DGO: A new DIRECT-type MATLAB toolbox for derivative-free global optimization

Linas Stripinis\textsuperscript{a}, Remigijus Paulavičiūs\textsuperscript{a,∗}

\textsuperscript{a}Vilnius University Institute of Data Science and Digital Technologies, Akademijos 4, LT-08663 Vilnius, Lithuania

\textbf{ARTICLE INFO}

\textbf{Keywords:} DIRECT-type algorithms, Derivative-free optimization, Global optimization, Hidden constraints, Parallel computing, Benchmarking, Optimization software

\textbf{ABSTRACT}

In this work, we introduce DGO, a new MATLAB toolbox for derivative-free global optimization. DGO collects various deterministic derivative-free DIRECT-type algorithms for box-constrained, generally-constrained, and problems with hidden constraints. Each sequential algorithm is implemented in two different ways: using static and dynamic data structures for more efficient information storage and organization. Furthermore, parallel schemes are applied to some promising algorithms within DGO. The toolbox is equipped with a graphical user interface (GUI), which ensures the user-friendly use of all functionalities available in DGO. Available features are demonstrated in detailed computational studies using a created comprehensive library of global optimization problems. Additionally, eleven classical engineering design problems are used to illustrate the potential of DGO to solve challenging real-world problems.

\section{1. Introduction}

The \textsc{direct} (D\textsc{iving} R\textsc{ectangles}) algorithm [35] is a well-known and widely used solution technique for derivative-free global optimization problems. The \textsc{direct} algorithm is an extension to classical Lipschitz optimization [57, 58, 64, 65, 71, 74], where the Lipschitz constant is not assumed to be known. This property makes \textsc{direct}-type methods especially attractive for the solution of various real-world optimization problems (see, e.g., [1, 2, 7, 13, 15, 23, 46, 56, 60, 79] and the references given therein). Moreover, a recent review and comparison in [68] revealed that, on average, \textsc{direct}-type algorithms’ performance is one of the best among all tested state-of-the-art derivative-free global optimization approaches. The \textsc{direct}-type algorithms often outperform algorithms belonging to other well-known classes, such as Genetic [32], Simulated annealing [38], and Particle swarm optimization [37].

While the original \textsc{direct} addresses only box-constrained optimization problems, various \textsc{direct}-type modifications and extensions have been proposed. Based on the type of constraints, \textsc{direct}-type algorithms can be classified into four main categories:

- Box-constrained (see, e.g., [20, 21, 33, 23, 35, 44, 45, 46, 47, 60, 69] and the references given therein);
- Linearly-constrained/symmetric (see, e.g., [25, 55, 59, 60, 61] and the references given therein);
- Generally-constrained (see, e.g., [12, 19, 33, 40, 79] and the references given therein);
- Containing hidden constraints (see, e.g., [22, 53, 77] and the references given therein).

\textsc{Matlab} [49] is one of the most broadly used mathematical computing environments in scientific and technical computing. Many widely used implementations of the original \textsc{direct} algorithm (see, e.g., [5, 18, 22]) as well as various later introduced \textsc{direct}-type extensions (see, e.g., [40, 43, 44, 60]), were developed using \textsc{matlab}. Motivated by this, we developed a \textsc{direct}-type global optimization toolbox (\textsc{dgo}) within the \textsc{matlab} environment. The toolbox is equipped with a graphical user interface (GUI), which links to a \textsc{directlib} library and ensures the user-friendly use of all functionalities available in DGO.

The first publicly available \textsc{direct} implementations and many others introduced later are typically using static data structures for data storage and organization. Our recent work [81] showed that the \textsc{matlab} implementation of the same \textsc{direct}-type algorithm based on dynamic data structures often has a significant advantage over the performance based on more straightforward static data structures. Therefore, each algorithm in \textsc{dgo} is implemented using both static and dynamic data structures. As various applications can benefit from parallel computing, therefore, the \textsc{spmd} (Single Program Multiple Data) parallel scheme (see [81] for more information on this) is used for the parallel implementation of some approaches.

\subsection{1.1. Contributions and structure}

We summarize our main contributions below:

- We develop a new \textsc{matlab} toolbox for derivative-free global optimization, consisting of 36 different \textsc{direct}-type algorithms (see Table 1 for the details).
- We implement each \textsc{direct}-type algorithm using two types of data structures, static and dynamic [28, 81].
- We design a user-friendly application with a graphical user interface (GUI).
- We adapt the \textsc{spmd} parallel scheme [81] for the parallel implementation of selected \textsc{direct}-type algorithms.
Linas Stripinis, Remigijus Paulavičius: Paper  Page 2 of 26

2. Theoretical and algorithmic backgrounds

This section provides the classification of existing DIRECT-type algorithms and describes in more detail algorithms implemented within our toolbox. The parallel scheme used in the implementation of some parallel algorithms is also discussed here. DGO toolbox is introduced and described in Section 3. The detailed computational study of the DGO toolbox using classical global optimization test and engineering design problems are presented in Sections 4 and 5, respectively. Finally, in Section 6, we conclude the work and discuss the possible future directions.

2.1. Classification of DIRECT-type implementations

We expand a library of test and engineering global optimization problems (DIRECTlib) [80].

We make MATLAB toolbox (DGO) open-source, i.e., freely available to anyone [76].

The rest of the paper is organized as follows. Section 2 provides the classification of existing DIRECT-type algorithms and describes in more detail algorithms implemented within our toolbox. The parallel scheme used in the implementation of some parallel algorithms is also discussed here. DGO toolbox is introduced and described in Section 3. The detailed computational study of the DGO toolbox using classical global optimization test and engineering design problems are presented in Sections 4 and 5, respectively. Finally, in Section 6, we conclude the work and discuss the possible future directions.

2.2. Theoretical and algorithmic backgrounds

This section provides the classification of existing DIRECT-type algorithms and describes in more detail algorithms implemented within our DGO toolbox. For a thorough review, we refer the reader to a recent survey [34].

The derivative-free DIRECT algorithm [35] is an efficient deterministic technique to solve global optimization [31, 72, 82] problems subject to simple box-constraints

\[
\min_{x \in D} f(x),
\]

where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) denotes the objective function and the feasible region is an \( n \)-dimensional hyper-rectangle \( D = [a, b] = \{ x \in \mathbb{R}^n : a_j \leq x_j \leq b_j, j = 1, \ldots, n \} \). The objective function \( f(x) \) is supposed to be Lipschitz-continuous (with an unknown Lipschitz constant) but can be nonlinear, non-differentiable, non-convex, and multimodal.

At the initial iteration, the DIRECT algorithm normalizes the feasible region \( D \) to be the unit hyper-cube \( \bar{D} \), and refers to original space \( D \) only when evaluating the objective function. Regardless of the dimension, the first evaluation of the objective function is done at the midpoint \( c_1 \) of \( D \). Then, the initial hyper-rectangle \( D_1 \) is trisected along all of its longest dimensions, as shown in Fig. 1. New points are sampled, and the objective function is evaluated at \( c_j = 1/3 I^n \pm c_1 \), where \( I^n \) is identity matrix, and \( j = 2, \ldots, 2n + 1 \). Furthermore, \( D \) trisected into \( 2n + 1 \) smaller non-overlapping hyper-rectangles, such that the lowest function values are placed in the largest measured hyper-rectangles.

The essential step in DIRECT is the identification of potentially optimal (the most promising) hyper-rectangles (POH) of the current partition, which at the iteration \( k \) is defined as

\[
P^k = \{ \bar{D}^k_1 : i \in I^k \},
\]

where \( \bar{D}^k_1 = [a^k_i, b^k_i] \) = \( \{ x \in \mathbb{R}^n : 0 \leq a^k_i \leq x_j \leq b^k_i \leq 1, j = 1, \ldots, n, \forall i \in I^k \} \) and \( I^k \) is the index set identifying the current partition \( P^k \). The next partition \( P^{k+1} \) is obtained after the subdivision of the selected potentially optimal hyper-rectangles from the current partition \( P^k \).

The selection procedure at the initial step is trivial as we have only one candidate \( \bar{D} \). To make the selection of potentially optimal hyper-rectangles in the future iterations, DIRECT assesses the goodness based on the lower bound estimates for the objective function \( f \) over each hyper-rectangle \( \bar{D}^k_1 \). The requirement of potential optimality is stated formally in Definition 1.
Definition 1. (Potentially optimal hyper-rectangle) Let \( c^k_j \) denote the center sampling point and \( \delta^k_j \) be a measure (equivalently, sometimes called distance or size) of the hyper-rectangle \( \bar{D}^k_j \). Let \( \varepsilon > 0 \) be a positive constant and \( \hat{f}(\bar{c}) \) be the best currently found value of the objective function. A hyper-rectangle \( \bar{D}^k_j, j \in I^k \) (where \( I^k \) is the index set identifying the current partition) is said to be potentially optimal if there exists some rate-of-change (Lipschitz) constant \( \bar{L} > 0 \) such that

\[
f(c^k_j) - \bar{L}\delta^k_j \leq f(c^k_i) - \bar{L}\delta^k_i, \quad \forall i \in I^k, \tag{2}
\]

\[
f(c^k_j) - \bar{L}\delta^k_j \leq \hat{f}(\bar{c}) - \varepsilon|\hat{f}(\bar{c})|, \tag{3}
\]

where the measure of the hyper-rectangle \( \bar{D}^k_j \) is

\[
\delta^k_j = \frac{1}{2}\|b^k_j - a^k_j\|_2. \tag{4}
\]

The hyper-rectangle \( \bar{D}^k_j \) is potentially optimal if the lower Lipschitz bound for the objective function computed by the left-hand side of (2) is the smallest one with some positive constant \( \bar{L} \) among the hyper-rectangles of the current partition \( P^k \). In (3), the parameter \( \varepsilon \) is used to protect from an excessive refinement of the local minima \([35, 55]\). Authors obtained good results for \( \varepsilon \) values ranging from \( 10^{-3} \) to \( 10^{-7} \) in \([35]\). A geometrical interpretation of the selection procedure is shown in Fig. 2. Each subsequent iteration \textsc{DIRECT} performs a selection of POHs, which are sampled, evaluated, and trisected. Almost all \textsc{DIRECT}-type extensions and modifications follow the same algorithmic framework, which is summarized in Algorithm 1.

### 2.1. \textsc{DIRECT}-type algorithms for box-constrained global optimization

Two broadly used public \textsc{MATLAB} implementations of the original \textsc{DIRECT} algorithm are \textsc{DIRECT v4.8} \([18]\) and \textsc{gbsolve} \([5]\) (see Table 1). Both implementations use vector-based static data memory management to store information, contrary to a tree-structure used in the original paper \([35]\). Interestingly, the performance (based on the number of function evaluations) of different \textsc{DIRECT} implementations using the same nine standard test problems slightly differs \([5]\). The most significant difference is how the selection of potentially optimal hyper-rectangles (the lower convex hull, as shown in Fig. 2) is implemented. Modified Graham’s scan algorithm \([66]\) is used for this in \([5]\), but Lemma 2.3 \([22]\) is used in \([18]\). Also, the numerical tolerances used in the implementations play an essential role \([5]\).
Many different DIRECT extensions have been suggested. Most of them focused on improving the selection of POHs, while others introduced new partitioning and sampling strategies. The summary of all box-constrained proposals considered in the DGO toolbox is given in Table 2. Most algorithms are based on the trisection of \( n \)-dimensional POHs, and just ADC, BIRECT, and both DISIMPL versions use different partitioning strategies. Below we briefly review the DIRECT-type approaches for box-constrained global optimization implemented in the current release of the DGO toolbox.

Adaptive diagonal curves (ADC) based algorithm was introduced in [69]. Independently of the problem dimension, the ADC algorithm evaluates the objective function \( f(x) \) only at two vertices \( a^{k}_{i} \) and \( b^{k}_{i} \) of the main diagonals of each hyper-rectangle \( D^{k}_{i} \). Additionally, a new two-phase technique balancing local and global information usage has also been incorporated. When the ADC algorithm performs a sufficient number of hyper-rectangle subdivisions near the current best point, the two-phase approach forces the new algorithm to switch to exploring a larger hyper-rectangle that could contain better solutions.

BIRECT (BIsecting RECTangles) [54] is motivated by the diagonal partitioning strategy [69, 70, 72]. The bisection is more appropriate to the trisection because of the shape of hyper-rectangles. Still, usual sampling strategies at the center or the diagonal’s endpoints are not appealing for bisection. In BIRECT, the objective function is evaluated at two points on the diagonal equidistant between themselves and a diagonal’s vertices. Such a sampling strategy entitles the reuse of the sampling points in descendant hyper-rectangles. Additionally, more comprehensive information about the objective function is considered compared to the central sampling strategy used in most DIRECT-type algorithms.

In DISIMPL [59], instead of hyper-rectangles, simplicial partitions are considered. At the first iteration, the hyper-cube \( D \) is partitioned into \( n! \) simplices by the standard face-to-face simplicial division based on the combinatorial vertex triangulation. After this, all simplices share the diagonal of the feasible region and have equal hyper-volume. In [59], we proposed two different sampling strategies. Both are included in the DGO toolbox: i) DISIMPL-C evaluating the objective function at the geometric center of the simplex; ii) DISIMPL-V evaluating the objective function on all unique vertices of the simplex. For box-constrained problems, the number of initial simplices increases speedily. Therefore, DISIMPL effectively can be used only for small box-constrained problems. However, the DISIMPL approach is auspicious (among all DIRECT-type methods) for symmetric optimization problems [59, 60] and problems with linear constraints [61].

In the DIRECT-restart algorithm [20], the authors introduced an adaptive scheme for the \( \epsilon \) parameter. Condition (3) is needed to stop the DIRECT from wasting function evaluations on minor hyper-rectangles where only a negligible improvement can be expected. The DIRECT-restart algorithm starts with \( \epsilon = 0 \), and the same value for \( \epsilon \) is maintained while improvement is achieved. However, if five consecutive iterations have no improvement in the best function value, the search may be stagnated around a local optimum. Therefore, the algorithm switches to \( \epsilon = 0.01 \) value, seeking to prevent an excessive local search.

The authors of MrDIRECT [45] and MrDIRECT075 [41] algorithms introduced three different levels to perform the selection procedure:

1. At level 2, DIRECT is run as usual, with \( \epsilon = 10^{-5} \).
2. At level 1, the selection is limited to only 90% of \( \tilde{D}^{k}_{i} \)
in the partition $P^k$: 10% of the largest measure hyper-rectangles, where the objective function $f$ values are worst, are ignored. A value of $\varepsilon = 10^{-7}$ is used.

3. At level 0, the selection is limited to only 10% of the best hyper-rectangles from level 1. At this level, a value of $\varepsilon = 0$ is used.

Both algorithms cycle through these levels using “W-cycle”: 21011012. The main difference between the proposed algorithms is that MrDIRECT uses fixed $\varepsilon = 10^{-4}$ value at all levels, while MrDIRECT$_{075}$ follows the rules described above.

In [1], authors relaxed the criteria of selection of POHs and proposed an aggressive version of the DIRECT algorithm. Aggressive DIRECT’s main idea is to select and divide at least one hyper-rectangle from each group of different ‘s main idea is to select and divide at

Aggressive DIRECT uses fixed $\varepsilon = 10^{-4}$ value at all levels, while MrDIRECT$_{075}$ follows the rules described above.

In [1], authors relaxed the criteria of selection of POHs and proposed an aggressive version of the DIRECT algorithm. Aggressive DIRECT’s main idea is to select and divide at least one hyper-rectangle from each group of different diameters $\delta^k_i$ having the lowest function value. Therefore, using Aggressive DIRECT in the situation presented in Fig. 2, a hyper-rectangle with the center point $c_3$ would also be selected and divided. The aggressive version performs much more function evaluations per iteration compared to other DIRECT-type methods. From the optimization point of view, such an approach seems less favorable since it “wastes” function evaluations by exploring unnecessary (non-potentially optimal) hyper-rectangles. However, such a strategy is much more appealing in a parallel environment, as was shown in [26, 27, 29, 86].

In [23] the algorithm named direct-1 was proposed. In most of DIRECT-type algorithms, the size of the hyper-rectangle is measured by a half-length of a diagonal (see (4)). In direct-1, the measure of a hyper-rectangle is evaluated by the length of its longest side. Such a measure corresponds to the infinity norm and allows the direct-1 algorithm to group more hyper-rectangles with the same measure. Thus, there are fewer distinct measures, and therefore, fewer potentially optimal hyper-rectangles are selected. Moreover, with direct-1 at most one hyper-rectangle selected from each group, even if there is more than one potentially optimal hyper-rectangle in the same group. Such a strategy allows a reduction in the number of divisions within a group.

In [21], the authors concluded that the original direct algorithm is sensitive to the objective function’s additive scaling. Additionally, the algorithm does not operate well when the values of the objective function are large enough. To overcome this, the authors proposed a scaling of function values by subtracting the median of the collected function values. direct-m replaces the equation (3) in Definition 1 to:

$$f(c_j) - \hat{\Delta_j} \leq \hat{f}(\hat{c}) - \varepsilon |\hat{f}(\hat{c}) - \hat{f}_{\text{median}}|. \quad \text{(5)}$$

Similarly, in [42], the authors extended the same idea in direct-a to reduce the objective function’s additive scaling. At each iteration, instead of the median value ($\hat{f}_{\text{median}}$), authors proposed to use the average value ($\hat{f}_{\text{average}}$):

$$f(c_j) - \Delta_j \leq \hat{f}(c) - \varepsilon |\hat{f}(c) - \hat{f}_{\text{average}}|. \quad \text{(6)}$$

Another extension of the DIRECT algorithm was proposed in [25]. Here, the authors introduced glbSolve-sym (glbSolve-sym2) as DIRECT extensions for symmetric Lipschitz continuous functions. When solving symmetric optimization problems, there exist equivalent subregions in the hyper-rectangle. To avoid exploration over equivalent subregions, for all newly generated hyper-rectangles, the algorithm determines which hyper-rectangles can be safely discarded (because of the problem’s symmetrical nature) from the further search.

In the PLOR algorithm [52], the set of all Lipschitz constants (here with the set of potentially optimal hyper-rectangles) is reduced to just two: with the maximal diameter ($\delta^k_{\text{max}}$) and the minimal one ($\delta^k_{\text{min}}$). In such a way, the PLOR approach is independent of any user-defined parameters and balances equally local and global searches during the optimization process.

In our recent DIRECT extension, DIRECT-GL [78], we introduced a new approach to identifying the extended set of potentially optimal hyper-rectangles. Here, using a novel two-step-based strategy, the set of the best hyper-rectangles is enlarged by adding more medium-measured hyper-rectangles with the smallest function value at their centers and, additionally, closest to the current minimum point. The first step of the selection procedure forces the DIRECT-GL algorithm to work more globally (compared to the selection used in DIRECT [35]). In contrast, the second step assures a faster and broader examination around the current minimum point. The original DIRECT-GL version performs a selection of potentially optimal hyper-rectangles in each iteration twice [78], and the algorithm separately handles the found independent sets $G$ (using Definition 2 from [78] - DIRECT-G) and $L$ (using Definition 3 from [78] - DIRECT-L). Following the same trend from [81], the version used in this paper slightly differs compared to [78]. In the current version of DIRECT-GL, identifying these two sets is performed in succession, and the unique union of these two sets ($S = G \cup L$) is used in Algorithm 1, Line 3. This modification was introduced, aiming to reduce the data dependency in the algorithm seeking a better parallelization.

Several different globally biased (gb-) versions of DIRECT-type algorithms were introduced and investigated [55, 56]. Proposed approaches are primarily oriented for solving multidimensional optimization problems and contain a phase that constrains itself to large subregions. The introduced step performs until a sufficient number of divisions of hyper-rectangles near the current best point is done. Once those subdivisions around the current best minima point are performed, the neighborhood contains only small measure hyper-rectangles and all larger ones located far away from it. Therefore, the two-phase strategy makes the DIRECT-type algorithms examine larger hyper-rectangles and return to the general phase only when an improved minimum is obtained. Within this toolbox, the proposed globally biased strategy is combined with glbSolve, DIRECT, DISIMPL-C, and DISIMPL-V algorithmic.
frameworks.

Finally, three different hybridized DIRECT-type algorithms are proposed (DIRECT-rev [33], DIRMIN [46], BIRMIN [56]). In our implementation, all algorithms are combined with the same local search routine – fmincon. The DIRMIN algorithm suggested running a local search starting from the midpoint of every potentially optimal rectangle. However, such an approach likely generates more local searches than necessary, as many of the starting points will converge to the same local optimum. The other authors of the DIRECT-rev and BIRMIN algorithms tried to minimize the usage of local searches. They suggested using fmincon only when some improvement in the best current solution is obtained. The authors in the [33] additionally incorporated the following two enhancements. First, in the DIRECT-rev algorithm, selected hyper-rectangles are trisected only on one longest side. Second, only one potentially optimal hyper-rectangle is selected if several equally good exist (the same diameter and objective values) in Definition 1.

2.2. DIRECT-type algorithms for generally constrained global optimization

The original DIRECT algorithm [35] solves optimization problems only with bounds on the variables. In this subsection, we consider a generally constrained global optimization problem of the form:

\[
\min_{x \in D} f(x)
\]

s.t. \( g(x) \leq 0 \), \( h(x) = 0 \),

where \( f : \mathbb{R}^n \rightarrow \mathbb{R}, g : \mathbb{R}^n \rightarrow \mathbb{R}^m, h : \mathbb{R}^n \rightarrow \mathbb{R}^r \) are (possibly nonlinear) continuous functions. The feasible region is a nonempty set, consisting of points that satisfy all constraints, i.e., \( D^\text{feas} = D \cap \Omega \neq \emptyset \), where \( \Omega = \{ x \in \mathbb{R}^n : g(x) \leq 0, h(x) = 0 \} \). As for the box-constrained problems, it is also assumed that the objective and all constraint functions are Lipschitz-continuous (with unknown Lipschitz constants) but can be nonlinear, non-differentiable, non-convex, and multimodal.

The first DIRECT-type algorithm for problems with general constraints was introduced in [33]. Finkel in [19] investigated three different constraint handling schemes within the DIRECT framework. The comparison revealed various disadvantages of the initial proposals. Recently, various new promising extensions for general global optimization problems were introduced (see, e.g., [3, 12, 40, 62, 63, 79] and the references given therein). Below we briefly review the approaches implemented in the current release of the DGO toolbox (see Table 1).

An exact L1 penalty approach DIRECT-L1 [18] is transforming the original constrained problem (7) in the form:

\[
\min_{x \in D} f(x) + \sum_{i=1}^{m} \max\{ g_i(x), 0 \} + \sum_{i=1}^{r} \gamma_i(h_i(x)),
\]

where \( \gamma_i \) are penalty parameters. Experiments in [19] showed promising results of this approach. Nevertheless, the biggest drawback is the users’ requirement to set penalty parameters for each constraint function manually. In practice, choosing penalty parameters is an essential task and can significantly impact the performance of the algorithm [19, 40, 60, 61, 79].

In [79], we have introduced a new DIRECT-type extension based on the DIRECT-GL [78] algorithm. The new DIRECT-GLce algorithm uses an auxiliary function approach that combines information on the objective and constraint functions and does not require any penalty parameters. The DIRECT-GLce algorithm works in two phases, where during the first phase, the algorithm finds feasible points and in the second phase improves a feasible solution. A separate step for handling infeasible initial points is beneficial when the feasible region is small compared to the entire search space. In the first phase, DIRECT-GLce samples the search space and minimizes the sum of constraint violations, i.e.:

\[
\min_{x \in D} \varphi(x),
\]

where

\[
\varphi(x) = \sum_{i=1}^{m} \max\{ g_i(x), 0 \} + \sum_{i=1}^{r} |h_i(x)|.
\]

The algorithm works in this phase until at least one feasible point \( x \in D^\text{feas} \) is found, where

\[
D^\text{feas} = \{ x : 0 \leq \varphi(x) \leq \epsilon_{\varphi}, x \in D \}.
\]

When feasible points are located, the effort is switched for improving the feasible solutions. In the second phase, DIRECT-GLce uses the transformed problem (7):

\[
\min_{x \in D^\text{feas}} f(x) + \bar{\varepsilon}(x, f^\text{feas}).
\]

\[
\bar{\varepsilon}(x, f^\text{feas}) = \begin{cases} 
0, & x \in D^\text{feas} \\
0, & x \in D^\text{inf}
\end{cases}
\]

\[
\begin{cases}
\varphi(x) + \Delta, & \text{otherwise,}
\end{cases}
\]

where

\[
D^\text{inf} = \{ x : f(x) \leq f^\text{feas}, \epsilon_{\varphi} < \varphi(x) \leq \epsilon_{\text{cons}}, x \in D \},
\]

and \( \epsilon_{\text{cons}} \) is a small tolerance for constraint function sum, which automatically varies during the optimization process. An auxiliary function \( \bar{\varepsilon}(x, f^\text{feas}) \) depends on the sum of the constraint functions and the parameter \( \Delta = | f(x) - f^\text{feas} | \), which is equal to the absolute value of the difference between the best feasible function value found so far (i.e., \( f^\text{feas} \)
<table>
<thead>
<tr>
<th>Alg.</th>
<th>Step</th>
<th>Partitioning scheme</th>
<th>Sampling scheme</th>
<th>Selection of POH &amp; comments on additional steps, if any</th>
<th>Input param.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggressive DIRECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-G</td>
<td></td>
<td>Hyper-rectangular partitions based on n-dimensional trisection</td>
<td>Samples midpoints of the hyper-rectangles</td>
<td>Uses enhanced global selection (Definition 2 in [78]).</td>
<td>No input parameters</td>
</tr>
<tr>
<td>DIRECT-L</td>
<td></td>
<td></td>
<td></td>
<td>Uses enhanced local selection (Definition 3 in [78]).</td>
<td></td>
</tr>
<tr>
<td>DIRECT-GL</td>
<td></td>
<td></td>
<td></td>
<td>Uses the unique union of two sets obtained using enhanced global and local selection (Definitions 2 and 3 [78]).</td>
<td></td>
</tr>
<tr>
<td>DIRECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-restart</td>
<td></td>
<td>Hyper-rectangular partitions based on 1-dimensional trisection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-rev</td>
<td></td>
<td>Hyper-rectangular partitions based on 1-dimensional trisection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRMIN</td>
<td></td>
<td>Hyper-rectangular partitions based on n-dimensional trisection</td>
<td></td>
<td>Uses Definition 1</td>
<td></td>
</tr>
<tr>
<td>PLOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>glbSolve</td>
<td></td>
<td>Hyper-rectangular partitions based on n-dimensional trisection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>glbSolve-sym</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>glbSolve-sym2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MrDIRECT</td>
<td></td>
<td></td>
<td></td>
<td>Discards unnecessary hyper-rectangles for symmetric functions.</td>
<td>Balance parameter $\epsilon$</td>
</tr>
<tr>
<td>MrDIRECT_075</td>
<td></td>
<td></td>
<td></td>
<td>Performs the selection of POH on three different sets (“levels”). MrDIRECT_075 uses different $\epsilon$ values at each level.</td>
<td></td>
</tr>
<tr>
<td>Gb-glbSolve</td>
<td></td>
<td></td>
<td></td>
<td>Uses an adaptive scheme for balancing the local and global search. Local minimization procedure fmincon is embedded into the BIRMIN algorithm.</td>
<td></td>
</tr>
<tr>
<td>Gb-DIRECT</td>
<td></td>
<td>Hyper-rectangular partitions based on 1-dimensional bisection</td>
<td>Samples hyper-rectangle at two points lying on diagonals</td>
<td>Uses the function convex hull to return all points on the convex hull, even redundant ones. convex hull is based on the extended version of the GRAHAMSHULL [66] algorithm.</td>
<td></td>
</tr>
<tr>
<td>BIRMIN</td>
<td></td>
<td></td>
<td>Sample hyper-rectangle at two vertices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIRECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td></td>
<td></td>
<td></td>
<td>Uses an adaptive scheme for balancing the local and global search.</td>
<td></td>
</tr>
<tr>
<td>DISIMPL-C</td>
<td></td>
<td>Simplex partitions based on n-dimensional trisection</td>
<td>Samples midpoints of the simplices</td>
<td>Uses an adaptive scheme for balancing the local and global search.</td>
<td></td>
</tr>
<tr>
<td>GB-DISIMPL-C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GB-DISIMPL-V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISIMPL-V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Summary of the main algorithmic characteristics of DIRECT-type methods for box constrained global optimization
and the objective value at an infeasible center point. The primary purpose of the parameter \( \Delta \) is to forbid the convergence of the algorithm to infeasible regions by penalizing the objective value obtained at infeasible points. In such a way, the formulation (12) does not require any penalty parameters and determines the convergence of the algorithm to a feasible solution. The value of \( \xi(x, f^{\text{feas}}) \) is updated when a smaller value of \( f^{\text{feas}} \) is found. This way, the new \( \text{DIRECT-GLc} \) algorithm divides more hyper-rectangles with center points lying close to the boundaries of the feasible region, i.e., the potential solution.

The proposed \( \text{DIRECT-GLc} \) algorithm has two extensions: The first one is \( \text{DIRECT-GLc} \) (see Table 1), which is a simplified version of \( \text{DIRECT-GLce} \) and instead of (12) minimizing the transformed problem:

\[
\min_{x \in D} f(x) + \xi(x, f^{\text{feas}}),
\]

\[
\xi(x, f^{\text{feas}}) = \begin{cases} 
0, & x \in D^{\text{feas}} \\
\varphi(x) + \Delta, & \text{otherwise},
\end{cases}
\]

Experimental investigation in [79] showed that the algorithm has the most wins among the algorithms used in the comparison, and it can solve about 50% of the problems with the highest efficiency. Unfortunately, \( \text{DIRECT-GLc} \) algorithm's efficiency decreasing, solving more challenging problems (with nonlinear constraints and \( n \geq 4 \)), where \( \text{DIRECT-GLce} \) is significantly better, so \( \text{DIRECT-GLc} \) should be used only for simpler optimization problems (with linear constraints and \( n \leq 4 \)). The second extension of the \( \text{DIRECT-GLce} \) algorithm is \( \text{DIRECT-GLce-min} \), where the algorithm is incorporated with MATLAB optimization solver \text{fmincon}. In [79] we observed, that by embedding a local minimization procedure into \( \text{DIRECT-GLce-min} \) (see Table 1) significantly reduces the total number of function evaluations compared to \( \text{DIRECT-GLce} \) and can significantly improve the quality of the final solution.

### 2.2.1. DIRECT-type algorithms for linearly constrained global optimization

Let us note that all previously described algorithms for a generally constrained problem can be directly applied to solve linearly constrained problems. This section emphasizes only applicable to problems with linear constraints, but not general ones.

In [61], we have extended the original simplicial partitioning based \text{DISIMPL} algorithm [59, 60] for such problems with linear constraints. Simplices may cover a search space defined by linear constraints. Therefore a simplicial approach may tackle such linear constraints in a very subtle way. In such a way, the new algorithms (\( \text{LC-DISIMPL-C} \) and \( \text{LC-DISIMPL-V} \)) [61] perform the search only in the feasible region, in contrast to other \( \text{DIRECT-type} \) approaches. Nevertheless, the authors in [61] showed that the feasible region's calculation requires solving \( 2n + m \) linear \( n \)-dimensional systems, and such operation is exponential in complexity. Therefore, the proposed algorithm can be effectively used for relatively small \( n \) and \( m \) values.

### 2.3. DIRECT-type algorithms for problems with hidden constraints

In this subsection, we consider the solution to the constrained global optimization problem:

\[
\min_{x \in D} f(x),
\]

where \( f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\} \) denotes an extended real-valued, most likely “black-box” objective function. A priori an unknown feasible region \( D^{\text{feas}} \) is defined as a non-empty set

\[
D^{\text{feas}} = D \cap D^{\text{hidden}} \neq \emptyset,
\]

and \( D^{\text{hidden}} \) are not given by explicit formulae hidden constraints. Such a problem formulation leads to a complex and analytically undefined feasible region.

Problems with hidden constraints often occur when the objective function is not everywhere defined [9]. Such situations arise, e.g., when computations within the objective function fail [7, 10, 14, 75], or when the evaluation of the objective function itself tests for feasibility conditions that are not given by closed-form inequalities [8, 39]. In the same vein as in [9], we call these internal to \( f \) constraints as “hidden constraints,” and assume that \( f \) fails to return a value when evaluated at \( x \notin D^{\text{feas}} \).

One of the first proposed modifications for such problems was the barrier method (\( \text{DIRECT-Barrier} \)) [22]. The \( \text{DIRECT-Barrier} \) algorithm is relatively straightforward and assigns a predefined high value to infeasible hyper-rectangles. However, such an approach produces other well-known problems discussed and reviewed by a few authors [19, 60, 79]. The main issue is that the barrier approach makes exploration around the edges of feasibility very slow. Significant penalties used by the barrier method ensure that no infeasible hyper-rectangle can be potentially optimal as long as there is the same measure hyper-rectangle with the feasible center midpoint. For \( \text{DIRECT-Barrier} \), the priority is the examination of regions where feasible points are found already. Another critical issue concluded in [19] is that hyper-rectangles, even with the sizeable feasible region, will not be explored in a reasonable number of function evaluations. To sum up, the barrier approach is not the best fit for the problem (15).

The second \( \text{DIRECT-type} \) approach for hidden constraints is based on Neighbourhood Assignment Strategy (NAS) [22]. \( \text{DIRECT-NAS} \)’s main idea is to assign the value at infeasible point \( x^{\text{inf}} \notin D^{\text{feas}} \) relative to the objective values attained in the feasible points from the neighborhood of \( x^{\text{inf}} \). \( \text{DIRECT-NAS} \) iterates over all infeasible midpoints by creating surrounding hyper-rectangles around them by keeping the same center points in every iteration. These hyper-rectangles are increased by doubling the length of each dimension. If inside the enlarged region exists more
than one feasible center point DIRECT-NAS, assigns the smallest function value to the infeasible midpoint plus a small epsilon $f(x^{feas}) + \epsilon f(x^{feas})$, where $\epsilon = 10^{-6}$ was proposed to use. If inside the enlarged region does not exist any feasible points, DIRECT-NAS assigns the largest objective function value found so far $f^{max} + \lambda$, where $\lambda = 1$ was proposed to use. Such a strategy does not allow the DIRECT-NAS algorithm to move beyond the feasible region by punishing infeasible midpoints with large values. However, the algorithm’s principal concern is the slow convergence, which is caused by many additional calculations.

Another recent idea to handle hidden constraints within the DIRECT framework is to use a subdividing step for infeasible hyper-rectangles. The proposed subDIRECT-Barrier [53] incorporates techniques from the previously mentioned barrier approach. Specifically, if the center point is identified as infeasible, then subDIRECT-Barrier assigns a considerable penalty value to it. An extra subdividing step is performed only in specific iterations, during which all infeasible hyper-rectangles are identified as potentially optimal and subdivided together with others POHs. The sub-dividing step can decompose the boundaries of the hidden constraints quite efficiently. Still, subDIRECT-Barrier has several apparent drawbacks. The algorithm performance strongly depends on when (how often) the subdividing step is performed. Therefore the number of new subdivisions can grow drastically, especially for higher dimensionality problems.

The most recent version for problems with hidden constraints DIRECT-GLh [77] is based on our previous DIRECT-GL [78] algorithm. For hyper-rectangles with infeasible midpoints, DIRECT-GLh assigns a value depending on how far the center is from the current best minima $x^{min}$. Such a technique does not require any additional computation. Simultaneously, distances from the $x^{min}$ point are already known, as they are used to selecting potential optimal hyper-rectangle schemes adapted from DIRECT-GL [78]. In such a way, DIRECT-GLh does not punish infeasible hyper-rectangles with large values (as was suggested by previous proposals), which are close to the $x^{min}$ and assures a faster and more comprehensive examination of hidden regions. Moreover, this approach employs additional procedures, which use an independent phase to efficiently handle infeasible initial points (see [77] for experimental justification).

2.4. Implementation of parallel DIRECT-type algorithms

We use the MathWorks official extension to the MATLAB language – the Parallel Computing Toolbox [49]. The Parallel Computing Toolbox provides several parallel programming paradigms [48], like threads, parallel for-loops and SPMD (Single Program Multiple Data). In [81], we introduced three parallel implementations of the DIRECT-GLce algorithm within MATLAB. We concluded that the version based on MATLAB spmd is the most efficient and significantly outperforms other implementations based on parfor-loops. Therefore, in the DGO toolbox for parallel implementation, we used the most efficient parallel scheme, based on [81]. Our implementations are based on the SPMD functionality within the Parallel Computing Toolbox, which is used to allocate the work across multiple labs in the MATLAB software environment. Each lab stores information on its main memory block and data is exchanged through the message passing over the interconnection network [49]. The master-slave paradigm is used to implement dynamic load balancing. The flowchart of the parallel algorithmic framework is illustrated in Fig. 3. One lab is the master, denoted by $M$, and the other labs are slaves $W_i, i = 1,\ldots, \rho - 1$. The master also acts as a slave. The master lab decides which hyper-rectangles will be sampled and subdivided and how these tasks will be distributed among all available slave labs. Additionally, the master lab is responsible for stopping the algorithm. The master lab also performs load balancing by distributing the selected hyper-rectangles to the rest of the slave labs. When the slave labs $W_i, i = 1,\ldots, \rho - 1$ receive tasks from the master lab, each performs the Sampling and Subdivision steps sequentially. Then finds a local set of potential optimal hyper-rectangles and sends local data for the further global Selection step back to the master lab and becomes idle until further instructions are received. Suppose any of the termination conditions are satisfied. In that case, all slave labs receive the notification that the master lab has become inactive, and the slave labs will terminate themselves without further messaging. We refer to [81] for a more detailed description and analysis of parallel schemes.

3. DGO toolbox

The sequential and parallel implementation of DIRECT-type algorithms presented in the previous sections forms the basis for our DGO toolbox. The toolbox consists of two main parts:

- DGO.mltxb - MATLAB toolbox package containing implementations of DIRECT-type algorithms (from Table 1), including an extensive DIRECT1b library of the box and generally constrained test and practical engineering global optimization problems, often used for benchmarking DIRECT-type algorithms.

- DGO.mlappinstall - A single MATLAB app installer file containing everything necessary to install and run the DGO toolbox, including a graphical user interface (GUI), a convenient way to use it.

3.1. Graphical user interface

After installation (using DGO.mlappinstall), DGO can be launched from MATLAB APPS, located in the toolbar. The graphical interface of the main DGO toolbox window is shown in Fig. 4. Application is divided into three main parts: i) selection of the type of test problem from the
3.2. MATLAB toolbox

After installation of the MATLAB toolbox (using DGO.mlbxs), all implemented DIRECT-type algorithms and test problems can be freely accessed in the command window of MATLAB. Unlike using GUI, algorithms from the command line require more programming knowledge, and configurations must be done manually. All algorithms can be run using the same style and syntax:

```matlab
[f_min, x_min, history] = algorithm(P, D, OPTS);
```

The left side of the equation specifies the output parameters. After the termination, the algorithm returns the best objective value (f_min), solution point (x_min), and history of the algorithmic performance during all iterations (history). The information presented here is: the iteration number, the total number of objective function evaluations, the current minimal value, and the total execution time.

On the right side, the algorithm name (algorithm) and at least two input parameters are needed to specify. The first one is the problem structure (P) consisting of an objective function:

```matlab
P.f = 'objfun';
```

If the problem involves additional constraints, they also must be specified:

```matlab
P.constraint = 'confun';
```

The second necessary parameter is the bound constraints for each dimension/variable:

```matlab
D (i,1) ≤ x_i ≤ D (i,2), i = 1...n;
```

The last parameter is an optional variable (OPTS) to customize the default settings.
4. Experimental investigation of **DIRECT**-type algorithms in the **DGO** toolbox

This section presents the performance evaluation of **DIRECT**-type algorithms on test and engineering design problems from the **DIRECTlib** [80] library. **DIRECTlib** consists of the box and generally constrained test and practical engineering global optimization problems for various **DIRECT**-type algorithms benchmarking. We distinguish the following classes (types) of global optimization problems:

- **BC** – Box-Constrained problems;
- **LC** – Linearly-Constrained problems;
- **GC** – Generally-Constrained problems.

A summary of all optimization problems in **DIRECTlib** and their properties is given in Appendix A, Table 12. We note that some test problems have several variants, e.g., Bohachevsky, Shekel, and some of them like, Alpine, Csendes, Griewank, can be used by changing the dimensionality of the problem. We used the following dimensions in our experimental setting: \( n = 2, 5, 10, 15 \).

All computations were carried out on a 6-core computer with 8th Generation Intel R Core\textsuperscript{TM} i7-8750H @ 2.20GHz Processor, 16 GB of RAM, and MATLAB R2020b. Performance analysis was carried out using physical cores only and disabled hyper-threading.

All global minima \( f^* \) are known for all test problems. Therefore, the investigated algorithms were stopped when it was generated such the point \( x \) with whom the percent error

\[
pe = 100\% \times \begin{cases} 
\frac{f(x) - f^*}{f^*}, & f^* \neq 0, \\
1, & f^* = 0,
\end{cases}
\]

is smaller than the tolerance value \( \epsilon_{pe} \), i.e., \( pe \leq \epsilon_{pe} \). Additionally, we stopped the tested algorithms when the number of function evaluations exceeded the prescribed maximal limit (equal to \( 2 \times 10^6 \)) or took more than twelve hours. In any of these situations, the final result is set to \( 2 \times 10^6 \) to further process results. Two different values for \( \epsilon_{pe} \) were considered: \( 10^{-2}, 10^{-8} \). All algorithms were tested using the \( \epsilon_{pe} = 10^{-2} \) value. Moreover, algorithms containing additional schemes to speed up the solution’s refinement have been tested using the \( \epsilon_{pe} = 10^{-8} \) value.

Additionally, we analyze and compare the algorithms’ performance by applying the performance profiles [17] to the convergence test (16). Performance profiles are broadly used for benchmarking and evaluating different solvers’ performance when running on a large set of problems. Benchmark results are generated by running a specific algorithm \( \nu \) (from a group of algorithms \( \mathcal{V} \) under consideration) for each problem \( p \) from a benchmark set \( \mathcal{P} \) and recording the performance measure of interest. Such measures could be, e.g., the number of function evaluations, the computation time, the number of iterations, or the memory used. In our case, we used a number of function evaluations and the execution (computation) time criteria.

Performance profiles assess the overall performance of solvers using a performance ratio \( (\lambda_{p,\nu}) \)

\[
\lambda_{p,\nu} = \frac{t_{\nu,p}}{\min\{t_{\nu,p} : p \in \mathcal{P}\}}
\]

where \( t_{\nu,p} > 0 \) is the number of functions evaluations (or the execution time) required to solve problem \( p \) by the algorithm \( \nu \) and \( \min\{t_{\nu,p} : \nu \in \mathcal{V}\} \) is the smallest number of function evaluations (or the lowest execution time) by an algorithm on this problem. Then, the performance profile \( (\chi_{\delta}(\beta)) \) of an algorithm \( \nu \in \mathcal{V} \) is given by the cumulative distribution

\[ \int_{0}^{\beta} \chi_{\delta}(\beta) d\beta \]
Table 3: The performance of algorithms from DG0 using the number of function evaluations ($f_{eval}$), the total execution time in seconds ($time$), and the total number of iterations ($iter$) criteria on a set of box-constrained problems (from DIRECT1b) when $\epsilon_{pc} = 10^{-2}$ value in stopping condition (16) was used.

function for the performance ratio

$$\chi_\alpha(\beta) = \frac{1}{|P|} \text{size}\{p \in P : \lambda_{p,o} \leq \beta\}, \quad \beta \geq 1, \quad (18)$$

where $|P|$ is the cardinality of $P$. Thus, $\chi_\alpha(\beta)$ is the probability for an algorithm $o \in \mathcal{V}$ that a performance ratio $\lambda_{p,o}$ for each $p \in P$ is within a factor of the best possible ratio.

The performance profiles seek to capture how well the certain algorithm $o$ performs compared to other algorithms from $\mathcal{V}$ on the set of problems from $P$. In particular, $\chi_\alpha(1)$ gives the fraction of the problems in $P$ for which algorithm $o$ is the winner, i.e., the best according to the $\lambda_{p,o}$ criterion. In general, algorithms with high values for $\chi_\alpha(\beta)$ are preferable [17, 79].

4.1. Comparison of DIRECT-type algorithms for box constrained optimization

In Table 3, we summarize experimental results when $\epsilon_{pc} = 10^{-2}$ was used. The smallest number of unsolved problems is achieved using DIRECT-GL (3/81), while the second-best algorithm is BIRMEN (5/81), a hybrid version and enriched with the local search subroutine fmincon. In column ‘Avg. # local searches’ we report the average number of local searches performed by each hybridized algorithm. Hybridization with fmincon allows solving more problems compared to, e.g., the globally biased version Gb-BIRECT (12/81). Among traditional DIRECT-type algorithms, the second-best position share BIRECT and DIRECT-G. Both algorithms failed to solve (10/81) problems.

Furthermore, hybridization allows significantly reducing the total number of function evaluations (see Average results and Median results column). Among the traditional algorithms, the BIRECT and PLOR were the most effective algorithms based on the median number of function evaluations (see Median results column). However, for PLOR, such performance needs to be interpreted correctly. As PLOR restricts POH set to only two hyper-rectangles per iteration, a lower number of function evaluations are required to get closer to the solution for simpler (low-dimensional) problems. However, looking at the average number of function evaluations, even restricted to the simplest subset of problems (see Average results ($n \leq 4$)), PLOR performance is among the worst. PLOR has failed on a larger number of simpler test problems compared to other approaches. In contrast, DIRECT-GL is only in a ninth-place based on the median number of function evaluation criteria, but is the only algorithm that solves all simpler ($n \leq 4$) problems and is the best performing algorithm, including hybridized versions. Moreover, on average, DIRECT-GL required approximately
84% percent fewer evaluations of the objective function than the second-best, BIRMIN algorithm, among all traditional DIRECT-type algorithms on a class of more challenging problems (see Average results \( n \geq 5 \)). Not surprisingly, within this class, the hybridized BIRMIN algorithm delivers the best average performance. Compared with the second-best DIRECT-type algorithm, DIRECT-GL, BIRMIN requires approximately 32% percent fewer evaluations.

Looking at the execution time (time), DIRECT-GL and Aggressive DIRECT are the best among all traditional algorithms. Aggressive DIRECT does not have a traditional POH selection procedure. Instead, the algorithm selects at least one candidate from each group of different hyper-rectangles. Therefore, the number of selected hyper-rectangles per iteration is larger, especially for higher dimensionality test problems. Consequently, the number of iterations (iter.) using Aggressive DIRECT is among the smallest. In overall, DIRECT-GL showed the most promising among all tested traditional DIRECT-type algorithms, while simplicial partition-based algorithms (various DISIMPL-C variations) are not appealing for box-constrained problems.

The performance profiles on the interval \([1, 10000]\) of all algorithms are shown in Fig. 5. Again, the performance profiles confirm that the hybridized algorithms (enriched with fmincon procedure) are more efficient than traditional DIRECT-type approaches. The DIRMIN algorithm has the most wins and can solve about 35% of problems with the highest efficiency, i.e., using the minor function evaluation (see the left side of Fig. 5). Nevertheless, when the performance ratio increases, the DIRECT-GL algorithm starts outperforming all algorithms, including hybridized ones. Although DIRECT-GL delivers the best overall performance (based on function evaluations), performance profiles in Fig. 5 reveal that other DIRECT-type extensions (PLOR, DIRECT-l, BIRECT, Gb-BIRECT) have more wins than DIRECT-GL. Moreover, they can solve about 60% of problems (mainly lower conditionality) using a smaller number of function evaluations. The two-step-based selection strategy in DIRECT-GL selects a more extensive set of POH. While for more straightforward problems, this is detrimental, but it helps locate a global solution faster to find it with higher accuracy. In terms of execution time (see the right-hand side in Fig. 5), DIRECT-GL is the fastest among all the traditional DIRECT-type algorithms (excluding hybridized).

As some algorithms have integrated schemes to help speed up the refinement of solutions with higher accuracy, we tested them when the solution with much higher precision \( \varepsilon_{pe} = 10^{-8} \) is sought. In Table 4, we summarize our experimental findings. First, the number of failed problems is much higher when higher accuracy is needed (see the Failed column in Tables 3 and 4). The smallest...
number of unsolved problems is achieved using DIRECT-GL (6/81), where all failed test problems belong to \((n \geq 5)\) class. The DIRECT-GL algorithm turns out to be more efficient even than hybrid methods. Overall, DIRECT-GL required approximately 16% fewer function evaluations and took 29% less time than the second-best algorithm, BIRMIN (see \textbf{Average results} column in Table 4). However, BIRMIN algorithm has the best median value (see \textbf{Median results} column), therefore solving at least half of the problems with the best performance. Finally, let us stress the inefficiency of the original DIRECT algorithm (DIRECT and glbSolve implementations). As the median value is more than 2,000,000, this means that DIRECT failed more than half of the test problems to solve.

Finally, the performance profiles on the interval \([1, 10000]\) of the selected algorithms are shown in Fig. 6. Once again, the performance profiles confirm that the hybridized algorithms are more efficient than traditional DIRECT-type approaches. However, when the performance ratio increases, DIRECT-GL outperforms all algorithms, including BIRMIN, in function evaluations and execution time.

### 4.2. Comparison of DIRECT-type algorithms for constrained global optimization

The comparison presented in this section was carried out using 80 global optimization test problems with various constraints. In DIRECT-1b, 35 test problems contain linear constraints, 39 problems have nonlinear constraints where 5 of them include equality constraints. All necessary details about the test problems are given in Appendix A, Table 12. We used the same stopping condition in these experimental investigations as in the previous ones, and the value \(\epsilon_{pe} = 10^{-8}\).

Let us stress that 5 of the test problems contain equality constraints, which we transform into inequality constraints as follows:

\[
    h(x) = 0 \rightarrow |h(x) - \epsilon_h| \leq 0
\]

where \(\epsilon_h > 0\) is a small tolerance for equality constraints. In our experiments, it was set to \(10^{-8}\).

#### 4.2.1. Test results on problems with hidden constraints

In the first part, we compared DIRECT-type versions devoted to problems with hidden constraints. In this experimental part, we have used all constrained test problems. However, we assume that any information about the constraint functions is unavailable. In the experimental investigation, the hidden search area \((D^{\text{hidden}})\) is defined as

\[
    D^{\text{hidden}} = \{x \in D : g(x) \leq 0, h(x) = 0\}
\]

Still, this information is unavailable for the tested algorithms and used only to determine whether a certain point is feasible or not. Obtained experimental results are summarized in the upper part of Table 5 (see ‘Performance of DIRECT-type algorithms for problems with hidden constraints’). First, let us note that 13 out of the 80 test problems contain complex constraints leading to a tiny feasible region. As the algorithms within this class do not use any information about constraint functions, none of the tested algorithms could find a single feasible point for any of these 13 test problems.

The best among all DIRECT-type algorithms for problems with hidden constraints is DIRECT-GLh (was unable to solve (18/80)), while the second-best is DIRECT-NAS (failed to solve (29/80)). The median number of function evaluations (see \(f_{\text{eval}}\) in \textbf{Median results}) is similar for both. Still, DIRECT-GLh is the best, primarily based on the number of iterations (\(\text{iter.}\)) and the execution time: DIRECT-GLh took around 4.5 times fewer iterations and approximately 33 times less execution time than the second best DIRECT-NAS, algorithm. The speed is the essential factor differentiating DIRECT-GLh from DIRECT-NAS.

For an extra subdividing step-based subDIRECT-Barrier, the user must define how often this step is activated. Unfortunately, the authors in [53] did not make any sensitivity analysis and guidance of this. In our experiments, we start the subdividing step at \(s_{\text{ub}}^{k}, k = 1, 2, \ldots\) iterations. We tested three different values for the variable \(\text{sub}^{k}\), i.e., \(\text{sub} = 2, 3\) and 5. Our
Figure 6: Performance profiles on [1, 10000] of selected DIRECT-type algorithms from DGO toolbox on the whole set of box-constrained optimization problems from D10lib when $\epsilon_{po} = 10^{-4}$ value in (16) was used.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Failed</th>
<th>Average results</th>
<th>Average results (Nonlin. constr.)</th>
<th>Average results (Lin. constr.)</th>
<th>Median results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>f&lt;sub&gt;eval&lt;/sub&gt;</td>
<td>time</td>
<td>iter</td>
<td>f&lt;sub&gt;eval&lt;/sub&gt;</td>
<td>time</td>
<td>iter</td>
</tr>
<tr>
<td>DIRECT-MIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-Barrier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subDIRECT-Barrier</td>
<td>sub = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subDIRECT-Barrier</td>
<td>sub = 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subDIRECT-Barrier</td>
<td>sub = 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-GL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-L&lt;sub&gt;1&lt;/sub&gt;</td>
<td>$\gamma = 10$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-L&lt;sub&gt;1&lt;/sub&gt;</td>
<td>$\gamma = 10^2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-L&lt;sub&gt;1&lt;/sub&gt;</td>
<td>$\gamma = 10^3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECT-L&lt;sub&gt;1&lt;/sub&gt;</td>
<td>$\gamma = 10^4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on function evaluations

- Performance of DIRECT-type algorithms for problems with hidden constraints
- Performance of DIRECT-type algorithms for generally constrained optimization problems
- Performance of DIRECT-type algorithms devoted for problems with linear constraints only

Table 5: The performance of DIRECT-type algorithms for problems with hidden constraints based on the number of function evaluations ($f_{eval}$), the total execution time in seconds (time), and the total number of iterations (iter.) criteria

experience showed that an extra subdividing step combined with a traditional barrier approach based on subDIRECT-Barrier did not significantly improve performance (based on the number of Failed problems and the Average results) over the original DIRECT-Barrier. The most obvious difference is that, on average, subDIRECT-Barrier subdivides much more POH per iteration because of an extra subdividing step, leading to a smaller number of iterations (iter.) and the execution time. subDIRECT-Barrier algorithm suffers solving larger dimension and problems where $L_{hidden}$ contains nonlinear constraints (see Average results (Nonlin. constr.)
proportion of solved problems vs. \( \beta \) for DIRECT-type algorithms on problems with linear constraints.

4.2.2. Test results on problems with general constraints

Better performance of DIRECT-type algorithms can be expected when the constraint function information is known. Let us note that all DIRECT-type algorithms considered in the previous section are included in this analysis. The new experimental results are added in the middle part of Table 5 (see ‘Performance of DIRECT-type algorithms for generally constrained optimization problems’). First, let us note that the recently proposed DIRECT-GLh and DIRECT-GLce algorithms of this class can much more successfully (compared to the algorithms for problems with hidden) solve problems contain complex and tiny feasible regions. The best average results among traditional DIRECT-type algorithms were obtained using the DIRECT-GLce (failed to solve 7/80). Interestingly, the performance based on the number of failed problems using DIRECT-GLc is worse, but based on the median results, it outperforms DIRECT-GLce quite clearly. This means that DIRECT-GLc is very effective while solving simpler problems, but the effectiveness drops when solving more complicated problems, e.g., higher dimensionality and nonlinear constraints.

While solving optimization problems with general constraints, the solution point is often located on the feasible region's boundaries. The common problem of several DIRECT-type algorithms is that hyper-rectangles with infeasible midpoints situated closely to the edges of feasibility are penalized with large values, resulting in a low probability of being elected POH. In this way, these algorithms converge very slowly.

Among all approaches within this class, DIRECT-GLce-min is the only hybridized version. Again, incorporating the local minimization procedure into DIRECT-GLce improves the performance significantly, e.g., it reduces the overall number of function evaluations approximately 7 times. Moreover, DIRECT-GLce-min fails to solve only one test problem.

Finally, in Figs. 8 and 9, the comparative performance of DIRECT-type algorithms (except hybridized DIRECT-GLce-min) using the performance profiles tool is demonstrated. Performance profiles confirm the same trends, i.e., the overall advantage of methods incorporating constraint information versus designed explicitly for problems with hidden constraints.

4.2.3. Test results on problems with linear constraints

In the final part, we test the performance of DIRECT-type algorithms on problems with linear constraints. We consider all previously tested algorithms and two specifically designed simplicial partitions based Lc-DISMPL-V and Lc-DISMPL-C [61]. The main advantage of using simplices is that they can cover a feasible region defined by linear constraints. Thus any infeasible areas are not involved in the search. Moreover, for most problems from DIRECTlib, the solution is located at the intersection of linear constraints. Therefore Lc-DISMPL-V finds it in the early or even in the first iteration. The performance profiles (see Fig. 10) reveal the overall effectiveness of the Lc-DISMPL-V algorithm for such problems with linear constraints. Nevertheless, the efficiency of simplicial partition-based algorithms suffers from the problem dimension. Three greater dimensionality problems with linear constraints were unsolved by the Lc-DISMPL-V algorithm (see the bottom part of Table 5, ‘Performance of DIRECT-type algorithms devoted for problems with linear constraints only’). Moreover, simplicial partition-based implementations are pretty slow. For example, on average, DIRECT-GLce is approximately 307 times faster than Lc-DISMPL-V. Moreover, DIRECT-GLce solved all test problems with linear constraints.

5. DGO performance on engineering problems

In this section, the algorithms from DGO were tested on eleven engineering design problems: tension/compression spring, three-bar truss, NASA speed reducer, pressure vessel, welded beam, and six different versions of the
The general nonlinear regression problem is box-constrained, while the others involve different types of constraints. Only the most promising algorithms (based on the results from Section 4) were considered. As the global minimums are known for all these engineering problems, we used the same stopping rule in the previous experiments. A detailed description of all engineering problems and mathematical formulations is given in Appendix B.

5.1. Tension/compression spring design problem

Here, we consider the tension-compression string design problem. This problem aims to minimize the string weight under the constraints on deflection, shear stress, surge frequency, and limits on the outside diameter. A detailed description of the practical problem can be found in [36], while in Appendix B.1 we give a short description and mathematical formulation.

A comparison of found solutions and performance metrics by the most promising algorithms from DGO is shown in Table 6. Four considered algorithms were able to solve this problem. Based on the number of function evaluation criteria, a slightly unexpectedly DIRECT-NAS was significantly better than other algorithms. The surprise comes from the fact that the algorithm does not use any information about the constraint functions. However, the DIRECT-NAS algorithm was approximate 4 times slower than the second-best method DIRECT-GLce.

Table 6: Performance of the most promising DIRECT-type algorithms from DGO on a tension/compression design problem

<table>
<thead>
<tr>
<th>Algorithm (input)</th>
<th>Iterations</th>
<th>$f_{val}$</th>
<th>Time (s)</th>
<th>$f_{val}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT-NAS</td>
<td>297</td>
<td>17,659</td>
<td>77.46</td>
<td>0.012680</td>
</tr>
<tr>
<td>DIRECT-Barrier</td>
<td>40,478</td>
<td>2,000,000</td>
<td>2,465.81</td>
<td>0.012867</td>
</tr>
<tr>
<td>subDIRECT-Barrier (sub = 2)</td>
<td>15,626</td>
<td>2,000,000</td>
<td>722.79</td>
<td>0.012867</td>
</tr>
<tr>
<td>subDIRECT-Barrier (sub = 5)</td>
<td>700</td>
<td>2,000,000</td>
<td>214.81</td>
<td>0.012683</td>
</tr>
<tr>
<td>DIRECT-GLh</td>
<td>675</td>
<td>423,209</td>
<td>47.53</td>
<td>0.012680</td>
</tr>
<tr>
<td>DIRECT-GLc</td>
<td>675</td>
<td>423,209</td>
<td>47.53</td>
<td>0.012680</td>
</tr>
<tr>
<td>DIRECT-GLc-min</td>
<td>624</td>
<td>178,115</td>
<td>20.45</td>
<td>0.012680</td>
</tr>
<tr>
<td>DIRECT-L1 ($p = 10^3$)</td>
<td>33,571</td>
<td>2,000,000</td>
<td>3,041.00</td>
<td>0.012867</td>
</tr>
</tbody>
</table>

5.2. Three-bar truss design problem

Here, we consider the three-bar truss design problem. The goal is to minimize the volume subject to stress...
constraints. The detailed description of the problem is given in [67], while in Appendix B.2, we provide a brief description and mathematical formulation.

A comparison of found solutions and performance metrics is shown in Table 7. Here, hybridized DIRECT-GLce-min was the most efficient optimizer. However, none of the algorithms had any difficulty solving this problem, and they found the solution in less than one-second time.

![Performance profiles on [1,200] of DIRECT-type algorithms for problems with general and hidden constraints on the subset of constrained optimization test problems containing linear constraints.](image)

5.3. NASA speed reducer design problem

Here we consider the NASA speed reducer design problem. The goal is to minimize the overall weight subject to constraints on the gear teeth’ bending stress, surface stress, transverse deflection of the shaft, and stresses in the shafts. A detailed description of the problem can be found in [67], while in Appendix B.3, we provide a short description and mathematical formulation.

A comparison of the found solutions and performance metrics is shown in Table 8. Only two algorithms (DIRECT-GLce and DIRECT-GLce-min) were able to tackle this problem. Again, the hybridized DIRECT-GLce-min algorithm showed the best performance. Note that the found solution with DIRECT-L1 is better than the best know value \( f_{\text{min}} \). However, the reported solution point is outside the feasible region and violates a few constraints.

![Proportion of solved problems](image)

Table 8: Performance of the most promising DIRECT-type algorithms from DGO on a NASA speed reducer design problem

5.4. Pressure vessel design problem

In this subsection, we consider a pressure vessel design problem, and the goal is to minimize the total cost of the material, forming, and to weld a cylindrical vessel. A detailed description of the problem can be found in [36], while in Appendix B.4, we provide a short description and mathematical formulation.

A comparison of the found solutions and performance metrics is shown in Table 9. Four algorithms solved this problem: DIRECT-NAS, DIRECT-GLh, DIRECT-GLce, and DIRECT-GLce-min, and hybridized DIRECT-GLce-min was the most efficient optimizer again. DIRECT-NAS is the best performing and outperformed the second-best by approximately 1.8 times, among traditional DIRECT-type algorithms. However, the DIRECT-NAS algorithm was about 26 times slower than the second-best method (DIRECT-GLh). As in the previous case, the DIRECT-L1 returned a better than the best know value \( f_{\text{min}} \) but the solution point lies outside the feasible region again.

5.5. Welded beam design problem

The fifth engineering problem is the welded beam design. Here, the goal is to minimize a welded beam for a minimum cost, subject to seven constraints. The detailed description is presented in [50, 51], while in Appendix B.5, we provide a short description and mathematical formulation.
The number of samples is considered challenging, especially with the increase in mathematical formulation. The problem is multimodal and while in Appendix B.6, we provide a short description and description of the problem can be found in [24, 56, 73], damped sinusoids to a series of observations. The detailed problem is considered in the form of fitting a sum of 5.6. General nonlinear regression problem

Once again, the two different values, and sinusoids ((correspond to , respectively)) among traditional DIRECT-type algorithms, but significantly suffered based on the execution time.

Again, the DIRECT–NAS algorithm showed the best performance (based on the total number of function evaluations) among traditional DIRECT-type algorithms, but significantly suffered based on the execution time.

Table 10: Performance of the most promising DIRECT-type algorithms from DGO on a welded beam design problem

A comparison of the algorithms is shown in Table 10. In total, five algorithms were able to solve the problem, and once again, the DIRECT–GLc-min was the most efficient one. Again, the DIRECT–NAS algorithm showed the best performance (based on the total number of function evaluations) among traditional DIRECT-type algorithms, but significantly suffered based on the execution time.

Motivated by the promising performance, we plan to extend this work to facilitate the broader adoption of DGO. We plan to include newly appearing promising DIRECT-type algorithms within this toolbox continuously. Another direction is extending the developed algorithms using a hybrid CPU-GPU scheme. Finally, we will consider advanced data structures for better data organization and reduced communication overhead.

Table 10: Performance of the most promising DIRECT-type algorithms from DGO on a welded beam design problem

6. Conclusion

This paper has introduced a new open-source DIRECT-type MATLAB toolbox (DGO) for derivative-free global optimization. The new toolbox combines various state-of-the-art DIRECT-type algorithms for the global solution of box-constrained, generally-constrained, and optimization problems with hidden constraints. All algorithms were implemented using two different data structures: static and dynamic. Additionally, several parallel schemes were adopted to promising algorithms. Furthermore, an online test library DGOlib, containing 119 global optimization test and engineering problems, has been presented.

The performance of various algorithms within DGO has been investigated via a detailed numerical study using the test problems from DGOlib. A further 11 examples of the use of the DGO for engineering design optimization have been investigated. The results demonstrate the promising performance of DGO in tackling these challenging problems. We also gave guidance on which algorithms to use for certain types of optimization problems.

Motivated by the promising performance, we plan to extend this work to facilitate the broader adoption of DGO. We plan to include newly appearing promising DIRECT-type algorithms within this toolbox continuously. Another direction is extending the developed algorithms using a hybrid CPU-GPU scheme. Finally, we will consider advanced data structures for better data organization and reduced communication overhead.

Source code statement

All implemented DIRECT-type algorithms (DGO toolbox) are available at the GitHub repository: https://github.com/blockchain-group/DGO, and can be used under GPL version 3.

Data statement

Data underlying this article (DGOlib) can be accessed on Zenodo at https://dx.doi.org/10.5281/zenodo.1218908, and used under the Creative Commons Attribution license.

References


### Table 11: Performance of the most promising DIRECT-type algorithms from DGO on nonlinear regression design problem

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iteration</th>
<th>$f_{	ext{best}}$</th>
<th>Time(s)</th>
<th>$f_{\text{best}}$</th>
<th>Iteration</th>
<th>$f_{	ext{best}}$</th>
<th>Time(s)</th>
<th>$f_{\text{best}}$</th>
<th>Iteration</th>
<th>$f_{	ext{best}}$</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggressive DIRECT</td>
<td>35</td>
<td>5,061</td>
<td>0.61</td>
<td>0.000008</td>
<td>311</td>
<td>$2 \times 10^{6}$</td>
<td>227.87</td>
<td>0.005848</td>
<td>213</td>
<td>$2 \times 10^{6}$</td>
<td>201.36</td>
</tr>
<tr>
<td>DIRECT-GL</td>
<td>27</td>
<td>1,617</td>
<td>0.12</td>
<td>0.000021</td>
<td>184</td>
<td>62,965</td>
<td>2.80</td>
<td>0.000099</td>
<td>652</td>
<td>543,207</td>
<td>35.01</td>
</tr>
<tr>
<td>DIRECT</td>
<td>32</td>
<td>501</td>
<td>0.24</td>
<td>0.000024</td>
<td>9,328</td>
<td>$2 \times 10^{6}$</td>
<td>1,125.17</td>
<td>0.001228</td>
<td>3,186</td>
<td>$2 \times 10^{6}$</td>
<td>553.10</td>
</tr>
<tr>
<td>FLOR</td>
<td>35</td>
<td>305</td>
<td>0.04</td>
<td>0.000001</td>
<td>203,19</td>
<td>$2 \times 10^{6}$</td>
<td>7,540.74</td>
<td>0.018320</td>
<td>93,457</td>
<td>$2 \times 10^{6}$</td>
<td>2,590.26</td>
</tr>
<tr>
<td>glbSolve</td>
<td>32</td>
<td>501</td>
<td>0.24</td>
<td>0.000021</td>
<td>2,805</td>
<td>1,978,935</td>
<td>762.10</td>
<td>0.000097</td>
<td>1,415</td>
<td>$2 \times 10^{6}$</td>
<td>453.64</td>
</tr>
<tr>
<td>DIRECT</td>
<td>58</td>
<td>616</td>
<td>0.43</td>
<td>0.000094</td>
<td>8,729</td>
<td>471,930</td>
<td>438.66</td>
<td>0.000096</td>
<td>36,991</td>
<td>$2 \times 10^{6}$</td>
<td>7,780.35</td>
</tr>
<tr>
<td>BIRMEN</td>
<td>5</td>
<td>101</td>
<td>0.03</td>
<td>1.04x10^{-13}</td>
<td>15</td>
<td>316</td>
<td>0.03</td>
<td>3.82x10^{-7}</td>
<td>85,458</td>
<td>722,243</td>
<td>5.915.73</td>
</tr>
<tr>
<td>DISIMPL-V</td>
<td>151</td>
<td>3,024</td>
<td>3.19</td>
<td>0.000086</td>
<td>178,562</td>
<td>43,215.10</td>
<td>0.014483</td>
<td>2</td>
<td>$2 \times 10^{6}$</td>
<td>43,000.00</td>
<td></td>
</tr>
</tbody>
</table>

N/A – not available


A. **DIRECTlib library**

A summary of all optimization problems in **DIRECTlib** and their properties is given in Table 12. The first column denotes the problem type, and the second contains the problem name. The third column contains the source of the problem. The fourth through eighth columns \((n, g, h, a, D)\) has the dimension \((n)\), number of inequality \((g)\), and equality \((h)\) constraints, number of active constraints \((a)\), and optimization domain \((D)\), respectively. Finally, the last column contains the best known optimal solution \((f^*)\).}

---

### References


### Table 12: Key characteristics of the DIRECT11b [80] test problems for box-constrained global optimization

<table>
<thead>
<tr>
<th>Problem</th>
<th>Source</th>
<th>Problem properties</th>
<th>$f^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ackley</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Beale</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bohachevsky 1</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bohachevsky 2</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bohachevsky 3</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Booth</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-1.0000</td>
</tr>
<tr>
<td>Branin</td>
<td>[30, 16, 84]</td>
<td>$n = 0$</td>
<td>-959.646</td>
</tr>
<tr>
<td>Bukin</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Csendes</td>
<td>[11]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Colville</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Cross-in-Tray</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>-1.0000</td>
</tr>
<tr>
<td>Dixon-Price</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-959.646</td>
</tr>
<tr>
<td>Drop-Wave</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Eason</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Eggholder</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Goldstein &amp; Price</td>
<td>[30, 16, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Griewank</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Hartman 3</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-3.8627</td>
</tr>
<tr>
<td>Hartman 6</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-3.3223</td>
</tr>
<tr>
<td>Holder Table</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>-19.2085</td>
</tr>
<tr>
<td>Hump</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-1.0316</td>
</tr>
<tr>
<td>Langerman</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>-4.5558</td>
</tr>
<tr>
<td>Levy</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Matyas</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>McCormick</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>-1.9132</td>
</tr>
<tr>
<td>Michalewicz</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-1.8013</td>
</tr>
<tr>
<td>Michalewicz</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-4.6876</td>
</tr>
<tr>
<td>Michalewicz</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-9.6601</td>
</tr>
<tr>
<td>Perm(β = 0.5)</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Perm(p = 10)</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Powell 4</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Power Sum</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Qing</td>
<td>[11]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-5.62</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>[30, 16, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Rotated Ellipsoid</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Schwenkel</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Shekel(m = 5)</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Shekel(m = 7)</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Shekel(m = 10)</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Shekel(m = 15)</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Shubert</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sphaler</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sphere</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Strybinski-Tang</td>
<td>[11]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sum of Powers</td>
<td>[84]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Trid</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-3.686</td>
</tr>
<tr>
<td>Trid</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-50.000</td>
</tr>
<tr>
<td>Zakharov</td>
<td>[30, 84]</td>
<td>$n = 0$</td>
<td>-210.000</td>
</tr>
<tr>
<td><strong>LC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bunnag 1</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bunnag 2</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bunnag 3</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bunnag 4</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bunnag 5</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bunnag 6</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Bunnag 7</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>G01</td>
<td>[83]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Genocop 9</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Genocop 10</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Genocop 11</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 1</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 2</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 3</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 4</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 5</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 6</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>Horst 7</td>
<td>[31]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut21</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut21 mod</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut24</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut35</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut36</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut42</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut55</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut64</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>hut76</td>
<td>[85]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
<tr>
<td>P14</td>
<td>[4]</td>
<td>$n = 0$</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Continued on next page
Table 12 Continued from previous page

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Name</th>
<th>Source</th>
<th>Problem properties</th>
<th>( f^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>s224</td>
<td>[85]</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s231</td>
<td>[85]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s232</td>
<td>[85]</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>s250</td>
<td>[85]</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s251</td>
<td>[85]</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>zecevic2</td>
<td>[85]</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>zecevic3</td>
<td>[85]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>zecevic4</td>
<td>[85]</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GC</td>
<td>circle</td>
<td>[85]</td>
<td>3</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>G02</td>
<td>[83]</td>
<td>20</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G03</td>
<td>[83]</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G04</td>
<td>[83]</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>G05</td>
<td>[83]</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G06</td>
<td>[83]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>G07</td>
<td>[83]</td>
<td>10</td>
<td>8</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>G08</td>
<td>[83]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G09</td>
<td>[83]</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>G10</td>
<td>[83]</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>G11</td>
<td>[83]</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G12</td>
<td>[83]</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G13</td>
<td>[83]</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G16</td>
<td>[83]</td>
<td>5</td>
<td>38</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>G18</td>
<td>[83]</td>
<td>9</td>
<td>13</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>G19</td>
<td>[83]</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G24</td>
<td>[83]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Goldstein &amp; Price*</td>
<td>[53]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Gomez</td>
<td>[4]</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Himmelblau</td>
<td>[6]</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P04</td>
<td>[83]</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>P02a</td>
<td>[8]</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>P02b</td>
<td>[8]</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>P02c</td>
<td>[8]</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>P02d</td>
<td>[8]</td>
<td>10</td>
<td>12</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>P03a</td>
<td>[4]</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>P03b</td>
<td>[4]</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P04</td>
<td>[4]</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P05</td>
<td>[4]</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P06</td>
<td>[4]</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P07</td>
<td>[4]</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P08</td>
<td>[4]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P09</td>
<td>[4]</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P10</td>
<td>[4]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P11</td>
<td>[4]</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P12</td>
<td>[4]</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P13</td>
<td>[4]</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P15</td>
<td>[4]</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>P16</td>
<td>[4]</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s365mod</td>
<td>[85]</td>
<td>7</td>
<td>9</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>T1</td>
<td>[19]</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>[19]</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
B. The mathematical formulation of engineering problems

B.1. Tension/compression spring design problem

The design variables of the tension/compression spring design problem [36] are the number of the wire diameter \( x_1 \), the winding diameter \( x_2 \), and active coils of the spring \( x_3 \). The objective function and the mechanical constraints are given by:

\[
\begin{align*}
\min f(x) &= x_1^2 x_2 (x_3 + 2) \\
\text{s.t. } g_1(x) &= 1 - \frac{x_3^2 x_1}{71875 x_1^4} \leq 0, \\
g_2(x) &= \frac{x_2 (4x_3 - x_1)}{12566 x_1^2} + \frac{2.46}{12566 x_1^2} - 1 \leq 0, \\
g_3(x) &= 1 - \frac{140.54 x_1}{x_3 x_2^2} \leq 0, \\
g_4(x) &= \frac{x_1 + x_2}{1.5} - 1 \leq 0
\end{align*}
\]

where \( 0.05 \leq x_1 \leq 0.2, 0.25 \leq x_2 \leq 1.3, 2 \leq x_3 \leq 15 \). The best known solution \( x^* = (0.05170517, 0.35710042, 11.28120672) \), where \( f(x^*) = 0.01267931 \). Two of the constraint functions are active (\( g_1 \) and \( g_2 \)).

B.2. Three-bar truss design problem

The three-bar truss design problem [67] has two design variables and three constraints. The optimization problem is formulated as follows:

\[
\begin{align*}
\min f(x) &= 100(2\sqrt{x_1} + x_2) \\
\text{s.t. } g_1(x) &= \sqrt{2x_1 + x_2} - 2 - 2 \leq 0, \\
g_2(x) &= \sqrt{\frac{x_1^2}{2} + 2x_1 x_2} - 2 \leq 0, \\
g_3(x) &= \frac{1}{x_1 + \sqrt{2x_2}} - 2 \leq 0
\end{align*}
\]

where \( 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1 \). The best known solution \( x^* = (0.78867512, 0.40824832) \), where \( f(x^*) = 263.89584535 \). One of the constraint functions is active (\( g_1 \)).

B.3. NASA speed reducer design problem

The design variables of the NASA speed reducer design problem [67] are the face width \( x_1 \), the module of teeth \( x_2 \), the number of teeth on the pinion \( x_3 \), the length of the first shaft between the bearings \( x_4 \), the distance of the second shaft between the bearings \( x_5 \), the diameter of the first shaft \( x_6 \), and, finally, the width of the second shaft \( x_7 \). The optimization problem is formulated as follows:

\[
\begin{align*}
\min f(x) &= 0.7854 x_1^2 (3.3333 x_2^2 + 14.9334 x_3 - 43.0934) \\
&- 1.508 x_1 (x_6^2 + x_7^2) + 7.4777 (x_6^2 + x_7^2) \\
&+ 0.7854 (x_4^2 + x_5^2)
\end{align*}
\]

\[
\begin{align*}
\text{s.t. } g_1(x) &= \frac{27}{x_1 x_3} - 1 \leq 0, \\
g_2(x) &= \frac{397.5}{x_1 x_3^2} - 1 \leq 0, \\
g_3(x) &= \frac{1.93 x_4^3}{x_2 x_3 x_5^3} - 1 \leq 0, \\
g_4(x) &= \frac{\left( \frac{745x_4}{x_2 x_3} \right)^2 + 16.9 \times 10^6}{110 x_5^3} - 1 \leq 0, \\
g_5(x) &= \frac{\left( \frac{745x_5}{x_2 x_3} \right)^2 + 157.5 \times 10^6}{85 x_4^3} - 1 \leq 0, \\
g_6(x) &= \frac{x_2 x_3}{40} - 1 \leq 0, \\
g_7(x) &= \frac{5 x_2}{x_1} - 1 \leq 0, \\
g_8(x) &= \frac{1.5 x_6 + 1.9}{x_4} - 1 \leq 0, \\
g_9(x) &= \frac{1.1 x_7 + 1.9}{x_5} - 1 \leq 0
\end{align*}
\]

where \( 2.6 \leq x_1 \leq 3.6, 0.7 \leq x_2 \leq 0.8, 17 \leq x_3 \leq 28, 7.3 \leq x_4 \leq 8.3, 7.8 \leq x_5 \leq 8.3, 2.9 \leq x_6 \leq 3.9, 5 \leq x_7 \leq 5.5 \). The best known solution \( x^* = (3.5, 0.7, 17, 7.3, 7.8, 3.35021468, 5.28668323) \), where \( f(x^*) = 2996.34817613 \). Three constraints are active (\( g_5, g_6 \) and \( g_8 \)).

B.4. Pressure vessel design problem

There are four design variables in the pressure vessel design problem [36](in inches): the thickness of the pressure vessel \( x_1 \), the thickness of the head \( x_2 \), the inner radius of the vessel \( x_3 \), and the length of the cylindrical component \( x_4 \). The optimization problem is formulated as follows:

\[
\begin{align*}
\min f(x) &= 0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2 + 3.1661 x_4^2 x_4 + 19.84 x_3^2 x_3 \\
\text{s.t. } g_1(x) &= -x_1 + 0.0193 x_3 \leq 0, \\
g_2(x) &= -x_2 + 0.00954 x_3 \leq 0, \\
g_3(x) &= -\pi x_2^2 x_4 - \frac{4}{3} \pi x_3^3 + 1296000 \leq 0, \\
g_4(x) &= x_4 - 240 \leq 0, \\
g_5(x) &= 1.1 - x_1 \leq 0, \\
g_6(x) &= 0.6 - x_2 \leq 0
\end{align*}
\]

where \( 1 \leq x_1 \leq 1.375, 0.625 \leq x_2 \leq 1, 25 \leq x_3 \leq 150, 25 \leq x_4 \leq 240 \). The best known solution \( x^* = (1.1, 0.625, 56.99481866, 51.00125165) \), where \( f(x^*) = 7163.73957163 \). Three constraints are active (\( g_1, g_3 \) and \( g_5 \)).

B.5. Welded beam design problem

The welded beam design problem [50, 51] is to design a welded beam at minimum cost, subject to some constraints

\[
\begin{align*}
\text{min } f(x) &= 0.8 x_1^2 (x_6^2 + x_7^2) + 7.4777 (x_6^2 + x_7^2) \\
&+ 0.7854 (x_4^2 + x_5^2)
\end{align*}
\]

\[
\begin{align*}
\text{s.t. } g_1(x) &= \frac{27}{x_1 x_3} - 1 \leq 0, \\
g_2(x) &= \frac{397.5}{x_1 x_3^2} - 1 \leq 0, \\
g_3(x) &= \frac{1.93 x_4^3}{x_2 x_3 x_5^3} - 1 \leq 0, \\
g_4(x) &= \frac{\left( \frac{745x_4}{x_2 x_3} \right)^2 + 16.9 \times 10^6}{110 x_5^3} - 1 \leq 0, \\
g_5(x) &= \frac{\left( \frac{745x_5}{x_2 x_3} \right)^2 + 157.5 \times 10^6}{85 x_4^3} - 1 \leq 0, \\
g_6(x) &= \frac{x_2 x_3}{40} - 1 \leq 0, \\
g_7(x) &= \frac{5 x_2}{x_1} - 1 \leq 0, \\
g_8(x) &= \frac{1.5 x_6 + 1.9}{x_4} - 1 \leq 0, \\
g_9(x) &= \frac{1.1 x_7 + 1.9}{x_5} - 1 \leq 0
\end{align*}
\]
The objective is to find a minimum fabrication cost. Considering the four design variables and constraints of shear stress τ, bending stress in the beam σ, buckling load on the bar P, and end deflection on the beam δ. The optimization model is summarized in the following equation:

\[
\begin{align*}
\min f(x) &= 1.10471x_1^2x_2 + 0.04811x_3x_4(14 + x_2) \\
\text{s.t. } g_1(x) &= (x_1 - 13600) \leq 0, \\
g_2(x) &= (x_1 - 3 \times 10^4) \leq 0, \\
g_3(x) &= x_1 - x_4 \leq 0, \\
g_4(x) &= 0.10471x_1^2 + 0.04811x_3x_4(14 + x_2) - 5 \leq 0, \\
g_5(x) &= \delta(x) - 0.25 \leq 0, \\
g_6(x) &= P - P_c(x) \leq 0, \\
g_7(x) &= 0.125 - x_1 \leq 0,
\end{align*}
\]

with:

\[
\begin{align*}
\tau(x) &= \sqrt{(\tau^1)^2 + (\tau^2)^2} / R + (\tau^2)^2, \\
\tau^1 &= \frac{P}{\sqrt{2x_1x_e}}, \tau^2 = \frac{MR}{J}, M = P(L + \frac{x_2}{2}), \\
R &= \sqrt{\frac{x_3^2}{4} + \frac{1}{4}x_1x_3^2}, \sigma(x) = \frac{6PL}{x_4x_3^2}, \\
J &= 2(\sqrt{2x_1x_2} \frac{x_3}{12} \frac{1}{4}(x_1x_3^2)), \delta(x) = \frac{4PL^3}{Ex_4x_3^3}, \\
P_c &= \frac{4.013E \frac{x_3^2x_4^6}{36}}{L^2}(1 - \frac{x_3}{2L} \sqrt{\frac{E}{4G}}), \\
P &= 6000, L = 14, E = 3 \times 10^7, G = 12 \times 10^6,
\end{align*}
\]

where

\[
-1 \leq x_3(q-1) + 1 \leq 0, \ 0 \leq x_3(q-1) + 2, 3(q-1) + 2 \leq 1, q = 1...s.
\]

is non-positive damping coefficients, \( a \) is frequencies, and \( \theta \) is phases of the sinusoids (hereafter, \( d_q \in [-1,0], \omega_q \in [0,1], \theta_q \in [0,1], q = 1...s \)). For signal approximation, the parameter \( x \) of the problem is determined to fit best the real-valued signal values observed in the uniformly distributed time moments \( t = 1, 2, ..., T \). The best known solution \( f(x^*) = 0 \), where \( x^* = (-0.2, 0.4, 0.3) \), for \( n = 3 \), \( x^* = (-0.2, 0.4, 0.3, -0.3, 0.3, 0.1) \), for \( n = 6 \), and \( x^* = (-0.4, 0.6, 0.2, -0.3, 0.3, 0.1, -0.2, 0.4, 0.3) \), for \( n = 9 \).

**B.6. General nonlinear regression problem**

Parameter estimation in the general nonlinear regression model [24, 56, 73] can be reduced to solving the minimization problem:

\[
\begin{align*}
\min f(x) &= \sum_{t=1}^{T}(\kappa(t) - \phi(x, t))^2 \\
\text{with:}
\end{align*}
\]

\[
\begin{align*}
\sum_{q=1}^{s} \kappa(t) &= e^{d_q} \sin(2\pi t \omega_q + \theta_q), \\
\sum_{q=1}^{s} \phi(x, t) &= e^{x_3(q-1) + 1} \sin(2\pi tx_3(q-1) + 2 + x_3(q-1) + 3)
\end{align*}
\]