UNIVERSITÉ DE NAMUR

naXys
Namur Center for Complex Systems

Numerical experience with a derivative-free trust-funnel method for nonlinear optimization problems with general nonlinear constraints

by Ph. R. Sampaio and Ph. L. Toint

Report naXys-03-2015 August 12, 2015

University of Namur, 61, rue de Bruxelles, B5000 Namur (Belgium)

http://www.unamur.be/sciences/naxys

Numerical experience with a derivative-free trust-funnel method for nonlinear optimization problems with general nonlinear constraints

Ph. R. Sampaio* and Ph. L. Toint†

August 12, 2015

Abstract

A trust-funnel method is proposed for solving nonlinear optimization problems with general nonlinear constraints. It extends the one presented by Gould and Toint (*Math. Prog.*, 122(1):155-196, 2010), originally proposed for equality-constrained optimization problems only, to problems with both equality and inequality constraints and where simple bounds are also considered. As the original one, our method makes use of neither filter nor penalty functions and considers the objective function and the constraints as independently as possible. To handle the bounds, an active-set approach is employed. We then exploit techniques developed for derivative-free optimization to obtain a method that can also be used to solve problems where the derivatives are unavailable or are available at a prohibitive cost. The resulting approach extends the DEFT-FUNNEL algorithm presented by Sampaio and Toint (*Comput. Optim. Appl.*, 61(1):25-49, 2015), which implements a derivative-free trust-funnel method for equality-constrained problems. Numerical experiments with the extended algorithm show that our approach compares favorably to other well-known model-based algorithms for derivative-free optimization.

Keywords: Constrained nonlinear optimization, trust-region method, trust funnel, derivative-free optimization.

1 Introduction

We consider the solution of the nonlinear optimization problem

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{s.t.:} & l^s \leq c(x) \leq u^s, \\ & l^x \leq x \leq u^x, \end{cases} \tag{1.1}$$

where we assume that $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable, and that f is bounded below on the feasible domain. The vectors l^s and u^s are lower and upper bounds, respectively, on the constraints' values $c(x)$, while l^x and u^x are bounds on the x variables, with $l^s \in (\mathbb{R} \cup -\infty)^m$, $u^s \in (\mathbb{R} \cup \infty)^m$, $l^x \in (\mathbb{R} \cup -\infty)^n$ and $u^x \in (\mathbb{R} \cup \infty)^n$.

*Namur Center for Complex Systems (naXys) and Department of Mathematics, University of Namur, 61, rue de Bruxelles, B-5000 Namur, Belgium. Email: phillipece@gmail.com

†Namur Center for Complex Systems (naXys) and Department of Mathematics, University of Namur, 61, rue de Bruxelles, B-5000 Namur, Belgium. Email: philippe.toint@unamur.be

By defining $f(x,s) \stackrel{\text{def}}{=} f(x)$ and $c(x,s) \stackrel{\text{def}}{=} c(x) - s$, the problem above may be rewritten as the following equality-constrained optimization problem with simple bounds

$$\begin{cases} \min\limits_{(x,s)} & f(x,s) \\ \text{s.t.:} & c(x,s) = 0, \\ & l^s \leq s \leq u^s, \\ & l^x \leq x \leq u^x, \end{cases} \tag{1.2}$$

which is the one we address throughout this paper.

The trust-funnel method was first introduced by Gould and Toint [11] (see also [10]) as a SQP (*Sequential Quadratic Programming*) algorithm for equality-constrained optimization problems whose convergence is driven by an adaptive bound imposed on the allowed infeasibility at each iteration. Such bound is monotonically decreased as the algorithm progresses, assuring its global convergence whilst seeking optimality; hence the name "trust funnel". It belongs to the class of trust-region methods and makes use of a composite-step approach to calculate a new direction at each iteration: a normal step is computed first in the hope of reducing the infeasibility measure ensuing from the constraint functions' values, and a tangent step is subsequently calculated with the aim of improving optimality of the iterates with regard to the objective function. These computations are carried out with the use of two different trust regions, one for each step component. The main idea is to consider the objective function and the constraints as independently as possible. The method is noticeable among others for constrained problems as a parameter-free alternative, for neither filter nor penalties are needed, freeing the user from common difficulties encountered when choosing the initial penalty parameter, for instance. An extension to problems with both equalities and inequalities was developed recently by Curtis *et al.* [7], who presented an interior-point trust-funnel algorithm for solving large-scale problems that may be characterized as a barrier-SQP method.

Although many methods have been designed for unconstrained DFO (*Derivative-Free Optimization*), the range of possibilities for the constrained case remains quite open. A derivative-free trust-funnel method called DEFT-FUNNEL (DErivative-Free Trust FUNNEL), proposed by Sampaio and Toint [22] for equality-constrained problems, has given good results in comparison with a filter-SQP method introduced by Colson [4] and a trust-region method by Powell named COBYLA (Constrained Optimization BY Linear Approximations) [19, 20], all centred on local polynomial interpolation models. Recently, another trust-region method by Powell [21] entitled LINCOA has been developed for linearly-constrained optimization without derivatives and makes use of quadratic interpolation-based models for the objective function. Many other methods that use direct search instead of trust-region techniques combined to interpolation-based models also have been developed for the constrained case (e.g., Lewis and Torczon [14, 15, 16], Audet and Dennis [1], Lucidi *et al.* [17], Yu and Li [24]).

The primary contribution of this paper is the extension of both the original trust-funnel method [11] and its derivative-free adaptation (DEFT-FUNNEL) [22] to problems with general nonlinear constraints as well as the presentation of numerical experience with the resulting algorithm. Particularly for the derivative-free algorithm, a subspace minimization approach proposed by Gratton *et al.* [12] that considers the poisedness of the interpolation set is employed in order to avoid its degeneration.

The final method described here features four main steps to solve problems with general nonlinear constraints, namely: a subspace minimization approach to handle the bounds on the x variables, which makes the DEFT-FUNNEL an active-set method, a bounded linear least-squares solver to calculate the normal step, a projected gradient method to calculate the tangent step and the control of the permitted infeasibility of the iterates through the funnel bound. The reason behind the choice of exploring subspaces defined by the active bounds is dual. Besides the fact that we aim to avoid treating the bounds on the x variables as general inequality constraints, the reduction of the dimension of the problem after having identified the active bounds helps to thwart a possible degeneration of

the interpolation set when the sample points become close to each other and thus affinely dependent, which happens often as the optimal solution is approached. The fact that the s variables play no role on the choice of the interpolation set vindicates the construction of the subspaces based upon the x variables only.

This paper is organized as follows. Section 2 introduces the proposed trust-funnel algorithm step by step. The subsection 2.1 explains the subspace minimization approach, while the subsections 2.2 and 2.3 give details on how the computation of the composite step components is conducted. The subsections 2.4 and 2.5 then address the whole mechanism of the algorithm itself. Section 3 concerns the application of the method to derivative-free optimization problems and Section 4 gives the description of the final method, which assembles the techniques elaborated in Sections 2 and 3. In Section 5, we discuss some initial numerical experiments and compare its performance to COBYLA's and LINCOA's in a selected set of test problems. Ultimately, we draw some final conclusions in Section 6.

Notation. Unless otherwise specified, our norm $\|\cdot\|$ is the standard Euclidean norm. Given any vector x, we denote its i-th component by $[x]_i$. 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 $\mathbb{R}^n$. Finally, given any subspace $\mathcal{S}$, we denote its dimension by $dim(\mathcal{S})$.

2 An active-set trust-funnel method

Our method generates a sequence of points $\{(x_k, s_k)\}$ such that, at each iteration k, the bound constraints below are satisfied

$$l^s \leq s_k \leq u^s, \tag{2.3}$$
$$l^x \leq x_k \leq u^x. \tag{2.4}$$

By using a composite-step approach, each trial step $d_k \stackrel{\text{def}}{=} (d_k^x, d_k^s)^T$ is decomposed as

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

where the *normal step* component $n_k \stackrel{\text{def}}{=} (n_k^x, n_k^s)^T$ aims to improve feasibility, and the *tangent step* component $t_k \stackrel{\text{def}}{=} (t_k^x, t_k^s)^T$ reduces the objective function model's value while preserving any gains in feasibility obtained through n_k.

In the following subsections, we will describe how each component is computed. Before that, we briefly explain how the subspace minimization is employed in our algorithm.

2.1 Subspace minimization

As in the method proposed by Gratton *et al.* [12] for bound-constrained optimization problems, our algorithm makes use of an active-set approach where the minimization is restricted to subspaces defined by the active x variables.

At each iteration k, we define the subspace $\mathcal{S}_k$ as follows

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

where $\mathcal{L}_k \stackrel{\text{def}}{=} \{i \mid [x_k]_i - [l^x]_i \leq \epsilon_b\}$ and $\mathcal{U}_k \stackrel{\text{def}}{=} \{i \mid [u^x]_i - [x_k]_i \leq \epsilon_b\}$ define the index sets of (nearly) active variables at their bounds, for some small constant $\epsilon_b > 0$ defined *a priori*. After that $\mathcal{S}_k$ has been

defined, the minimization at iteration k is then restricted to the new subspace $\mathcal{S}_k$. Once a direction d_k for (x_k, s_k) has been computed, we set $(x_{k+1}, s_{k+1}) = (x_k, s_k) + d_k$ if k is a successful iteration; otherwise, we set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$.

If a solution $(\tilde{x}_k, \tilde{s}_k)$ for the subproblem defined by $\mathcal{S}_k$ satisfies the optimality conditions for the subproblem, we check whether it is also optimal for the original problem (2.5). If it is, the solution encountered is returned to the user and the algorithm halts; otherwise, it proceeds in the full space by computing a new direction for $(\tilde{x}_k, \tilde{s}_k)$ and repeats the above process at iteration $k+1$ by defining $\mathcal{S}_{k+1}$.

2.2 The normal step

For any point (x, s), we measure the constraint violation by

$$v(x, s) \stackrel{\text{def}}{=} \tfrac{1}{2}\|c(x, s)\|^2. \tag{2.5}$$

Analogous to the trust-funnel method for equality-constrained problems proposed in [11] and in [22], we also have a funnel bound $v_k^{\max}$ for v such that, for each iteration k,

$$v_k \leq v_k^{\max},$$

where $v_k \stackrel{\text{def}}{=} v(x_k, s_k)$. As this bound is monotonically decreased, the algorithm is driven towards feasibility, guaranteeing the convergence of the algorithm.

In order to ensure that the normal step is indeed "normal", we add the following condition that asks that it mostly lies in the space spanned by the columns of the matrix $J(x_k, s_k)^T$:

$$\|n_k\|_\infty \leq \kappa_n \|c(x_k, s_k)\|, \tag{2.6}$$

for some $\kappa_n \geq 1$. We then perform the calculation of n_k by solving the trust-region bound-constrained linear least-squares problem

$$\left\{ \begin{array}{rc} \min\limits_{n=(n^x,n^s)} & \tfrac{1}{2}\|c(x_k, s_k) + J(x_k, s_k)n\|^2 \\ \text{s.t.:} & l^s \leq s_k + n^s \leq u^s, \\ & l^x \leq x_k + n^x \leq u^x, \\ & x_k + n^x \in \mathcal{S}_k, \\ & n \in \mathcal{N}_k, \end{array} \right. \tag{2.7}$$

where $J(x, s) \stackrel{\text{def}}{=} (J(x) \ -I)$ represents the Jacobian of $c(x, s)$ with respect to (x, s) and

$$\mathcal{N}_k \stackrel{\text{def}}{=} \{z \in \mathbb{R}^{n+m} \mid \|z\|_\infty \leq \min[\, \Delta_k^c,\, \kappa_n \|c(x_k, s_k)\| \,]\}, \tag{2.8}$$

for some trust-region radius $\Delta_k^c > 0$.

2.3 The tangent step

After attempting to reduce infeasibility through the normal step, a tangent step is calculated with the aim of improving optimality. The computation of the latter is conducted carefully so that the gains in feasibility obtained by the former are not abandoned without good reasons.

The SQP model for the function f is defined as

$$\psi_k((x_k, s_k) + d) \stackrel{\text{def}}{=} f_k + \langle g_k, d\rangle + \tfrac{1}{2}\langle d, B_k d\rangle, \tag{2.9}$$

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

$$\begin{aligned}\mathcal{L}(x, s, \mu, z^s, w^s, z^x, w^x) = f(x) + \langle \mu, c(x, s))\rangle + \langle w^s, s - u^s\rangle + \langle z^s, l^s - s\rangle \\ + \langle w^x, x - u^x\rangle + \langle z^x, l^x - x\rangle\end{aligned}$$

with respect to (x, s), given by

$$B_k = \begin{pmatrix} H_k + \sum_{i=1}^m [\hat{\mu}_k]_i C_{ik} & 0 \\ 0 & 0 \end{pmatrix}, \tag{2.10}$$

where z^s and w^s are the Lagrange multipliers associated to the lower and upper bounds, respectively, on the slack variables s, and z^x and w^x, the Lagrange multipliers associated to the lower and upper bounds on the x variables. In (2.10), H_k is a bounded symmetric approximation of $\nabla^2_{xx} f(x_k, s_k) = \nabla^2 f(x_k)$, the matrices C_{ik} are bounded symmetric approximations of the constraints' Hessians $\nabla^2_{xx} c_{ik}(x_k, s_k) = \nabla^2 c_{ik}(x_k)$ and the vector $\hat{\mu}_k$ may be viewed as a local approximation of the Lagrange multipliers with respect to the equality constraints $c(x, s)$.

By using the decomposition $d_k = n_k + t_k$, we then have that

$$\psi_k((x_k, s_k) + n_k + t) \;\; = \;\; \psi_k((x_k, s_k) + n_k) + \langle g_k^N, t\rangle + \tfrac{1}{2}\langle t, B_k t\rangle, \tag{2.11}$$

where

$$g_k^N \stackrel{\text{def}}{=} g_k + B_k\, n_k. \tag{2.12}$$

In the interest of assuring that (2.11) it is a proper local approximation for the function $f((x_k, s_k) + n_k + t)$, the complete step $d = n_k + t$ must belong to

$$\mathcal{T}_k \stackrel{\text{def}}{=} \{d \in \mathbb{R}^{n+m} \mid \|d\|_\infty \leq \Delta_k^f\}, \tag{2.13}$$

for some radius Δ_k^f. The minimization of (2.11) should then be restricted to the intersection of $\mathcal{N}_k$ and $\mathcal{T}_k$, which imposes that the *tangent step* t_k results in a complete step $d_k = n_k + t_k$ that satisfies the inclusion

$$d_k \in \mathcal{R}_k \stackrel{\text{def}}{=} \mathcal{N}_k \cap \mathcal{T}_k \stackrel{\text{def}}{=} \{d \in \mathbb{R}^{n+m} \mid \|d\|_\infty \leq \Delta_k\}, \tag{2.14}$$

where the radius Δ_k of $\mathcal{R}_k$ is thus given by

$$\Delta_k = \min[\Delta_k^c, \Delta_k^f]. \tag{2.15}$$

Similarly to the computation of the normal step, we intend to remain in the subspace $\mathcal{S}_k$ after walking along the tangent direction. We accomplish that by imposing the following condition

$$x_k + n_k^x + t^x \in \mathcal{S}_k.$$

Additionaly, the conditions (2.3) and (2.4) must be satisfied at the final point $(x_k, s_k) + d_k$, i.e. we must have

$$\begin{aligned} l^s \leq s_k + n_k^s + t^s \leq u^s, \\ l^x \leq x_k + n_k^x + t^x \leq u^x. \end{aligned}$$

Due to (2.14), we ask n_k to belong to $\mathcal{R}_k$ before attempting the computation of t_k by requiring that

$$\|n_k\|_\infty \leq \kappa_{\mathcal{R}} \Delta_k, \tag{2.16}$$

for some $\kappa_{\mathcal{R}} \in (0,1)$. If (2.16) happens, which means that there is "enough space left" to make another step without crossing the trust region border, the tangent step is finally computed by solving the following problem

$$\left\{\begin{array}{rc} \min\limits_{t=(t^x,t^s)} & \langle g_k^N, t\rangle + \frac{1}{2}\langle t, B_k t\rangle \\ \text{s.t.:} & J(x_k,s_k)t = 0, \\ & l^s \leq s_k + n_k^s + t^s \leq u^s, \\ & l^x \leq x_k + n_k^x + t^x \leq u^x, \\ & x_k + n_k^x + t^x \in \mathcal{S}_k. \\ & n_k + t \in \mathcal{R}_k. \end{array}\right. \tag{2.17}$$

We define our f-criticality measure as

$$\pi_k^f \stackrel{\text{def}}{=} -\langle g_k^N, r_k\rangle, \tag{2.18}$$

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

$$\left\{\begin{array}{rc} \min\limits_{r=(r^x,r^s)} & \langle g_k^N, r\rangle \\ \text{s.t.:} & J(x_k,s_k)r = 0, \\ & l^s \leq s_k + n_k^s + r^s \leq u^s, \\ & l^x \leq x_k + n_k^x + r^x \leq u^x, \\ & x_k + n_k^x + r^x \in \mathcal{S}_k. \\ & \|r\|_\infty \leq 1. \end{array}\right. \tag{2.19}$$

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 bounds.

A new local estimate of the Lagrange multipliers $(\mu_k, z_k^s, w_k^s, z_k^x, w_k^x)$ are computed by solving the following bound-constrained linear least-squares problem

$$\left\{\begin{array}{rc} \min\limits_{(\mu,\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x)} & \frac{1}{2}\|\mathcal{M}_k(\mu,\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x)\|^2. \\ \text{s.t.:} & \hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x \geq 0, \end{array}\right. \tag{2.20}$$

where

$$\begin{aligned} \mathcal{M}_k(\mu,\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x) \stackrel{\text{def}}{=} & \begin{pmatrix} g_k^N \\ 0 \end{pmatrix} + \begin{pmatrix} J(x_k)^T \\ -I \end{pmatrix}\mu + \begin{pmatrix} 0 \\ I_w^s \end{pmatrix}\hat{w}^s + \begin{pmatrix} 0 \\ -I_z^s \end{pmatrix}\hat{z}^s \\ & + \begin{pmatrix} I_w^x \\ 0 \end{pmatrix}\hat{w}^x + \begin{pmatrix} -I_z^x \\ 0 \end{pmatrix}\hat{z}^x, \end{aligned}$$

the matrix I is the $m \times m$ identity matrix, the matrices I_z^s and I_w^s are obtained from I 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_z^x and I_w^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 $(\hat{z}^s,\hat{w}^s,\hat{z}^x,\hat{w}^x)$ are those in (z^s,w^s,z^x,w^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.

2.4 Which steps to compute and retain

We now explain when the normal step n_k and the tangent step t_k should be computed at iteration k given the constraint violation and the measure of optimality described in the previous subsection.

The normal step is computed when $k = 0$ or the current constraint violation is "significant", which is now formally defined by the conditions

$$\|c(x_k, s_k)\| > \omega_n(\pi^f_{k-1}) \text{ or } v_k > \kappa_{vv} v^{\max}_k, \tag{2.21}$$

where ω_n is some bounding function, $\kappa_{vv} \in (0, 1)$ is a constant. If (2.21) fails, the computation of the normal step is not required and so 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 v at (x_k, s_k) onto the region delimited by the bounds as

$$\pi^v_k \stackrel{\text{def}}{=} -\langle J(x_k, s_k)^T c(x_k, s_k), b_k\rangle,$$

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

$$\begin{cases} \min\limits_{b=(b^x,b^s)} & \langle J(x_k, s_k)^T c(x_k, s_k), b\rangle \\ \text{s.t.:} & l^s \le s_k + b^s \le u^s, \\ & l^x \le x_k + b^x \le u^x, \\ & x_k + b^x \in \mathcal{S}_k, \\ & \|b\|_\infty \le 1. \end{cases} \tag{2.22}$$

We say that (x_k, s_k) is an infeasible stationary point if $c(x_k, s_k) \neq 0$ and $\pi^v_k = 0$.

The procedure for the calculation of normal step is given in the algorithm below.

Algorithm 2.1: NormalStep(x_k, s_k, π^v_k, π^f_{k-1}, v_k, $v^{\max}_k$)

Step 1: If $c(x_k, s_k) \neq 0$ and $\pi^v_k = 0$, **STOP** (infeasible stationary point).

Step 2: If $k = 0$ or if (2.21) holds, compute a normal step n_k by solving (2.7). Otherwise, set $n_k = 0$.

If the solution of (2.19) is $r_k = 0$, then by (2.18) we have $\pi^f_k = 0$. In this case, the computation of the tangent step is skipped, and we simply set $t_k = 0$. If π^f_k is unsubstantial compared to the current infeasibility, i.e. for a given a monotonic bounding function ω_t, the condition

$$\pi^f_k > \omega_t(\|c(x_k, s_k)\|) \tag{2.23}$$

fails, then the current iterate is still too far from feasibility to worry about optimality, and we again skip the tangent step computation by setting $t_k = 0$.

While (2.23) and (2.21) together provide considerable flexibility in our algorithm in that a normal or tangent step is only computed when relevant, our setting also produces the possibility that both these conditions fail. In this case, we have that $d_k = n_k + t_k$ is identically zero, and the sole computation in the iteration is that of the new Lagrange multipliers (2.20). Finally, we may evaluate the usefulness of the tangent step t_k after (or during) its computation, in the sense that we would like a relatively large tangent step to cause a clear decrease in the model (2.11) of the objective function. We therefore check whether the conditions

$$\|t_k\| > \kappa_{\mathcal{CS}} \|n_k\| \tag{2.24}$$

and

$$\delta_k^f \stackrel{\text{def}}{=} \delta_k^{f,t} + \delta_k^{f,n} \geq \kappa_\delta \delta_k^{f,t}, \tag{2.25}$$

where

$$\delta_k^{f,t} \stackrel{\text{def}}{=} \psi_k((x_k, s_k) + n_k) - \psi_k((x_k, s_k) + n_k + t_k) \tag{2.26}$$

and

$$\delta_k^{f,n} \stackrel{\text{def}}{=} \psi_k(x_k, s_k) - \psi_k((x_k, s_k) + n_k), \tag{2.27}$$

are satisfied for some $\kappa_{\mathcal{CS}} > 1$ and for $\kappa_\delta \in (0, 1)$. The inequality (2.25) indicates that the *predicted* improvement in the objective function obtained in the tangent step is not negligible compared to the *predicted* change in f resulting from the normal step. If (2.24) holds but (2.25) fails, the tangent step is not useful in the sense just discussed, and we choose to ignore it by resetting $t_k = 0$.

The computation of the tangent step is described algorithmically as follows.

Algorithm 2.2: TangentStep(x_k, s_k, n_k)

Step 1: If (2.16) holds, then

- **Step 1.1:** select a vector $\hat{\mu}_k$ and define B_k as in (2.10);
- **Step 1.2:** compute μ_k by solving (2.20);
- **Step 1.3:** compute the modified Cauchy direction r_k by solving (2.19) and define π_k^f as (2.18);
- **Step 1.4:** if (2.23) holds, compute a tangent step t_k by solving (2.17).

Step 2: If (2.16) fails, set $\mu_k = \mu_{k-1}$. In this case, or if (2.23) fails, or if (2.24) holds but (2.25) fails, set $t_k = 0$ and $d_k = n_k$.

Step 3: Define $(x_k^+, s_k^+) = (x_k, s_k) + d_k$.

2.5 Iterations types

Once we have computed the step d_k and the trial point

$$(x_k^+, s_k^+) \stackrel{\text{def}}{=} (x_k, s_k) + d_k, \tag{2.28}$$

we are left with the task of accepting or rejecting it. If $n_k = t_k = 0$, iteration k is said to be a μ-iteration because the only computation potentially performed is that of a new vector of Lagrange multiplier estimates. We will say that iteration k is an f-iteration if $t_k \neq 0$, (2.25) holds, and

$$v(x_k^+, s_k^+) \leq v_k^{\max}. \tag{2.29}$$

Condition (2.29) ensures that the step maintains feasibility within reasonable bounds. Thus the iteration's expected major achievement is, in this case, a decrease in the value of the objective function f, hence its name. If iteration k is neither a μ-iteration nor a f-iteration, then it is said to be a c-iteration. If (2.25) fails, then the expected major achievement (or failure) of iteration k is, *a contrario*, to improve feasibility, which is also the case when the step only contains its normal component.

The main idea behind the technique for accepting the trial point is to measure whether the major expected achievement of the iteration has been realized.

- If iteration k is a μ-iteration, we do not have any other choice than to restart with $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ using the new multipliers. We then define

$$\Delta^f_{k+1} = \Delta^f_k \text{ and } \Delta^c_{k+1} = \Delta^c_k \tag{2.30}$$

  and keep the current value of the maximal infeasibility $v^{\max}_{k+1} = v^{\max}_k$.

- If iteration k is an f-iteration, we accept the trial point (i.e. $(x_{k+1}, s_{k+1}) = (x^+_k, s^+_k)$) if

$$\rho^f_k \stackrel{\rm def}{=} \frac{f(x_k, s_k) - f(x^+_k, s^+_k)}{\delta^f_k} \geq \eta_1, \tag{2.31}$$

  and reject it (i.e. $(x_{k+1}, s_{k+1}) = (x_k, s_k)$), otherwise. The value of the maximal infeasibility measure is left unchanged, that is $v^{\max}_{k+1} = v^{\max}_k$. Note that $\delta^f_k > 0$ (because of (2.26) and (2.25)) unless (x_k, s_k) is first-order critical, and hence that condition (2.31) is well-defined.

- If iteration k is a c-iteration, we accept the trial point if the improvement in feasibility is comparable to its predicted value

$$\delta^c_k \stackrel{\rm def}{=} \tfrac{1}{2}\|c(x_k, s_k)\|^2 - \tfrac{1}{2}\|c(x_k, s_k) + J(x_k, s_k)d_k\|^2,$$

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

$$n_k \neq 0, \quad \delta^c_k \geq \kappa_{cn}\delta^{c,n}_k \text{ and } \rho^c_k \stackrel{\rm def}{=} \frac{v(x_k, s_k) - v(x^+_k, s^+_k)}{\delta^c_k} \geq \eta_1, \tag{2.32}$$

  for some $\kappa_{cn} \in (0, 1 - \kappa_{tg}]$ and where

$$\delta^{c,n}_k \stackrel{\rm def}{=} \tfrac{1}{2}\|c(x_k, s_k)\|^2 - \tfrac{1}{2}\|c(x_k, s_k) + J(x_k, s_k)n_k\|^2. \tag{2.33}$$

  If (2.32) fails, the trial point is rejected. We update the value of the maximal infeasibility by

$$v^{\max}_{k+1} = \begin{cases} \max\left[\kappa_{tx1}v^{\max}_k, v(x^+_k, s^+_k) + \kappa_{tx2}(v(x_k, s_k) - v(x^+_k, s^+_k))\right] & \text{if (2.32) hold,} \\ v^{\max}_k & \text{otherwise,} \end{cases} \tag{2.34}$$

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

Note that we only check the third condition in (2.32) *after* the first two conditions have been verified. Assuming that $n_k \neq 0$, the Cauchy condition (2.33) and $c(x_k, s_k) \neq 0$ ensure that $\delta^{c,n}_k > 0$ provided $\pi^v_k \neq 0$. Thus the third condition is well defined, unless (x_k, s_k) is an infeasible stationary point, in which case the algorithm is terminated.

3 Application to derivative-free optimization

Our algorithm makes use of surrogate models $m^f(x)$ and $m^c(x) = (m^c_1(x), m^c_2(x), \ldots, m^c_m(x))$ built from polynomial interpolation over a set of sample points $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$ that replace the functions $f(x)$ and $c(x) = (c_1(x), c_2(x), \ldots, c_m(x))$. Therefore, at each iteration k, the following interpolation conditions are satisfied

$$\begin{aligned} m^f(y^i) &= f(y^i), \\ m^c_j(y^i) &= c_j(y^i), \end{aligned} \tag{3.35}$$

for all $y^i \in \mathcal{Y}_k$.

Whenever $\mathcal{Y}_k$ changes, the models m_k^c and m_k^f are updated to satisfy the interpolation conditions (3.35) for the new set $\mathcal{Y}_{k+1}$, thereby implying that new function evaluations of f and c are carried out for the additional points obtained at iteration k.

The algorithm developed in this work employs the now standard idea of starting with incomplete interpolation models with linear accuracy and then enhancing them with curvature information, which provides an actual accuracy at least as good as that for linear models and, hopefully, better. We thus consider underdetermined quadratic interpolation — i.e. $n+1 \leq |\mathcal{Y}| \leq (n+1)(n+2)/2$ — with initial sample sets that are poised for linear interpolation.

As there are many possibilities for building underdetermined quadratic interpolation-based models, we consider three distinct ways whose complete descriptions are available in Conn, Scheinberg and Vicente [6], namely: subbasis selection approach, minimum ℓ_2-norm or Euclidian norm models (referred to as minimum-norm interpolating polynomials in Section 5.1 of [6]), whose coefficients vector associated to the natural basis is the minimum Euclidian norm solution of the linear system resulting from the interpolation conditions, and minimum Frobenius norm models where the coefficients related to the quadratic part of the natural basis are minimized (see the optimization problem (5.6) in [6]), which is equivalent to the minimization of the Frobenius norm of the Hessian of the models. In addition, we also consider least-squares regression for the construction of quadratic models, in which case one has $n+1 \leq |\mathcal{Y}| \leq (n+1)(n+2)$, giving the user a total of four possibilities for model building.

For any function $f : \mathbb{R}^n \to \mathbb{R}$ and set $\mathcal{Y} = \{y^0, \ldots, y^p\} \subset \mathbb{R}^n$, there exists an unique polynomial $m(x)$ that interpolates $f(x)$ on $\mathcal{Y}$ such that $m(x) = \sum_{i=0}^{p} f(y^i)\ell_i(x)$, where $\{\ell_i(x)\}_{i=0}^{p}$ is the basis of Lagrange polynomials, if $\mathcal{Y}$ satisfies a property of nonsingularity called *poisedness* (see Definition 3.1, page 37, in [6]). Before going further into details of the algorithm, we first introduce a condition to the interpolation set that is used to measure the error between the original functions and the interpolation models as well as their derivatives.

Definition 3.1. *Let $\mathcal{Y} = \{y^0, y^1, \ldots, y^p\}$ be a poised interpolation 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 basis of Lagrange polynomials associated with $\mathcal{Y}$. Then, the set $\mathcal{Y}$ is said to be Λ-poised in $\mathcal{B}$ for $\mathcal{P}_n^d$ (in the interpolation sense) if and only if*

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

As it is shown in [6], the error bounds for at most fully quadratic models 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 $B(\mathcal{Y})$ containing $\mathcal{Y}$ for the first derivatives and quadratic for the function values.

3.1 Recursive call in subspaces

As pointed out before, by reducing the dimension of the problem we attempt to diminish the chances of a possible degeneration of the interpolation set when the sample points become too close to each other and thus affinely dependent. Such case might happen, for instance, as the optimal solution is approached.

In our algorithm, we apply the recursive approach found in the derivative-free method proposed by Gratton *et al.* [12] for bound-constrained optimization problems. Once the subspace $\mathcal{S}_k$ has been defined at iteration k, the algorithm calls itself recursively and the dimension of the problem is then reduced to $\hat{n} = n - |\mathcal{L}_k \cup \mathcal{U}_k|$, where n denotes here the dimension of $\mathbb{R}^n$. A new well-poised

interpolation set $\mathcal{Z}_k$ is then constructed from a suitable choice of points in $\mathcal{X}_k \cap \mathcal{S}_k$, where $\mathcal{X}_k$ is the set of all points obtained up to iteration k. In order to save function evaluations in the building process of the new interpolation set, all the points in $\mathcal{X}_k$ that are nearly but not in $\mathcal{S}_k$ are projected onto $\mathcal{S}_k$ and used to build $\mathcal{Z}_k$ with their model values instead of their real function values. In a more formal description, we define the set of the points that are close to (but not on) the active bounds at x_k as

$$\mathcal{A}_k \stackrel{\rm def}{=} \left\{ y \in \mathcal{X}_k \;\middle|\; \begin{array}{c} 0 \leq |\, [y]_i - [l^x]_i \,| \leq \epsilon_b \;\; \text{for} \;\; i \in \mathcal{L}_k \;\; \text{and} \\ 0 \leq |\, [u^x]_i - [y]_i \,| \leq \epsilon_b \;\; \text{for} \;\; i \in \mathcal{U}_k \end{array} \right\},$$

where, for at least one i, the strict inequality

$$0 < |\, [y]_i - [l^x]_i \,|, \; i \in \mathcal{L}_k,$$

or

$$0 < |\, [u^x]_i - [y]_i \,|, \; i \in \mathcal{U}_k,$$

must hold. We then project all the points $y \in \mathcal{A}_k$ onto $\mathcal{S}_k$, obtaining new "dummy" points y_s that are added to $\mathcal{X}_k$ with associated values $m_k^f(y_s)$ and $m_k^c(y_s)$ rather than the values of the original functions. As it is shown in Section 3.3, these dummy points are progressively replaced by other points with true function values with high priority during the minimization in $\mathcal{S}_k$. We denote by $\mathcal{D}um_k$ the set of dummy points at iteration k.

Convergence in a subspace is only declared if the interpolation set contains no dummy points. If a solution has been found for a subspace and there are still dummy points in the interpolation set, evaluations of the original functions $f(x)$ and $c(x)$ at such points are carried out and the interpolating models are recomputed from the original function values. Once convergence has been declared in a subspace $\mathcal{S}_k$, the $|\mathcal{L}_k \cup \mathcal{U}_k|$ fixed components x_i associated with the active bounds and the component x of the approximate solution found in $\mathcal{S}_k$ of dimension $\hat{n} = n - |\mathcal{L}_k \cup \mathcal{U}_k|$ are assembled to compose a full-dimensional vector $x_{\mathcal{S}}^*$ in $\mathbb{R}^n$. The algorithm then checks whether $(x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$ is optimal for the full-dimensional problem or not. Firstly, a full-space interpolation set of degree $n+1$ is built in an ϵ-neighborhood around the point $x_{\mathcal{S}}^*$. Subsequently, the corresponding interpolating models m_k^f and m_k^c are recomputed and the f-criticality measure π_{k-1}^f is calculated anew using information of the updated models. Finally, the criticality step in the full space is then entered.

The following algorithm gives a full description of the ideas explained above.

Algorithm 3.1: SubspaceMinimization($\mathcal{X}_k$, $\mathcal{Y}_k$, x_k, s_k, Δ_k^f, Δ_k^c, $v_k^{\max}$)

Step 1: Check for (nearly) active bounds at x_k and define $\mathcal{S}_k$. If there is no (nearly) active bound or if $\mathcal{S}_k$ has already been explored, go to Step 6. If all bounds are active, go to Step 5.

Step 2: Project points in $\mathcal{X}_k$ which lie close to the (nearly) active bounds on $\mathcal{S}_k$ and associate with them suitable function values estimates. Add new dummy points to $\mathcal{D}um_k$, if any.

Step 3: Build a new interpolation set $\mathcal{Z}_k$ in $\mathcal{S}_k$ including the projected points, if any.

Step 4: Call recursively DEFT-FUNNEL($\mathcal{S}_k$, $\mathcal{X}_k$, $\mathcal{Z}_k$, x_k, s_k, Δ_k^f, Δ_k^c, $v_k^{\max}$) and let $(x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$ be the solution of the subspace problem after adding the fixed components.

Step 5: If $dim(\mathcal{S}_k) < n$, return $(x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$. Otherwise, reset $(x_k, s_k) = (x_{\mathcal{S}}^*, s_{\mathcal{S}}^*)$, construct new set $\mathcal{Y}_k$ around x_k, build m_k^f and m_k^c and recompute π_{k-1}^f.

Step 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 $\Delta^c_{k+1} = \gamma\Delta^c_k$, set $\Delta_{k+1} = \min[\Delta^f_{k+1}, \Delta^c_{k+1}]$ and build a new poised set $\mathcal{Y}_{k+1}$ in $\mathcal{B}(x_{k+1}; \Delta_{k+1})$.

3.2 Criticality step

Convergence is declared when 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 3.1. The criticality step in DEFT-FUNNEL is described in the next algorithm.

Algorithm 3.2: CriticalityStep($\mathcal{Y}_k$, π^f_{k-1}, ϵ_i)

Step 1: Define $\hat{m}^f_i = m^f_k$, $\hat{m}^c_i = m^c_k$ and $\hat{\pi}^f_i = \pi^f_{k-1}$.

Step 2: If $\|c(x_k, s_k)\| \leq \epsilon_i$ and $\hat{\pi}^f_i \leq \epsilon_i$, set $\epsilon_{i+1} = \max\left[\alpha\|c(x_k, s_k)\|, \alpha\hat{\pi}^f_i, \epsilon\right]$ 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 $\hat{m}^f_i$ and $\hat{m}^c_i$, calculate $\hat{r}_i$ and $\hat{\pi}^f_i$ and increment i by one. If $\|c(x_k, s_k)\| \leq \epsilon$ and $\hat{\pi}^f_i \leq \epsilon$, return (x_k, s_k); otherwise, start Step 2 again;

Step 3: Set $m^f_k = \hat{m}^f_i$, $m^c_k = \hat{m}^c_i$, $\pi^f_{k-1} = \hat{\pi}^f_i$, $\Delta_k = \beta \max\left[\|c(x_k, s_k)\|, \pi^f_{k-1}\right]$ and define $\vartheta_i = x_k$ if a new model has been computed.

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

Many derivative-based trust-regions methods decrease the trust region radius at unsuccessful iterations to converge. However, in derivative-free optimization, unsuccessful iterations in trust-region methods might be due to a bad quality of the interpolating model rather than a large trust region size. In order to maintain the quality of the surrogate models without resorting to costly model improvement steps at unsuccessful iterations, which might require another optimization problem to be globally solved, we apply a self-correcting geometry scheme proposed by Scheinberg and Toint [23] for the management of the geometry of the interpolation set. Such scheme is based upon the idea that unsuccessful trial points can still be useful as they may improve the geometry of the interpolation set. At such iterations, the trust region radii are reduced only if the algorithm fails to improve the geometry of the interpolation set by replacing one of its points by the trial point, thereby implying that there is no lack of poisedness in this case. Firstly, an interpolation point that is far from the current point x_k is sought to be replaced by the trial point x^+_k. If there is no such a far point, one tries to replace an interpolation point $y^{k,j}$ whose associated Lagrange polynomial value at x^+_k in absolute value, $|\ell_{k,j}(x^+_k)|$, is bigger than a predefined threshold Λ. By doing that, one attempts to obtain a Λ-poised set $\mathcal{Y}_{k+1}$, or, since at most one interpolation point is replaced by iteration, improve its poisedness, at least. If no interpolation point is replaced, the trust regions are then shrunk.

A slight modification on the scheme is used on DEFT-FUNNEL. Following the idea proposed in [12], the trust region radii can also be reduced in cases where it is possible to improve poisedness. The

number of reductions of Δ_k^f and Δ_k^c allowed in such cases, however, is limited by constants $\nu_f^{\max} > 0$ and $\nu_c^{\max} > 0$ predefined by the user as a means to prevent the trust regions from becoming too small.

The management of the interpolation set in DEFT-FUNNEL is described in the next algorithm. It depends on the criterion used to define successful iterations, which is passed to the algorithm through the parameter *criterion*.

Algorithm 3.3: UpdateInterpolationSet($\mathcal{Y}_k$, x_k, x_k^+, Δ_k, ϵ_i, *criterion*)

Step 1: Augment the interpolation set. If $|\mathcal{Y}_k| < p_{\max}$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \cup \{x_k^+\}$.

Step 2: Replace a dummy interpolation point. If $|\mathcal{Y}_k| = p_{\max}$ and the set

$$\mathcal{D}_k \stackrel{\rm def}{=} \{y^{k,j} \in \mathcal{D}um_k \cap \mathcal{Y}_k\} \tag{3.36}$$

is non-empty, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ where $\ell_{k,r}(x_k^+) \neq 0$ and r is an index of any point in $\mathcal{D}_k$ such that

$$r = \arg\max_j \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.37}$$

Step 3: Successful iteration. If $|\mathcal{Y}_k| = p_{\max}$, $\mathcal{D}_k = \emptyset$, and *criterion* holds, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y^{k,r}\} \cup \{x_k^+\}$ for

$$y^{k,r} = \arg\max_{y^{k,j} \in \mathcal{Y}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.38}$$

Step 4: Replace a far interpolation point. If $|\mathcal{Y}_k| = p_{\max}$, $\mathcal{D}_k = \emptyset$, *criterion* fails, either $x_k \neq \vartheta_i$ or $\Delta_k \leq \epsilon_i$, and the set

$$\mathcal{F}_k \stackrel{\rm def}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k \mid > \zeta\Delta \text{ and } \ell_{k,j}(x_k^+) \neq 0\} \tag{3.39}$$

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

$$y^{k,r} = \arg\max_{y^{k,j} \in \mathcal{F}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.40}$$

Step 5: 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 $\mathcal{D}_k \cup \mathcal{F}_k$ is empty, and the set

$$\mathcal{C}_k \stackrel{\rm def}{=} \{y^{k,j} \in \mathcal{Y}_k \text{ such that } \|y^{k,j} - x_k\| \leq \zeta\Delta \text{ and } |\ell_{k,j}(x_k^+)| > \Lambda\} \tag{3.41}$$

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

$$y^{k,r} = \arg\max_{y^{k,j} \in \mathcal{C}_k} \|y^{k,j} - x_k^+\|^2 |\ell_{k,j}(x_k^+)|. \tag{3.42}$$

Step 6: No replacements. If $|\mathcal{Y}_k| = p_{\max}$, *criterion* fails and either $[x_k = \vartheta_i \text{ and } \Delta_k > \epsilon_i]$ or $\mathcal{D}_k \cup \mathcal{F}_k \cup \mathcal{C}_k = \emptyset$, then define $\mathcal{Y}_{k+1} = \mathcal{Y}_k$.

We now give the details of the procedures related to f- and c-iterations, starting with the former. The operations involved in such procedures concern the acceptance of the trial point and the update of the trust regions radii. The relation between the trust region updating strategy at unsuccessful iterations and the management of the interpolation set discussed above is now formally described.

Algorithm 3.4: f-iteration(x_k, s_k, x_k^+, s_k^+, Δ_k^f, Δ_k^c)

Step 1: Successful iteration. If $\rho_k^f \geq \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$ and $\nu_f = 0$.

If $\rho_k^f \geq \eta_2$, set $\Delta_{k+1}^f = \min[\max[\gamma_2\|d_k\|, \Delta_k^f], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^f = \Delta_k^f$.

If $v(x_k^+, s_k^+) < \eta_3\, v_k^{\max}$, set $\Delta_{k+1}^c = \min[\max[\gamma_2\|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.

Step 2: Unsuccessful iteration. If $\rho_k^f < \eta_1$, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta_{k+1}^c = \Delta_k^c$.

If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, set $\Delta_{k+1}^f = \gamma_1\|d_k\|$ and $\nu_f = \nu_f + 1$ if $\nu_f \leq \nu_f^{\max}$ or $\Delta_{k+1}^f = \Delta_k^f$ otherwise.

If $\mathcal{Y}_{k+1} = \mathcal{Y}_k$, set $\Delta_{k+1}^f = \gamma_1\|d_k\|$.

The operations related to c-iterations follow below.

Algorithm 3.5: c-iteration(x_k, s_k, x_k^+, s_k^+, Δ_k^f, Δ_k^c)

Step 1: Successful iteration. If (2.32) holds, set $(x_{k+1}, s_{k+1}) = (x_k^+, s_k^+)$, $\Delta_{k+1}^f = \Delta_k^f$ and $\nu_c = 0$.
If $\rho_k^c \geq \eta_2$, set $\Delta_{k+1}^c = \min[\max[\gamma_2\|n_k\|, \Delta_k^c], \Delta^{\max}]$; otherwise, set $\Delta_{k+1}^c = \Delta_k^c$.

Step 2: Unsuccessful iteration. If (2.32) fails, set $(x_{k+1}, s_{k+1}) = (x_k, s_k)$ and $\Delta_{k+1}^f = \Delta_k^f$.

If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$ and $\nu_c \leq \nu_c^{\max}$, set $\Delta_{k+1}^c = \gamma_1\|n_k\|$ if $\|n_k\| \neq 0$, or $\Delta_{k+1}^c = \gamma_1\Delta_k^c$ otherwise ($\|n_k\| = 0$). If $\nu_c > \nu_c^{\max}$, set $\Delta_{k+1}^c = \Delta_k^c$. If $\nu_c \leq \nu_c^{\max}$, update $\nu_c = \nu_c + 1$.

If $\mathcal{Y}_{k+1} = \mathcal{Y}_k$, set $\Delta_{k+1}^c = \gamma_1\|n_k\|$ if $\|n_k\| \neq 0$, or $\Delta_{k+1}^c = \gamma_1\Delta_k^c$ otherwise ($\|n_k\| = 0$).

4 The algorithm

We now provide a formal description of our complete algorithm for solving nonlinear optimization problems with general nonlinear constraints without derivatives.

Algorithm 4.1: DEFT-FUNNEL($\mathcal{S}$, $\mathcal{X}$, $\mathcal{Y}$, x, s, Δ^f, Δ^c, $v^{\max}$)

Step 0: Initialization. Choose an initial vector of Lagrange multipliers μ_{-1} and parameters $\epsilon > 0$, $\epsilon_0 > 0$, $\Delta_0^f > 0$, $\Delta_0^c > 0$, $\alpha \in (0,1)$, $0 < \gamma_1 < 1 < \gamma_2$, $\zeta \geq 1$, $0 < \eta_1 < \eta_2 < 1$, $\Lambda > 1$, $\beta > 0$, $\eta_3 > 0$. Define $\Delta_0 = \min[\Delta_0^f, \Delta_0^c] \leq \Delta^{\max}$. Initialize $\mathcal{Y}_0$, with $x_0 \in \mathcal{Y}_0 \subset \mathcal{B}(x_0, \Delta_0)$ and

$|\mathcal{Y}_0| \geq n+1$, as well as the maximum number of interpolation points $p_{\max} \geq |\mathcal{Y}_0|$. Compute the associated models m_0^f and m_0^c around x_0 and Lagrange polynomials $\{l_{0,j}\}_{j=0}^p$. Define $v_0^{\max} = \max[\kappa_{ca}, \kappa_{cr} v(x_0, s_0)]$, where $\kappa_{ca} > 0$ and $\kappa_{cr} > 1$. Compute r_{-1} by solving (2.19) with normal step $n_{-1} = 0$ and define π_{-1}^f as in (2.18). Define $\nu_f^{\max} > 0$ and $\nu_c^{\max} > 0$ and set $\nu_f = \nu_c = 0$. Set $k = 0$ and $i = 0$.

Step 1: SubspaceMinimization($\mathcal{X}_k$, $\mathcal{Y}_k$, x_k, s_k, Δ_k^f, Δ_k^c, $v_k^{\max}$).

Step 2: CriticalityStep($\mathcal{Y}_k$, π_{k-1}^f, ϵ_i).

Step 3: NormalStep(x_k, s_k, π_k^v, π_{k-1}^f, v_k, $v_k^{\max}$).

Step 4: TangentStep(x_k, s_k, n_k).

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

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

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

Step 6: Conclude an f-iteration. If $t_k \neq 0$ and (2.25) and (2.29) hold,

Step 6.1: UpdateInterpolationSet($\mathcal{Y}_k$,x_k, x_k^+, Δ_k, ϵ_i, '$\rho_k^f \geq \eta_1$');

Step 6.2: f-iteration(x_k, s_k, x_k^+, s_k^+, Δ_k^f, Δ_k^c);

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

Step 7: Conclude a c-iteration. If either $n_k \neq 0$ and $t_k = 0$, or either one of (2.25) or (2.29) fails,

Step 7.1: UpdateInterpolationSet($\mathcal{Y}_k$,x_k, x_k^+, Δ_k, ϵ_i, '(2.32)');

Step 7.2: c-iteration(x_k, s_k, x_k^+, s_k^+, Δ_k^f, Δ_k^c);

Step 7.3: Set $\Delta_{k+1} = \min[\Delta_{k+1}^f, \Delta_{k+1}^c]$ and update $v_k^{\max}$ using (2.34).

Step 8: Update the models and the Lagrange polynomials. If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation models m_{k+1}^f and m_{k+1}^c 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.

---

5 Numerical experiments

We implemented DEFT-FUNNEL in Matlab and tested it first on a set of 80 small-scale constrained problems from the CUTEst [9] collection. The problems contain at least one equality or inequality constraint, many of them containing both types and some containing simple bounds as well. 63 of the problems in our experiments are in the range of the first 113 test examples from the Hock-Schittowski collection [13], while the remaining are nonlinearly constrained optimization problems found in [3].

For the calculation of the normal step and the approximate Lagrange multipliers, we used a Matlab code named BLLS developed in collaboration with Anke Tröltzsch for solving bound-constrained linear least-squares problems. This method is intended for small-dimensional problems and is an active-set algorithm where the unconstrained problem is solved at each iteration in the subspace defined by the currently active bounds, which are determined by a projected Cauchy step. As for the computation of

the tangent step, we used a nonmonotone spectral projected gradient method [2] to solve the (possibly) indefinite quadratic subproblems (2.17). Finally, we define the bounding functions $\omega_n(t)$ and $\omega_t(t)$ as

$$\omega_n(t) \stackrel{\text{def}}{=} 0.01 \min[1,\ t] \quad \text{and} \quad \omega_t(t) \stackrel{\text{def}}{=} 0.01 \min[1,\ t^2]. \tag{5.43}$$

We compare the results of the four variants of our method with regard to the way of building the interpolating models — subbasis selection, mininum Frobenius norm, minimum ℓ_2-norm and regression — to those obtained with the popular Fortran code COBYLA [19], a trust-region method for constrained problems that models the objective and constraint functions by linear interpolation. The only criterion for comparison is the number of calls on the routine that evaluates the objective function and the constraints at the same time at the required points, which is motivated by the fact that these costs very often dominate those of all other algorithmic computations. Therefore, we do not count evaluations of the objective function and the constraints separately, but rather the number of calls to the single routine that calculates both simultaneously. Comparisons of the computational results obtained by DEFT-FUNNEL for equality-constrained problems only also have been made with another method named CDFO, a derivative-free filter-SQP algorithm proposed in [4]. The results are reported in [22] and show that DEFT-FUNNEL was superior to CDFO in our tests.

The threshold ϵ for declaring convergence in the criticality step in DEFT-FUNNEL is set to $\epsilon = 10^{-4}$. The stopping criterion used in COBYLA is based on the trust region radius ρ from the interval $[\rho_{end}, \rho_{beg}]$, where ρ_{beg} and ρ_{end} are constants predefined by the user. The parameter ρ is decreased by a constant factor during the execution of the algorithm and is never increased. The algorithm stops when $\rho = \rho_{end}$. Therefore, ρ_{end} should have the magnitude of the required accuracy in the final values of the variables. In our experiments, we set $\rho_{end} = 10^{-4}$.

In DEFT-FUNNEL, we fixed the trust-region parameters to $\Delta_0 = 1$, $\eta_1 = 0.0001$, $\eta_2 = 0.9$, $\eta_3 = 0.5$, $\gamma_1 = 0.5$, $\gamma_2 = 2.5$ and $\Delta^{\max} = 10^{10}$. The parameter ζ used in the definition of the sets $\mathcal{F}_k$ and $\mathcal{C}_k$ of far points and close points, respectively, is set to $\zeta = 1$. For the limit number of times to reduce the trust regions sizes when a far or close interpolation point is replaced at unsuccessful iterations, we choose $\nu_f^{\max} = \nu_c^{\max} = 10$. We set $p_{\max} = (n+1)(n+2)/2$ for the subbasis, minimum Frobenius norm and minimum ℓ_2-norm approaches, and $p_{\max} = (n+1)(n+2)$ for the regression case. Finally, we set $\epsilon_0 = 0.01$ as the initial value for the loop in the criticality step, $\alpha = 0.1$ and $\beta = 1$.

The performance of the methods are first compared by means of their *performance profiles* [8]. Denote the set of solvers by $\mathcal{S}$, and the set of problems by $\mathcal{P}$. We compare the performance on problem $p \in \mathcal{P}$ by solver $s \in \mathcal{S}$ with the best performance by any solver on this problem; that is, we use the *performance ratio*

$$r_{p,s} \stackrel{\text{def}}{=} \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}},$$

where $t_{p,s}$ is the number of function evaluations required to solve problem p by solver s. If we define

$$\rho_s(\tau) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{P}|} |\{p \in \mathcal{P} : r_{p,s} \leq \tau\}|,$$

then $\rho_s(\tau)$ is an approximation to the probability for solver $s \in \mathcal{S}$ that a performance ratio $r_{p,s}$ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio.

In Figure 5.1, we provide the performance profiles of the four variants of DEFT-FUNNEL, whereas, in Figure 5.2, we compare the performance of each variant of our method to COBYLA's individually. As it can be seen, DEFT-FUNNEL has shown superior results on the set of test problems when the interpolating models are built from subbasis selection, minimum Frobenius norm and minimum ℓ_2-norm approaches. For the regression variant, Figure 5.2d reveals that COBYLA was faster than DEFT-FUNNEL in most of the problems, although the latter was able to solve more problems than the former for large values of τ.

Figure 5.1: *Log*$_2$-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models on a set of 80 problems from CUTEst.

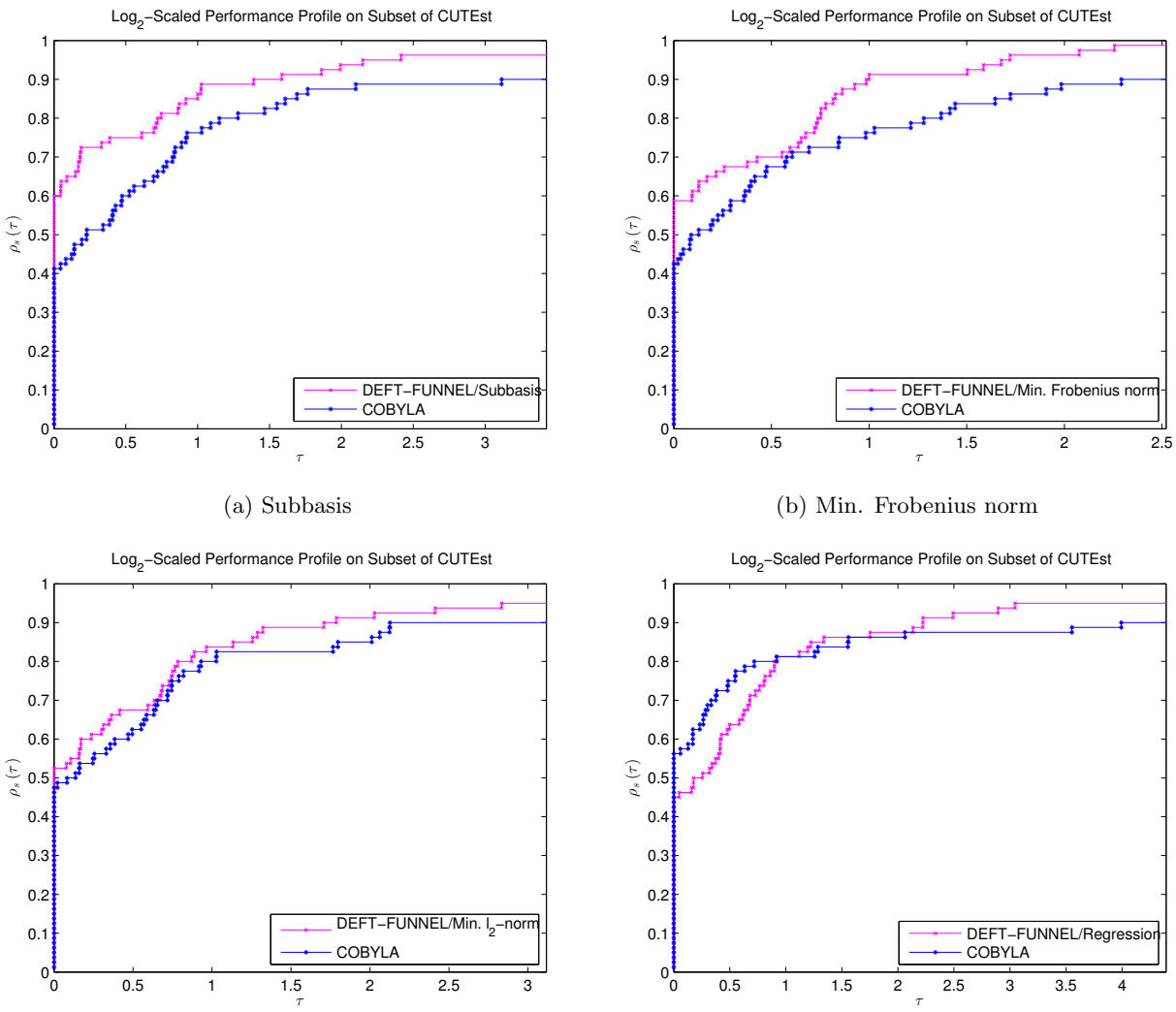
We also compare the methods by using *data profiles*. Such profiles were proposed by Moré and Wild [18] as a means to provide the user an accurate view of the performance of derivative-free solvers when the computational budget is limited. Let $t_{p,s}$ be the number of function evaluations required to solve problem p by solver s as before and denote by n_p the dimension of the vector x in the problem p. The data profile of solver $s \in \mathcal{S}$ is defined by

$$d_s(\tau) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} : \frac{t_{p,s}}{n_p + 1} \leq \tau \right\} \right|.$$

Because of the scaling choice of dividing by n_p+1, we may interpret $d_s(\tau)$ as the percentage of problems that can be solved with τ simplex gradient estimates.

In Figure 5.3, the data profiles show that the subbasis, minimum Frobenius norm and minimum ℓ_2-norm variants have similar performance to COBYLA when the computational budget is limited to a value between 0 and 20. In particular, when only 20 simplex gradient estimates are allowed, they solve roughly 80% of the problems. However, as the budget limit increases, the difference in the performance between these three variants and COBYLA becomes progressively larger, revealing the superiority of DEFT-FUNNEL. For the regression variant, it is difficult to tell which method wins from Figure 5.3d. For a very small budget (up to ~ 17 simplex gradient estimates), both solve approximately the same amount of problems. For a budget of k simplex gradients where $\kappa \in (17, 35]$, COBYLA solves more problems than DEFT-FUNNEL, while for larger values of κ DEFT-FUNNEL solves more problems.

We also tested DEFT-FUNNEL on a set of 20 small-scale linearly constrained optimization problems from CUTEst and compared its results with those of the software LINCOA, a newly developed trust-region method by Powell [21] for linearly constrained optimization without derivatives. His software combines active-set methods with truncated conjugate gradients and uses quadratic interpolation models for the objective function. The user is required to provide the Jacobian matrix and the vector of constants in the right side of the constraints as inputs, as well as a feasible starting point. Since many of CUTEst problems provide an infeasible starting point, we chose a feasible one instead for the sake of comparison. The stopping criterion used in LINCOA is the same found in COBYLA, i.e. the algorithm stops when the trust region radius $\rho \in [\rho_{end}, \rho_{beg}]$ reaches the smallest value allowed ρ_{end}.



(a) Subbasis

(b) Min. Frobenius norm

(c) Min. ℓ_2-norm

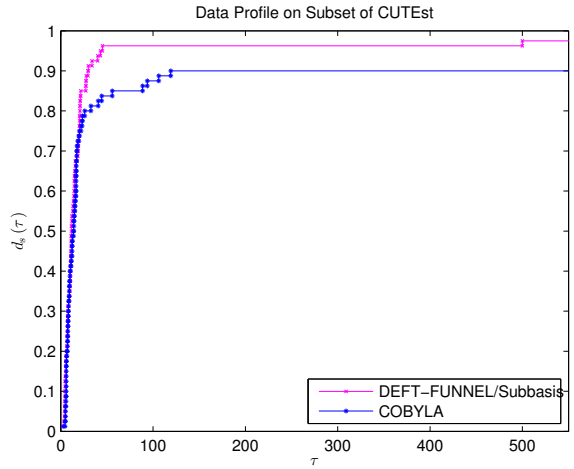
(d) Regression

Figure 5.2: *Log*$_2$-scaled performance profiles of the methods DEFT-FUNNEL (with different approaches to build the models) and COBYLA on a set of 80 problems from CUTEst.

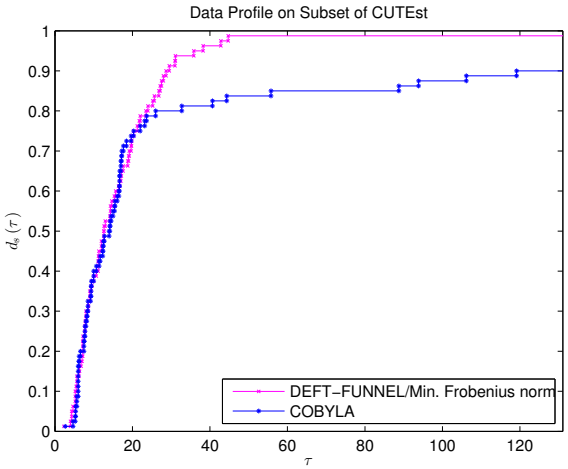
In our experiments, we set $\rho_{end} = 10^{-4}$. Regarding the parameter setting in DEFT-FUNNEL, the same values were kept for comparison with LINCOA.

In Figure 5.4, we provide the performance profiles of the four variants of DEFT-FUNNEL for the 20 linearly constrained problems, whereas, in Figure 5.5, we compare the performance of each variant of our method to LINCOA's individually. All the variants of DEFT-FUNNEL except the minimum ℓ_2-norm approach have solved all the 20 linearly constrained problems. The results reported in Figures 5.5a and 5.5b reveal a superior performance of LINCOA over DEFT-FUNNEL/Subbasis and DEFT-FUNNEL/Frobenius. On the other hand, Figure 5.5c shows that DEFT-FUNNEL/Min. ℓ_2-norm was faster than LINCOA and, in Figure 5.5d, it can be seen that DEFT-FUNNEL/Regression also was slightly faster, although the latter presented superior performance for large values of τ, which indicates more robustness.

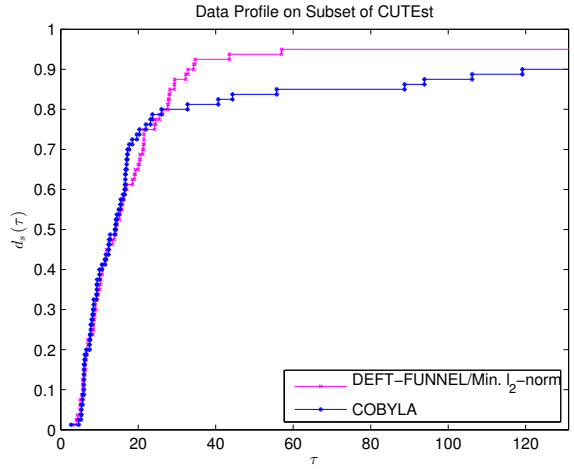
Data profiles of DEFT-FUNNEL and LINCOA are presented in Figure 5.6 and show that, for very small budgets (less than 10 simplex gradient estimates), DEFT-FUNNEL was able to solve more



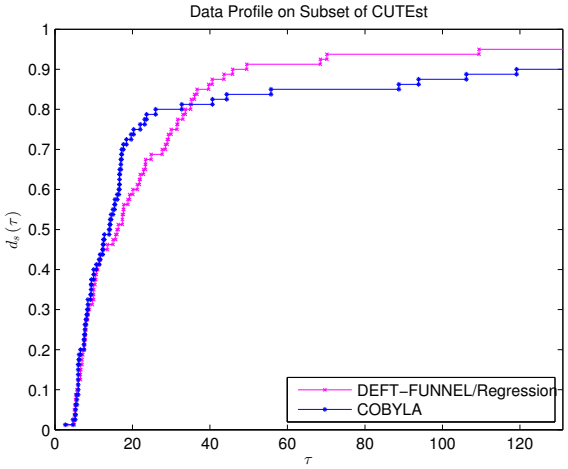
(a) Subbasis



(b) Min. Frobenius norm



(c) Min. ℓ_2-norm



(d) Regression

Figure 5.3: Data profiles of the methods DEFT-FUNNEL (with different approaches to build the models) and COBYLA on a set of 80 problems from CUTEst.

problems than LINCOA, while the latter performed better than the former for larger budgets.

We conclude this section by noticing that the performance profiles in the Figures 5.2 and 5.5 and the data profiles in the Figures 5.3 and 5.6 give clear indication that DEFT-FUNNEL provides encouraging results for small-scale nonlinear optimization problems with general nonlinear constraints. For the case where the user knows that the constraint functions are linear and he is able to provide their gradients, it might be interesting to the handle those constraints separately rather than lumping them and other general nonlinear constraints together in $c(x)$. The number of function evaluations required by each method to converge in our numerical experiments are reported in the appendix.

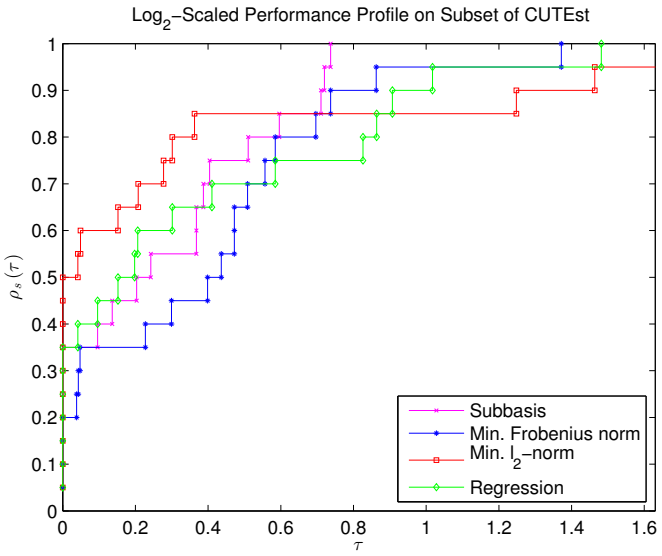
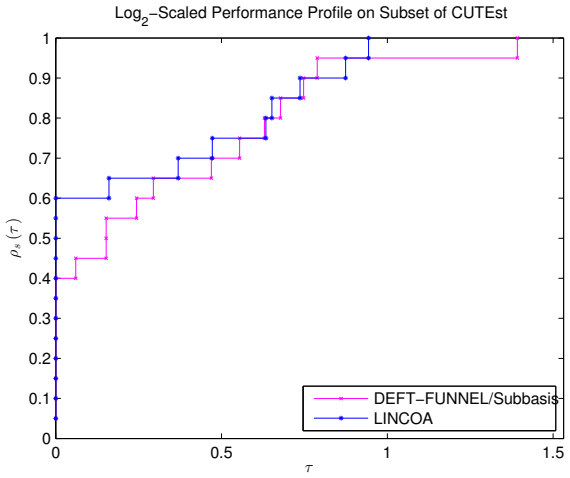
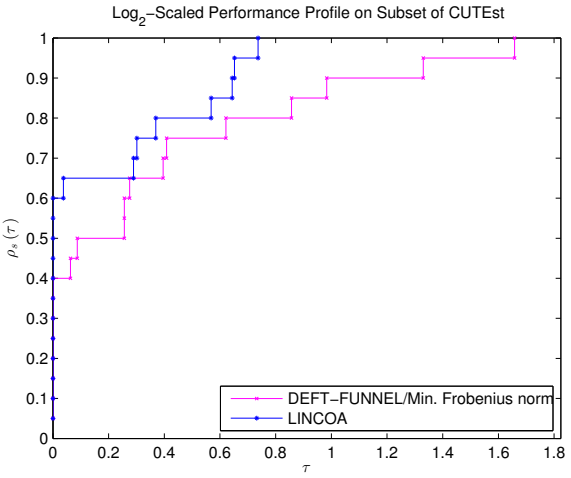


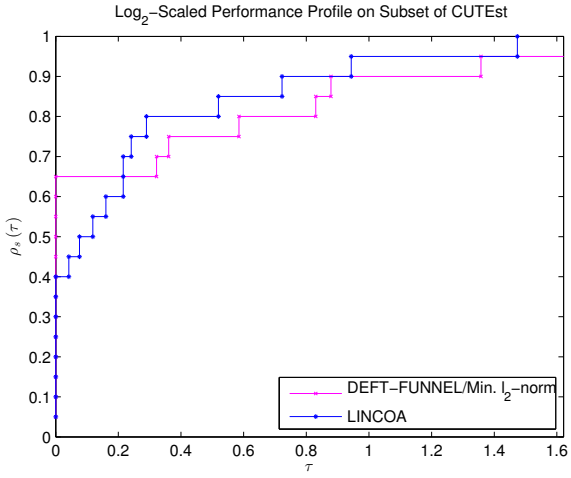
Figure 5.4: Log_2-scaled performance profiles of DEFT-FUNNEL with different approaches to build the models on a set of 20 linearly constrained problems from CUTEst.



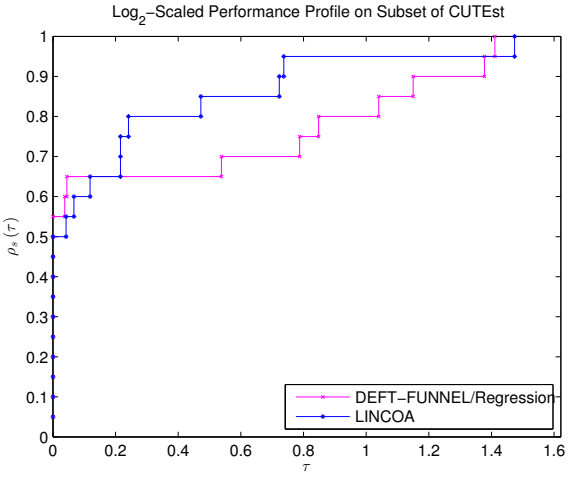
(a) Subbasis



(b) Min. Frobenius norm

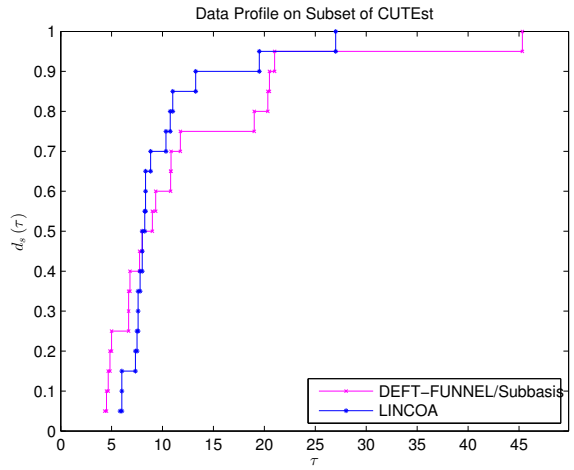


(c) Min. ℓ_2-norm

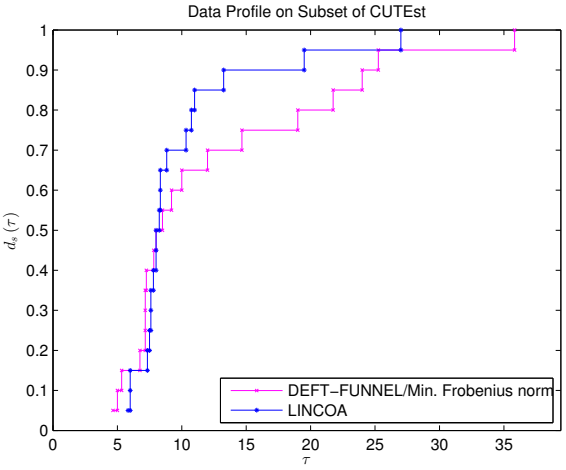


(d) Regression

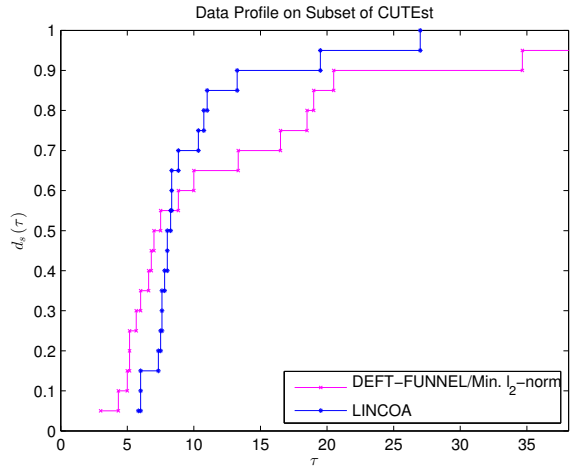
Figure 5.5: *Log*$_2$-scaled performance profiles of the methods DEFT-FUNNEL (with different approaches to build the models) and LINCOA a set of 20 linearly constrained problems from CUTEst.



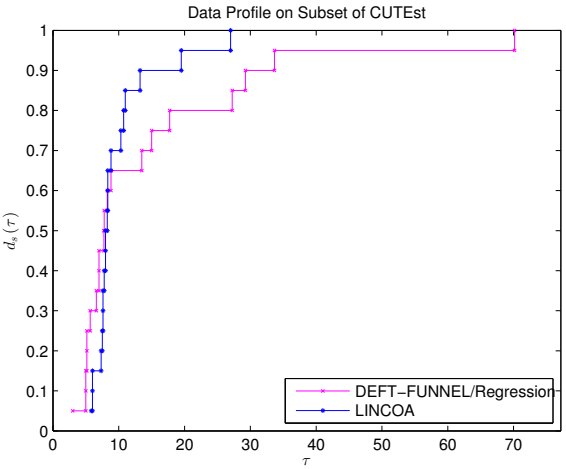
(a) Subbasis



(b) Min. Frobenius norm



(c) Min. ℓ_2-norm



(d) Regression

Figure 5.6: Data profiles of the methods DEFT-FUNNEL (with different approaches to build the models) and LINCOA on a set of 20 linearly constrained problems from CUTEst.

6 Conclusions

We have presented a SQP algorithm for solving nonlinear optimization problems with general nonlinear constraints. It extends the one introduced by Gould and Toint [11] for problems with equality constraints only. When simple bounds are given, the algorithm makes use of an active-set approach to perform minimization on the subspaces defined by the active bounds.

We also applied our algorithm to the field of derivative-free optimization by employing an ensemble of underlying techniques such as multivariate polynomial interpolation for the construction of surrogate models, a self-correcting geometry mechanism for the maintenance of interpolation set and the reduction of the dimension of the problem when entering the subspaces by using a recursive subspace minimization approach proposed in Gratton *et al.* [12]. The final method, named DEFT-FUNNEL, extends the one proposed by Sampaio and Toint [22] and makes use of neither the derivatives of the objective function nor the derivatives of the constraints. It also considers both the objective and constraints as black-box functions. Numerical experiments on a set of 80 small-scale nonlinear optimization problems with general nonlinear constraints and on a set of 20 small-scale linearly constrained optimization problems were performed and revealed good results of the new algorithm in practice.

For future research, we plan to consider the possibility of having two different interpolations sets for the objective function and the constraints and analyze its performance for cases where the constraints are linear, while the objective function is of higher degree. Convergence results are also of interest and are left as a development line for future work.

Acknowledgements

The first author gratefully acknowledges a CERUNA-UNamur scholarship.

References

[1] C. Audet and J. E. Dennis. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14(4):980–1010, 2004.

[2] E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[3] P. T. Boggs and J. W. Tolle. A strategy for global convergence in a sequential quadratic programming algorithm. *SIAM Journal on Numerical Analysis*, 26(3):600–623, 1989.

[4] B. Colson. *Trust-Region Algorithms for Derivative-Free Optimization and Nonlinear Bilevel Programming.* PhD thesis, Department of Mathematics, FUNDP - University of Namur, Namur, Belgium, 2004.

[5] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods.* MOS-SIAM Series on Optimization, Philadelphia, 2000.

[6] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization.* MPS-SIAM Book Series on Optimization, Philadelphia, 2009.

[7] F. Curtis, N. I. M. Gould, D. Robinson, and Ph. L. Toint. An interior-point trust-funnel algorithm for nonlinear optimization. Technical Report naXys-02-2014, Department of Mathematics, University of Namur, 2014.

[8] E. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–13, 2002.

[9] N. I. M. Gould, D. Orban, and Ph. L. Toint. Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, pages 1–13, 2014.

[10] N. I. M. Gould, D. P. Robinson, and Ph. L. Toint. Corrigendum: Nonlinear programming without a penalty function or a filter. Technical report, Namur Centre for Complex Systems (naXys), University of Namur, Namur, Belgium, 2011.

[11] N. I. M. Gould and Ph. L. Toint. Nonlinear programming without a penalty function or a filter. *Mathematical Programming*, 122(1):155–196, 2010.

[12] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for bound-constrained nonlinear optimization without derivatives. *Optimization Methods and Software*, 26(4-5):875–896, 2011.

[13] W. Hock and K. Schittkowski. Test example for nonlinear programming codes. *Journal of Optimization Theory and Applications*, 30(1):127–129, 1980.

[14] R. M. Lewis and V. Torczon. Pattern search algorithms for linearly constrained minimization. *SIAM Journal on Optimization*, 10:917–941, 2000.

[15] R. M. Lewis and V. Torczon. A globally convergent augmented langragian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4):1075–1089, 2002.

[16] R. M. Lewis and V. Torczon. Active set identification for linearly constrained minimization without explicit derivatives. *SIAM Journal on Optimization*, 20(3):1378–1405, 2009.

[17] S. Lucidi, M. Sciandrone, and P. Tseng. Objective-derivative-free methods for constrained optimization. *Mathematical Programming, Series A*, 92:37–59, 2002.

[18] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[19] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275, pages 51–67, Dordrecht, The Netherlands, 1994. Kluwer Academic Publishers.

[20] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.

[21] M. J. D. Powell. On fast trust region methods for quadratic models with linear constraints. Technical Report DAMTP 2014/NA02, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2014.

[22] Ph. R. Sampaio and Ph. L. Toint. A derivative-free trust-funnel method for equality-constrained nonlinear optimization. *Computational Optimization and Applications*, 61(1):25–49, 2015.

[23] K. Scheinberg and Ph. L. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM Journal on Optimization*, 20(6):3512–3532, 2010.

[24] W.-C. Yu and Y.-X. Li. A direct search method by the local positive basis for the nonlinearly constrained optimization. *Chinese Annals of Mathematics*, 2:269–280, 1981.

A Appendix

The number of function evaluations required by DEFT-FUNNEL, COBYLA and LINCOA to converge in our numerical experiments are reported here. We denote by n the number of variables in the problem and by $m_{\rm B}$, $m_{\rm EQ}$ and $m_{\rm IQ}$ the number of constraints of the type bound, equality and inequality, respectively. We indicate by "NaN" (standing for not a number) the problems where convergence to a solution was not achieved.

Table 1.2: Number of function evaluations required by DEFT-FUNNEL and COBYLA to converge on a set of 80 problems with general nonlinear constraints.

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	COBYLA
HS6	2	0	1	0	21	27	27	32	24
HS7	2	0	1	0	31	23	25	49	32
HS8	2	0	2	0	13	13	13	15	18
HS9	2	0	1	0	24	24	57	45	8
HS10	2	0	0	1	34	34	32	25	61
HS11	2	0	0	1	26	26	26	18	53
HS12	2	0	0	1	64	59	64	47	35
HS13	2	2	0	1	36	47	32	48	50
HS14	2	0	1	1	18	17	17	18	18
HS15	2	1	0	2	40	39	28	99	122
HS16	2	3	0	2	13	13	9	9	17
HS17	2	3	0	2	34	32	51	70	30
HS18	2	4	0	2	15	77	15	95	51
HS19	2	4	0	2	17	21	17	23	28
HS20	2	2	0	3	17	17	12	15	18
HS21	2	4	0	1	14	19	61	33	25
HS22	2	0	0	2	34	28	19	16	18
HS23	2	4	0	5	36	36	43	16	18
HS24	2	2	0	3	14	14	18	21	14
HS26	3	0	1	0	109	85	98	140	30
HS27	3	0	1	0	61	79	57	116	NaN
HS28	3	0	1	0	31	29	30	31	30
HS29	3	0	0	1	59	68	77	77	69
HS30	3	6	0	1	14	16	34	46	60
HS31	3	6	0	1	63	67	63	40	56
HS32	3	3	1	1	55	36	28	28	21
HS33	3	4	0	2	17	21	40	40	25
HS34	3	6	0	2	38	38	36	40	NaN
HS35	3	3	0	1	84	101	82	71	51
HS36	3	6	0	1	18	27	20	20	31
HS37	3	6	0	2	76	87	66	117	67
HS39	4	0	2	0	58	63	59	87	38
HS40	4	0	3	0	50	43	42	43	47
HS41	4	8	1	0	66	71	35	38	NaN
HS42	4	0	2	0	32	31	31	33	46
HS43	4	0	0	3	79	94	77	36	86
HS44	4	4	0	6	54	60	33	33	33
HS46	5	0	2	0	164	122	197	275	37
HS47	5	0	3	0	125	103	169	211	266
HS48	5	0	2	0	29	47	80	81	32
HS49	5	0	2	0	272	215	208	421	51
HS50	5	0	3	0	122	114	111	202	60
HS51	5	0	3	0	65	43	53	53	32
HS52	5	0	3	0	44	43	34	34	142
HS53	5	10	3	0	29	43	58	58	85
HS55	6	8	6	0	134	115	178	160	NaN
HS56	7	0	4	0	148	224	221	349	262
HS60	3	6	1	0	81	88	86	85	50
HS61	3	0	2	0	23	22	65	47	NaN
HS62	3	6	1	0	82	96	NaN	NaN	NaN
HS63	3	3	2	0	30	21	86	88	57
HS64	3	3	0	1	160	153	174	274	355
HS65	3	6	0	1	120	110	111	120	66
HS66	3	6	0	2	60	45	97	126	58
HS71	4	8	1	1	104	99	73	109	62

Table 1.3: Number of function evaluations required by DEFT-FUNNEL and COBYLA to converge on a set of 80 problems with general nonlinear constraints. (Continued)

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	COBYLA
HS73	4	4	1	2	35	40	80	180	41
HS76	4	4	0	3	45	50	34	39	57
HS77	5	0	2	0	166	141	168	171	132
HS78	5	0	3	0	66	67	62	62	102
HS79	5	0	3	0	106	76	74	149	93
HS80	5	10	3	0	59	67	91	113	100
HS81	5	10	3	0	116	257	342	NaN	156
HS93	6	6	0	2	302	313	240	284	834
HS100LNP	7	0	2	0	174	236	NaN	NaN	133
HS106	8	16	0	6	NaN	NaN	NaN	985	NaN
HS108	9	1	0	13	146	176	200	201	161
HS113	10	0	0	8	225	342	323	257	203
BT1	2	0	1	0	NaN	86	NaN	NaN	18
BT2	3	0	1	0	118	124	129	198	223
BT3	5	0	3	0	40	86	62	63	74
BT4	3	0	2	0	32	31	22	49	31
BT5	3	0	2	0	30	21	20	21	NaN
BT6	5	0	2	0	203	163	176	238	51
BT7	5	0	3	0	86	86	83	166	139
BT8	5	0	2	0	63	88	69	106	56
BT9	4	0	2	0	58	63	59	87	77
BT10	2	0	2	0	7	7	19	24	17
BT11	5	0	3	0	89	92	127	220	118
BT12	5	0	3	0	65	115	129	48	563
BT13	5	1	1	0	NaN	161	158	40	637

Table 1.4: Number of function evaluations required by DEFT-FUNNEL and LINCOA to converge on a set of 20 linearly constrained problems.

Problem	n	m_{B}	m_{EQ}	m_{IQ}	Subbasis	Min. Frobenius norm	Min. ℓ_2-norm	Regression	LINCOA
HS9	2	0	1	0	24	24	57	45	31
HS21	2	4	0	1	13	16	13	15	25
HS24	2	2	0	3	14	14	18	21	22
HS28	3	0	1	0	31	29	30	31	43
HS35	3	3	0	1	84	101	82	71	32
HS36	3	6	0	1	18	27	20	20	33
HS37	3	6	0	2	76	87	66	117	44
HS44	4	4	0	6	54	60	33	33	39
HS48	5	0	2	0	29	47	80	81	45
HS49	5	0	2	0	272	215	208	421	152
HS50	5	0	3	0	122	114	111	202	117
HS51	5	0	3	0	65	43	53	53	53
HS52	5	0	3	0	56	88	34	34	35
HS53	5	10	3	0	40	43	31	31	36
HS62	3	6	1	0	82	96	NaN	109	53
HS76	4	4	0	3	45	50	34	39	38
BT3	5	0	3	0	40	43	31	31	36
HATFLDH	4	8	0	13	34	46	35	35	38
STANCMIN	3	3	0	2	47	34	40	33	32
SIMPLLPA	2	2	0	2	15	15	9	9	25