The initial point was randomly drawn from the standard normal distribution. Laguna [25] reports that the Ackley function with 30 variables was elusive. Its global minimum could not be found by several genetic algorithms tested. Table 5 shows that our approach is appealing. It never failed, although sometimes the number of total function evaluations is huge due to the size of the population, but again the number of function evaluations per individual in $P$ dropped sensibly and the rate of success was above 87%. The number of minimizers was between 14 and 18 with a size population of 20.

We repeat the testing for the Ackley’s function with 5 variables and the nonlinear constraints

$$\sum_{j=1}^{5} x_j^2 \leq 36$$

$$x_1^2 = x_2$$

We tried NOMAD on the Ackley’s function with constraints (24) and failed in 68% of the runs with the message “constraint violation measure > 1”. The results
reported for our algorithm in Table 5 are very encouraging. We note that an increase of \( \delta \) (used in line 3 of the algorithm described in Figure 5) might decrease the number of function evaluations. In addition we note that in one instance of our experiments 94% success was achieved with \( \delta = 0.5 \) against 87% with \( \delta = 0.1 \), but no conclusive statement can be given without further experimentation.

We now turn our attention to the problem of finding the lowest energy distribution of \( p \) electrons on a sphere. This is known as the Thomson’s problem and it is included in the Dolan More benchmark collection [12]. They claim that this function has an exponential number of local minima and also report that some modern packages, that make use of first/second order derivatives, have failed in solving this problem. Moreover, standard DFO packages, like NOMAD, cannot be used due to the presence of an equality constraint. The problem is formulated as:

Given \( p \) points \( (x_k, y_k, z_k), k = 1, \ldots, p \) we want to:

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=1}^{p-1} \sum_{j=k+1}^{p} \left[ (x_k - x_j)^2 + (y_k - y_j)^2 + (z_k - z_j)^2 \right]^{-\frac{1}{2}} \\
\text{subject to:} & \quad x_k^2 + y_k^2 + z_k^2 = 1, \quad k = 1, \ldots, p
\end{align*}
\]

(25)

This problem was coded in C, compiled with Devc++ version 5.8.2 and run on an Intel CPU 3.30GHz. A critical part of the algorithm is the choice of the directions of search. At each point \( p_k = (x_k, y_k, z_k) \) we define 3 orthogonal directions \( d_{k1}, d_{k2}, \) and \( p_k \). The latter direction is always infeasible, it points outwards from the sphere, and we work with the points \( \Pi_S(p_k \pm \tau d_{kj}) \), \( j = 1, 2 \), where \( \Pi_S(p) \) is the projection of \( p \) on the sphere \( S \), which is a trivial operation. This strategy forces the algorithm to generate feasible points on the sphere and no penalization is necessary.

Due to the intrinsic characteristics of this problem the following parameters were used in the tests: \( |P| = 1, \tau = 0.1, \text{MaxCount} = 0 \); which means that we try to find the best local minimum with our algorithm, but excluding the Addis-Locatelly strategy in order to avoid a very large number of function evaluations. Table 6 shows the results: the running time was remarkably competitive with those reported in [12], which compute derivatives. Albeit we never report the known global solution, all the local minima found exhibit a function value that is never above 3% of the known global minimum.

<table>
<thead>
<tr>
<th>#var</th>
<th>global (known)</th>
<th>minimum value</th>
<th>Func.Evals</th>
<th>time avg (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>low - avg - high</td>
<td>low - avg - high</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>413</td>
<td>415.33 - 416.72 - 418.93</td>
<td>16596 - 28778 - 43917</td>
<td>0.89</td>
</tr>
<tr>
<td>150</td>
<td>1055.18</td>
<td>1068.25 - 1072.04 - 1085.54</td>
<td>20267 - 51333 - 102760</td>
<td>1.59</td>
</tr>
<tr>
<td>300</td>
<td>4448.41</td>
<td>4507.24 - 4532.94 - 4565.75</td>
<td>40249 - 66811 - 134005</td>
<td>9.44</td>
</tr>
<tr>
<td>600</td>
<td>18439</td>
<td>18776.18 - 18861.28 - 18912.23</td>
<td>51400 - 75996 - 118617</td>
<td>37.82</td>
</tr>
</tbody>
</table>
5.1 Real application

As a final test we try out the well-known application from chemical engineering that models the optimal design of multi-product multi-batch plants, where fixed amounts $q_k, k = 1, \ldots, N$ of $N$ products must be manufactured at a minimum cost. The plant has $M$ processing stages and the model should provide $z_j, v_j, j = 1, \ldots, M$, which are, respectively, the number and sizes of parallel units for each stage. The model must find as well, the optimum values for $x_k, y_k, k = 1, \ldots, N$, which are, respectively, the batch size and cycle times for each product. The model gathers the following data:

- $N$, the number of products, and $M$, the number of processing stages,
- $q_k, k = 1, \ldots, N$, the quantities to be manufactured for each product,
- $t_{k,j}, s_{k,j}$, the processing time and size factor of product $k$ at stage $j$, and
- $H$, the time horizon

and must satisfy the constraints given in Model-M in Figure 6. Both a convex and a non-convex version have been formulated for solving this problem [7, 21, 22]. A complete formulation of both versions is included in the MINLP collection [15] and can be downloaded from http://www.minlp.org/library/problem/index.php?id=48. For the sake of completeness we reproduce the model with its data in Figure 6.

This paper solves the more challenging non-convex model to test the performance of our algorithm on this kind of problems with multiple local minima. We must point out from the onset that the quasi-convex constraints $z_j y_k \geq t_{k,j}, k = 1, \ldots, 5; \ j = 1, \ldots, 6$ are converted into linear constraints if the integer variables $z_j, j = 1, \ldots, 6$ are kept fixed. This observation will be the key for our algorithm.

\[
\begin{align*}
\text{minimize} \quad & f(z, x, y, v) = 250 \sum_{k=1}^{5} z_k v_k^6 \\
& v_j \geq s_{j,k} x_k, \ k = 1, \ldots, 5; \ j = 1, \ldots, 6 \\
& z_j y_k \geq t_{k,j}, \ k = 1, \ldots, 5; \ j = 1, \ldots, 6 \\
& \sum_{k=1}^{5} q_k x_k / x_k \leq 6000 \\
& z_j \in \{1, 2, 3, 4\}; \ 300 \leq v_j \leq 3000, \ j = 1, \ldots, 6
\end{align*}
\]

<table>
<thead>
<tr>
<th>Matrix S = s(k,j)</th>
<th>Matrix T = t(k,j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.9 2.0 5.2 4.9 6.1 4.2</td>
<td>6.4 4.7 8.3 3.9 2.1 1.2</td>
</tr>
<tr>
<td>0.7 0.8 0.9 3.4 2.1 2.5</td>
<td>6.8 6.4 6.5 4.4 2.3 3.2</td>
</tr>
<tr>
<td>0.7 2.6 1.6 3.6 3.2 2.9</td>
<td>1.0 6.3 5.4 11.9 5.7 6.2</td>
</tr>
<tr>
<td>4.7 2.3 1.6 2.7 1.2 2.5</td>
<td>3.2 3.0 3.5 3.3 2.8 3.4</td>
</tr>
<tr>
<td>1.2 3.6 2.4 4.5 1.6 2.1</td>
<td>2.1 2.5 4.2 3.6 3.7 2.2</td>
</tr>
</tbody>
</table>

Vector q: 250000 150000 180000 160000 120000

Fig. 6 A model with integer and continuous variables

Model-M has the following characteristics. It has

- 16 continuous variables denoted by ($v \in \mathbb{R}^6, y \in \mathbb{R}^5, x \in \mathbb{R}^5$),
– 6 integer variables denoted by \( z_j, j = 1, \ldots, 6 \),
– bound constraints on the variable \( v \) given by (30),
– 30 linear constraints given by (27),
– 30 quasi-convex constraints, given by (28),
– 1 hard constraint given by (29), and
– a non-convex objective function given by (26).

Model-M has several local minima. In [22] the authors report 4 local minima with functions values ranging from the global solution of 285506.5 up to 349864.6.

To solve the model we require a starting point \( z_0^k \in \{1, 2, 3, 4\} \), \( k = 1, \ldots, M \), which is either provided by the user, or generated by the algorithm described in Figure 4, applied to Model-M without the integrality constraints on \( z \), and we can define for example \( z_0 = \lceil \bar{z} \rceil \), where \( \bar{z} \) is the optimum solution of the relaxation of Model-M.

### 5.2 Algorithm for solving Model-M

Figure 7 is a pseudo code of an algorithm that attempts to find the global minimum of Model-M. All variables in the minimization problems (31) are continuous, because the integer variables in the aggregated problems have a fixed value, that is, \( z^i = z^i_1, \ldots, z^i_6 \). Moreover, the quasi-convex constraints (28) are reduced to easy to handle linear constraints; in fact we observe that \( \gamma_k = \max_k (\frac{k_{ij}^i}{\gamma_j^i}) \) is feasible. For this application we used \( \rho = 500 \) and solve iteratively problem (31) with a fixed \( z^i \),

\[
\begin{align*}
\min_{x,y,v} & \quad 250 \sum_{k=1}^{5} z_{k}^{0.6} + 500 \max \left[ \sum_{k=1}^{5} q_k y_k / x_k - 6000, 0 \right] \\
\text{s.t.} & \quad v_j \geq s_{kj} x_k, \ k = 1, \ldots, 5; j = 1, \ldots, 6 \\
& \quad \gamma_j^i y_k \geq k_{ij}, \ k = 1, \ldots, 5; j = 1, \ldots, 6 \\
& \quad 300 \leq v_j \leq 3000, \ j = 1, \ldots, 6 
\end{align*}
\]  

(31)

From now on, the feasible set of (31) will be denoted by \( \mathcal{E} \). The algorithm described in Figure 7 is very simple indeed and it is based on a well-known property of a local minimum of functions defined on discrete variables. This strategy for solving a MINLP problem was used successfully by Pachón in his recent dissertation [31]. We define \( f(z) \) as a function of the integer variable \( z \) and try to find its local minimum. The algorithm generates a non-monotone sequence \( f(z^0), \ldots, f(z^i), \ldots \), because we preserve the hill climbing feature; indeed any feasible \( z^{i+1} \) is accepted if \( f(z^{i+1}) < \phi_i \), and the updating of \( \phi_i \) mimics the updating of \( \phi \) for continuous variables; so \( \phi_i \geq f(z^i) \), and \( \phi_i \) approaches \( f(z^*) \). The algorithm stops when it finds that \( z^* \) is a local minimum. We accept that \( z^* \) is a weak local minimizer if \( f(z) \geq f(z^*) \) for all \( z \in \mathcal{F}(z^*) \), where \( \mathcal{F}(\cdot) \) is a feasible neighborhood that we have defined beforehand. In our case

\[
\mathcal{F}(z) = \{ \bar{z} \in \mathcal{E} : \| \bar{z} - z \|_1 = 1 \}.
\]  

(32)
To find a global minimum is not, in general, guaranteed. To improve the expectation we could enlarge the neighborhood, for example

\[ \mathcal{N}(z) = \{ \hat{z} \in \mathcal{E} : ||\hat{z} - z||_1 \leq q \}, \]

where \( q \) is an integer bigger than one. We have used (32), but in all likelihood the most appropriate definition for a neighborhood will depend on the application, but this topic will lead us away from our objectives and will be considered in a forthcoming paper. Our augmented problem (31) will generate a finite sequence \( \{z^i, x^i, y^i, v^i\}_{i \in \mathbb{K}} \) of feasible points to model-M, which is an important feature for the practitioner. By convention \( f(\hat{z}) = \infty \) when there exists no \((x, v, u)\) for which \((\hat{z}, x, v, u) \in \mathcal{E}\).

**Lemma 6** If there exists some \((x, v, y)\) such that \((z^0, x, v, y) \in \mathcal{E}\) and if all conditions of lemma 5 hold (for problem (31)), then all members of the sequence \( \{(z^i, x^i, v^i, y^i)\} \) generated by the algorithm remain feasible for Model-M.

**Proof** Since \( \mathcal{E} \) is bounded [7], problem (31) is well defined and its solution satisfies the constraints of Model-M (Figure 6) by Corollary 2. □

It is well known that any optimization algorithm will have a better performance in regards to the number of function evaluations if the starting point is close to the solution. In our experiments we use the algorithm in figure 4 to provide an initial point. We simply relax the integer variables to obtain a lower bound for (31). We then start the algorithm with the rounded values for the integer variables, that is, \( z^0_k = [\bar{z}_k + 0.5], k = 1, \ldots, N \), where \( \bar{z} \) is the optimum continuous solution. We used MaxCount = 0 and \( \tau = 1 \) for solving all instances of (31). When this procedure was used the algorithm always achieved the global minimum. For this problem in particular the continuous solution was \( \bar{z} = (1.741 1.547 2.258 2.158 1.198 1.235) \), and our initial point was the rounded \( \hat{z} \), that is, \( z^0 = (2 2 2 2 1 1) \).

Unfortunately, this point is not feasible so \( f(z^0) = \infty \); but the optimal solution \( z^1 = (2 2 3 2 1 1) \in \mathcal{N}(z^0) \) and the solution was immediately found. We cannot draw conclusive statements about the behavior of the algorithm in MINLP; but this open a line of research that we will pursue. Any technique used to determine the starting point does not prevent the algorithm from converging to a stationary point in the neighborhood defined by (32), or in any other finite neighborhood.
6 Final remarks

We have described a Non Monotone Direct Search algorithm that extends previous works to the solution of optimization problems subjected to linear and non linear equalities, a feature that was absent from MADS until the recent work published while this paper was under review [6]. The algorithm solves a problem with linear constraints by constructing a cone of feasible directions through pivot transformations, a technique well known by both theorists and practitioners. Convergence is ensured for strictly differentiable functions, which allows the search on a bounded number of feasible directions at each iteration. As our previous numerical experiments on box constrained problems was highly competitive [17], no efforts were devoted to generate a dense set of search directions, although our implementation sparingly use random searches. The algorithm solves a problem with nonlinear equalities and inequalities by aggregating these constraints as an exact penalty to the objective function. The paper proves that stationary points of the augmented function are also stationary for the original problem. We obtained promising numerical results on a set of problems that have been identified in the literature as difficult because might cause an optimization technique to fail when trying to find its minimizer. It is worth mentioning that, in general, not even a dense set of feasible directions of search ensures convergence to a minimum of non smooth problems. This shortcoming is shared by all Direct Search techniques known to date.

The algorithm was the heart of a strategy for solving satisfactorily a non-convex Mixed Integer Non-Linear Problem arisen from an application of optimization models in Chemical Engineering [7].

This study also reveals that setting appropriate parameters when using our algorithm strongly depends on the practitioner’s aims. Searching for multiple minima, with no previous knowledge of their number and location is indeed a hard problem whose solution may consume a large budget. To find multiple minima it was necessary to work with an initial population of starting points. Increasing the population size helps, as long as this figure is not way above the unknown number of local minima. The algorithm introduces strategies to decrease the population size, but more research is needed. Searching for multiple minimizers, including the global one, inevitably implies a larger number of function evaluations than searching for the global minimizer only; which in turn requires more function evaluations than searching for a local minimizer.

Finally parallelism has not been analyzed because we are working on some technicalities and practical ways to enforce and implement the theoretical ideas, but parallelism seems to be a natural line of research to pursue. Recent papers [13, 26] employ parallelism to detect multiple local minimizers.

We hope this work will encourage researchers and practitioners to the use and further development of DFO techniques.

Acknowledgements This research has been funded by the Galician Regional Government, Spain under project R2014/037. Dr. Ildemaro García thanks the Decanato de Investigación y Desarrollo at USB for providing him with the hardware to carry out the experiments.
One of the referees pointed out to us reference [14] and gave us clues to correct a flaw. Section 2.3 was rewritten following his advise. This enhanced the paper noticeably.

References