Computational and Applied Mathematics (2021) 40:176
https://doi.org/10.1007/s40314-021-01562-y

DEFT-FUNNEL: an open-source global optimization solver for constrained grey-box and black-box problems

Phillipe R. Sampaio[1]

Received: 19 August 2020 / Revised: 11 June 2021 / Accepted: 12 June 2021 / Published online: 28 June 2021

Abstract
The fast-growing need for grey-box and black-box optimization methods for constrained global optimization problems in fields such as medicine, chemistry, engineering and artificial intelligence, has led to the development of new efficient algorithms for finding the best possible solution. In this work, we present DEFT-FUNNEL, an open-source global optimization algorithm for general constrained grey-box and black-box problems that belongs to the class of trust-region sequential quadratic optimization algorithms. Polynomial interpolation models are used as surrogates for the black-box functions and a clustering-based multistart strategy is applied for searching for the global minima. Numerical experiments show that DEFT-FUNNEL compares favorably with state-of-the-art methods on two sets of benchmark problems: one set containing problems where every function is a black box and another set with problems where some of the functions and their derivatives are known to the solver. The code as well as the test sets used for experiments are available at the Github repository http://github.com/phrsampaio/deft-funnel.

Keywords Global optimization · Constrained nonlinear optimization · Black-box optimization · Grey-box optimization · Derivative-free optimization · Simulation-based optimization

Mathematics Subject Classification 90-04 · 90C56 · 90C26

1 Introduction

The proposed solver searches for a global minimum of the optimization problem

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{s.t.:} & l^c \leq c(x) \leq u^c, \\ & l^h \leq h(x) \leq u^h, \\ & l^x \leq x \leq u^x, \end{cases} \tag{1}$$

Communicated by Ernesto G. Birgin.

✉ Phillipe R. Sampaio
sampaio.phillipe@gmail.com

[1] Veolia Research and Innovation, Maisons-Laffitte, France

where $f : \mathbb{R}^n \to \Re$ might be a black box, $c : \mathbb{R}^n \to \mathbb{R}^q$ are black-box constraint functions and $h : \mathbb{R}^n \to \mathbb{R}^l$ are white-box constraint functions, i.e. their analytical expressions as well as their derivatives are available. The vectors l^c, l^h, u^c and u^h are lower and upper bounds on the constraints values $c(x)$ and $h(x)$, while l^x and u^x are bounds on the x variables, with $l^c \in (\mathbb{R} \cup -\infty)^q$, $l^h \in (\mathbb{R} \cup -\infty)^l$, $u^c \in (\mathbb{R} \cup \infty)^q$, $u^h \in (\mathbb{R} \cup \infty)^l$, $l^x \in (\mathbb{R} \cup -\infty)^n$ and $u^x \in (\mathbb{R} \cup \infty)^n$. We also address the case where there are no white-box constraint functions h. Finally, we assume that the bound constraints are unrelaxable, i.e. feasibility must be maintained throughout the iterations, while the other general constraints are relaxable. The definition of relaxable constraint employed in this paper follows the one proposed by Digabel and Wild (2015), i.e. a relaxable constraint is a constraint that does not need to be satisfied in order to obtain meaningful outputs from the simulations in order to compute the objective and the constraints.

When at least one of the functions in (1) has a closed form (i.e. either the objective function is a white box or there is at least one white-box constraint function in the problem), it is said to be a grey-box problem. If no information about the functions is given at all, which means that the objective function is a black box and that there are no white-box constraints, the problem is known as a black-box problem. Both grey-box and black-box optimization belong to the field of derivative-free optimization (DFO) (Audet and Hare 2017; Conn et al. 2009), where the derivatives of the functions are not available. DFO problems are encountered in real-life applications in various fields such as engineering, medicine, science and artificial intelligence. The black boxes are often the result of an expensive simulation or a proprietary code, in which case automatic differentiation (Griewank 2003; Griewank and Walther 2008) is not applicable.

Many optimizations methods have been developed for finding stationary points or local minima of (1) when both the objective and the constraints functions are black boxes (e.g., Amaioua et al. 2018; Audet et al. 2015; Bueno et al. 2013; Echebest et al. 2017; Lewis and Torczon 2002; Powell 1994; Sampaio and Toint 2015, 2016). However, a local minimum is not enough sometimes and so one needs to search for a global minimum (Floudas et al. 1999; Floudas 2000). A reduced number of methods have been proposed to find a global minimum of black-box problems with nonlinear constraints (see, for instance, Boukouvala et al. 2017; Jones et al. 1998; Regis and Shoemaker 2005; Regis 2011, 2014; Regis and Shoemaker 2007). Moreover, many global optimization methods for constrained black-box problems proposed in the literature or used in industry are unavailable to the public and are not open source. We refer the reader to the survey papers by Rios and Sahinidis (2013) and by Larson et al. (2019) and to the textbooks by Conn et al. (2009) and by Audet and Hare (2017) for a comprehensive review on DFO algorithms for different types of problems.

In the case of constrained grey-box problems, especially those found in industrial applications, it is a good idea to exploit the available information about the white boxes because such problems are usually hard to be solved (i.e. highly nonlinear, multimodal and with very expensive functions) and the available derivatives can be very helpful to drive the optimization solver towards local and global minima. Therefore, one would expect the solver to use any information given as input in order to attain the global minimum as fast as possible. Unfortunately, even less global optimization solvers exist for such problems today. A common approach of optimization researchers, engineers and practioners is to consider all the functions as black boxes and to use a black-box optimization algorithm to solve the problem. Two of the few methods that exploit the available information are ARGONAUT (Boukouvala et al. 2017) and a trust-region two-phase algorithm proposed by Bajaj et al. (2018). In ARGONAUT, the black-box functions are replaced by surrogate models and a global optimization algorithm is used to solve the problem to global optimality in order to

find a lower bound while a local optimization algorithm is applied with different starting points to find the upper bounds. The surrogate models are updated only after the resolution of the problem by using the function values of the global minimum and local minima found by the solver within a clustering procedure that defines new sample points. After updating the models, the problem is solved again with the updated models and this process is repeated until convergence is declared. In the two-phase algorithm described by Bajaj et al. (2018), radial basis functions (RBF) are used as surrogate models for the black-box functions. Moreover, as in DEFT-FUNNEL, the self-correcting geometry approach proposed by (Scheinberg and Toint 2010) is applied to the management of the interpolation set. The algorithm of Bajaj et al. (2018) is composed of a feasibility phase, where the goal is to find a feasible point, and an optimization phase, where the feasible point found in the first phase is used as a starting point to find a global minimum.

The algorithm from Bajaj et al. (2018) is the one sharing more elements in common with DEFT-FUNNEL. However, these two methods differ since DEFT-FUNNEL combines a multistart strategy with a sequential quadratic optimization (SQO) algorithm in order to find a global minimum while the algorithm from Bajaj et al. (2018) applies a global optimization solver. Furthermore, DEFT-FUNNEL employs polynomial models rather than RBF models since the former is well suited for the SQO algorithm used in its local search. Despite the good performance of the algorithms proposed in Boukouvala et al. (2017) and Bajaj et al. (2018), neither is freely available or open source.

Contributions. This paper proposes a new global optimization solver for general constrained grey-box and black-box problems written in Matlab (MATLAB 2015b) that exploits any provided white-box functions given as inputs and that employs surrogate models built from polynomial interpolation in a trust-region-based SQO algorithm. Differently from the ARGONAUT approach, the surrogate models are updated during the optimization process as soon as new information from the evaluation of the functions at the iterates becomes available. Furthermore, the proposed solver, named DEFT-FUNNEL, is open source and freely available at the Github repository http://github.com/phrsampaio/deft-funnel. It is based on the works by Sampaio and Toint (2015, 2016) and it extends their original DFO algorithm to grey-box problems and to the search for a global minimum. As its previous versions, it does not require feasible starting points. To our knowledge, DEFT-FUNNEL is the first open-source global optimization solver for general constrained grey-box and black-box problems that exploits the derivative information available from the white-box functions. It is also the first one of the class of trust-funnel algorithms (Gould and Toint 2010) to be used in the search for global minima in both derivative-based and derivative-free optimization.

This paper serves also as the first release of the DEFT-FUNNEL code. Notice also that some modifications and additions have been made to the local search SQO algorithm with respect to the one presented in Sampaio and Toint (2016). In particular, some changes were done in the condition for the normal step calculation, in the criticality step and in the maintenance of the interpolation set, all of them being described in due course. Furthermore, we have also added a second-order correction step.

The extension to global optimization is based on the multi-level single linkage (MLSL) method (Kan and Timmer 1987a, b), a well-known stochastic multistart strategy that combines random sampling, a clustering-based approach for the selection of the starting points and local searches in order to identify all local minima under specific conditions. In DEFT-FUNNEL, it is used for selecting the starting points of the local searches done with the trust-funnel SQO algorithm.

Organization. The outline of this paper is as follows. Section 2 introduces the MLSL method while in Sect. 3 the DEFT-FUNNEL solver is presented in detail. In Sect. 4, numerical

results on a set of benchmark problems for global optimization are shown and the performance of DEFT-FUNNEL is compared with those of other state-of-the-art algorithms in a black-box setting. Moreover, numerical results on a set of grey-box problems are also analyzed. Finally, some conclusions about the proposed solver are drawn in Sect. 5.

Notation. Unless otherwise specified, the norm $\|\cdot\|$ is the standard Euclidean norm. Given any vector $x \in \mathbb{R}^n$, we denote its i-th component by $[x]_i$. We define $[x]^+ = \max(0, x)$ where the max operation is done componentwise. We let $\mathcal{B}(z; \Delta)$ denote the closed Euclidian ball centered at z, with radius $\Delta > 0$. Given any set $\mathcal{A}$, $|\mathcal{A}|$ denotes the cardinality of $\mathcal{A}$. By $\mathcal{P}_n^d$, we mean the space of all polynomials of degree at most d in $\mathbb{R}^n$. The Greek symbol π is used for the mathematical constant, also referred to as Archimedes'constant, as well as for defining the optimality measures of the normal and tangent subproblems, π_k^v and π_k^f, respectively — both cases are explicitly stated for avoiding confusion. Finally, given any subspace $\mathcal{S}$, we denote its dimension by $dim(\mathcal{S})$.

2 Multi-level single linkage

The MLSL method (Kan and Timmer 1987a, b) is a stochastic multistart strategy originally designed for bound-constrained global optimization problems as below

$$\begin{cases} \min \quad f(x) \\ \text{s.t.: } x \in \Omega, \end{cases} \tag{2}$$

where $\Omega \subseteq \mathbb{R}^n$ is a convex, compact set containing global minima in its interior and is defined by lower and upper bounds. It was later extended to problems with general constraints by Sendín et al. (2009). As in most of stochastic multistart methods, MLSL consists of a global phase, where random points are sampled from a probabilistic distribution, and a local phase, where selected points from the global phase are used as starting points for local searches. MLSL aims at avoiding unnecessary and costly local searches that culminate in the same local minimum. To achieve this goal, sample points are drawn from an uniform distribution in the global phase and then a local search procedure is applied to each of them except if there is another sample point within a critical distance with smaller objective function value or if the current point is a previously detected local minimum. The method is fully described in Algorithm 2.1.

Algorithm 2.1: MLSL

$\mathcal{L}^* = \emptyset$
for $k = 1$ to $\ldots$ **do**
  Generate N random points uniformly distributed in Ω.
  Rank the sample points by increasing value of f.
  for $i = 1$ to γkN **do** (where $0 < \gamma \leq 1$)
    if $x_i \notin \mathcal{L}^*$ and $\nexists x_j$ such that $\|x_j - x_i\| \leq r_k$ and $f(x_j) < f(x_i)$ **then**
      $\mathcal{L}^* = \mathcal{L}^* \cup LocalSearch(x_i)$
    end if
  end for
end for
return the best local minimum found in $\mathcal{L}^*$

The ranking of the sample points at line 4 is optional and does not affect the convergence properties of the algorithm. It is mainly useful when $\gamma < 1$, which means that preference is given to the γkN starting points having the lowest objective function values, as one might believe that they are more likely to be closer to a local minimum that those points with higher objective function values.

The critical distance r_k is defined as

$$r_k \stackrel{\text{def}}{=} \pi^{-1/2} \left(\Gamma\left(1 + \frac{n}{2}\right) m(\Omega) \frac{\sigma \log kN}{kN} \right)^{1/n}, \tag{3}$$

for some $\sigma > 0$ and where π here is the Archimedes' constant, Γ is the gamma function defined by $\Gamma(x) = \int_0^\infty e^{-t} t^{x-1}\, dt$ and $m(\Omega)$ is the Lebesgue measure of the set Ω. The details of this formula are elaborated on Kan and Timmer (1987a, b) and are not part of the scope of this paper. The method is centred on the idea of exploring the region of attraction of all local minima, which is formally defined below.

Definition 1 Given a local search procedure $\mathcal{P}$, we define a region of attraction $\mathcal{R}(x^*)$ in Ω to be the set of all points in Ω starting from which $\mathcal{P}$ will arrive at x^*.

The ideal multistart method is the one that runs a local search only once at the region of attraction of every local minimum. However, two types of errors might occur in practice (Locatelli 1998):

- **Error 1.** The same local minimum x^* has been found after applying local search to two or more points belonging to the same region of attraction of x^*.
- **Error 2.** The region of attraction of a local minimum x^* contains at least one sampled point, but local search has never been applied to points in this region.

In Kan and Timmer (1987a, b), the authors demonstrate the following theoretical properties of MLSL that are directly linked to the errors above:

- **Property 1**. [Theorem 8 in Kan and Timmer (1987a) and Theorem 1 in Kan and Timmer (1987b)] If $\sigma > 4$ in (3), then, even if the sampling continues forever, the total number of local searches ever started by MLSL is finite with probability 1.
- **Property 2**. [Theorem 12 in Kan and Timmer (1987a) and Theorem 2 in Kan and Timmer (1987b)] If r_k tends to 0 with increasing k, then any local minimum x^* will be found within a finite number of iterations with probability 1.

Property 1 states that the number of possible occurrences of Error 1 is finite while Property 2 says that Error 2 never happens. Due to its strong theoretical results and good practical performance, MLSL became one of the most reliable and popular multistart methods of late.

Finally, we note that MLSL has already been applied into global black-box optimization problems with general constraints before (Armstrong and Favorite 2016; Sendín et al. 2009). In particular, Armstrong and Favorite (2016) proposes the method MLSL-MADS, which integrates MLSL with a mesh adaptive direct search (MADS) method (Audet and Dennis 2006) to find multiple local minima of an inverse transport problem involving black-box functions.

3 The DEFT-FUNNEL solver

First, the function $z : \mathbb{R}^n \to \mathbb{R}^{q+l}$ is defined as $z(x) \stackrel{\text{def}}{=} (c(x), h(x))$, i.e. it includes all the constraint functions of the original problem (1). Then, by defining $f(x, s) \stackrel{\text{def}}{=} f(x)$ and $z(x, s) \stackrel{\text{def}}{=} z(x) - s$, the problem (1) is rewritten as

$$\begin{cases} \min\limits_{(x,s)} & f(x,s) \\ \text{s.t.:} & z(x,s) = 0, \\ & l^s \leq s \leq u^s, \\ & l^x \leq x \leq u^x, \end{cases} \tag{4}$$

where $s \in \mathbb{R}^{q+l}$ are slack variables and $l^s \in (\mathbb{R} \cup -\infty)^{q+l}$ and $u^s \in (\mathbb{R} \cup \infty)^{q+l}$ are the lower and upper bounds of the modified problem with $l^s = \left[l^c \; l^h\right]^T$ and $u^s = \left[u^c \; u^h\right]^T$. We highlight that the rewriting of the original problem (1) as (4) is done within the solver and that the user does not need to interfere.

DEFT-FUNNEL is composed of a global search and a local search that are combined to solve the problem (4). In the next two sections, we elaborate on each of these search steps.

3.1 Global search

As mentioned previously, the global search in DEFT-FUNNEL relies on the MLSL multistart strategy. However, different from the Algorithm 2.1, the global search in DEFT-FUNNEL makes use of a merit function Φ rather than the objective function f in order to decide which starting points are selected for the local search. Moreover, the sampling of N random points is done within the set $\{x \in \mathbb{R}^n \mid l^x \leq x \leq u^x\}$. If there exists an index i such that l_i^x is not defined by the user, the negative value -10 is used by default during the sampling. Analogously, if there exists an index i such that u_i^x is not defined by the user, the positive value 10 is used by default.

The global search is implemented in the function `deft_funnel_multistart`, which is called by typing the following line in the Matlab command window:

```
[best_sol, best_fval, best_indicators, total_eval, nb_local_searches, fL
    ] = deft_funnel_multistart(@f, @c, @h, @dev_f, @dev_h, n, nb_cons_c,
     nb_cons_h)
```

The inputs and outputs of `deft_funnel_multistart` are detailed in Table 1.

The merit function Φ employed in DEFT-FUNNEL is the well-known ℓ_1 penalty functon which is defined as follows:

$$\Phi(x) \overset{\text{def}}{=} f(x) + \lambda \sum_{i=1}^{m} \left(\left[z_i(x) - \left[u^s\right]_i\right]^+ + \left[\left[l^s\right]_i - z_i(x)\right]^+ \right), \tag{5}$$

where $m = q + l$ is the total number of constraints excluding bound constraints on x, λ is the penalty parameter and $z_i(x) = (c_i(x), h_i(x))$, $i = 1, \ldots, m$. One of the advantages of this penalty function over others is that it is exact, that is, for sufficiently large values of λ, the local minimum of Φ subject only to the bound constraints on the x variables is also the local minimum of the original constrained problem (1). Note that, although Φ is nondifferentiable, it is only used in the global search for selecting the starting points for the local searches.

The *LocalSearch* algorithm at line 7 is started by calling the function `deft_funnel` whose inputs and outputs are given in the next section.

3.2 Local search

Before describing in detail each component of the local search, we give a global view of its full algorithm in what follows. The algorithm is based on the one described in Sampaio and Toint (2016). It is a trust-region SQO method that makes use of a funnel upper bound $v_k^{\max}$

Table 1 Inputs and outputs of the function `deft_funnel_multistart`

	Name	Description
Mandatory Inputs	f	Function handle of the objective function
	c	Function handle of the black-box constraints if any or an empty array
	h	Function handle of the white-box constraints if any or an empty array
	dev_f	Function handle of the derivatives of f if white box or an empty array
	dev_h	Function handle of the derivatives of h if any or an empty array
	n	Number of decision variables
	nb_cons_c	Number of black-box constraints (bound constraints not included)
	nb_cons_h	Number of white-box constraints (bound constraints not included)
Optional Inputs	lsbounds	Vector of lower bounds for the constraints
	usbounds	Vector of upper bounds for the constraints
	lxbounds	Vector of lower bounds for the x variables
	uxbounds	Vector of upper bounds for the x variables
	maxeval	Max. number of evaluations (default: 5000*n)
	maxeval_ls	Max. number of evaluations per local search (default: maxeval*0.7)
	whichmodel	Approach to build the surrogate models
	f_global_optimum	Known objective function value of the global optimum
Outputs	best_sol	Best feasible solution found
	best_fval	Objective function value of "best_sol"
	best_indicators	Indicators of "best_sol"
	total_eval	Number of evaluations used
	nb_local_searches	Number of local searches done
	fL	Objective function values of all local minima found

on the infeasibility of the iterates in order to ensure convergence. At each iteration k, the iterate is defined by the point (x_k, s_k), where $x_k \in \mathcal{Y}_k$, with $\mathcal{Y}_k$ being the interpolation set at iteration k. Every iterate satisfies the following bound constraints:

$$l^s \leq s_k \leq u^s, \tag{6}$$
$$l^x \leq x_k \leq u^x. \tag{7}$$

At each iteration, the local search algorithm checks if there are (nearly) active bounds at x_k. If there is at least one active bound and if some conditions are satisfied, the local search algorithm calls itself recursively with a new subspace built from the active bound(s) as well as a new interpolation set that belongs to the new subspace. This step is done by the SubspaceMinimization subroutine, which is detailed in Sect. 3.2.2.

Next, the subroutine CriticalityStep checks if either the trust-region size is too small or the last search direction is too small. If neither holds, it checks if both feasibility and optimality have been achieved in the current subspace. If any of these criticality conditions is satisfied,

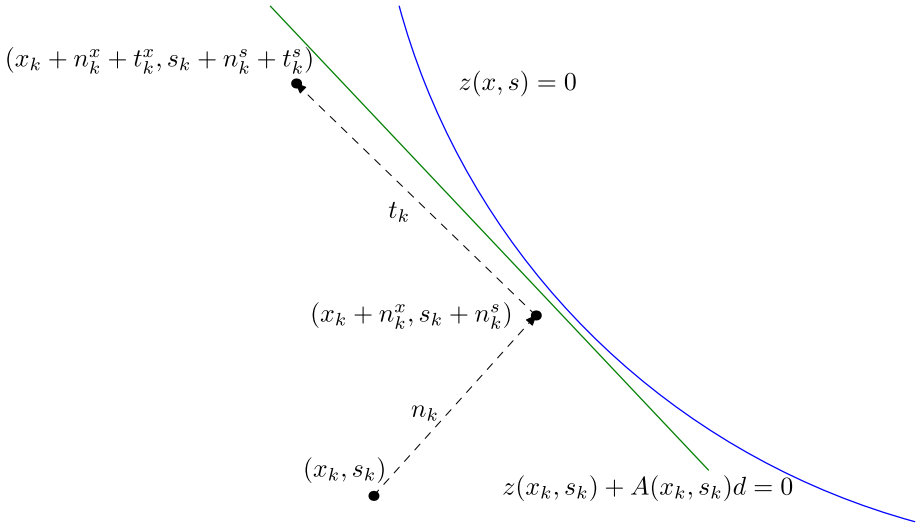


Fig. 1 Illustration of the composite step $d_k = n_k + t_k$. The normal step n_k attempts to improve feasibility by reducing the linearized constraint violation at (x_k, s_k), whereas the tangent step aims at minimizing the objective function without deteriorating the gains in feasibility obtained through the normal step. Here, $A(x_k, s_k)$ denotes the Jacobian of $z(x, s)$ at (x_k, s_k)

the local search returns the current iterate (x_k, s_k). The subroutine CriticalityStep is discussed in depth in Sect. 3.2.7.

If criticality has not been attained at (x_k, s_k), a new step $d_k \stackrel{\text{def}}{=} (d_k^x, d_k^s)^T$ is computed. Each full step of the trust-funnel algorithm is decomposed as

$$d_k = \begin{pmatrix} d_k^x \\ d_k^s \end{pmatrix} = \begin{pmatrix} n_k^x \\ n_k^s \end{pmatrix} + \begin{pmatrix} t_k^x \\ t_k^s \end{pmatrix} = n_k + t_k, \tag{8}$$

where the normal step component n_k aims to improve feasibility and the tangent step component t_k reduces the objective function model without worsening the constraint violation up to first order. This is done by requiring the tangent step to lie in the null space of the approximated Jacobian of the constraints and by requiring the predicted improvement in the objective function obtained in the tangent step to not be negligible compared to the predicted change in f resulting from the normal step. The full composite step d_k is illustrated in Fig. 1. As it is explained in the next subsections, the computation of the composite step in the proposed algorithm does not involve the function z itself but rather its surrogate model.

The step n_k is computed by the subroutine NormalStep, which considers a trust-region bound Δ_k^z whose update rule is based on the improvement on feasibility obtained at iteration k. The computation of the normal step is fully described in Sect. 3.2.3. The step t_k is computed by the subroutine TangentStep, which considers the trust-region bound $\Delta_k = \min[\Delta_k^f, \Delta_k^z]$, where Δ_k^f is a trust-region bound that is updated according to the improvement on optimality obtained at iteration k. The calculation of the tangent step is described in detail in Sect. 3.2.4.

Depending on the contributions of the current iteration in terms of optimality and feasibility, the iteration is classified into three types: μ-iteration, f-iteration and z-iteration. If $d_k = 0$, no contribution has been made to optimality or feasibility and thus iteration k is

said to be a μ-iteration. If iteration k has mainly contributed to optimality, it is said to be a f-iteration. Otherwise, it is defined as a z-iteration.

After having computed a trial point $x_k + d_k$, the algorithm proceeds by checking the iteration type and whether the iteration was successful in a sense to be defined in Sects. 3.2.6 and 3.2.6. The iterate and the trust regions are then updated correspondingly, while the interpolation set is updated by the subroutine UpdateInterpolationSet according to a self-correcting geometry scheme described in Sect. 3.3. The subroutine $f-$iteration is responsible for updating the trust regions Δ_k^f and Δ_k^z if iteration k is a f-iteration, while the subroutine $z-$iteration does the same for Δ_k^z if iteration k is a z-iteration. The details of both subroutines are given in Sects. 3.2.6 and 3.3. Finally, if $\mathcal{Y}_k$ has been modified, the surrogate models are updated to satisfy the interpolation conditions for the new set $\mathcal{Y}_{k+1}$, implying that new function evaluations are carried out for the additional point obtained at iteration k.

The complete local search algorithm is presented below, while its subroutines are fully explained in the next sections.

Algorithm 3.1: LocalSearch

0: Initialization. Choose an initial vector of Lagrange multipliers μ_{-1}, initial trust-region radii $\Delta_0^f > 0$ and $\Delta_0^z > 0$, a fixed accuracy threshold $\epsilon > 0$ for declaring convergence on subspaces with dimension smaller than n, and an initial accuracy threshold $\epsilon_0 > 0$ used for checking the criticality conditions on the full space. Define $\Delta_0 = \min[\Delta_0^f, \Delta_0^z] \leq \Delta^{\max}$. Initialize $\mathcal{Y}_0$, with $x_0 \in \mathcal{Y}_0 \subset \mathcal{B}(x_0; \Delta_0)$ and $p_{\max} \geq |\mathcal{Y}_0| \geq n+1$, where $p_{\max} = (n+1)(n+2)/2$ if underdetermined quadratic models are considered, and $p_{\max} = (n+1)(n+2)$, in case where regression models are chosen. Compute the associated models m_0^f and m_0^c around x_0 and Lagrange polynomials $\{l_{0,j}\}_{j=0}^p$. Set $k = 0$ (updated at Step 8) and $i = 0$ (updated within CriticalityStep).

1: SubspaceMinimization.

2: CriticalityStep.

3: NormalStep.

4: TangentStep.

5: Conclude a μ-iteration. If $n_k = t_k = 0$, then

5.1: set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, $\Delta_{k+1}^f = \Delta_k^f$ and $\Delta_{k+1}^z = \Delta_k^z$;

5.2: set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^z]$, $v_{k+1}^{\max} = v_k^{\max}$ and $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

6: Conclude an f-iteration. If $t_k \neq 0$ and the improvement in the objective function is significant and the infeasibility at the trial point is lower than $v_k^{\max}$, then

6.1: UpdateInterpolationSet;

6.2: f-iteration;

6.3: Set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^z]$ and $v_{k+1}^{\max} = v_k^{\max}$.

7: Conclude a z-iteration. If the iteration is neither a μ-iteration nor a f-iteration, then

7.1: UpdateInterpolationSet;

7.2: z-iteration;

7.3: Set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^z]$ and update $v_k^{\max}$ if the improvement in feasibility is significant.

8: Update the models and the Lagrange polynomials. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation models m^f_{k+1} and m^c_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$ and the associated Lagrange polynomials $\{l_{k+1,j}\}_{j=0}^{p}$. Increment k by one and go to Step 1.

The index k indicates the iteration number, which is incremented in the end of iteration, at Step 8. However, whenever a recursive call takes place within SubspaceMinimization at Step 1, the iteration index k continues to be incremented within the new subspace. This means that the current value of k is passed as input to LocalSearch and that its final value is also returned by LocalSearch.

As it is explained in Sect. 3.2.7, the index i is incremented in CriticalityStep within an inner loop that aims at ensuring that the derivative information of the models is not too different from that of the original functions. Since i and k are incremented differently, they do not equal to each other.

The local search is started inside the multistart strategy loop in the global search, but it can also be called directly by the user in order to use DEFT-FUNNEL without multistart. This is done by typing the following line at the Matlab command window:

```
[best_sol, best_fval, best_indicators, total_eval, nb_local_searches, fL
    ] = deft_funnel_multistart(@f, @c, @h, @dev_f, @dev_h, n, nb_cons_c,
     nb_cons_h)
```

The inputs and outputs of the function `deft_funnel` are detailed below in Table 2. Many other additional parameters can be set directly in the function `deft_funnel_set_parameters`. Those are related to the trust-region mechanism, to the interpolation set maintenance and to criticality step thresholds.

Once the initial interpolation set has been built using one of the methods described in the next subsection, the algorithm calls the function `deft_funnel_main`. In fact, `deft_funnel` serves only as a wrapper for the main function of the local search, `deft_funnel_main`, doing all the data preprocessing and paramaters setting needed in the initialization process. All the main steps such as the subspace minimization step, the criticality step and the computation of the new directions are part of the scope of `deft_funnel_main`.

3.2.1 Building the surrogate models

The local search algorithm starts by building an initial interpolation set either from a simplex or by drawing samples from an uniform distribution. The construction of the interpolation set is done in the function `deft_funnel_build_initial_sample_set`, which is called only once during a local search within `deft_funnel`. The choice between random sampling and simplex is done in `deft_funnel` when calling the function `deft_funnel_set_parameters`, which defines the majority of parameters of the local search. If random sampling is chosen, it checks if the resulting interpolation set is well poised and, if not, it is updated using the Algorithm 6.3 described in Chapter 6 in Conn et al. (2009), which is implemented in `deft_funnel_repair_Y`.

For the sake of simplicity, we assume henceforth that the objective function is also a black box. Let $\mathcal{Y}_0 = \{y^0, y^1, \ldots, y^p\}$ be a poised set of sample points with an initial point $x_0 \in \mathcal{Y}_0$, where p denotes the cardinality of $\mathcal{Y}_k$. As described in Sect. 3.3, p can

Table 2 Inputs and outputs of the function `deft_funnel`

	Name	Description
Mandatory inputs	f	Function handle of the objective function
	c	Function handle of the black-box constraints if any or an empty array
	h	Function handle of the white-box constraints if any or an empty array
	dev_f	Function handle of the derivatives of f if white box or an empty array
	dev_h	Function handle of the derivatives of h if any or an empty array
	x0	Starting point (no need to be feasible)
	nb_cons_c	Number of black-box constraints (bound constraints not included)
	nb_cons_h	Number of white-box constraints (bound constraints not included)
Optional inputs	lsbounds	Vector of lower bounds for the constraints
	usbounds	Vector of upper bounds for the constraints
	lxbounds	Vector of lower bounds for the x variables
	uxbounds	Vector of upper bounds for the x variables
	maxeval	Max. number of evaluations (default: 500*n)
	type_f	String 'BB' if f is a black box (default) or 'WB' otherwise
	whichmodel	Approach to build the surrogate models
Outputs	x	The best approximation found to a local minimum
	fx	The value of the objective function at x
	mu	Local estimates for the Lagrange multipliers
	indicators	Feasibility and optimality indicators
	evaluations	Number of calls to the objective function and constraints
	iterate	Info about the best point found as well as the coordinates of all past iterates
	exit_algo	Output signal (0: terminated with success; -1: terminated with errors)

increase over the iterations as the interpolation set is augmented with new trial points. The next step of our algorithm is to replace the objective function $f(x)$ and the black-box constraint functions $c(x) = (c_1(x), c_2(x), \ldots, c_q(x))$ by surrogate models $m^f(x)$ and $m^c(x) = (m^{c_1}(x), m^{c_2}(x), \ldots, m^{c_q}(x))$, respectively, built from the solution of the interpolation system

$$M(\phi, \mathcal{Y})\alpha_\phi = \Upsilon(\mathcal{Y}), \tag{9}$$

where

$$M(\phi, \mathcal{Y}) = \begin{pmatrix} \phi_0(y^0) & \phi_1(y^0) & \cdots & \phi_b(y^0) \\ \phi_0(y^1) & \phi_1(y^1) & \cdots & \phi_b(y^1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(y^p) & \phi_1(y^p) & \cdots & \phi_b(y^p) \end{pmatrix}, \quad \Upsilon(\mathcal{Y}) = \begin{pmatrix} \Upsilon(y^0) \\ \Upsilon(y^1) \\ \vdots \\ \Upsilon(y^p) \end{pmatrix},$$

where $\phi = \{\phi_0, \ldots, \phi_b\}$ is the basis of monomials in $\mathcal{P}_n^d$ and $\Upsilon(x)$ is replaced by the objective function $f(x)$, if building $m^f(x)$, or some black-box constraint function $c_j(x)$, if building m^{c_j}, for $j = 1, \ldots, q$.

If $M(\phi, \mathcal{Y})$ is square, (9) becomes an interpolation problem. If, in addition, $M(\phi, \mathcal{Y})$ is nonsingular for some basis $\phi \in \mathcal{P}_n^d$, the set $\mathcal{Y}$ is said to be poised for interpolation in $\mathbb{R}^n$. Poisedness is important since, given a function $f : \mathbb{R}^n \to \mathbb{R}$ and a poised set $\mathcal{Y}$, it follows that the interpolating polynomial m(x) exists and is unique (see, Conn et al. 2009, Lemma 3.2). In the underdetermined case, $p < b$, the set $\mathcal{Y}$ is said to be poised if $M(\phi, \mathcal{Y})$ has full row rank. As for the regression case, the set $\mathcal{Y}$ is said to be poised for polynomial least-squares regression in $\mathbb{R}^n$ if the corresponding matrix $M(\phi, \mathcal{Y})$ has full column rank for some basis $\phi \in \mathcal{P}_n^d$.

We consider underdetermined quadratic interpolation models that are fully linear and that are enhanced with curvature information along the optimization process. Since the linear system (9) is potentially underdetermined, the resulting interpolating polynomials may not be unique and so we provide to the user four approaches to construct the models $m^f(x)$ and $m^{c_j}(x)$ that can be chosen by passing a number from 1 to 4 to the input 'whichmodel': 1 - subbasis selection approach; 2 - minimum ℓ_2-norm models; 3 - minimum Frobenius norm models; and 4 - regression (recommended for noisy functions).

The subbasis selection consists in considering only $p+1$ columns of the matrix $M(\phi, \mathcal{Y})$ in the linear system (9) in order to have a square matrix, which is equivalent to choosing a subbasis $\hat{\phi}$ of ϕ with only $p+1$ elements. This is done in DEFT-FUNNEL by selecting the first $p+1$ columns. In the second approach to build the models, the minimum ℓ_2-norm solution of the system (9) is computed. In the third approach, we minimize the Frobenius norm of the Hessian of the surrogate model m(x) since it plays an important role on the error bounds for quadratic models (see Conn et al. 2009, Theorem 5.4). This is done by solving the following optimization problem in $\alpha_\phi = (\alpha_L, \alpha_Q)$, where α_L and ϕ_L are related to the linear components of the natural basis ϕ, while α_Q and ϕ_Q, to the quadratic ones:

$$\begin{aligned} &\min && \tfrac{1}{2}\|\alpha_Q\|_2^2 \\ &\text{s.t.: } && M(\phi_L, \mathcal{Y})\alpha_L + M(\phi_Q, \mathcal{Y})\alpha_Q = \Upsilon(\mathcal{Y}). \end{aligned} \tag{10}$$

Finally, regression models are built by obtaining the least-squares solution of the system (9). Further details about each approach can be found in Conn et al. (2009). In DEFT-FUNNEL, minimum ℓ_2-norm models are the default option since they obtained better performance in past numerical experiments than the others (Sampaio and Toint 2015, 2016).

If $|\mathcal{Y}_0| = n + 1$, where the initial interpolation points are affinely independent, a linear model rather than an underdetermined quadratic model is built for each function. The reason is that, despite both having error bounds that are linear in Δ for the first derivatives, the error bound for the latter includes also the norm of the model Hessian, as stated in Lemma 2.2 in Zhang et al. (2010), which makes it worse than the former.

Whenever $n+1 < |\mathcal{Y}_k| \leq (n+1)(n+2)/2 = p^{\max}$, the algorithm builds underdetermined quadratic models based on the choice of the user between the approaches described above. If regression models are considered instead, we set $p^{\max} = (n+1)(n+2)$, which means that the sample set is allowed to have twice the number of sample points required for fully quadratic interpolation models. Notice that having a number of sample points larger than the required for quadratic interpolation can also worsen the local quality of the interpolation models as the sample set could contain points that are too far from the iterate, which is not ideal for models built for local approximation.

It is also possible to choose the initial degree of the models between fully linear, quadratic with a diagonal Hessian or fully quadratic. This is done within `deft_funnel` by setting the input argument `cur_degree` in the call to `deft_funnel_set_parameters` to one of the following options: `model_size.plin`, `model_size.pdiag` or `model_size.pquad`.

The interpolation system (9) is solved using a QR factorization of the matrix $M(\phi, \mathcal{Y})$ within the function `deft_funnel_computeP`, which is called by `deft_funnel_build_models`.

In order to evaluate the error of the surrogate models and their derivatives with respect to the original functions f and c, we make use of the measure of well poisedness of $\mathcal{Y}$ given below. Notice, however, that this definition is generalized for the interpolation, minimum-norm and regression cases and that the construction of the Lagrange polynomials differs for each case.

Definition 2 Let $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$ be a poised set and $\mathcal{P}_n^d$ be a space of polynomials of degree less than or equal to d on $\mathbb{R}^n$. Let $\Lambda > 0$ and $\{\ell_0(x), \ell_1(x), \ldots, \ell_p(x)\}$ be the set of Lagrange polynomials associated with $\mathcal{Y}$. Then, the set $\mathcal{Y}$ is said to be Λ-poised in $\mathcal{B} \in \mathbb{R}^n$ for $\mathcal{P}_n^d$ if and only if

$$\max_{0 \le i \le p} \max_{x \in \mathcal{B}} |\ell_i(x)| \le \Lambda. \tag{11}$$

As it is shown in Conn et al. (2009), the error bound between at most fully quadratic models and the original functions as well as the error bound between their gradients depend linearly on the constant Λ; the smaller it is, the better the interpolation models approximate the original functions. We also note that the error bounds for undetermined quadratic interpolation models are linear in the diameter of the smallest ball containing $\mathcal{Y}$ for the first derivatives and quadratic for the function values.

The coefficients of each Lagrange polynomial are given by the solution of linear system

$$M(\phi, \hat{\mathcal{Y}})\lambda_j = e_{j+1}, \quad j = 0, \ldots, p, \tag{12}$$

where e_{j+1} is the $(j+1)$-th column of the identity matrix of order $q+1$ and $\hat{\mathcal{Y}}$ is a shifted and scaled version of $\mathcal{Y}$ that is contained in a ball of radius one centered at the origin given by

$$\hat{\mathcal{Y}} = \{0, \hat{y}^1, \ldots, \hat{y}^p\} = \{0, (y^1 - y^0)/\Delta, \ldots, (y^p - y^0)/\Delta\} \subset \mathcal{B}(0; 1), \tag{13}$$

where

$$\Delta = \Delta(\mathcal{Y}) = \max_{1 \le i \le p} \|y^i - y^0\|. \tag{14}$$

By scaling $\mathcal{Y}$, the condition number of $M(\phi, \hat{\mathcal{Y}})$, where ϕ is the basis of monomials, can be used for measuring the well poisedness of the set $\mathcal{Y}$ (see Conn et al. 2009, Chapter 3, page 48). Since the Λ-poisedness depends on the region $\mathcal{B}$ in which poisedness is considered, we also shift the scaled sample set.

When $p < b$, the set of Lagrange polynomials associated with $\mathcal{Y}$ in Definition 2 is obtained by the minimum ℓ_2-norm solution of (12). These Lagrange polynomials are an extension of the standard Lagrange polynomials of the interpolation case. On the other hand, when $p > b$, the set of regression Lagrange polynomials for $\mathcal{Y}$ is given by the least-squares solution of (12), or equivalently,

$$\min_{\lambda_j} \|M(\phi, \hat{\mathcal{Y}})\lambda_j - e_{j+1}\|^2, \quad j = 0, \ldots, p. \tag{15}$$

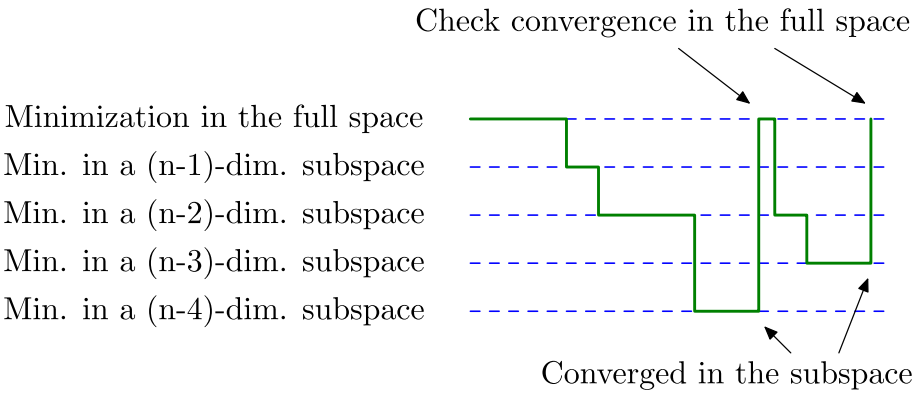


Fig. 2 Subspace minimization procedure. The algorithm calls itself recursively in the order to solve the problem in a new subspace. If convergence is attained, it goes back to check if the solution found is also optimal in the full space

Finally, since the constraint functions $c(x)$ are replaced by surrogate models $m^c(x)$ in the algorithm, we define the models $m^z(x) \stackrel{\text{def}}{=} (m^c(x), h(x))$ and $m^z(x, s) \stackrel{\text{def}}{=} m^z(x) - s$, which are those used for computing new directions.

3.2.2 Subspace minimization

In this subsection, we explain how the subspace minimization is employed in our algorithm. We define the subspace $\mathcal{S}_k$ at iteration k as

$$\mathcal{S}_k \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n \mid [x]_i = \left[l^x\right]_i \text{ for } i \in \mathcal{L}_k \text{ and } [x]_i = \left[u^x\right]_i \text{ for } i \in \mathcal{U}_k\},$$

where $\mathcal{L}_k \stackrel{\text{def}}{=} \{i \mid [x_k]_i - \left[l^x\right]_i \leq \epsilon_b\}$ and $\mathcal{U}_k \stackrel{\text{def}}{=} \{i \mid \left[u^x\right]_i - [x_k]_i \leq \epsilon_b\}$ define the index sets of (nearly) active variables at their bounds, for some small constant $\epsilon_b > 0$. If $\mathcal{L}_k \neq \emptyset$ or $\mathcal{U}_k \neq \emptyset$, a new well-poised interpolation set $\mathcal{Z}_k$ is built by the function `deft_funnel_choose_lin` in the following way. First, it builds $\mathcal{Z}_k$ by selecting all points in $\mathcal{X}_k \cap \mathcal{S}_k$ that are inside the current trust region $\mathcal{B} = \mathcal{B}(x_k; \Delta_k)$, where $\mathcal{X}_k$ is the set of all points obtained up to iteration k. Then, in order to complete the set with a total number of $n_{\mathcal{S}} + 1$ interpolation points to build a linear model in $\mathcal{S}_k$, where $n_{\mathcal{S}} = dim(\mathcal{S}_k) = n - |\mathcal{L}_k \cup \mathcal{U}_k|$, it proceeds by generating random points within $\mathcal{B} \cap \mathcal{S}_k$ and including them in $\mathcal{Z}_k$. In order to make $\mathcal{Z}_k$ a Λ-poised set for a given $\Lambda > 1$, it applies the Algorithm 6.2 in Conn et al. (2009) for improving well poisedness via Lagrange polynomials. In practice, each new interpolation point y^j whose Lagrange polynomial $\ell_j(x)$ satisfies the condition $\max_{x \in \mathcal{B} \cap \mathcal{S}_k} |\ell_j(x)| > \Lambda$ is optimally replaced by a point that maximizes the corresponding Lagrange polynomial in $\mathcal{B} \cap \mathcal{S}_k$. After a new Λ-poised interpolation set $\mathcal{Z}_k$ has been built, a recursive call is made in order to solve the problem in the new subspace $\mathcal{S}_k$ with the new set $\mathcal{Z}_k$.

If the algorithm converges in a subspace $\mathcal{S}_k$ with an optimal solution $(\tilde{x}, \tilde{s})$, it checks if the latter is also optimal for the full-space problem, in which case the algorithm stops. If not, the algorithm continues by attempting to compute a new direction in the full space. This procedure is illustrated in Fig. 2.

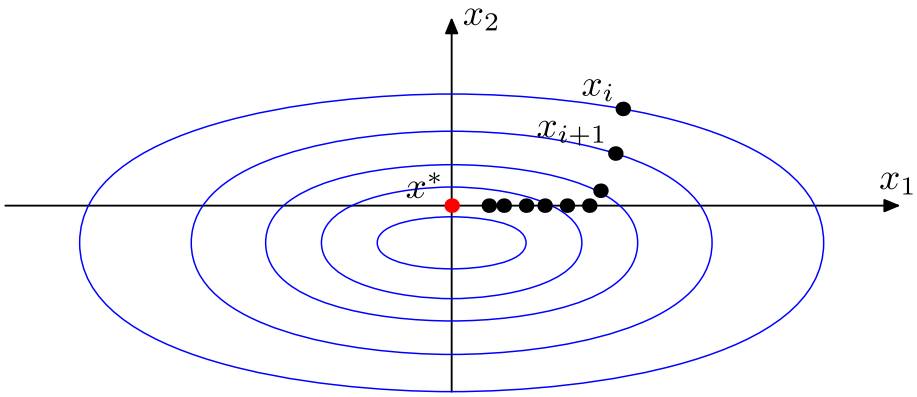


Fig. 3 Illustration of a scenario where the interpolation set becomes degenerated as the optimal solution is approached. This example considers a two-dimensional problem with the bound constraint $[x]_2 \geq 0$, which is active at the solution x^* and at the iterates close to it

The dimensionality reduction of the problem mitigates the chances of degeneration of the interpolation set when the sample points become too close to each other and often affinely dependent. Figure 3 gives an example of this scenario as the optimal solution is approached.

In order to check the criticality in the full-space problem, a full-space interpolation set of degree $n + 1$ is built in an ϵ-neighborhood around the point $x^*_{\mathcal{S}}$, which is obtained by assembling the subspace solution $\tilde{x}$ and the $|\mathcal{L}_k \cup \mathcal{U}_k|$ fixed components $[x_k]_i$, where $i \in \mathcal{L}_k \cup \mathcal{U}_k$. The models m^f_k and m^c_k are then updated and the criticality step is entered.

The complete subspace minimization step is described in Algorithm 3.2 and it is implemented in the function `deft_funnel_subspace_min`, which is called inside `deft_funnel_main`.

Algorithm 3.2: SubspaceMinimization($\hat{\mathcal{S}}$, x_k, s_k, Δ^f_k, Δ^z_k, $v^{\max}_k$)

1: Check for (nearly) active bounds at x_k and define $\mathcal{S}_k$. If there is no (nearly) active bound or if $\mathcal{S}$ has already been explored, go to Step 6. If all bounds are active, go to Step 5.
2: Build a new interpolation set $\mathcal{Z}_k$ in $\mathcal{S}_k$.
3: Call recursively LocalSearch($\mathcal{S}_k$, $\mathcal{Z}_k$, x_k, s_k, Δ^f_k, Δ^z_k, $v^{\max}_k$) and mark $\mathcal{S}_k$ as explored.
4: Let $(x^*_{\mathcal{S}}, s^*_{\mathcal{S}})$ be the solution of the subspace problem after adding the fixed components and let k be the current iteration number returned by the local search (which has been updated during the recursive call in Step 3).
5: If $dim(\hat{\mathcal{S}}) < n$, return $(x^*_{\mathcal{S}}, s^*_{\mathcal{S}})$. Otherwise, set $(x_k, s_k) = (x^*_{\mathcal{S}}, s^*_{\mathcal{S}})$, construct new set $\mathcal{Y}_k$ around x_k, build m^f_k and m^c_k and recompute π^f_{k-1} (optimality measure).
6: If $\mathcal{S}_k$ has already been explored, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, reduce the trust regions radii $\Delta^f_{k+1} = \gamma \Delta^f_k$ and $\Delta^z_{k+1} = \gamma \Delta^z_k$, set $\Delta_{k+1} = \min[\Delta^f_{k+1}, \Delta^z_{k+1}]$ and build a new poised set $\mathcal{Y}_{k+1}$ in $\mathcal{B}(x_{k+1}; \Delta_{k+1})$.

As a final commentary on this section, notice that whenever a recursive call takes place in Step 3 of Algorithm 3.2, the iteration index k continues to be incremented within the new subspace, as already mentioned in Sect. 3.2. For this reason, the current value of k is retrieved in Step 4, after LocalSearch has returned a solution. In other words, if the iteration number

equals $k = iter$ for a certain $iter$ before Step 3, it does not necessarily equals $k = iter + 1$ after Step 3.

3.2.3 The normal step

The normal step aims at reducing the constraint violation at (x, s) as defined by

$$v(x, s) \stackrel{\text{def}}{=} \tfrac{1}{2}\|z(x, s)\|^2. \tag{16}$$

To ensure that the step n_k is normal to the approximately linearized constraint $m^z(x_k, s_k) + J(x_k, s_k)n = 0$, where $J(x, s) \stackrel{\text{def}}{=} (J(x) \ -I_m)$ is the Jacobian of $m^z(x, s)$ with respect to (x, s), the matrix I_m is the $m \times m$ identity matrix and $J(x)$ is the Jacobian of $m^z(x)$ with respect to x, we require that

$$\|n_k\|_\infty \leq \kappa_n \|m^z(x_k, s_k)\|, \tag{17}$$

for some $\kappa_n \geq 1$.

The computation of n_k is done by solving the constrained linear least-squares subproblem

$$\begin{cases} \min\limits_{n=(n^x,n^s)} & \tfrac{1}{2}\|m^z(x_k, s_k) + J(x_k, s_k)n\|^2 \\ \text{s.t.:} & l^s \leq s_k + n^s \leq u^s, \\ & l^x \leq x_k + n^x \leq u^x, \\ & x_k + n^x \in \mathcal{S}_k, \\ & n \in \mathcal{N}_k, \end{cases} \tag{18}$$

where

$$\mathcal{N}_k \stackrel{\text{def}}{=} \{n \in \mathbb{R}^{n+m} \mid \|n\|_\infty \leq \min\left[\ \Delta_k^z,\ \kappa_n \|m^z(x_k, s_k)\|\ \right]\}, \tag{19}$$

for some trust-region radius $\Delta_k^z > 0$. Finally, a funnel bound $v_k^{\max}$ is imposed on the constraint violation $v_k \stackrel{\text{def}}{=} v(x_k, s_k)$ for the acceptance of new iterates to ensure the convergence towards feasibility.

We notice that, although a linear approximation of the constraints is used for calculating the normal step, the second-order information of the quadratic interpolation model $m^z(x, s)$ is still used in the SQO model employed in the tangent step problem as is shown next.

The subproblem (18) is solved within the function `deft_funnel_normal_step`, which makes use of an original active-set algorithm where the unconstrained problem is solved at each iteration in the subspace defined by the currently active bounds, themselves being determined by a projected Cauchy step. Each subspace solution is then computed using a SVD decomposition of the reduced matrix. This algorithm is implemented in the function `deft_funnel_blls` and is intended for small-scale bound-constrained linear least-squares problems.

3.2.4 The tangent step

The tangent step is a direction that improves optimality and it is computed by using a SQO model for the problem (4) after the normal step calculation. The quadratic model for the objective function is defined as

$$\psi_k((x_k, s_k) + d) \stackrel{\text{def}}{=} m^f(x_k, s_k) + \langle g_k, d\rangle + \tfrac{1}{2}\langle d, B_k d\rangle, \tag{20}$$

where $m^f(x_k, s_k) \stackrel{\text{def}}{=} m^f(x_k)$, $g_k \stackrel{\text{def}}{=} \nabla_{(x,s)} m^f(x_k, s_k)$, and B_k is the approximate Hessian of the Lagrangian function

$$\mathcal{L}(x, s, \mu, \xi^s, \tau^s, \xi^x, \tau^x) = m^f(x, s) + \langle \mu, m^z(x, s))\rangle + \langle \tau^s, s - u^s\rangle + \langle \xi^s, l^s - s\rangle + \langle \tau^x, x - u^x\rangle + \langle \xi^x, l^x - x\rangle$$

with respect to (x, s), given by

$$B_k = \begin{pmatrix} H_k + \sum_{i=1}^m [\hat{\mu}_k]_i Z_{ik} & 0 \\ 0 & 0 \end{pmatrix}, \tag{21}$$

where ξ^s and τ^s are the Lagrange multipliers associated to the lower and upper bounds, respectively, on the slack variables s, and ξ^x and τ^x are the Lagrange multipliers associated to the lower and upper bounds on the x variables. In (21), $H_k = \nabla^2_{xx} m^f(x_k, s_k)$, $Z_{ik} = \nabla^2_{xx} m^z_{ik}(x_k, s_k)$ and the vector $\hat{\mu}_k$ may be viewed as a local approximation of the Lagrange multipliers with respect to the equality constraints $m^z(x, s) = 0$.

By applying (8) into (20), we obtain

$$\psi_k((x_k, s_k) + n_k + t) = \psi_k((x_k, s_k) + n_k) + \langle g_k^N, t\rangle + \tfrac{1}{2}\langle t, B_k t\rangle, \tag{22}$$

where

$$g_k^N \stackrel{\text{def}}{=} g_k + B_k n_k. \tag{23}$$

Since (22) is a local approximation for the function $f((x_k, s_k) + n_k + t)$, a trust region with radius Δ_k^f is used for the complete step $d = n_k + t$:

$$\mathcal{T}_k \stackrel{\text{def}}{=} \{d \in \mathbb{R}^{n+m} \mid \|d\|_\infty \le \Delta_k^f\}. \tag{24}$$

Moreover, given that the normal step was also calculated using local models, it makes sense to remain in the intersection of both trust regions, which implies that

$$d_k \in \mathcal{R}_k \stackrel{\text{def}}{=} \mathcal{N}_k \cap \mathcal{T}_k \stackrel{\text{def}}{=} \{d \in \mathbb{R}^{n+m} \mid \|d\|_\infty \le \Delta_k\}, \tag{25}$$

where $\Delta_k = \min[\Delta_k^z, \Delta_k^f]$.

In order to make sure that there is still enough space left for the tangent step within $\mathcal{R}_k$, we first check if the following constraint on the normal step is satisfied:

$$\|n_k\|_\infty \le \kappa_{\mathcal{R}} \Delta_k, \tag{26}$$

for some $\kappa_{\mathcal{R}} \in (0, 1)$. If (26) holds, the tangent step is calculated by solving the following subproblem

$$\begin{cases} \min\limits_{t=(t^x, t^s)} & \langle g_k^N, t\rangle + \frac{1}{2}\langle t, B_k t\rangle \\ \text{s.t.:} & J(x_k, s_k)t = 0, \\ & l^s \le s_k + n_k^s + t^s \le u^s, \\ & l^x \le x_k + n_k^x + t^x \le u^x, \\ & x_k + n_k^x + t^x \in \mathcal{S}_k. \\ & n_k + t \in \mathcal{R}_k, \end{cases} \tag{27}$$

where we require that the new iterate $x_k + d_k^x$ belongs to subspace $\mathcal{S}_k$ and that it satisfies the bound constraints (6) and (7). In the Matlab code, the tangent step is calculated by the function `deft_funnel_tangent_step`, which in turn calls either the Matlab solver `linprog` or our implementation of the nonmonotone spectral projected gradient method (Birgin et al.

2000) in `deft_funnel_spg` to solve the subproblem (27). The choice between both solvers is based on whether $\|B_k\| \leq \epsilon$, for a small $\epsilon > 0$, in which case we assume that the problem is linear and therefore `linprog` is used. Finally, as in the original trust-funnel method of Gould and Toint (2010), inexact tangent steps are allowed.

The f-criticality measure is defined as

$$\pi_k^f \stackrel{\text{def}}{=} -\langle g_k^N, r_k \rangle, \tag{28}$$

where r_k is the projected Cauchy direction obtained by solving the linear optimization problem

$$\begin{cases} \min\limits_{r=(r^x,r^s)} & \langle g_k^N, r \rangle \\ \text{s.t.:} & J(x_k, s_k)r = 0, \\ & l^s \leq s_k + n_k^s + r^s \leq u^s, \\ & l^x \leq x_k + n_k^x + r^x \leq u^x, \\ & x_k + n_k^x + r^x \in \mathcal{S}_k. \\ & \|r\|_\infty \leq 1. \end{cases} \tag{29}$$

By definition, π_k^f measures how much decrease could be obtained locally along the projection of the negative of the approximate gradient g_k^N onto the nullspace of $J(x_k, s_k)$ intersected with the region delimited by the bound constraints on $x_k + n_k^x + r^s$ and $s_k + n_k^s + r^x$ in (29). This measure is computed in `deft_funnel_compute_optimality`, which uses `lingprog` to solve the subproblem (29).

A new local estimate of the Lagrange multipliers $(\mu_k, \xi_k^s, \tau_k^s, \xi_k^x, \tau_k^x)$ is computed by solving the following problem:

$$\begin{cases} \min\limits_{(\mu,\hat{\xi}^s,\hat{\tau}^s,\hat{\xi}^x,\hat{\tau}^x)} & \frac{1}{2}\|\mathcal{M}_k(\mu, \hat{\xi}^s, \hat{\tau}^s, \hat{\xi}^x, \hat{\tau}^x)\|^2 \\ \text{s.t.:} & \hat{\xi}^s, \hat{\tau}^s, \hat{\xi}^x, \hat{\tau}^x \geq 0, \end{cases} \tag{30}$$

where

$$\begin{aligned} \mathcal{M}_k(\mu, \hat{\xi}^s, \hat{\tau}^s, \hat{\xi}^x, \hat{\tau}^x) \stackrel{\text{def}}{=} & \begin{pmatrix} g_k^N \\ 0 \end{pmatrix} + \begin{pmatrix} J(x_k)^T \\ -I_m \end{pmatrix} \mu + \begin{pmatrix} 0 \\ I_\tau^s \end{pmatrix} \hat{\tau}^s + \begin{pmatrix} 0 \\ -I_\xi^s \end{pmatrix} \hat{\xi}^s \\ & + \begin{pmatrix} I_\tau^x \\ 0 \end{pmatrix} \hat{\tau}^x + \begin{pmatrix} -I_\xi^x \\ 0 \end{pmatrix} \hat{\xi}^x, \end{aligned}$$

the matrices I_ξ^s and I_τ^s are obtained from I_m by removing the columns whose indices are not associated to any active (lower and upper, respectively) bound at $s_k + n_k^s$, the matrices I_ξ^x and I_τ^x are obtained from the $n \times n$ identity matrix by removing the columns whose indices are not associated to any active (lower and upper, respectively) bound at $x_k + n_k^x$, and the Lagrange multipliers $(\hat{\xi}^s, \hat{\tau}^s, \hat{\xi}^x, \hat{\tau}^x)$ are those in $(\xi^s, \tau^s, \xi^x, \tau^x)$ associated to active bounds at $s_k + n_k^s$ and $x_k + n_k^x$. All the other Lagrange multipliers are set to zero.

The subproblem (30) is also solved using the active-set algorithm implemented in the function `deft_funnel_blls`.

3.2.5 Which steps to compute and retain

The algorithm computes normal and tangent steps depending on the measures of feasibility and optimality at each iteration. Differently from Sampaio and Toint (2015, 2016), where

the computation of the normal steps depends on the measure of optimality, here the normal step is computed whenever the following condition holds

$$\|z(x_k, s_k)\| > \epsilon, \tag{31}$$

for some small $\epsilon > 0$ (i.e. preference is always given to feasibility). This choice is based on the fact that, in many real-life problems with expensive functions and a small budget, one seeks to find a feasible solution as fast as possible and that a solution having a smaller objective function value than the current strategy is already enough. If (31) fails, we set $n_k = 0$.

We define a v-criticality measure that indicates how much decrease could be obtained locally along the projection of the negative gradient of the Gauss-Newton model of m^z at (x_k, s_k) onto the region delimited by the bounds as

$$\pi_k^v \stackrel{\text{def}}{=} -\langle J(x_k, s_k)^T z(x_k, s_k), b_k \rangle,$$

where the projected Cauchy step b_k is given by the solution of

$$\begin{cases} \min\limits_{b=(b^x,b^s)} & \langle J(x_k, s_k)^T z(x_k, s_k), b \rangle \\ \text{s.t.:} & l^s \leq s_k + b^s \leq u^s, \\ & l^x \leq x_k + b^x \leq u^x, \\ & x_k + b^x \in \mathcal{S}_k, \\ & \|b\|_\infty \leq 1. \end{cases} \tag{32}$$

We say that (x_k, s_k) is an infeasible stationary point if $z(x_k, s_k) \neq 0$ and $\pi_k^v = 0$, in which case the algorithm terminates.

The procedure for the calculation of the normal step is given in the algorithm below. In the code, it is implemented in the function `deft_funnel_normal_step`, which calls the algorithm in `deft_funnel_blls` in order to solve the normal step subproblem (18).

Algorithm 3.3: NormalStep(x_k, s_k, π_k^v, v_k, $v_k^{\max}$)

1: If $z(x_k, s_k) \neq 0$ and $\pi_k^v = 0$, **STOP** (infeasible stationary point).
2: If (31) holds, compute a normal step n_k by solving (18). Otherwise, set $n_k = 0$.

If the solution of (29) is $r_k = 0$, then by (28) we have $\pi_k^f = 0$, in which case we set $t_k = 0$. If the current iterate is farther from feasibility than from optimality, i.e., for a given a monotonic bounding function ω_t, the condition

$$\pi_k^f > \omega_t(\|z(x_k, s_k)\|) \tag{33}$$

fails, then we skip the tangent step computation by setting $t_k = 0$.

After the computation of the tangent step, the usefulness of the latter is evaluated by checking if the conditions

$$\|t_k\| > \kappa_{\mathcal{ZS}} \|n_k\| \tag{34}$$

and

$$\delta_k^f \stackrel{\text{def}}{=} \delta_k^{f,t} + \delta_k^{f,n} \geq \kappa_\delta \delta_k^{f,t}, \tag{35}$$

where

$$\delta_k^{f,t} \stackrel{\text{def}}{=} \psi_k((x_k, s_k) + n_k) - \psi_k((x_k, s_k) + n_k + t_k) \tag{36}$$

and

$$\delta_k^{f,n} \stackrel{\text{def}}{=} \psi_k(x_k, s_k) - \psi_k((x_k, s_k) + n_k), \tag{37}$$

are satisfied for some $\kappa_{\mathcal{ZS}} > 1$ and $\kappa_\delta \in (0, 1)$. The inequality (35) indicates that the *predicted* improvement in the objective function obtained in the tangent step is substantial compared to the *predicted* change in f resulting from the normal step. If (34) holds but (35) does not, the tangent step is not useful in the sense just discussed, and we choose to ignore it by resetting $t_k = 0$.

The tangent step procedure is stated in Algorithm 3.4 and is implemented in the function `deft_funnel_tangent_step`.

Algorithm 3.4: TangentStep(x_k, s_k, n_k)

1: If (26) holds, then
  1.1: select a vector $\hat{\mu}_k$ and define B_k as in (21);
  1.2: compute μ_k by solving (30);
  1.3: compute the modified Cauchy direction r_k by solving (29) and define π_k^f as (28);
  1.4: if (33) holds, compute a tangent step t_k by solving (27) and set $d_k = n_k + t_k$.
2: If (26) fails, set $\mu_k = \mu_{k-1}$. In this case, or if (33) fails, or if (34) holds but (35) fails, set $t_k = 0$ and $d_k = n_k$.
3: Define $(x_k^+, s_k^+) = (x_k, s_k) + d_k$.

3.2.6 Iteration types

As mentioned previously, each iteration is classified into one of three types depending on the contributions made in terms of optimality and feasibility, namely: μ-iteration, f-iteration and z-iteration. This is done by checking if some conditions hold for the trial point defined as

$$(x_k^+, s_k^+) \stackrel{\text{def}}{=} (x_k, s_k) + d_k. \tag{38}$$

3.2.6.1 μ-iteration

If $d_k = 0$, then it means that $n_k = 0$, that is, (26) holds. In this case, the Lagrange multipliers estimates are potentially the only new values that have been computed. For this reason, iteration k is said to be a μ-iteration with reference to the Lagrange multipliers μ associated to the constraints $m^z(x, s) = 0$. Notice, however, that not only new μ_k values have been computed, but all the other Lagrange multipliers $(\xi_k^s, \tau_k^s, \xi_k^x, \tau_k^x)$ as well.

In this case, we set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, $\Delta_{k+1}^f = \Delta_k^f$, $\Delta_{k+1}^z = \Delta_k^z$, $v_{k+1}^{\max} = v_k^{\max}$ and we use the new multipliers to build a new SQO model in (20).

Since null steps $d_k = 0$ might be due to the poor quality of the interpolation models, we check the Λ-poisedness in μ-iterations and attempt to improve it whenever the following condition holds

$$\Lambda\, \Delta(\mathcal{Y}_k) > \epsilon_\mu, \tag{39}$$

where

$$\Delta(\mathcal{Y}_k) \stackrel{\text{def}}{=} \max_j \|y^{k,j} - x_k\|,$$

Λ is estimated by solving the maximization problem (11) present in Definition 2 and $\epsilon_\mu > 0$. The inequality (39) gives an estimate of the error bound for the interpolation models. If (39) holds, we try to reduce the value at the left side by modifying the sample set $\mathcal{Y}_k$. Firstly, we choose a constant $\xi \in (0, 1)$ and replace all points $y^{k,j} \in \mathcal{Y}_k$ such that

$$\|y^{k,j} - x_k\| > \xi \Delta(\mathcal{Y}_k)$$

by new points $y_*^{k,j}$ that (approximately) maximize $|\ell_{j_k}(x)|$ in $\mathcal{B}(x_k; \xi\Delta(\mathcal{Y}_k))$. Then we use the Algorithm 6.3 described in Chapter 6 in Conn et al. (2009) with the smaller region $\mathcal{B}$ to improve Λ-poisedness of the new sample set. This procedure is implemented in the Matlab code by the function `deft_funnel_repair_sample_set`.

3.2.6.2 f-iteration

If iteration k has mainly contributed to optimality, it is called an f-iteration. Formally, this happens when $t_k \neq 0$, (35) holds, and

$$v(x_k^+, s_k^+) \leq v_k^{\max}. \tag{40}$$

Convergence of the algorithm towards feasibility is ensured by condition (40), which limits the constraint violation with the funnel bound.

In this case, we set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$ if

$$\rho_k^f \stackrel{\text{def}}{=} \frac{f(x_k, s_k) - f(x_k^+, s_k^+)}{\delta_k^f} \geq \eta_1, \tag{41}$$

for $\eta_1 \in (0, 1)$, and $(x_{k+1}, s_{k+1}) = (x_k, s_k)$, otherwise. Note that $\delta_k^f > 0$ (because of (36) and (35)) unless (x_k, s_k) is first-order critical, and hence condition (41) is well defined. As for the value of the funnel bound in this case, we set $v_{k+1}^{\max} = v_k^{\max}$.

Since our method can suffer from the Maratos effect (Maratos 1978), we also apply a second-order correction (see Section 15.6, in Nocedal and Wright 2006, for more details) to the normal step whenever the complete direction d_k is unsuccessful at improving optimality, i.e., whenever the condition (41) fails. As the latter might be due to the inadequate local approximation of the constraint functions, this effect may be overcome with a second-order step $\hat{n}$ that is calculated in `deft_funnel_sec_order_correction` by solving the following subproblem

$$\begin{cases} \min\limits_{\hat{n}=(\hat{n}^x,\hat{n}^s)} & \frac{1}{2}\|m^z(x_k^+, s_k^+) + J(x_k, s_k)\hat{n}\|^2 \\ \text{s.t.:} & l^s \leq s_k^+ + \hat{n}^s \leq u^s, \\ & l^x \leq x_k^+ + \hat{n}^x \leq u^x, \\ & x_k^+ + \hat{n}^x \in \mathcal{S}_k, \\ & \hat{n} \in \hat{\mathcal{N}}_k, \end{cases} \tag{42}$$

where

$$\hat{\mathcal{N}}_k \stackrel{\text{def}}{=} \{\hat{n} \in \mathbb{R}^{n+m} \mid \|\hat{n}\|_\infty \leq \min\left[\, \Delta_k^z,\ \kappa_n \|m^z(x_k^+, s_k^+)\| \,\right]\}. \tag{43}$$

After the second-order step $\hat{n}$ has been computed, the conditions

$$v((x_k^+, s_k^+) + \hat{n}) \leq v_k^{\max} \tag{44}$$

and

$$\rho_k^{fC} \stackrel{\text{def}}{=} \frac{f(x_k, s_k) - f((x_k^+, s_k^+) + \hat{n})}{\delta_k^f} \geq \eta_1 \tag{45}$$

still need to be satisfied for the f-iteration with the augmented step $(x_k^+, s_k^+) + \hat{n}$ to be successful. If (44) and (45) hold, we set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+) + \hat{n}$.

3.2.6.3 z-iteration

If iteration k is neither a μ-iteration nor a f-iteration, then it is said to be a z-iteration. This means that the major contribution of iteration k is to improve feasibility, which happens when either $t_k = 0$ or when (35) fails.

The trial point is accepted if the improvement in feasibility is comparable to its predicted value

$$\delta_k^z \stackrel{\text{def}}{=} \tfrac{1}{2}\|m^z(x_k, s_k)\|^2 - \tfrac{1}{2}\|m^z(x_k, s_k) + J(x_k, s_k)d_k\|^2,$$

and the latter is itself comparable to its predicted decrease along the normal step, that is

$$n_k \neq 0, \quad \delta_k^z \geq \kappa_{zn}\delta_k^{z,n} \quad \text{and} \quad \rho_k^z \stackrel{\text{def}}{=} \frac{v(x_k, s_k) - v(x_k^+, s_k^+)}{\delta_k^z} \geq \eta_1, \tag{46}$$

for some $\kappa_{zn} \in (0, 1)$ and where

$$\delta_k^{z,n} \stackrel{\text{def}}{=} \tfrac{1}{2}\|m^z(x_k, s_k)\|^2 - \tfrac{1}{2}\|m^z(x_k, s_k) + J(x_k, s_k)n_k\|^2. \tag{47}$$

If (46) is not satisfied, the trial point (x_k^+, s_k^+) is rejected.

Finally, the funnel bound is updated as follows

$$v_{k+1}^{\max} = \begin{cases} \max\left[\kappa_{tx1}v_k^{\max}, v(x_k^+, s_k^+) + \kappa_{tx2}(v(x_k, s_k) - v(x_k^+, s_k^+))\right] & \text{if (46) hold,} \\ v_k^{\max} & \text{otherwise,} \end{cases} \tag{48}$$

for some $\kappa_{tx1} \in (0, 1)$ and $\kappa_{tx2} \in (0, 1)$.

3.2.7 Criticality step

Two different criticality steps are employed: one for the subspaces $\mathcal{S}_k$ with $dim(\mathcal{S}_k) < n$ and one for the full space ($dim(\mathcal{S}_k) = n$). In the latter, convergence is declared whenever at least one of the following conditions is satisfied: (1) the trust-region radius Δ_k is too small, (2) the computed direction d_k is too small or (3) both feasibility and optimality have been achieved and the error between the real functions and the models is expected to be sufficiently small. As it was mentioned before, this error is directly linked to the Λ-poisedness measure given in Definition 2. In the subspace, we are less demanding and only ask that either Δ_k be very small or both feasibility and optimality have been achieved without checking the models error though.

The complete criticality step in DEFT-FUNNEL is described in the next algorithm.

Algorithm 3.5: CriticalityStep($\mathcal{S}_k$, $\mathcal{Y}_k$, π_{k-1}^f, α, β, Δ_k, d_k, ϵ, ϵ_i)

1: If $dim(\mathcal{S}_k) < 0$,

1.1: If $\Delta_k \leq \epsilon\|(x_k, s_k)\|$, return (x_k, s_k).

1.2: If $\|z(x_k, s_k)\| \leq \epsilon$ and $\pi_{k-1}^f \leq \epsilon$, return (x_k, s_k).

2: If $dim(\mathcal{S}_k) = n$,

2.1: If $\Delta_k \leq \epsilon\|(x_k, s_k)\|$ or $\|d_k\| \leq \epsilon\|(x_k, s_k)\|$, return (x_k, s_k).
2.2: Define $\hat{m}_i^f = m_k^f$, $\hat{m}_i^c = m_k^c$ and $\hat{\pi}_i^f = \pi_{k-1}^f$.
2.3: If $\|z(x_k, s_k)\| \leq \epsilon_i$ and $\hat{\pi}_i^f \leq \epsilon_i$, set $\epsilon_{i+1} = \max\left[\alpha\|z(x_k, s_k)\|, \alpha\hat{\pi}_i^f, \epsilon\right]$ for a fixed input parameter $\alpha \in (0, 1)$ and modify $\mathcal{Y}_k$ as needed to ensure it is Λ-poised in $\mathcal{B}(x_k; \epsilon_{i+1})$. If $\mathcal{Y}_k$ was modified, compute new models $\hat{m}_i^f$ and $\hat{m}_i^c$, calculate $\hat{r}_i$ and $\hat{\pi}_i^f$ and increment i by one. If $\|z(x_k, s_k)\| \leq \epsilon$ and $\hat{\pi}_i^f \leq \epsilon$, return (x_k, s_k); otherwise, start Step 2.3 again;
2.4: Set $m_k^f = \hat{m}_i^f$, $m_k^c = \hat{m}_i^c$, $\pi_{k-1}^f = \hat{\pi}_i^f$, $\Delta_k = \beta \max\left[\|z(x_k, s_k)\|, \pi_{k-1}^f\right]$, where $\beta > 0$ is a fixed input parameter and define $\vartheta_i = x_k$ if a new model has been computed.

In order to check feasibility and optimality in the full space, the algorithm employs a dynamic threshold ϵ_i that is expected to fall below the fixed accuracy threshold ϵ whenever criticality is verified in the current iterate. The initial value of ϵ_i is set in Step 0 of Algorithm 3.1 when $i = 0$. The implicit loop in Step 2.3 may be viewed as a model improvement step that aims to ensure that the derivative information of the surrogate models do not differ too much from that of the original functions. This inner iteration follows the same idea of the criticality test in Algorithm 2 of Scheinberg and Toint (2010).

Notice that the definition of ϑ_i at Step 2.4 of Algorithm 3.5 serves to indicate the point at which well-poised models m^c and m^f are known and the current trust region is larger than the region at which these models are poised. This variable is then used in the update rule of the interpolation set, as it is detailed in the next section.

3.3 Maintenance of the interpolation set and trust-region updating strategy

The management of the geometry of the interpolation set is based on the self-correcting geometry scheme proposed in Scheinberg and Toint (2010), where unsuccessful trial points are used to improve the geometry of the interpolation set. It depends on the criterion used to define successful iterations, which is passed to the algorithm through the parameter *criterion*, which depends on the iteration type (μ, f or z). For f-iterations, the inequality $\rho_k^f \geq \eta_1$ must be satisfied, whereas for z-iterations the conditions in (46) must hold.

In general, unsuccessful trial points replace other sample points in the interpolation set which maximize a combined criteria of distance and poisedness involving the trial point. However, this replacement is avoided when the iterate is a point ϑ_i (defined in CriticalityStep, Algorithm 3.5) at which well-poised models m^c and m^f are already known. Finally, the proposed solver do not make use of "dummy" interpolation points resulting from projections onto the subspaces as in Sampaio and Toint (2016). The whole procedure is described in Algorithm 3.6. As before, the maximum cardinality of the interpolation set, $p^{\max}$, is $(n + 1)(n + 2)/2$.

Algorithm 3.6: UpdateInterpolationSet($\mathcal{Y}_k$, x_k, x_k^+, Δ_k, ϑ_i, ϵ_i, ***criterion***)

1: Augment the interpolation set. If $|\mathcal{Y}_k| < p^{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$.
2: Successful iteration. If $|\mathcal{Y}_k| = p^{\max}$ and *criterion* holds, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{49}$$

3: Replace a far interpolation point. If $|\mathcal{Y}_k| = p^{\max}$, *criterion* fails, either $x_k \neq \vartheta_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| > \zeta \Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\} \tag{50}$$

is non-empty, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{51}$$

4: Replace a close interpolation point. If $|\mathcal{Y}_k| = p^{\max}$, *criterion* fails, either $x_k \neq \vartheta_i$ or $\Delta_k \leq \epsilon_i$, the set $\mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\text{def}}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| \leq \zeta \Delta \text{ and } |\ell_{k,j}(x_k^+)| > \Lambda\} \tag{52}$$

is non-empty, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$, where

$$y^{k,r} = \arg \max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{53}$$

5: No replacements. If $|\mathcal{Y}_k| = p^{\max}$, *criterion* fails and either $[x_k = \vartheta_i$ and $\Delta_k > \epsilon_i]$ or $\mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

The trust-region update strategies associated to f- and z-iterations are now described. Following the idea proposed in Gratton et al. (2011), the trust-region radii are allowed to decrease even when the interpolation set has been changed after the replacement of a far point or a close point at unsuccessful iterations. However, the number of times it can be shrunk in this case is limited to $\nu_f^{\max}$ and $\nu_z^{\max}$ as a means to prevent the trust regions from becoming too small too quickly. If the interpolation set has not been updated, the algorithm understands that the lack of success is not due to the surrogate models but rather due to the trust region size, and thus it reduces the latter.

Algorithm 3.7: f-iteration(x_k, s_k, x_k^+, s_k^+, Δ_k^f, Δ_k^z)

1: Successful iteration. If $\rho_k^f \geq \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$ and $\nu_f = 0$. The radius of $\mathcal{T}_k$ in (24) is updated by

$$\Delta_{k+1}^f = \begin{cases} \min\left[\max[\gamma_2 \|d_k\|, \Delta_k^f], \Delta^{\max}\right] & \text{if } \rho_k^f \geq \eta_2, \\ \Delta_k^f & \text{if } \rho_k^f \in [\eta_1, \eta_2), \end{cases} \tag{54}$$

The radius of $\mathcal{N}_k$ in (19) is updated by

$$\Delta_{k+1}^z = \begin{cases} \min\left[\max[\gamma_2\|n_k\|, \Delta_k^z], \Delta^{\max}\right] & \text{if } v(x_k^+, s_k^+) < \eta_3\, v_k^{\max}, \\ \Delta_k^z & \text{otherwise.} \end{cases} \tag{55}$$

2: Unsuccessful iteration. If $\rho_k^f < \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta_{k+1}^z = \delta_k^z$. The radius of $\mathcal{T}_k$ is updated by

$$\Delta_{k+1}^f = \begin{cases} \gamma_1\|d_k\| & \text{if either } (\mathcal{Y}_{k+1} \neq \mathcal{Y}_k \text{ and } v_f \leq \nu_f^{\max}) \\ & \quad \text{or } \mathcal{Y}_{k+1} = \mathcal{Y}_k, \\ \Delta_k^f & \text{if } \mathcal{Y}_{k+1} \neq \mathcal{Y}_k \text{ and } v_f > \nu_f^{\max}, \end{cases} \tag{56}$$

If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$ and $v_f \leq \nu_f^{\max}$, update $v_f = v_f + 1$.

The operations related to z-iterations follow below.

Algorithm 3.8: z-iteration$(x_k, s_k, x_k^+, s_k^+, \Delta_k^f, \Delta_k^z)$

1: Successful iteration. If (46) holds, set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$, $\Delta_{k+1}^f = \Delta_k^f$ and $v_z = 0$. The radius of $\mathcal{N}_k$ is updated by

$$\Delta_{k+1}^z = \begin{cases} \min\left[\max[\gamma_2\|n_k\|, \Delta_k^z], \Delta^{\max}\right] & \text{if } \rho_k^z \geq \eta_2, \\ \Delta_k^z & \text{if } \rho_k^z \in [\eta_1, \eta_2). \end{cases} \tag{57}$$

2: Unsuccessful iteration. If (46) fails, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta_{k+1}^f = \Delta_k^f$. The radius of $\mathcal{N}_k$ is updated by

$$\Delta_{k+1}^z = \begin{cases} \gamma_1\|n_k\| & \text{if } \|n_k\| \neq 0 \text{ and either } (\mathcal{Y}_{k+1} \neq \mathcal{Y}_k \text{ and } v_z \leq \nu_z^{\max}) \\ & \quad \text{or } \mathcal{Y}_{k+1} = \mathcal{Y}_k, \\ \Delta_k^z & \text{if } \mathcal{Y}_{k+1} \neq \mathcal{Y}_k \text{ and } v_z > \nu_z^{\max}, \\ \gamma_1\Delta_k^z & \text{if } \|n_k\| = 0, \end{cases} \tag{58}$$

If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$ and $v_z \leq \nu_z^{\max}$, update $v_z = v_z + 1$.

3.4 Parameter tuning and user goals

Like any other algorithm, DEFT-FUNNEL performance can be affected by how its parameters are tuned. In this section, we discuss about which parameters might have a major impact on its performance and how they should be tuned depending on the user goals. We consider four main aspects concerning the resolution of black-box problems: the budget, the priority level given to feasibility, the type of the objective function and the priority level given to global optimality. We can then describe the possible scenarios in decreasing order of difficulty as below:

- **Budget:** very low, limited or unlimited.
- **Priority to feasibility:** high or low.
- **Objective function:** highly multimodal, multimodal or likely unimodal.
- **Priority given to global optimality:** high or low.

Clearly, if the objective function is highly multimodal and the budget is limited, one should give priority to a multistart strategy with a high number of samples per iteration where global optimality is important. This can be done via two ways: the first one is by reducing the budget for each local search through the optional input maxeval_ls, which equals maxeval*0.7 by default; for instance, one could try setting maxeval_ls = maxeval*0.4, so that each local search uses up to 40% of the total budget only. The other possibility is to increase the number N of random points generated in the global search (see Step 3 in Algorithm 2.1). In case where attaining the global minimum is not a condition, a better approach would be to give more budget to each local search so that the chances to reach a local minimum are higher. If the objective function is not highly multimodal, one should search for a good compromise between spending the budget on each local search and on the sampling of the multistart strategy.

Notice also that when the budget is too small and it is very hard to find a feasible solution, it may be a good idea to compute a tangent step only when feasibility has been achieved. This is due to the fact that tangent steps may still deteriorate the gains in feasibility obtained through normal steps. When this happens, more normal steps must be computed which requires more function evaluations. Therefore, instead of spending the budget with both tangent and normal steps without guarantee of feasibility in the end, it is a better strategy to compute only normal steps in the beginning so that the chances of obtaining a feasible solution are higher in the end. For this purpose, the user should set the constant value $\kappa_{\mathcal{R}} = 0$ in (26). By doing so, the tangent step will be computed only if the normal step equals zero, which by (31) happens when the iterate is feasible. In the code, this can be done by setting `kappa_r` to zero in the function `deft_funnel_set_parameters`.

4 Numerical experiments

The numerical experiments are divided into two sections: the first one is focused on the evaluation of the performance of DEFT-FUNNEL on black-box optimization problems, while the second one aims at analyzing the benefits of the exploitation of white-box functions on grey-box problems. In all experiments with DEFT-FUNNEL, minimum ℓ_2-norm models were employed. This choice is based on past numerical experiments (Sampaio and Toint 2015, 2016) where minimum ℓ_2-norm models performed better than the other approaches. The criticality step threshold was set to $\epsilon = 10^{-4}$ and the trust-region constants were defined as $\eta_1 = 10^{-4}$, $\eta_2 = 0.9$, $\eta_3 = 0.6$, $\gamma_1 = 0.5$, $\gamma_2 = 2.0$, $\nu_f^{\max} = 20 \times n$ and $\nu_z^{\max} = 20 \times n$, where n is the number of variables.

DEFT-FUNNEL is compared with two popular algorithms for constrained black-box optimization: NOMAD (Le Digabel 2011), a state-of-the-art C++ implementation of the MADS method, and the Pattern Search (PS) algorithm from the Matlab Global Optimization Toolbox (MATLAB 2015b). Both solvers belong to the class of direct-search methods for constrained black-box optimization algorithms. Besides their popularity, the choice for these solvers is based on the fact that both codes are publicly available for any user, a criterion that reduces the number of options significantly. Notice also that this section is neither intended to be an extensive benchmark on black-box optimization solvers nor to show the superiority of

one solver over another—as it would require, among other things, a careful hyperparameter tuning of the other solvers as well—but rather to present encouraging results from DEFT-FUNNEL in a set of meaningful and difficult tests based on real-world applications.

Since NOMAD and PS are local optimization algorithms, they have been coupled with the Latin Hypercube Sampling (LHS) (McKay et al. 1979) method in order to achieve global optimality, a very common strategy found among practitioners in industry. In NOMAD, all constraints were treated as relaxable constraints by using the Progressive Barrier approach of the solver. Our experiments cover therefore two of the most popular approaches for solving black-box problems: surrogate-based methods and direct-search methods. Other DFO algorithms have already been compared against DEFT-FUNNEL in previous papers (see Sampaio and Toint 2015, 2016) on a much larger set of test problems mainly designed for local optimization.

4.1 Black-box optimization problems

The three methods are compared on a set of 14 well-known benchmark problems for constrained global optimization, including four industrial design problems—Welded Beam Design (WB4) (Deb 2000), Gas Transmission Compressor Design (GTCD4) (Beightler and Phillips 1976), Pressure Vessel Design (PVD4) (Coello and Montes 2002) and Speed Reducer Design (SR7) (Floudas and Pardalos 1990)—and the Harley pooling problem (problem 5.2.2 from Floudas et al. 1999), which is originally a maximization problem and that has been converted into a minimization one. Besides the test problems originated from industrial applications, we have collected problems with different characteristics (multimodal, nonlinear, separable and non-separable, with connected and disconnected feasible regions) to have a broader view of the performance of the algorithms in various kinds of scenarios. For instance, the Hesse problem (Hesse 1973) is the result of the combination of 3 separable problems with 18 local minima and 1 global minimum, while the Gómez #3 problem, listed as the third problem in Gómez and Levy (1982), consists of many disconnected feasible regions, thus having many local minima. The test problems G3-G11 are taken from the widely known benchmark list in Michalewicz and Schoenauer (1996). Table 3 gives the number of decision variables, the number of constraints and the best known feasible objective function value of each test problem.

Two types of black-box experiments were conducted in order to compare the algorithms. In the first type, a small budget of 100 black-box calls is given to each algorithm in order to evaluate how they perform on difficult problems with highly expensive functions. In such scenarios, many algorithms have difficulties to obtain even local minima depending on the test problem. In the second type of experiments, we analyze their ability and speed to achieve global minima rather than local minima by allowing larger budgets that range from $100 \times n$ to $400 \times n$, where n is the number of variables. Every function is considered as black box in both types of experiments. Finally, each algorithm is run 50 times on each test problem. For NOMAD and PS, each run is done on a different starting point obtained from LHS.

Only approximate feasible solutions of the problem (1) are considered when comparing the best objective function values obtained by the algorithms, i.e. we require that each optimal solution x^* satisfy $cv(x^*) \leq 10^{-4}$, where

$$cv(x) \stackrel{\text{def}}{=} \max\left[\left[z(x) - u^s\right]^+, \left[l^s - z(x)\right]^+\right]. \tag{59}$$

In the next two subsections, we show the results for the two types of experiments in the black-box setting.

Table 3 Problem name, number of decision variables, number of constraints (simple bounds not included) and the best known feasible objective function value of each test problem

Test problem	Number of variables	Number of constraints	Best known feasible objective function value
Harley (Harley Pooling Problem)	9	6	−600
WB4 (Welded Beam Design)	4	6	1.7250
GTCD4 (Gas Transmission Compressor Design)	4	1	2964893.85
PVD4 (Pressure Vessel Design)	4	3	5804.45
SR7 (Speed Reducer Design)	7	11	2994.42
Hesse	6	6	−310
Gómez #3	2	1	−0.9711
G3	2	1	−1
G4	5	6	−30665.539
G6	2	2	−6961.8139
G7	10	8	24.3062
G8	2	2	−0.0958
G9	7	4	680.6301
G11	2	1	0.75000455

4.1.1 Budget-driven experiments

The results of the first type of experiments are shown in Table 4. In the second column, f_{OPT} denotes the objective function value of the global minimum of the problem when it is known or the best known objective function value otherwise. For each solver, we show the best, the average and the worst objective function values obtained in 50 runs on every test problem.

As it can be seen in Table 4, DEFT-FUNNEL attained f_{OPT} in 10 out of 14 problems, while NOMAD did it in 5 problems and PS in only 4. Besides, when considering the best value found by each solver, DEFT-FUNNEL was superior to the others or equal to the best solver in all problems. In the average and worse cases, DEFT-FUNNEL also presented a very good performance; in particular, its worse performance was inferior to all others' in only 4 problems, while its average performance was superior to others in 9 problems. Although NOMAD did not reach f_{OPT} often, it presented the second best average-case performance among all methods, while PS presented the worst. Finally, NOMAD and PS were unable to reach a feasible solution in the Harley pooling problem.

4.1.2 Experiments driven by global minima

This section evaluates the ability of each solver to find global minima rather than local minima. The results of the second type of experiments for each test problem are presented individually, which allows a better analysis of the evolution of each solver performance over the number of function evaluations allowed. Each figure is thus associated to one single test problem and shows the average progress curve of each solver after 50 trials as a function of the budget.

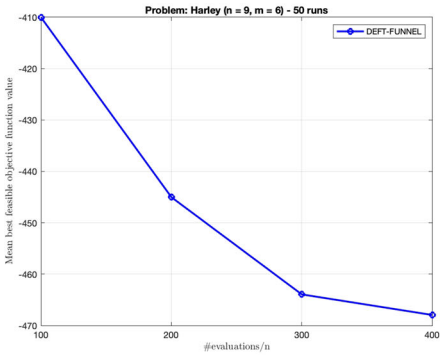
In Fig. 4, we show the results on test problems Harley, WB4, GTCD4 and PVD4. DEFT-FUNNEL and NOMAD presented the best performance among the three methods, with

Table 4 Results for the first type of experiments on a budget of 100 black-box calls. For each solver, it shows the best, the average and the worst objective function values obtained in 50 runs. For each column and each problem, the method with the lowest objective function value is in bold

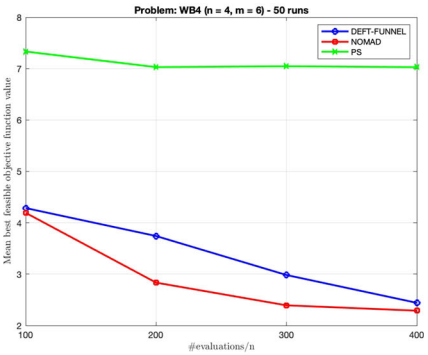
Prob	f_{OPT}	Solver	Best	Avg.	Worst
Harley	−600	NOMAD	None	None	None
		PS	None	None	None
		DEFT-FUNNEL	**−600**	**−17.1508**	**301.9904**
WB4	1.7250	NOMAD	3.0824	**6.4501**	11.7399
		PS	3.6274	7.2408	11.0038
		DEFT-FUNNEL	**3.0773**	6.7615	**8.1626**
GTCD4	2964893.85	NOMAD	3781751.7488	**7303454.2325**	13628063.6983
		PS	4953046.0849	12212940.1950	13786675.7772
		DEFT-FUNNEL	**3707234.9964**	8188878.0194	**13503809.4223**
PVD4	5804.45	NOMAD	6063.7060	7521.6839	**9630.1232**
		PS	5877.6483	7650.8416	10827.5206
		DEFT-FUNNEL	**5804.3761**	**7360.2882**	10033.0231
SR7	2994.42	NOMAD	3044.4672	**3146.3952**	**3347.5176**
		PS	3134.0525	3516.1761	5677.6224
		DEFT-FUNNEL	**3003.7577**	3489.8743	4583.1364
Hesse	−310	NOMAD	−308.4200	−194.4412	−44.3316
		PS	−302.1163	−162.5650	**−49.4364**
		DEFT-FUNNEL	**-310**	**−234.5969**	−24
Gómez #3	−0.9711	NOMAD	−0.9709	**−0.8626**	−0.2094
		PS	−0.9700	−0.7774	**−0.4293**
		DEFT-FUNNEL	**−0.9711**	−0.0689	3.23333
G3	−1	NOMAD	−0.9600	−0.0384	−0.0000
		PS	**−1**	−0.8162	**−0.0831**
		DEFT-FUNNEL	**−1**	**−0.8967**	−0.0182
G4	−30665.539	NOMAD	−30961.7716	−30565.9896	**−29792.9259**
		PS	−30814.5885	−29422.4521	−27838.8541
		DEFT-FUNNEL	**−31025.6056**	**−30980.3524**	−29246.5608
G6	−6961.8139	NOMAD	−6961.8147	−6327.9568	−1537.6319
		PS	−6252.2652	−3220.0072	−1206.1356
		DEFT-FUNNEL	**−6961.8165**	**−6961.8158**	**−6961.8146**
G7	24.3062091	NOMAD	111.7315	587.7964	1918.5181
		PS	197.7475	434.6940	**687.2492**
		DEFT-FUNNEL	**24.3011**	**52.1011**	185.5706
G8	−0.095825	NOMAD	**−0.0958**	**−0.0786**	**−0.0212**
		PS	−0.0953	−0.0097	0.0185
		DEFT-FUNNEL	**−0.0958**	−0.0471	0.0008

Table 4 continued

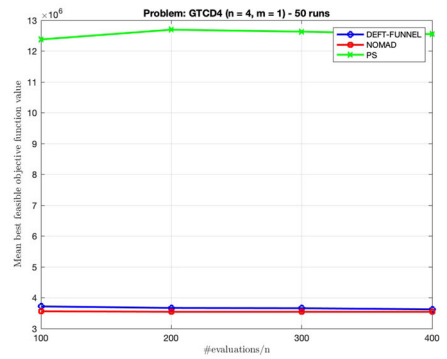
Prob	f_{OPT}	Solver	Best	Avg.	Worst
G9	680.6300573	NOMAD	910.0862	34279.7451	686827.9075
		PS	953.2964	5336.9157	12000.4159
		DEFT-FUNNEL	**797.1996**	**1403.7992**	**2668.9271**
G11	0.75000455	NOMAD	0.7501	0.9238	1
		PS	0.7502	0.8937	0.9997
		DEFT-FUNNEL	**0.7499**	**0.7513**	**0.8091**



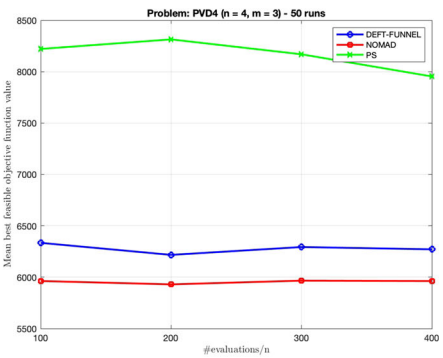
(a) Test problem: Harley.



(b) Test problem: WB4.



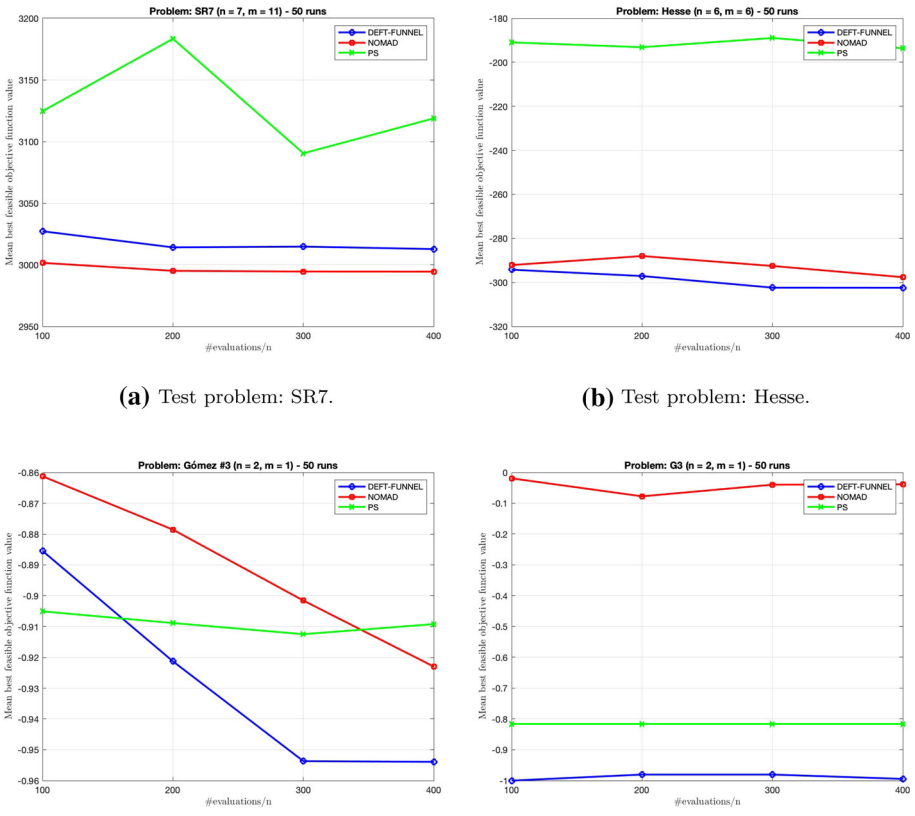
(c) Test problem: GTCD4.



(d) Test problem: PVD4.

Fig. 4 Mean best approximately feasible objective function value obtained by each solver on 50 trials with budgets of $100 \times n$, $200 \times n$, $300 \times n$ and $400 \times n$ black-box function evaluations

NOMAD being superior in 2 out of 4 problems. PS not only was inferior to the other two methods, but it also seemed not to be affected by the number of black-box calls allowed, as it can be seen in Fig. 4a–c. The solutions found by PS had often much larger objective function values than those obtained by DEFT-FUNNEL and NOMAD. Moreover, PS and NOMAD could not find a feasible solution for the Harley problem regardless of the number of black-box calls.



(a) Test problem: SR7.

(b) Test problem: Hesse.

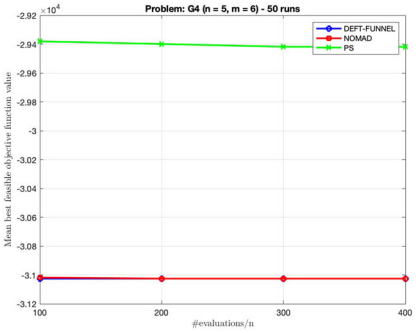
(c) Test problem: Gómez #3.

(d) Test problem: G3.

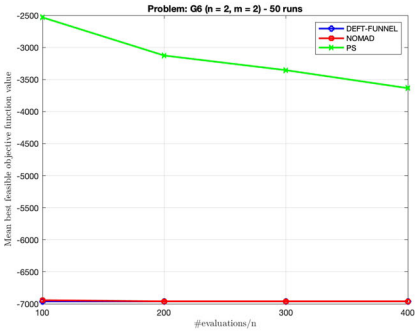
Fig. 5 Mean best approximately feasible objective function value obtained by each solver on 50 trials with budgets of $100 \times n$, $200 \times n$, $300 \times n$ and $400 \times n$ black-box function evaluations

Figure 5 shows the results on test problems SR7, Hesse, Gómez #3 and G3. DEFT-FUNNEL and NOMAD had similar performances on SR7 and Hesse problems with little difference between both solvers, while PS had the poorest performance. On the other hand, the performance of DEFT-FUNNEL on Gómez #3 and G3 was significantly better than the other methods; in particular, DEFT-FUNNEL attained f_{OPT} in G3 in, practically, all tests. Finally, PS performance on SR7 shows that allowing more black-box calls was not helpful and that the objective function value even increased.

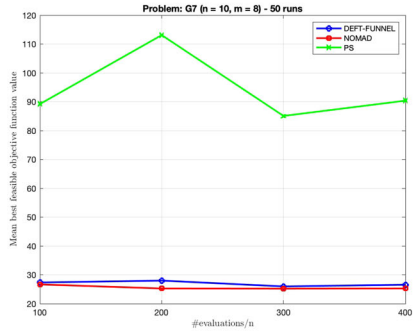
Figure 6 shows the results on test problems G4, G6, G7, G8, G9 and G11. DEFT-FUNNEL was superior to all other methods, attaining f_{OPT} on 4 problems (G4, G6, G8, G11) independently of the number of evaluations allowed. NOMAD was the second best, followed by PS. DEFT-FUNNEL and NOMAD had similar performances on G7 and G9, where both reached f_{OPT} when the number of black-box calls were higher than $200 \times n$. Besides that PS did not find a feasible solution on Harley pooling problem, there is a significant increase on its objective function value as the number of evaluations grows on test problems SR7 and G7. These 3 problems are the ones with the largest number of constraints, which could be a reason for its poor performance.



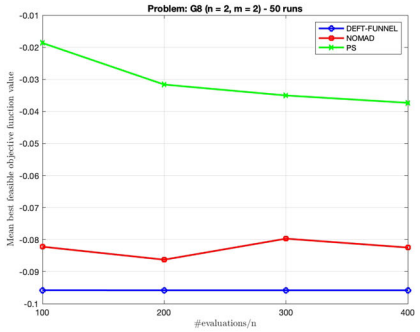
(a) Test problem: G4.



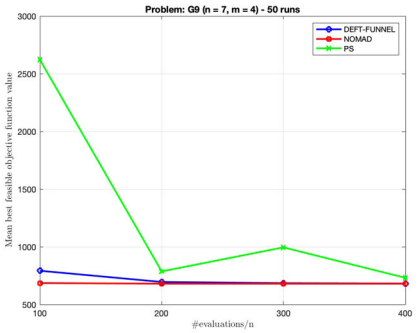
(b) Test problem: G6.



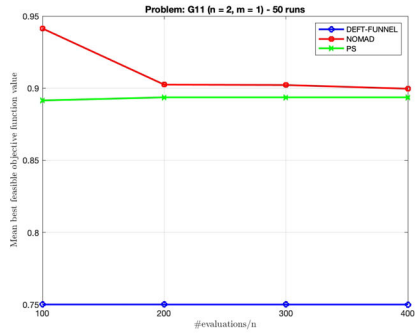
(c) Test problem: G7.



(d) Test problem: G8.



(e) Test problem: G9.



(f) Test problem: G11.

Fig. 6 Mean best approximately feasible objective function value obtained by each solver on 50 trials with budgets of $100 \times n$, $200 \times n$, $300 \times n$ and $400 \times n$ black-box function evaluations

Table 5 Problem name, number of decision variables, number of black-box constraints, number of white-box constraints, type of objective function—black box (BB) or white box (WB)—and the best known feasible objective function value of each test problem

Test problem	Number of variables	Number of BB constraints	Number of WB constraints	Type of Obj. function	Best known feasible bj. function value
WB4	4	6	0	WB	1.7250
GTCD4	4	0	1	BB	2964,893.85
SR7	7	9	2	WB	2994.42
Hesse	6	3	3	WB	−310

Table 6 Experiments with grey-box problems with a budget of 100 black-box calls. For each solver, it shows the best, the average and the worst objective function values obtained in 50 runs. DEFT-FUNNEL GB denotes the results on the grey-box problems while DEFT-FUNNEL BB denotes the version of DEFT-FUNNEL with the WB functions treated as BB functions

Prob	f_{OPT}	Solver	Best	Avg.	Worst
WB4	1.7250	NOMAD	3.0824	6.4501	11.7399
		PS	3.6274	7.2408	11.0038
	DEFT-FUNNEL GB	**2.3590**	**6.0765**	**6.3947**	
	DEFT-FUNNEL BB	3.0773	6.7615	8.1626	
GTCD4	2964893.85	NOMAD	3781751.7488	7303454.2325	13628063.6983
		PS	4953046.0849	12212940.1950	13786675.7772
		DEFT-FUNNEL GB	**3697286.5280**	**6904032.0742**	**10408130.0338**
		DEFT-FUNNEL BB	3707234.9964	8188878.0194	13503809.4223
SR7	2994.42	NOMAD	3044.4672	3146.3952	3347.5176
		PS	3134.0525	3516.1761	5677.6224
		DEFT-FUNNEL GB	**2994.4244**	**3083.8930**	**3288.8933**
		DEFT-FUNNEL BB	3003.7577	3489.8743	4583.1364
Hesse	−310	NOMAD	−308.4200	−194.4412	−44.3316
		PS	302.1163	−162.5650	**−49.4364**
		DEFT-FUNNEL GB	**−310**	**−241.2927**	−27
		DEFT-FUNNEL BB	**−310**	−234.5969	−24

4.2 Grey-box optimization problems

In this section, DEFT-FUNNEL is compared with NOMAD and PS on a set of artificial grey-box problems built by selecting a test problem and defining some of its functions as black boxes and the remaining as white boxes. Among the 5 grey-box problems considered in the experiments, 3 were used in the black-box experiments described in the previous subsection, where all functions were considered as black boxes, namely: Hesse, SR7 and GTCD4. The reuse of these test problems allows for evaluating the performance improvement of DEFT-FUNNEL in cases where some information is available and for comparing its performance with that obtained in the black-box setting. The remaining grey-box problems are the problems 21 and 23 from the well-known Hock-Schittkowski collection (Hock and Schittkowski 1980), denoted here as HS21 and HS23, respectively. Both HS21 and HS23

have nonlinear objective functions; however, in HS21 the objective function is defined as white box, while in HS23 it is defined as black box. The only constraint present in HS21 is defined as black box, while in HS23 there is a balance between both categories among the constraints. We have therefore attempted to cover different possibilities related to the type of functions in a grey-box setting. The derivatives of all functions defined as white boxes were given as input to DEFT-FUNNEL. The reader can find more details about the tested grey-box problems in Table 5.

In Table 6, the results obtained with a budget of 100 black-box calls are presented. In order to see the improvement on the DEFT-FUNNEL results when some functions are defined as white boxes, both the results obtained on the grey-box problem, denoted by DEFT-FUNNEL GB in Table 6, and those obtained on the corresponding black-box problem, denoted by DEFT-FUNNEL BB, are provided. The DEFT-FUNNEL BB results contain the same values already presented in Table 4.

It can be seen in Table 6 that DEFT-FUNNEL GB was the only one to reach f_{OPT} in two problems, having also the best average-case and worst-case performances in general. When comparing the black-box and grey-box results obtained by DEFT-FUNNEL on problems WB4, GTCD4 and SR7, it is evident that the available information from the white-box functions contributed to improve its performance. Not only the best objective function values on these problems were improved, allowing for reaching the global minimum on SR7, but also the average and worst ones. Since DEFT-FUNNEL had already attained the global minimum on Hesse in the black-box setting, the only expected improvement would be in the average and worst cases, which did not happen in the experiments. This is probably due to the fact that this a multimodal problem with 18 local minima, being a combination of 3 separable problems and having 1 global minimum. Therefore, even if information about the function is partially available, the problem remains very difficult to solve due to its nature.

5 Conclusions

This paper introduced a new global optimization solver for general constrained grey-box and black-box optimization problems named DEFT-FUNNEL. It combines a stochastic multistart strategy with a trust-funnel sequential quadratic optimization algorithm where the black-box functions are replaced by surrogate models built from polynomial interpolation. The proposed solver is able to exploit available information from white-box functions rather than considering them as black boxes. Its code is open source and freely available at the Github repository http://github.com/phrsampaio/deft-funnel. Unlike many black-box optimization algorithms, it can handle both inequality and equality constraints and it does not require feasible starting points. Moreover, the constraints are treated individually rather than grouped into a penalty function.

The numerical experiments have shown that DEFT-FUNNEL compares favourably with other state-of-the-art algorithms available for the optimization community on a collection of well-known benchmark problems. The reported numerical results indicate that the proposed approach is very promising for grey-box and black-box optimization. Future research works include extensions for multiobjective optimization and mixed-integer nonlinear optimization as well as as parallel version of the solver.

References

Amaioua N, Audet C, Conn AR, Digabel SL (2018) Efficient solution of quadratically constrained quadratic subproblems within the mesh adaptive direct search algorithm. Eur J Oper Res 268(1):13–24. https://doi.org/10.1016/j.ejor.2017.10.058. Retrieved from http://www.sciencedirect.com/science/article/pii/S0377221717309876

Armstrong JC, Favorite JA (2016) Using a derivative-free optimization method for multiple solutions of inverse transport problems. Optim Eng 17(1):105–125. https://doi.org/10.1007/s11081-015-9306-x

Audet C, Dennis J (2006) Mesh adaptive direct search algorithms for constrained optimization. SIAM J Optim 17(1):188–217. https://doi.org/10.1137/040603371

Audet C, Digabel SL, Peyrega M (2015) Linear equalities in blackbox optimization. Comput Optim Appl 61(1):1–23. https://doi.org/10.1007/s10589-014-9708-2

Audet C, Hare W (2017) Derivative-free and blackbox optimization. Springer, Cham. https://doi.org/10.1007/978-3-319-68913-5

Bajaj I, Iyer SS, Hasan MMF (2018) A trust region-based two phase algorithm for constrained black-box and grey-box optimization with infeasible initial point. Comput Chem Eng 116:306–321. https://doi.org/10.1016/j.compchemeng.2017.12.011, Retrieved from http://www.sciencedirect.com/science/article/pii/S0098135417304404

Beightler CS, Phillips DT (1976) Applied geometric programming. Wiley, New York

Birgin EG, Martínez JM, Raydan M (2000) Nonmonotone spectral projected gradient methods on convex sets. SIAM J Optim 10(4):1196–1211

Boukouvala F, Hasan MMF, Floudas CA (2017) Global optimization of general constrained grey-box models: new method and its application to constrained pdes for pressure swing adsorption. J Global Optim 67(1):3–42. https://doi.org/10.1007/s10898-015-0376-2

Bueno L, Friedlander A, Martínez J, Sobral F (2013) Inexact restoration method for derivative-free optimization with smooth constraints. SIAM J Optim 23(2):1189–1213. https://doi.org/10.1137/110856253

Coello CAC, Montes EM (2002) Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Adv Eng Inf 16(3):193–203. https://doi.org/10.1016/S1474-0346(02)00011-3. Retrieved from http://www.sciencedirect.com/science/article/pii/S1474034602000113

Conn AR, Scheinberg K, Vicente LN (2009) Introduction to derivative-free optimization. MPS-SIAM Book Series on Optimization, Philadelphia

Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186(2):311–338. https://doi.org/10.1016/S0045-7825(99)00389-8. Retrieved from http://www.sciencedirect.com/science/article/pii/S0045782599003898

Digabel SL, Wild SM (2015) A taxonomy of constraints in simulation-based optimization. arXiv:1505.07881

Echebest N, Schuverdt ML, Vignau RP (2017) An inexact restoration derivative-free filter method for nonlinear programming. Comput Appl Math 36(1):693–718. https://doi.org/10.1007/s40314-015-0253-0

Floudas C (2000) Deterministic global optimization: theory, methods and applications. Springer, Boston, MA. https://doi.org/10.1007/978-1-4757-4949-6

Floudas C, Pardalos P, Adjiman C, Esposito WR, Gümüs ZH, Harding, ST, Schweiger CA (1999) Handbook of test problems in local and global optimization. Springer US. https://doi.org/10.1007/978-1-4757-3040-1

Floudas CA, Pardalos PM (1990) A collection of test problems for constrained global optimization algorithms. Springer, Berlin

Gómez S, Levy AV (1982) The tunnelling method for solving the constrained global optimization problem with several non-connected feasible regions. In: Hennart JP (ed) Numerical analysis. Springer, Heidelberg, pp 34–47

Gould NIM, Toint PL (2010) Nonlinear programming without a penalty function or a filter. Math Program 122(1):155–196. https://doi.org/10.1007/s10107-008-0244-7

Gratton S, Toint PL, Tröltzsch A (2011) An active-set trust-region method for bound-constrained nonlinear optimization without derivatives. Optim Methods Softw 26(4–5):875–896

Griewank A (2003) A mathematical view of automatic differentiation. Acta Numerica 12:321–398. https://doi.org/10.1017/S0962492902000132

Griewank A, Walther A (2008) Evaluating derivatives (Second ed), Society for Industrial and Applied Mathematics. Retrieved from https://epubs.siam.org/doi/abs/10.1137/1.9780898717761. https://doi.org/10.1137/1.9780898717761

Hesse R (1973) A heuristic search procedure for estimating a global solution of nonconvex programming problems. Oper Res 21(6):1267–1280. https://doi.org/10.1287/opre.21.6.1267

Hock W, Schittkowski K (1980) Test examples for nonlinear programming codes. J Optim Theory Appl 30(1):127–129. https://doi.org/10.1007/BF00934594

Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. J Global Optim 13(4):455–492. https://doi.org/10.1023/A:1008306431147
Kan AHGR, Timmer GT (1987a) Stochastic global optimization methods part i: Clustering methods. Math Program 39(1):27–56. https://doi.org/10.1007/BF02592070
Kan AHGR, Timmer GT (1987b) Stochastic global optimization methods part ii: multi level methods. Math Program 39(1):57–78. https://doi.org/10.1007/BF02592071
Larson J, Menickelly M, Wild SM (2019) Derivative-free optimization methods. Acta Numer 28:287–404. https://doi.org/10.1017/S0962492919000060
Le Digabel S (2011) Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm. ACM Trans Math Softw 37(4):1–15
Lewis R, Torczon V (2002) A globally convergent augmented lagrangian pattern search algorithm for optimization with general constraints and simple bounds. SIAM J Optim 12(4):1075–1089. https://doi.org/10.1137/S1052623498339727
Locatelli M (1998) Relaxing the assumptions of the multilevel single linkage algorithm. J Global Optim 13(1):25–42. https://doi.org/10.1023/A:1008246031222
Maratos N (1978) Exact penalty function algorithms for finite dimensional and control optimization problems., Department of Control Theory, Imperial College London. Retrieved from https://books.google.fr/books?id=Ar2AtgAACAAJ
MATLAB (2015b) Natick, Massachusetts, The MathWorks Inc
McKay MD, Beckman RJ, Conover WJ (1979) A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2):239–245. Retrieved from http://www.jstor.org/stable/1268522
Michalewicz Z, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4(1):1–32. https://doi.org/10.1162/evco.1996.4.1.1
Nocedal J, Wright SJ (2006) Numerical optimization, 2nd edn. Springer, New York
Powell MJD (1994) A direct search optimization method that models the objective and constraint functions by linear interpolation. In: Gomez S, Hennart JP (eds), Advances in optimization and numerical analysis (pp. 51–67). Springer, Dordrecht. https://doi.org/10.1007/978-94-015-8330-5_4
Regis RG (2011) Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. Comput Oper Res 38(5):837–853. https://doi.org/10.1016/j.cor.2010.09.013. Retrieved from http://www.sciencedirect.com/science/article/pii/S030505481000208X
Regis RG (2014) Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points. Eng Optim 46(2):218–243. https://doi.org/10.1080/0305215X.2013.765000
Regis RG, Shoemaker CA (2005) Constrained global optimization of expensive black box functions using radial basis functions. J Global Optim 31(1):153–171. https://doi.org/10.1007/s10898-004-0570-0
Regis RG, Shoemaker CA (2007) A stochastic radial basis function method for the global optimization of expensive functions. INFORMS J Comput 19(4):497–509. https://doi.org/10.1287/ijoc.1060.0182
Rios LM, Sahinidis NV (2013) Derivative-free optimization: a review of algorithms and comparison of software implementations. J Global Optim 56(3):1247–1293. https://doi.org/10.1007/s10898-012-9951-y
Sampaio PR, Toint PL (2015) A derivative-free trust-funnel method for equality-constrained nonlinear optimization. Comput Optim Appl 61(1):25–49. https://doi.org/10.1007/s10589-014-9715-3
Sampaio PR, Toint PL (2016) Numerical experience with a derivative-free trust-funnel method for nonlinear optimization problems with general nonlinear constraints. Optim Methods Softw 31(3):511–534. https://doi.org/10.1080/10556788.2015.1135919
Scheinberg K, Toint PL (2010) Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. SIAM J Optim 20(6):3512–3532
Sendín JOH, Banga JR, Csendes T (2009) Extensions of a multistart clustering algorithm for constrained global optimization problems. Ind Eng Chem Res 48(6):3014–3023. https://doi.org/10.1021/ie800319m
Zhang H, Conn A, Scheinberg K (2010) A derivative-free algorithm for least-squares minimization. SIAM J Optim 20(6):3555–3576. https://doi.org/10.1137/09075531X