A Unified Funnel Restoration SQP Algorithm*

David Kiessling[1], Sven Leyffer[2], and Charlie Vanaret[2,3]

[1]Department of Mechanical Engineering, KU Leuven and Flanders Make @ KU Leuven, Leuven, Belgium
[2]Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA
[3]Mathematical Algorithmic Intelligence Division, Zuse-Institut Berlin, Germany

Abstract

We consider nonlinearly constrained optimization problems and discuss a generic double-loop framework consisting of four algorithmic ingredients that unifies a broad range of nonlinear optimization solvers. This framework has been implemented in the open-source solver `Uno`, a Swiss Army knife-like C++ optimization framework that unifies many nonlinearly constrained nonconvex optimization solvers. We illustrate the framework with a sequential quadratic programming (SQP) algorithm that maintains an acceptable upper bound on the constraint violation, called a funnel, that is monotonically decreased to control the feasibility of the iterates. Infeasible quadratic subproblems are handled by a feasibility restoration strategy. Globalization is controlled by a line search or a trust-region method. We prove global convergence of the trust-region funnel SQP method, building on known results from filter methods. We implement the algorithm in `Uno`, and we provide extensive test results for the trust-region line-search funnel SQP on small `CUTEst` instances.

1 Introduction and Background

We focus on algorithms for solving nonlinearly constrained optimization problems (NCOs) of the form

$$\min_{x \in \mathbb{R}^n} f(x), \quad \text{s.t.} \quad c(x) = 0, \quad x \geq 0, \tag{NCO}$$

*This work has been carried out within the framework of the Flanders Make SBO project DIRAC: Deterministic and Inexpensive Realizations of Advanced Control. This work was also supported by the Applied Mathematics activity within the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.

where $f\colon \mathbb{R}^n \to \mathbb{R}$ and $c\colon \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable and possibly nonconvex. Problems with general inequalities $l \leq c(x) \leq u$ with $l, u \in \mathbb{R}^m$ can be formulated as NCO via the introduction of slack variables.

NCOs arise in many important applications, such as optimal control problems [3], partial differential equation constrained optimization problems [1, 20] that model topology optimization [2], inverse problems [4], or control problems [27]. In addition, NCOs arise as subproblems in more complex optimization problems such as mixed-integer nonlinear optimization [22] and optimization problems with complementarity constraints [25].

Solution methods for NCOs are iterative and generate a sequence of iterates $x^{(k)}$ for $k \geq 0$ that (hopefully) converges to a stationary point of NCO or a stationary point of the constraint violation $\|c(x)\|$ under mild assumptions when started near a stationary point. In general, however, we must safeguard our methods to ensure convergence from remote starting points (which we refer to as "global convergence" in the remainder). Iterative methods for NCOs share common algorithmic features, such as how new iterates are computed and how global convergence is ensured.

Here we present a generic framework for solving NCOs as a double-loop algorithm. The outer loop computes new iterates that converge to a stationary point, while the inner loop implements the convergence safeguards. This perspective allows us to easily interpret many existing NCO methods within this framework and *forms the basis of an open-source C++ implementation that unifies many existing solvers for NCOs, called* `Uno` [28, 29]. `Uno` is a reliable and modular solver for NCO that is meant to be extended to other areas such as optimization with equilibrium constraints (see, e.g., [12, 13, 25]) or robust optimization [23].

In this paper we concentrate on `funnel methods` to promote global convergence. We relate the funnel idea to filter methods [11, 14] and illustrate how it fits into the double-loop framework. This allows us to quickly develop a funnel implementation by modifying the existing filter implementation within `Uno`. The funnel method has been studied in [18, 19] for solving equality-constrained NCOs with a Byrd–Omojokun trust-region method. An iteration consists of the solution of several subproblems that yield tangential and normal step decomposition. Iterations are divided into three different kinds, namely, f-type, v-type, and y-type iterations, which account for an improvement in optimality, a reduction of infeasibility, or an update of the Lagrange multipliers. In his Ph.D. thesis [26], Samedi used a funnel in the context of equality-constrained optimization to design an algorithm with favorable worst-case complexity bounds. The first funnel method for solving inequality-constrained NCOs was introduced in [6]: the problem was transformed into a barrier problem and solved with a Lagrange–Newton method. The approach uses matrix-free and inexact methods and performs well on large-scale NCOs. Funnel methods can be traced back to the Ph.D. thesis of Zoppke-Donaldson [32] that implements an SQP method that employs a tolerance tube and reports encouraging numerical results but without any convergence analysis.

We look at funnel methods through the lens of filter methods, from which we derive our global convergence proof. We wish to demonstrate that the funnel method obtains performance similar to that of the filter method, while being simpler to implement. Filter methods were introduced in [11] in the context of a trust-region SQP algorithm that switches to a feasibility restoration phase when the quadratic subproblem is infeasible. The method was implemented in the solver `filterSQP`, which obtained excellent performance on the `CUTEst` [17] test set. A filter line-search interior-point method was implemented within the solver `IPOPT`, one of the most successful open-source NCO

solvers [31]. Respective global convergence proofs were given in [14], [10], and [30]. The latter two papers serve as a baseline for proving global convergence results.

Contributions. We make a number of important contributions in this paper: (i) we investigate the funnel method from the perspective of filter methods; (ii) we consider equality and inequality-constrained problems (if necessary by introducing additional slack variables); (iii) we prove global convergence of a trust-region restoration funnel method; and (iv) we implement a trust-region restoration funnel method and a line-search restoration funnel method in the `Uno` solver, and we provide extensive numerical results on a subset of the `CUTEst` test set. To our knowledge, this is the first time that line-search funnel methods have been considered.

Outline. This paper is structured as follows. In the remainder of this section we introduce our notation and necessary optimality conditions. Next, we present the double-loop framework and the main algorithmic components that make up a generic NCO method, using the funnel method as an exemplar. We then review the funnel method in more detail and prove global convergence. We present numerical results on a subset of the `CUTEst` test set (with detailed tables of function and derivative evaluations for the trust-region and line-search methods in the appendices). In the final section we summarize our conclusions.

Notation and Necessary Optimality Conditions

The jth component of a vector $x \in \mathbb{R}^n$ is denoted by a subscript, x_j. The kth iteration index is given by a superscript, $x^{(k)}$. For brevity, quantities evaluated at a given iterate $x^{(k)}$ are denoted by a superscript as well, for example, $f^{(k)} \overset{\text{def}}{=} f(x^{(k)})$.

We start by defining the Fritz John function (or scaled Lagrangian) of (NCO) at (x, ρ, λ, μ):

$$\mathcal{L}(x, \rho, \lambda, \mu) \overset{\text{def}}{=} \rho f(x) - \lambda^T c(x) - \mu^T x = \rho f(x) - \sum_{j=1}^{m} \lambda_j c_j(x) - \sum_{i=1}^{n} \mu_i x_i, \tag{1}$$

where $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^n$ are the Lagrange multipliers of the equality constraints and the bound constraints, respectively, and $\rho \in \{0, 1\}$. We use the scaled Lagrangian because it allows us to treat feasible and infeasible stationary points of (NCO) in a unified way. When $\rho = 1$, we obtain the standard Lagrangian. The gradient of the scaled Lagrangian with respect to x at a point (x, ρ, λ, μ) is denoted by

$$\nabla_x \mathcal{L}(x, \rho, \lambda, \mu) \overset{\text{def}}{=} \rho \nabla f(x) - \sum_{j=1}^{m} \lambda_j \nabla c_j(x) - \mu, \tag{2}$$

where $\nabla f(x) \in \mathbb{R}^n$ is the gradient of f and $\nabla c(x)^T \in \mathbb{R}^{m \times n}$ is the Jacobian of c.

The Hessian of the scaled Lagrangian with respect to x at (x, ρ, λ) is defined by

$$W_\rho(x, \lambda) \overset{\text{def}}{=} \nabla^2_{xx} \mathcal{L}(x, \rho, \lambda, \mu) = \rho \nabla^2 f(x) - \sum_{j=1}^{m} \lambda_j \nabla^2 c_j(x), \tag{3}$$

where $\nabla^2 f(x) \in \mathbb{R}^{n \times n}$ is the Hessian of f and $\nabla^2 c_j(x) \in \mathbb{R}^{n \times n}$ is the Hessian of c_j.

We say that a feasible point x of (NCO) fulfills the Mangasarian–Fromowitz constraint qualification (MFCQ) if the gradients of the constraints $c(x)$ are linearly independent at x and if there

exists $d \in \mathbb{R}^n$ such that $\nabla c(x)^T d = 0$ and $d_i > 0$ if $x_i = 0$ (d points into the interior of the feasible set). Necessary optimality conditions for (NCO) are given in the following definition.

Definition 1 (KKT conditions [24])**.** *If MFCQ holds at an optimal point x^* of* (NCO)*, the first-order necessary optimality conditions of* (NCO) *at x^* state that there exist multipliers $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^n$ such that*

$$\nabla_x \mathcal{L}(x^*, 1, \lambda^*, \mu^*) = 0, \quad c(x^*) = 0, \quad x^* \geq 0, \quad \mu^* \geq 0, \quad x^* \odot \mu^* = 0,$$

where $\odot$ denotes the Hadamard (componentwise) product. These conditions are called the Karush–Kuhn–Tucker (KKT) conditions.

2 Unified Abstraction of Nonlinearly Constrained Optimization

In [29] we introduced an abstract framework to unify the workflows of iterative methods for NCO, arguing that most methods can be assembled by combining the following four generic ingredients within a double-loop framework:

1. A constraint relaxation strategy is a systematic way to reformulate (NCO) with relaxed constraints,for example, feasibility restoration or ℓ_1 relaxation.
2. A subproblem method is a local approximation of the reformulated NCO at the current primal-dual iterate, for example, inequality-constrained quadratic problems (QPs) in an SQP framework or a primal-dual interior-point Newton system.
3. A globalization strategy assesses whether a trial iterate makes sufficient progress toward a solution, for example, filter method or ℓ_1 merit function.
4. A globalization mechanism defines the action that an algorithm takes when a trial iterate is not acceptable, for example, line-search or trust-region method.

The double-loop framework portrayed in Algorithm 1 shows how the four ingredients interact with one another. This abstract framework is implemented in `Uno`; it offers robust, off-the-shelf strategies that are independent and agnostic of each other. Strategy combinations can be assembled on the fly with no programming effort from the user. In particular, `Uno` implements three presets that mimic existing solvers: a filterSQP [11] preset (a trust-region restoration filter SQP method); an IPOPT [30, 31] preset (a line-search restoration filter barrier method); and a Byrd [5] preset (a line-search ℓ_1-merit Sℓ_1QP method).

This paper uses the same abstraction as in [29] but restricts the presentation to SQP methods solving inequality-constrained QP subproblems and feasibility restoration to ensure consistent QPs. The funnel method is investigated as a globalization strategy for different algorithmic configurations and unified both for trust-region and line-search methods. In the remainder of this section we provide details of these algorithmic ingredients.

Algorithm 1: Abstract double-loop framework for iterative methods for (NCO).

Data: initial point $(x^{(0)}, \lambda^{(0)}, \mu^{(0)})$
Set $k \leftarrow 0$
while *termination criteria at* $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$ *not met* **do**
  repeat *globalization mechanism*
    Solve feasible subproblem(s) that approximate(s) (NCO) at $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$
    Assemble trial iterate $(\hat{x}^{(k+1)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$
  until $(\hat{x}^{(k+1)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$ *is acceptable*
  Update $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)}) \leftarrow (\hat{x}^{(k+1)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$
  $k \leftarrow k + 1$
Result: $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$

2.1 Subproblem Method: Sequential Quadratic Programming

SQP is a second-order iterative method for finding a local solution for (NCO). At iteration k, it solves a local quadratic approximation of (NCO) at the primal-dual iterate $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$:

$$\begin{aligned} \min_{d} \quad & \tfrac{1}{2} d^T W_1^{(k)} d + (\nabla f^{(k)})^T d \\ \text{s.t.} \quad & c^{(k)} + (\nabla c^{(k)})^T d = 0 \\ & x^{(k)} + d \geq 0. \end{aligned} \tag{QP($x^{(k)}$)}$$

The primal-dual solution of (QP($x^{(k)}$)) is denoted by $(d^{(k)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$. The trial iterate for a given step size $\alpha \in (0, 1]$ is given by $\hat{x}^{(k+1)} = x^{(k)} + \alpha d^{(k)}$. If the trial iterate is accepted by the globalization strategy, we move to the trial iterate and solve the next QP. Otherwise, the globalization mechanism (e.g., a line search) defines a new trial iterate that is more likely to be acceptable. The process terminates either when an approximate stationary (KKT) point is found or when a stationary point of the constraint violation is found or with an indication that a constraint qualification fails.

Under appropriate assumptions and close to a solution, SQP cnverges with $\alpha = 1$ and achieves superlinear or quadratic local convergence [24]. Far from the solution, additional safeguards need to be taken to ensure global convergence. SQP can suffer from ill-posedness; if the exact Hessian is indefinite, the QP can be unbounded. Moreover, the linearization of the constraints in (NCO) can be inconsistent. Both cases yield iterations that are not well defined. Additionally, taking full steps ($\alpha = 1$) does not necessarily result in a converging method. A globalization mechanism needs to be incorporated, as well as a globalization strategy that ensures descent for the objective function and improvement in constraint violation. The next section discusses various methods to address these issues.

2.2 Globalization Mechanism: Trust Region or Line Search

Standard globalization mechanisms include trust-region methods and line-search methods. We discuss each scheme in turn and show that they can be interpreted as inner iterations.

2.2.1 Trust-Region Methods

Trust-region methods limit the length of the direction d a priori by imposing the trust-region constraint $\|d\| \leq \Delta_{\mathrm{TR}}^{(l)}$, where $\Delta_{\mathrm{TR}}^{(l)} > 0$ is the trust-region radius. Various norms are possible; in this paper we consider only the ℓ_∞ norm because it is most easily implemented within a QP subproblem. The step $d^{(k,l)}$ is obtained by solving the trust-region subproblem at the current primal-dual iterate $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$:

$$\begin{array}{ll} \min\limits_{d} & \frac{1}{2} d^T W_1^{(k)} d + (\nabla f^{(k)})^T d \\ \text{s.t.} & c^{(k)} + (\nabla c^{(k)})^T d = 0 \\ & x^{(k)} + d \geq 0 \\ & \|d\|_\infty \leq \Delta_{\mathrm{TR}}^{(l)}. \end{array} \qquad (QP(x^{(k)}, \Delta_{\mathrm{TR}}^{(l)}))$$

The radius $\Delta_{\mathrm{TR}}^{(l)}$ is reduced until the trial iterate $\hat{x}^{(k+1,l)} \stackrel{\mathrm{def}}{=} x^{(k)} + d^{(k,l)}$ is accepted by the globalization strategy or until $(QP(x^{(k)}, \Delta_{\mathrm{TR}}^{(l)}))$ becomes infeasible. Usually, $\Delta_{\mathrm{TR}}^{(l)}$ is increased in successful iterations if the trust region is active at $d^{(k,l)}$, and it is decreased to a value smaller than $\min(\Delta_{\mathrm{TR}}^{(l)}, \|d^{(k,l)}\|_\infty)$ when the trial iterate is rejected. A positive definite Hessian $W_1^{(k)}$ is not required (as long as the QP solver can handle problems with an indefinite Hessian), because directions of negative curvature are bounded by the trust region.

2.2.2 Line Search Methods

Line-search methods solve $\mathrm{QP}(x^{(k)})$ and search for an acceptable iterate along $d^{(k)}$ by varying the step size $\alpha^{(k,l)} \in (0, 1]$: the trial iterate is denoted by $\hat{x}^{(k+1,l)} \stackrel{\mathrm{def}}{=} x^{(k)} + \alpha^{(k,l)} d^{(k)}$. Here we opt for a backtracking line search that seeks the largest step size in the sequence $a^{(k,l)} \in \{2^{-l} \mid l = 0, 1, \ldots\}$ such that the trial iterate is acceptable to the globalization strategy. A positive definite approximation of the Hessian $W^{(k)}$ is required to guarantee the well-posedness of $\mathrm{QP}(x^{(k)})$.

The key difference with trust-region methods is that no additional QPs need be solved if a trial iterate is rejected. On the other hand, the step $\alpha^{(k,l)} d^{(k)}$ is usually not optimal (or even feasible) within the equivalent trust region, $\|d\|_\infty \leq \alpha^{(k,l)} \|d^{(k)}\|_\infty$.

2.3 Constraint Relaxation Strategy: Feasibility Restoration

An infeasible quadratic subproblem results from inconsistent linearized or bound constraints. This situation can indicate that (NCO) is infeasible. In this case the method reverts to the *feasibility restoration phase*: the original objective is temporarily discarded, and the following feasibility problem (e.g., with the ℓ_1 norm) is solved instead:

$$\begin{array}{ll} \min\limits_{x} & \|c(x)\|_1 \\ \text{s.t.} & x \geq 0. \end{array} \qquad (4)$$

Other subproblems are also possible and do not influence the proof of global convergence. Feasibility restoration improves feasibility until a minimum of the constraint violation is obtained or the subproblem becomes feasible again, in which case solving the original problem (the *optimality phase*) is resumed. Any (local) solution $x^* \geq 0$ of (4) with $\|c(x^*)\|_1 > 0$ is an indication that

(NCO) is (locally) infeasible. Because (4) is essentially a bound-constrained problem, we will not analyze the convergence to infeasible stationary points here and instead assume without loss of generality that a convergent globalization is available.

In our implementation we use a smooth reformulation of the ℓ_1 feasibility problem (possibly with a trust-region constraint). The feasible quadratic subproblem is given by

$$\begin{aligned} \min_{d,u,v} \quad & \tfrac{1}{2} d^T W_0^{(k)} d \;+\; e^T u + e^T v \\ \text{s.t.} \quad & c^{(k)} + (\nabla c^{(k)})^T d \;- u + v = 0 \\ & x^{(k)} + d \geq 0, \quad \left(\|d\|_\infty \leq \Delta_{\mathrm{TR}}^{(l)} \right) \\ & u \geq 0,\; v \geq 0, \end{aligned} \tag{$FQP(x^{(k)}, \Delta_{\mathrm{TR}}^{(l)})$}$$

where we indicate the presence/absence of the trust-region constraint for trust-region/line-search methods, respectively. Here e is a vector of ones of appropriate size. We note that introducing the elastic variables u, v makes the constraint Jacobian full rank, which guarantees linear independence constraint qualification. Moreover, we add the trust-region bound only to the original variables, thus ensuring that $(FQP(x^{(k)}, \Delta_{\mathrm{TR}}^{(l)}))$ is feasible for any trust-region radius. For the line-search variant, we will refer to $(FQP(x^{(k)}))$.

2.4 Globalization Strategies

Constrained optimization is concerned with the realization of two competing goals: minimizing the constraint violation and minimizing the objective value. Filter methods [11, 14] interpret (NCO) as a bi-objective optimization problem. Instead of combining the objective and the constraint violation in a merit function, they decompose the optimization problem into two separate goals: reducing the objective function and reducing the constraint violation (the latter takes precedence). The great advantage is that they do not require a priori knowledge or the update of a penalty parameter. Funnel methods adaptively define an acceptable threshold of constraint violation and accept steps that satisfy a sufficient decrease condition either for the constraint violation or for the objective. The rationale for funnel methods is that close to a local minimum and near the feasible set, we can expect the QP model to be a good predictor for the decrease of the objective, while far from the feasible set the QP step is more likely to reduce the constraint violation. Both the funnel and filter methods employ a natural switching condition that is capable of recognizing these two scenarios. In the following, we first discuss the common progress measure for the funnel and filter methods, then introduce them in detail, and highlight their close connections and similarities.

2.4.1 Progress Measures

Without loss of generality, the bound constraints on x are always feasible throughout SQP iterations. To quantify progress regarding the constraint violation or the objective, we monitor the measure of infeasibility $h(x) \stackrel{\text{def}}{=} \|c(x)\|_1$ and the objective $f(x)$ throughout the optimization process. Local models of $h(x)$ and $f(x)$ at an iterate $x^{(k)}$ are defined by

$$m_h^{(k)}(d) \stackrel{\text{def}}{=} \left\| c^{(k)} + (\nabla c^{(k)})^T d \right\|_1, \tag{5a}$$

$$m_f^{(k)}(d) \stackrel{\text{def}}{=} \frac{1}{2} d^T W_1^{(k)} d + (\nabla f^{(k)})^T d + f^{(k)}. \tag{5b}$$

We define the respective *predicted reductions* by

$$\Delta m_h^{(k)}(d) \stackrel{\text{def}}{=} m_h^{(k)}(0) - m_h^{(k)}(d) = h^{(k)} - m_h^{(k)}(d), \tag{6a}$$

$$\Delta m_f^{(k)}(d) \stackrel{\text{def}}{=} m_f^{(k)}(0) - m_f^{(k)}(d) = -\frac{1}{2} d^T W_1^{(k)} d - (\nabla f^{(k)})^T d. \tag{6b}$$

The *actual reductions* are defined by

$$\Delta f^{(k)}(d) \stackrel{\text{def}}{=} f^{(k)} - f(x^{(k)} + d), \tag{7a}$$

$$\Delta h^{(k)}(d) \stackrel{\text{def}}{=} h^{(k)} - h(x^{(k)} + d). \tag{7b}$$

2.4.2 Funnel Method

A funnel (illustrated in Figure 1) describes a relaxation of the feasible set that allows a constraint violation up to a given upper bound $\tau^{(k)} > 0$. At iteration k, a necessary condition for acceptance of the trial iterate is the *funnel condition*

$$h(x) \leq \tau^{(k)}. \tag{8}$$

The initial funnel width is given by

$$\tau^{(0)} = \max\left[\bar{\tau}, \bar{\kappa} h^{(0)}\right] \tag{9}$$

with $\bar{\tau} > 0$ and $\bar{\kappa} > 1$ to ensure that the initial point is acceptable. All iterates stay within the funnel whose width is monotonically non-increasing; that is, $\tau^{(k+1)} \leq \tau^{(k)}$ for all $k \geq 0$. This property combined with a constraint relaxation strategy controls the feasibility of the iterates and ensures a feasible limit point.

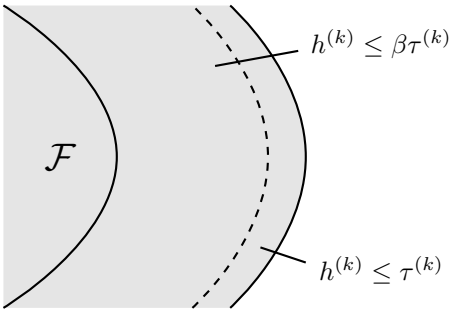


Figure 1: Funnel (in gray) around the feasible set $\mathcal{F}$. The frontier of the funnel envelope (12) is shown as a dashed curve.

Similarly to filter methods, a switching condition (with $\delta \in (0, 1)$)

$$\Delta m_f^{(k)}(d) \geq \delta (h^{(k)})^2 \tag{10}$$

ensures that the algorithm does not take infinitely small steps and thus avoids convergence toward infeasible points. It distinguishes between two types of iterations: f-type iterations that improve optimality and h-type iterations that improve feasibility:

- If the switching condition (10) is satisfied, the trial iterate is accepted if an Armijo-type sufficient decrease condition

$$\Delta f^{(k)} \geq \sigma \Delta m_f^{(k)}(d) \tag{11}$$

holds with $\sigma \in (0,1)$, that is, if the current iterate yields a sufficient decrease of the objective function. This is an f-type iteration.

- If the switching condition (10) is violated and if the funnel sufficient decrease condition

$$h(\hat{x}^{(k+1,l)}) \leq \beta \tau^{(k)} \tag{12}$$

holds with $\beta \in (0,1)$, the trial iterate is accepted, and the funnel width is decreased:

$$\tau^{(k+1)} = (1-\kappa) h(\hat{x}^{(k+1,l)}) + \kappa \tau^{(k)} \tag{13}$$

for $\kappa \in (0,1)$. This is an h-type iteration.

Otherwise, if none of the above conditions is satisfied, the step is rejected, and either the trust-region radius or the step size is reduced. The complete funnel method is summarized in Algorithm 2 in the context of feasibility restoration. Here, x^{resto} is the point at which the algorithm switches from the optimality phase to the feasibility restoration phase; it is set in Algorithms 4 and 5. In the feasibility restoration phase, we simply enforce the Armijo condition on the constraint violation.

Algorithm 2: Restoration funnel acceptance test

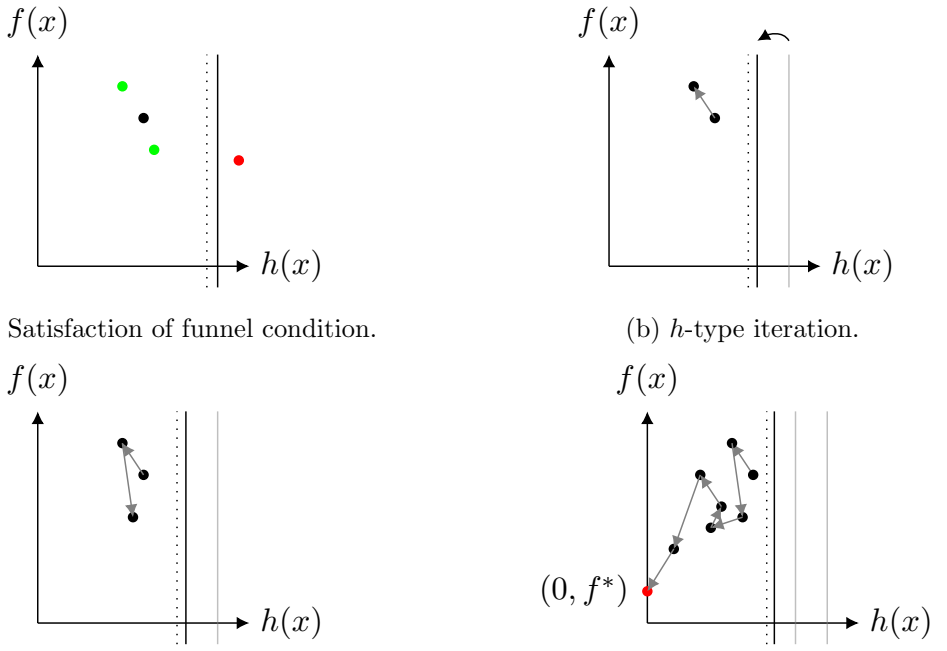
Input: trial iterate $\hat{x}^{(k+1)}$, direction $d^{(k,l)}$

acceptable $\leftarrow$ *false* — *constraint relaxation*
if $\|d^{(k,l)}\| = 0$ **then**
  acceptable $\leftarrow$ *true* — `// KKT point found`
else
  if *phase* = *Restoration and subproblem feasible and* $h(\hat{x}^{(k+1,l)}) \leq \beta \min(\tau^{(k)}, h(x^{resto}))$ **then**
    phase $\leftarrow$ Optimality
    $\tau^{(k+1)} \leftarrow (1-\kappa) h(\hat{x}^{(k+1)}) + \kappa \tau^{(k)}$ — *globalization strategy*
  if *phase* = *Optimality* **then**
    if *trial iterate is acceptable to funnel:* $h(\hat{x}^{(k+1)}) \leq \tau^{(k)}$ **then** — *globalization strategy*
      if *switching condition is satisfied:* $\Delta m_f^{(k)}(d) \geq \delta \left(h^{(k)}\right)^2$ **then** — `// f-type step`
        if *Armijo condition is satisfied:* $f^{(k)} - f(\hat{x}^{(k+1)}) \geq \sigma \Delta m_f^{(k)}(d)$ **then**
          acceptable $\leftarrow$ *true*
      else if *funnel is sufficiently reduced:* $h(\hat{x}^{(k+1)}) \leq \beta \tau^{(k)}$ **then** — `// h-type step`
        acceptable $\leftarrow$ *true*
        $\tau^{(k+1)} \leftarrow (1-\kappa) h(\hat{x}^{(k+1)}) + \kappa \tau^{(k)}$
  else if *phase* = *Restoration* **then**
    if *Armijo condition is satisfied:* $h^{(k)} - h(\hat{x}^{(k+1)}) \geq \sigma \Delta m_h^{(k)}(d)$ **then** — *globalization strategy*
      acceptable $\leftarrow$ *true*

return *acceptable*

Inspired by [26], the principle of the funnel is illustrated in Figure 2. In Figure 2a, the funnel width is the solid vertical line, and the funnel envelope is represented by the dotted line. The

black dot is the current iterate, the green dots show possible acceptable iterates, and the red dot lies outside of the funnel and is rejected. Figure 2b represents an h-type iteration: the switching condition is not satisfied, but the funnel sufficient decrease condition is satisfied. The trial iterate is accepted, and the funnel width is decreased according to (13). This guarantees that the feasibility of the iterates is driven to zero. The previous funnel width is shown in gray. Figure 2c represents an f-type iteration: both the switching condition and the Armijo sufficient decrease condition are satisfied. The trial iterate is accepted, but the funnel width is not updated. Figure 2d shows the convergence of the funnel method to the optimal solution.



(a) Satisfaction of funnel condition.

(b) h-type iteration.

(c) f-type iteration.

(d) Convergence of funnel method.

Figure 2: Illustration of the funnel method [26].

A note on the funnel reduction mechanism. In the first paper on funnel methods [19], the funnel update was given by

$$\tau^{(k+1)} = \max \left[\beta \tau^{(k)}, (1-\kappa) h(\hat{x}^{(k+1,l)}) + \kappa h^{(k)} \right]. \tag{14}$$

This keeps the iterates within the funnel if $h(\hat{x}^{(k+1,l)}) \leq h^{(k)}$. In our case, this cannot be guaranteed since we also allow steps that increase both optimality and infeasibility. We tried out different update strategies inspired by [19], but all had similar performance. Therefore, we opted for (13), which is the second term in the max. More important is the choice of the parameter κ. If κ is large such that the funnel width is slowly decreased, the funnel method allows for too much non-monotonicity, and performance can be degraded. We note that we decrease the funnel width only in h-type iterations.

2.4.3 Filter Methods

Using the notion of [30], we can define a filter as a taboo region in the $\{(h, f) \in \mathbb{R}^2 \colon\ h \geq 0\}$ half-plane, defined by a list of points $(h(x_p), f(x_p))$ (typically, previous iterates). The filter at iteration k is denoted by $\mathcal{F}^{(k)}$. A trial iterate $\hat{x}^{(k+1,l)}$ is acceptable to the filter if it sufficiently decreases feasibility or optimality (or both),

$$h(\hat{x}^{(k+1,l)}) \leq \beta h(x_p) \quad \text{or} \quad f(\hat{x}^{(k+1,l)}) \leq f(x_p) - \gamma h(\hat{x}^{(k+1,l)}) \quad \text{for all} \quad (h(x_p), f(x_p)) \in \mathcal{F}^{(k)},$$

where $\beta, \gamma > 0$. The first condition is similar to (12), and the second condition is a sufficient decrease of the objective. During the optimization process, it is ensured that every iterate does not lie within the taboo region. Often the filter is initialized with an upper bound $h_{\max}$ on the constraint violation that mimics a filter entry $(h_{\max}, -\infty)$. The acceptability of a trial iterate with respect to $h_{\max}$ is given by

$$h(\hat{x}^{(k+1,l)}) \leq \beta h_{\max}. \tag{15}$$

Filter methods also distinguish between f-type and h-type iterations by means of a switching condition. In the case of an f-type iteration, an Armijo condition checks for sufficient decrease, and the filter remains unchanged. In h-type iterations, the trial iterate is accepted if it satisfies sufficient reduction with respect to the current iterate:

$$f(\hat{x}^{(k+1)}) \leq f^{(k)} - \gamma h(\hat{x}^{(k+1)}) \quad \text{or} \quad h(\hat{x}^{(k+1)}) \leq \beta h^{(k)},$$

in which case the current iterate is added to the filter. This prevents the algorithm from cycling. We note that in the unconstrained case, we have to take more care to accept steps, because $h(\hat{x}^{(k+1)}) \leq \beta h^{(k)}$ holds trivially in that case, and we instead enforce a sufficient reduction condition on f as usual for unconstrained optimization.

2.4.4 Connections between Funnel and Filter Methods

An interesting connection exists between filter and funnel methods. In practical implementations of filter methods, we limit the number of filter entries to a maximum number, say $N_{\mathcal{F}}$, and initialize the filter with an upper bound on the constraint violation $h_{\max} = \max\{\kappa_1 h(x^{(0)}), \kappa_2\}$, where $\kappa_1, \kappa_2 > 1$ are constants that ensure that the initial point $x^{(0)}$ is acceptable. If the capacity $N_{\mathcal{F}}$ of the filter is reached, we reduce the upper bound by replacing it with an upper bound derived from the filter entry that has the maximum constraint violation $(h^+, f^+) \in \mathcal{F}$, resulting in a new upper bound $(h^+, -\infty)$. This approach frees one filter entry and ensures that the current point remains filter-acceptable, and we cannot cycle.

We can now interpret the funnel as a filter with a single entry, $N_{\mathcal{F}} = 1$, that corresponds to the upper bound on the constraint violation, and we update this upper bound on h-type iterations. The main difference between this interpretation and the actual funnel method is the update rule (13). This interpretation allows us to provide a streamlined convergence proof.

3 A Unified Funnel Restoration SQP Algorithm

The complete method is presented in Algorithms 2, 3, 4, and 5. It follows the double-loop framework of Algorithm 1: the outer loop updates the current iterate and the inner loop computes an

acceptable point (either in the optimality phase or in the feasibility restoration phase). Algorithm 2 implements the funnel globalization strategy; Algorithms 3 and 4 implement the double-loop SQP with a trust-region and with a line-search globalization mechanism, respectively; and, Algorithm 5 implements the direction computation. We deliberately split the two methods into separate components to emphasize the modularity of the Uno framework.

Algorithm 3: Uno: trust-region funnel restoration SQP method.

Input: initial primal-dual iterate $(x^{(0)}, \lambda^{(0)}, \mu^{(0)})$
$k \leftarrow 0$
termination $\leftarrow$ *false*
phase $\leftarrow$ Optimality — *constraint relaxation*
$\tau^{(0)} = \max\left[\bar{\tau}, \bar{\kappa} h^{(0)}\right]$ — *globalization strategy*
repeat
  Set inner iteration counter $l \leftarrow 0$ — *globalization mechanism*
  repeat
    Compute primal-dual solution $(d^{(k,l)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$ (Algorithm 5) — *constraint relaxation*
    Assemble trial iterate $\hat{x}^{(k+1,l)} \stackrel{\text{def}}{=} x^{(k)} + d^{(k,l)}$
    Reset the multipliers corresponding to the active trust region
    Determine whether trial iterate is *acceptable* (Algorithm 2) — *constraint relaxation*
    if *acceptable* **then**
      if *trust region is active at* $d^{(k,l)}$ **then** increase radius $\Delta_{\text{TR}}^{(l)}$
    else
      Decrease radius $\Delta_{\text{TR}}^{(l)}$
      $l \leftarrow l + 1$
  until $(\hat{x}^{(k+1,l)}, \hat{\lambda}^{(k+1,l)}, \hat{\mu}^{(k+1,l)})$ *is acceptable*
  Update $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)}) \leftarrow (\hat{x}^{(k+1,l)}, \hat{\lambda}^{(k+1,l)}, \hat{\mu}^{(k+1,l)})$
  if *termination criteria satisfied at* $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)})$ **then** *termination* $\leftarrow$ *true*
  $k \leftarrow k + 1$
until *termination*
return $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$

4 Global Convergence of Trust-Region Funnel SQP Algorithm

In this section we prove that the trust-region funnel SQP method converges either to a feasible point, which is a KKT point if a constraint qualification holds, or to a stationary point of the constraint violation. We start by stating our basic assumptions.

Assumption 1 (Standard assumptions [11])**.**

1. *All points attained by the funnel algorithm lie in a nonempty compact set* X*.*
2. *The functions* f *and* c *are twice continuously differentiable on an open set containing* X*.*
3. *The Hessian matrices of the objective and constraints are uniformly bounded for all* $x \in X$*; in other words, there exists an* $M > 0$ *such that the Hessian matrices* $H(x)$ *satisfy* $\|H(x)\|_2 \leq M$ *for all* $x \in X$*.*

In particular, these assumptions ensure that the problem functions are bounded and that the objective function f is bounded from below.

Algorithm 4: Uno: line-search funnel restoration SQP method.

Input: initial primal-dual iterate $(x^{(0)}, \lambda^{(0)}, \mu^{(0)})$
$k \leftarrow 0$
termination $\leftarrow$ *false*
phase $\leftarrow$ Optimality — *constraint relaxation*
$\tau^{(0)} = \max\left[\bar{\tau}, \bar{\kappa} h^{(0)}\right]$ — *globalization strategy*
repeat
  Compute primal-dual solution $(d^{(k)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$ (Algorithm 5) — *constraint relaxation*
  $\alpha^{(0)} \leftarrow 1$
  Set inner iteration counter $l \leftarrow 0$
  repeat
    Assemble trial iterate $\hat{x}^{(k+1,l)} \stackrel{\text{def}}{=} x^{(k)} + \alpha^{(l)} d^{(k)}$
    Determine whether trial iterate is *acceptable* (Algorithm 2) — *constraint relaxation*
    if *not acceptable* **then**
      Decrease step length $\alpha^{(l)}$
      $l \leftarrow l + 1$
    if $\alpha^{(l)} < \alpha_{\min}$ *(step-size too small)* **then**
      phase $\leftarrow$ Restoration — *constraint relaxation*
      $x^{resto} \leftarrow x^{(k)}$
      Compute primal-dual solution $(d^{(k)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$ (Algorithm 5)
      $\alpha^{(l)} \leftarrow 1$
  until $(\hat{x}^{(k+1,l)}, \hat{\lambda}^{(k+1,l)}, \hat{\mu}^{(k+1,l)})$ *is acceptable*
  Update $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)}) \leftarrow (\hat{x}^{(k+1,l)}, \hat{\lambda}^{(k+1,l)}, \hat{\mu}^{(k+1,l)})$
  if *termination criteria satisfied at* $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)})$ **then** *termination* $\leftarrow$ *true*
  $k \leftarrow k + 1$
until *termination*
return $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$

Algorithm 5: Uno: QP direction computation

Input: current iterate $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$
if *phase* = *Optimality* **then** — *constraint relaxation*
  $(d^{(k,l)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)}) \leftarrow$ solve subproblem QP$(x^{(k)})$ or $QP(x^{(k)}, \Delta_{\text{TR}}^{(l)})$
  if *subproblem infeasible* **then**
    phase $\leftarrow$ Restoration
    $x^{resto} \leftarrow x^{(k)}$
    $\lambda^{(k)} \leftarrow 0$
if *phase* = *Restoration* **then**
  $(d^{(k,l)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)}) \leftarrow$ solve feasibility subproblem $(FQP(x^{(k)}))$ or $FQP(x^{(k)}, \Delta_{\text{TR}}^{(l)})$
return $(d^{(k,l)}, \hat{\lambda}^{(k+1)}, \hat{\mu}^{(k+1)})$

4.1 Convergence to Feasibility

The convergence to a feasible point is independent of whether we use a trust-region or a line search mechanism and instead relies on the switching condition, the boundedness of f, and the funnel mechanism. In Theorem 1 we prove that the algorithm generates a sequence of iterates that converges toward feasibility. This theorem implicitly assumes that the algorithm is well defined

(i.e., that the inner iteration terminates) and generates an infinite sequence, which is shown below in Lemma 5 for the trust-region method (a similar result follows easily for the line-search method because we switch to a restoration phase once the step size α becomes smaller than $\alpha_{\min}$).

Theorem 1. *Assume that the algorithm does not terminate finitely at a KKT point, and consider sequences $\{\tau^{(k)}\}$, $\{h^{(k)}\}$, and $\{f^{(k)}\}$ such that $h^{(k)} \geq 0$, $f^{(k)}$ is bounded below and $h^{(k+n)} \leq \beta\tau^{(k)}$ for all $k, n \in \mathbb{N}$. Furthermore, let constants $\beta \in (0,1)$, $\kappa \in (0,1)$ be given. It holds either that*

$$\textit{(h-type step)} \quad h^{(k+1)} \leq \beta\tau^{(k)} \quad \textit{and} \quad \tau^{(k+1)} = (1-\kappa)h^{(k+1)} + \kappa\tau^{(k)} \tag{16}$$

or that

$$\textit{(f-type step)} \quad \tau^{(k+1)} = \tau^{(k)} \quad \textit{and} \quad f^{(k)} - f^{(k+1)} \geq \sigma\delta(h^{(k)})^2. \tag{17}$$

In both cases, it follows that $h^{(k)} \to 0$ for $k \to \infty$.

Proof. We study two cases, depending on whether there exists an infinite sequence of h-type steps.

1. If the number of h-type updates (16) is infinite, the funnel update rule implies that

$$\begin{aligned}\tau^{(k+1)} &= (1-\kappa)h^{(k+1)} + \kappa\tau^{(k)} \\ &\leq (1-\kappa)\beta\tau^{(k)} + \kappa\tau^{(k)} \\ &\leq (1-(1-\beta)(1-\kappa))\,\tau^{(k)}.\end{aligned}$$

   Therefore, $\tau^{(k+1)} \leq \theta\tau^{(k)}$, where $\theta \stackrel{\text{def}}{=} 1-(1-\beta)(1-\kappa) \in (0,1)$. Thus $\tau^{(k)} \to 0$ for $k \to \infty$.

2. If the number of h-type steps is finite, there exists a $\overline{k} \in \mathbb{N}$ such that for all $k \geq \overline{k}$ only f-type updates are performed, that is, both the switching condition $\Delta m_f^{(k)}(d) \geq \delta(h^{(k)})^2$ and the Armijo sufficient decrease condition $\Delta f^{(k)} \geq \sigma\Delta m_f^{(k)}(d)$ are satisfied. In the case of the line-search mechanism we have $\Delta f^{(k)} \geq \alpha\sigma\Delta m_f^{(k)}(d) > \alpha_{\min}\sigma\Delta m_f^{(k)}(d)$, because the step was not a restoration step. We then sum the switching condition (17) over k from $\overline{k}$ to a given $\overline{k}+N$ and observe that

$$\sum_{k=\overline{k}}^{\overline{k}+N} \left(f^{(k)} - f^{(k+1)}\right) \geq \sigma\delta \sum_{k=\overline{k}}^{\overline{k}+N} (h^{(k)})^2,$$

   which is equivalent to

$$f^{(\overline{k})} - f^{(\overline{k}+N)} \geq \sigma\delta \sum_{k=\overline{k}}^{\overline{k}+N} (h^{(k)})^2,$$

   where the right-hand side is multiplied by $\alpha_{\min} > 0$ for the line-search mechanism. For $N \to \infty$, $\sum_{k=\overline{k}}^{\overline{k}+N} (h^{(k)})^2$ is bounded (because f is bounded below by Assumption 1), and therefore $h^{(k)} \to 0$.

□

4.2 Global Convergence of Trust-Region Funnel Method to Stationary Points

The remainder of the global convergence analysis for trust-region and line-search methods differs significantly, and here we analyze only trust-region methods.

The trust-region funnel SQP method has three possible outcomes: (1) it terminates finitely or converges in the limit at a KKT point that satisfies MFCQ; (2) it converges to a Fritz John point at which MFCQ fails; or (3) it converges to a stationary point of the constraint violation. Outcome (3) is as strong as we can hope for in some sense, because finding a feasible point of (NCO) is just as hard as finding a stationary point.

In our analysis we concentrate on outcomes (1) and (2) and consider only infinite sequences. Similar to the global convergence paper of the filter method [14], we first show properties in a neighborhood of the feasible set. We start with two lemmas proven in [14] (due to a different definition of (NCO), we give an adjusted proof).

Lemma 1 (Lemma 2 in [14])**.** *Consider minimizing a quadratic function $\phi(\alpha)$, $\phi\colon \mathbb{R} \to \mathbb{R}$ on the interval $\alpha \in [0,1]$ when $\phi'(0) < 0$. A necessary and sufficient condition for the minimizer to be at $\alpha = 1$ is $\phi'' + \phi'(0) \leq 0$. In this case it follows that $\phi(0) - \phi(1) \geq -\frac{1}{2}\phi'(0)$.*

Lemma 2. *Let the standard assumptions hold, and let d be a feasible point of $QP(x^{(k)}, \Delta_{TR}^{(l)})$. It then follows that*

$$\Delta f^{(k)} \geq \Delta m_f^{(k)} - n(\Delta_{TR}^{(l)})^2 M, \tag{18}$$

$$\left| c_i\left(x^{(k)} + d\right)\right| \leq \frac{1}{2} n(\Delta_{TR}^{(l)})^2 M, \quad i \in \{1, \ldots, m\}. \tag{19}$$

Proof. Relation (18) is derived in [14]. For $i \in \{1, \ldots, m\}$, there exists z on the line segment from $x^{(k)}$ to $x^{(k)} + d$ such that

$$c_i\left(x^{(k)} + d\right) = c_i^{(k)} + (\nabla c_i^{(k)})^T d + \frac{1}{2} d^T \nabla^2 c_i(z) d = \frac{1}{2} d^T \nabla^2 c_i(z) d$$

by feasibility of d. Taking the absolute value and using the estimates similar to [14], we get (19). □

Lemma 3. *Let standard assumptions hold. If d solves $QP(x^{(k)}, \Delta_{TR}^{(l)})$, $x^{(k)} + d$ is acceptable to the funnel if $\Delta_{TR}^2 \leq 2\beta\tau^{(k)}/(mnM)$.*

Proof. It holds that

$$h(x^{(k)} + d) = \|c(x^{(k)} + d)\|_1 = \sum_{i=1}^{m} |c_i(x^{(k)} + d)| \leq \sum_{i=1}^{m} \frac{1}{2} n \Delta_{\mathrm{TR}}^2 M = \frac{1}{2} nm \Delta_{\mathrm{TR}}^2 M.$$

If $\Delta_{\mathrm{TR}}^2 \leq 2\beta\tau^{(k)}/(nmM)$, then $h(x^{(k)} + d) \leq \beta\tau^{(k)}$; in other words, the step is acceptable to the funnel. □

We note that Lemma 3 actually proves the funnel sufficient condition.

Lemma 4. *Let standard assumptions hold, and let $x^\circ \in X$ be a feasible point of problem* (NCO) *at which MFCQ holds but which is not a KKT point. Then, there exist a neighborhood $\mathcal{N}^\circ$ of x° and positive constants ε, μ, and κ such that for all $x \in \mathcal{N}^\circ \cap X$ and all Δ_{TR} for which*

$$\mu h(x) \leq \Delta_{TR} \leq \kappa$$

it follows that $QP(x^{(k)}, \Delta_{TR}^{(l)})$ *has a feasible solution* d *at which the predicted reduction* (6) *satisfies*

$$\Delta m_f \geq \frac{1}{3}\Delta_{TR}\varepsilon, \tag{20}$$

the sufficient reduction condition (11) *holds, and the actual reduction* (7) *satisfies*

$$\Delta f \geq \Delta_{TR}\sigma h(x+d). \tag{21}$$

Proof. The proof follows directly from the proof of [14, Lemma 4]. □

Lemma 5. *Let standard assumptions hold. Then the inner iterations terminate finitely.*

Proof. The active set at a feasible point x is the set of all bound constraints that hold with equality:

$$\mathcal{A}(x) \stackrel{\text{def}}{=} \{i \in \{1,\ldots,n\} \mid x_i = 0\}.$$

We denote by d the global solution of $QP(x^{(k)}, \Delta_{\text{TR}}^{(l)})$. If $x^{(k)}$ is a KKT point of (NCO), $d = 0$, and the inner loop terminates. Otherwise, we consider two cases:

1. $h^{(k)} > 0$: There exists an $i \in \{1,\ldots,m\}$ such that, without loss of generality, $c_i^{(k)} > 0$. For all d such that $\|d\|_\infty \leq \Delta_{\text{TR}}$, it holds that

   $$c_i^{(k)} + (\nabla c_i^{(k)})^T d \geq c_i^{(k)} - \Delta_{\text{TR}}\|\nabla c_i^{(k)}\|_1.$$

   If either $\|\nabla c_i^{(k)}\| = 0$ or $\Delta_{\text{TR}} < |c_i^{(k)}|/\|\nabla c_i^{(k)}\|_1$, we have $c_i^{(k)} - \Delta_{\text{TR}}\|\nabla c_i^{(k)}\|_1 > 0$. Thus for Δ_{TR} sufficiently small, $QP(x^{(k)}, \Delta_{\text{TR}}^{(l)})$ is infeasible, and the inner loop terminates finitely.

2. $h^{(k)} = 0$: Inactive bound constraints will stay inactive for any direction $\|d\|_\infty \leq \Delta_{\text{TR}}$ for sufficiently small Δ_{TR}. Therefore, we only consider equality and active inequality constraints. Since $x^{(k)}$ is not a KKT point, there exists s with $\|s\| = 1$ and $\eta > 0$ such that

   $$s^T\nabla f^{(k)} = -\eta < 0, \quad s^T\nabla c_i^{(k)} = 0 \quad \text{for} \quad i \in \{1,\ldots,m\}, \quad s_i \geq 0 \quad \text{for} \quad i \in \mathcal{A}^{(k)}.$$

   We consider the QP-feasible line segment $\alpha\Delta_{\text{TR}}s$, $\alpha \in [0,1]$. We construct the function $\phi(\alpha) = m_f^{(k)}(\alpha\Delta_{\text{TR}}s)$ with the properties

   $$\phi'(0) = -\Delta_{\text{TR}}\eta, \quad \phi'' = \Delta_{\text{TR}}^2 s^T W_1^{(k)} s \leq \Delta_{\text{TR}}^2 M.$$

   Hence, if $\Delta_{\text{TR}} \leq \eta/M$, then $\phi'(0) + \phi'' \leq 0$. From Lemma 1, it follows that $\phi(0) - \phi(1) \geq \frac{1}{2}\Delta_{\text{TR}}\eta$. Because d is globally optimal for $QP(x^{(k)}, \Delta_{\text{TR}}^{(l)})$, it holds that

   $$\Delta m_f^{(k)}(d) \geq \Delta m_f^{(k)}(\Delta_{\text{TR}}s) = m_f^{(k)}(0) - m_f^{(k)}(\Delta_{\text{TR}}s) = \phi(0) - \phi(1) \geq \frac{1}{2}\Delta_{\text{TR}}\eta > 0. \tag{22}$$

   If we choose $\Delta_{\text{TR}} \leq \dfrac{(1-\sigma)\eta}{2nM}$, then, combining (18) and (22), we have

   $$\begin{aligned}
   \Delta f^{(k)} &\stackrel{(18)}{\geq} \Delta m_f^{(k)} - n\Delta_{\text{TR}}^{(l)}\Delta_{\text{TR}}^{(l)}M \\
   &\geq \Delta m_f^{(k)} - n\frac{(1-\sigma)\eta}{2nM}\Delta_{\text{TR}}^{(l)}M = \Delta m_f^{(k)} - (1-\sigma)\frac{1}{2}\Delta_{\text{TR}}^{(l)}\eta \\
   &\stackrel{(22)}{\geq} \Delta m_f^{(k)} - (1-\sigma)\Delta m_f^{(k)}(d) = \sigma\Delta m_f^{(k)} \\
   &> 0 = h^{(k)}.
   \end{aligned}$$

These are the necessary conditions of an f-type iteration. If $\Delta_{\mathrm{TR}}^2 \leq \dfrac{2\beta\tau^{(k)}}{mnM}$, $x^{(k)} + d$ is acceptable to the funnel. Thus, if Δ_{TR} is chosen small enough, an f-type step is taken.

□

Remark 1. *We note that the condition that d be the global solution of $QP(x^{(k)}, \Delta_{TR}^{(l)})$ holds if the Hessian is positive definite on the null space of the linearized equality constraints. Otherwise, it can be replaced by a Cauchy-point construction that explicitly produces a unit step, s in* (22)*, and then requires that $QP(x^{(k)}, \Delta_{TR}^{(l)})$ compute a step that is at least as good as the Cauchy prediction.*

Theorem 2. *If standard assumptions hold, the funnel SQP algorithm has one of the following outcomes:*

1. *The restoration phase fails to find a point x that is both acceptable to the funnel and for which $QP(x^{(k)}, \Delta_{TR}^{(l)})$ has a feasible direction for some $\Delta_{TR} \geq \Delta_{TR}^{\circ}$. In this case the restoration phase converges to an infeasible limit point.*
2. *A KKT point of problem* (NCO) *is found ($d = 0$ solves $QP(x^{(k)}, \Delta_{TR}^{(l)})$ for some k).*
3. *There exists an accumulation point that is feasible and either is a KKT point or fails to satisfy MFCQ.*

Proof. It is enough to consider case 3. As proven in Lemma 5, the inner loop is finite, and therefore the trust-region funnel SQP method produces an infinite sequence of iterates. All iterates lie in the compact set X; therefore, there exists a converging subsequence. We consider two cases:

1. The main sequence contains an infinite number of h-type steps: in this case we pick a subsequence containing purely h-type steps. For h-type iterations, we know $h^{(k)} \to 0$ and $\tau^{(k)} \to 0$. In particular, $\tau^{(k+1)} < \tau^{(k)}$. Therefore, there exists a converging subsequence whose index set is denoted by $\mathcal{S}$ and whose limit point is denoted by x^{∞}. For $k \in \mathcal{S}$, it holds that

$$x^{(k)} \to x^{\infty}, \quad h^{(k)} \to 0, \quad \tau^{(k+1)} < \tau^{(k)}.$$

   Thus, x^{∞} must be feasible. We take another case distinction.

   (a) MFCQ is not satisfied at x^{∞}; then the claim holds.

   (b) MFCQ is satisfied at x^{∞}. For a proof by contradiction, we assume that x^{∞} is not a KKT point. The vectors ∇c_i^{∞} for $i \in \{1, \ldots, m\}$ are linearly independent, and the MFCQ set is not empty. For sufficiently large $k \in \mathcal{S}$, $x^{(k)}$ lies in the neighborhood $\mathcal{N}^{\infty}$. If $QP(x^{(k)}, \Delta_{\mathrm{TR}})$ has a feasible solution and

$$\mu h^{(k)} \leq \Delta_{\mathrm{TR}} \leq \min\left\{\sqrt{\frac{2\beta h_{\max}^{(k)}}{mnM}}, \kappa\right\},$$

   the algorithm performs an f-type iteration. For k sufficiently large, we get

$$\mu h^{(k)} \leq \Delta_{\mathrm{TR}} \leq \sqrt{\frac{2\beta h_{\max}^{(k)}}{mnM}}.$$

We see that the upper bound is more than twice the lower bound. Given the update rule, we find a Δ_{TR} that lies in the interval. Therefore an f-type step occurs, which is a contradiction; that is, x^∞ is a KKT point.

2. The main sequence contains finitely many h-type steps. Therefore, there exists an index $\overline{k}$ such that for all $k > \overline{k}$ the sequence $\{f^{(k)}\}$ is strictly monotonically decreasing. From Theorem 1 we know $h^{(k)} \to 0$ for $k \geq \overline{k}$. We deduce that any accumulation point x^∞ is feasible. Because $f(x)$ is bounded on X and $\{f^{(k)}\}$ is converging and, in particular, it is a Cauchy sequence, it follows that $\sum_{k \geq \overline{k}} \Delta f^{(k)}$ is converging. Consider again two cases:

   (a) MFCQ is not satisfied; then there is nothing to show.

   (b) MFCQ is satisfied, and we proceed again by deriving a contradiction. A sufficient condition for accepting an f-type step is that Δ_{TR} lies in the following interval:

$$\mu h^{(k)} \leq \Delta_{\mathrm{TR}} \leq \min\left\{\sqrt{\frac{2\beta\tau^{(\overline{k})}}{mnM}}, \kappa\right\}.$$

   The funnel parameter $\tau^{(\overline{k})}$ on the right side is constant; therefore the right-hand side is constant, and we denote it by $\overline{\Delta_{\mathrm{TR}}} > 0$. For sufficiently large k, we can guarantee that the right side is greater than twice the lower bound. The inner loop decreases Δ_{TR} such that either it falls into the interval and will be accepted as an f-type step or it is already accepted beforehand. This guarantees that a trust-region radius $\Delta_{\mathrm{TR}}^{(k)} \geq \min\{\frac{1}{2}\overline{\Delta_{\mathrm{TR}}}, \Delta_{\mathrm{TR}}^\circ\}$ is picked. We deduce from (20) and (11) that $\Delta f^{(k)} \geq \frac{1}{3}\sigma\varepsilon \min\{\frac{1}{2}\overline{\Delta_{\mathrm{TR}}}, \Delta_{\mathrm{TR}}^\circ\}$, which is a contradiction to the convergence of $\sum_{k \geq \bar{k}} \Delta f^{(k)}$. Hence, x^∞ must be a KKT point.

□

5 Simulation Results

The funnel method has been implemented in `Uno` as a globalization strategy and is available at `https://github.com/david0oo/Uno/tree/funnel_method`. We compare four different algorithmic configurations on a subset of 278 small instances of the `CUTEst` test set [17]. All four versions employ the restoration SQP method but differ in their globalization strategy (filter or funnel) and their globalization mechanism (line-search or trust-region method).

We have excluded unconstrained problems from the test set because filter and funnel methods behave identically on these problems (the switching condition (10) is always satisfied, and the same Armijo condition is enforced). In our numerical experiments we compare the funnel method with the default filter methods of `Uno`: we start from a bird's-eye position and move to a successively more detailed analysis. We close this section with some test results on the Maratos effect.

5.1 Implementation Details

The algorithmic parameters of the funnel method and the filter method are chosen as in Table 1. Both methods are initialized with the same parameters to guarantee that they start in similar

conditions. Following the discussion of the funnel update strategy in Section 2.4.2, we found that balancing the update of the funnel width is important; in other words, the funnel reduction should be neither too strict nor too slow. Therefore, we picked $\kappa = 0.5$, which worked well for our simulations. The filter has a maximum capacity of 50 entries. For both methods, the initial upper bound on the constraint violation is set according to Equation (9). The maximum number of outer iterations is set to $4,000$.

Table 1: Parameter values of the funnel method and the filter method.

	Parameter	Value	Parameter	Value	Parameter	Value
Funnel method	$\bar{\tau}$	100	$\bar{\kappa}$	1.25	κ	0.5
	δ	0.999	σ	10^{-4}	β	0.99
Filter method	$\bar{\tau}$	100	$\bar{\kappa}$	1.25	β	0.999

The QP solver available in Uno is bqpd [8,9], a reliable null-space method for indefinite quadratic optimization. The trust-region method uses the exact Hessian matrix. A sufficient condition for the well-posedness of the line-search subproblems is ensured by iteratively adding a multiple of the identity to $W^{(k)}$ until the resulting matrix is positive definite [24, p. 51]. The inertia of the matrix is computed by MA57 [21].

Uno terminates at (x^*, λ^*, μ^*) with one of the following outcomes:

- **KKT point found**, if the final iterate satisfies an approximate KKT condition:

$$\|\nabla \mathcal{L}(x^*, 1, \lambda^*, \mu^*)\| \leq \varepsilon, \quad \|c(x^*)\| \leq \varepsilon, \quad \|x^* \odot \mu^*\| \leq \varepsilon,$$

  where the tolerance ε is set to 10^{-6} for our numerical results.

- **Infeasible stationary point found**, if an approximate KKT condition of the feasibility problem holds:

$$\left\| \begin{pmatrix} \nabla \mathcal{L}(x^*, 0, \lambda^*, \mu^*) \\ e + \lambda^* - \mu_u^* \\ e - \lambda^* - \mu_v^* \end{pmatrix} \right\| \leq \varepsilon, \quad \|c(x^*)\| > \varepsilon, \quad \left\| \begin{pmatrix} x^* \odot \mu^* \\ u^* \odot \mu_u^* \\ v^* \odot \mu_v^* \end{pmatrix} \right\| \leq \varepsilon,$$

  where $u, v \geq 0$ are the elastic variables introduced in $FQP(x^{(k)}, \Delta_{\text{TR}}^{(l)})$ and $\mu_u, \mu_v \geq 0$ are the corresponding bound multipliers.

- **Maximum number of iterations**, if we reached the iteration bound.

- **Unbounded solution**, if $f(x^*) < -10^{20}$ for an ε-feasible point ($\|c(x^*)\| \leq \varepsilon$).

- **Small feasible step**, if the trust-region radius $\Delta_{\text{TR}} \leq 10^{-16}$ and $\|c(x^*)\| \leq \varepsilon$. This indicates that the problem may be ill-conditioned or may not be differentiable.

We consider a problem to be solved correctly if both methods found a KKT point or converged to an infeasible stationary point. If one of the methods converged to an infeasible point while the

other found a KKT point, the first method failed to solve the problem. Other reasons for failure include the step size becoming too small or an excess of maximum number of iterations. The simulations were carried out on an Intel Core i7-10810U CPU, and the corresponding log files are available at `https://github.com/david0oo/uno_funnel_results`.

5.2 Comparison between Funnel and Filter

Detailed results of this comparison are tabulated in Appendices A and B. Figure 3 shows a comparison of the four algorithmic configurations as a Dolan–Moré performance profile [7] for the number of constraint evaluations. A point (α, β) on the graph means that a method solves the fraction β of all test problems within α times the number of evaluations of the virtual best solver—the solver that performs best for every instance [16]. The higher and the more to the left, the better. The performance profiles demonstrate that the funnel method slightly outperforms the filter method in terms of Hessian and constraint evaluations. For these two metrics, line-search methods perform much worse than trust-region methods. The cause seems to stem from the convexification procedure: the trust-region methods solve indefinite QPs and exploit the negative curvature of the problems. If the trust-region QPs are convexified as are the line-search QPs, the performance similarly degrades.

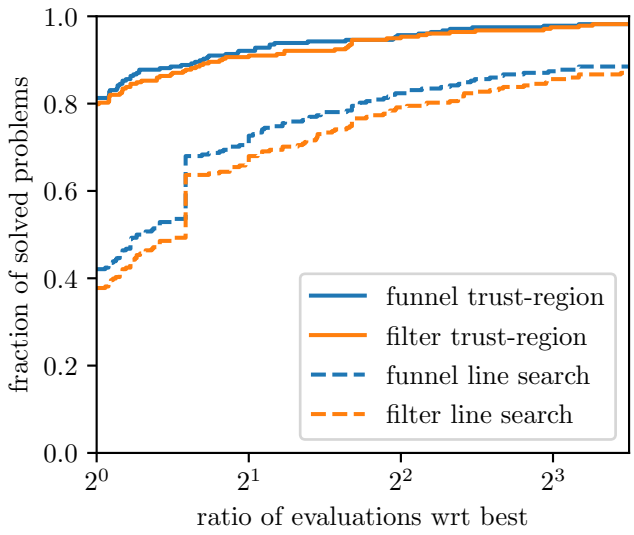
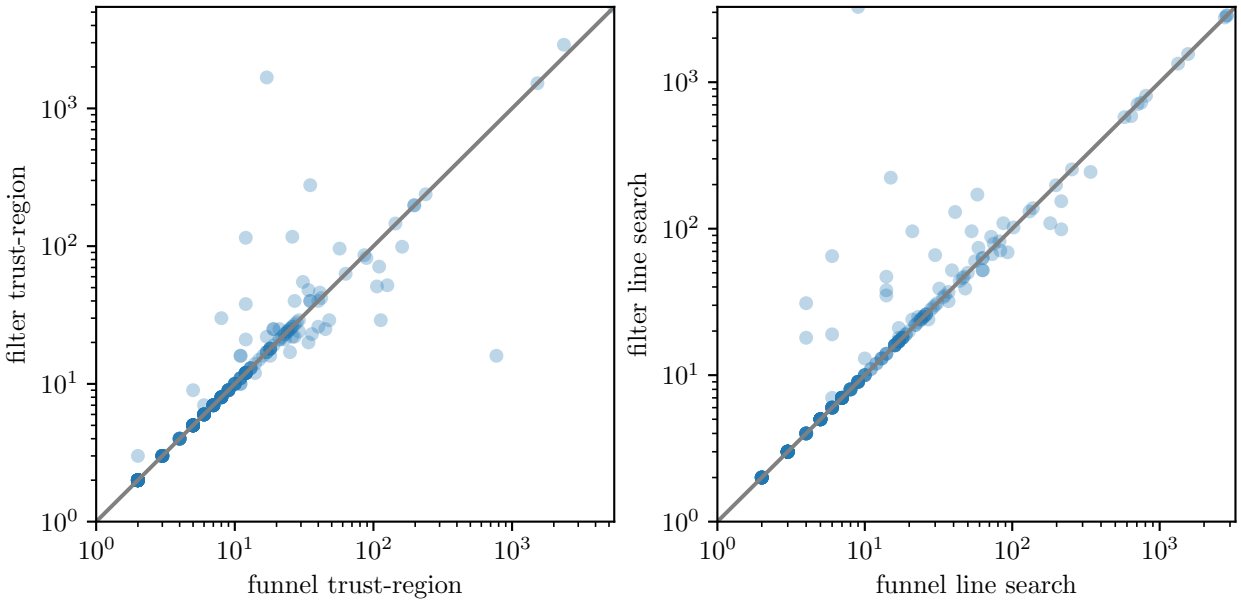


Figure 3: Performance profiles for all four algorithmic configurations with respect to constraint evaluations.

Figure 4 presents a distribution plot of the filter and funnel methods with trust region (left) and line search (right) on the `CUTEst` instances, using the number of constraint evaluations as the metric. Each point corresponds to one problem instance. Points on the diagonal correspond to instances where the funnel method and the filter method took the same number of iterations, points above the diagonal correspond to instances where the funnel method was faster than the filter method, and points below the diagonal correspond to instances where the filter method was faster. We observe that on many instances, the two methods behave almost identically. For the line-search methods, we observe that the funnel method has more wins (points above the diagonal) than the filter method, which is also observed in the performance profiles.



(a) Trust-region methods.

(b) Line-search methods.

Figure 4: Distribution plot of constraint evaluations for funnel vs filter globalization strategies.

Next, we compared the number of iterations and the number of evaluations of all functions and derivatives. This more detailed analysis reveals that both trust-region methods take different iteration paths for only 49 instances. The results of the line-search methods also differ for only 49 of the problems. In Figures 5 and 6 we plot for each of these instances a vertical bar that shows the number of f-type (blue), h-type (orange), and restoration steps (green) for the trust-region and line-search methods, respectively. A missing bar means that the algorithm failed to solve the problem. Overall, the ratio of the three iteration types seems to be similar, except for a few outliers. We observe that fewer problems can be solved by line-search methods than by trust-region methods, as is also shown in the performance profiles.

5.3 Insights from Illustrative Examples

We finish our discussion of the numerical results by zooming in to two illustrative examples that highlight the similarities and differences between filter and funnel methods.

5.3.1 `powellbs` Example

The `powellbs` instance for the trust-region methods is given by

$$\begin{aligned} \min_{x\in\mathbb{R}^2} \quad & 0 \\ \text{s.t.} \quad & -1 + 10000x_1x_2 = 0, \\ & -1.0001 + e^{-x_1} + e^{-x_2} = 0, \end{aligned}$$

with initial point $x^{(0)} = (0, 1)^T$. The optimal solution is approximately $(1.1 \times 10^{-5}, 9.1)^T$. Figure 7 shows the infeasibility and the funnel width (in log scale) as functions of the outer iterations for

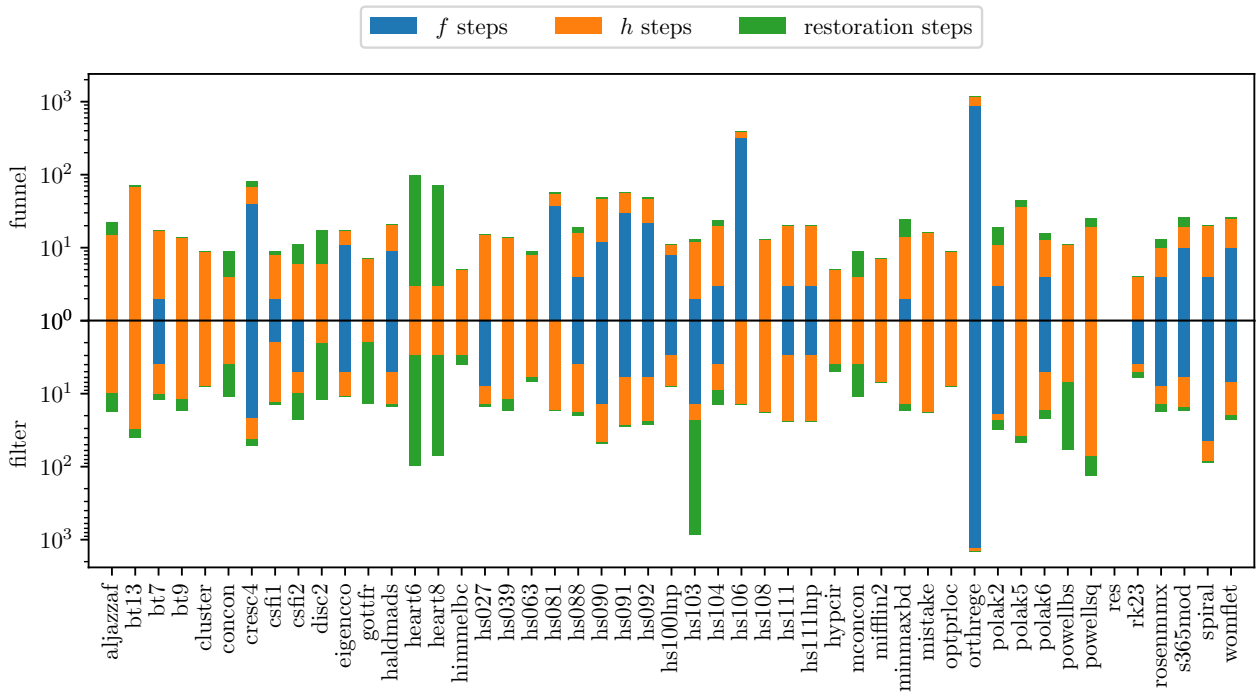


Figure 5: Comparison of the number of f-type (blue), h-type (orange), and restoration steps (green) for the trust-region funnel method and the trust-region filter method.

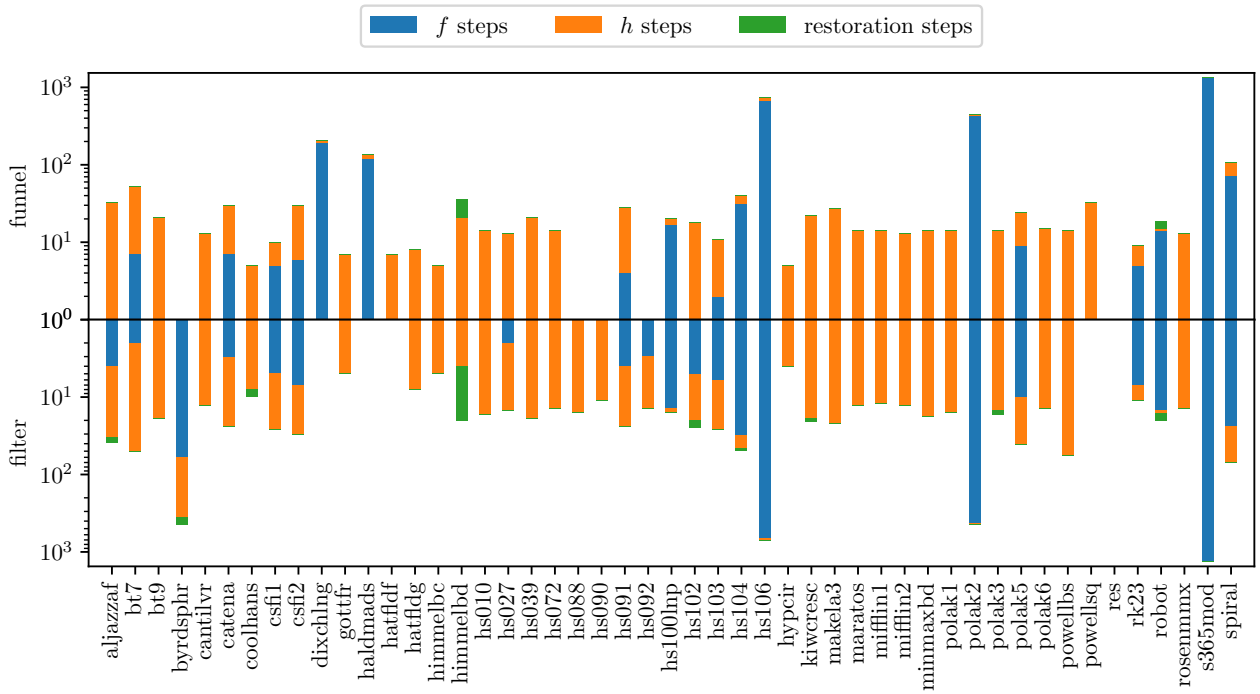


Figure 6: Comparison of the number of f-type (blue), h-type (orange), and restoration steps (green) for the line-search funnel method and the line-search filter method.

the trust-region funnel and filter methods. The funnel width decreases linearly, because only h-type steps are taken. Interestingly, both methods are identical until iteration 5. From iteration 6 onwards, the funnel allows more non-monotonicity, and the method enters the regime of quadratic convergence, while the filter blocks progress. This causes infeasibility of the QP and triggers the switch to feasibility restoration. Many iterations are required to find a filter-acceptable point, after which the convergence is quadratic.

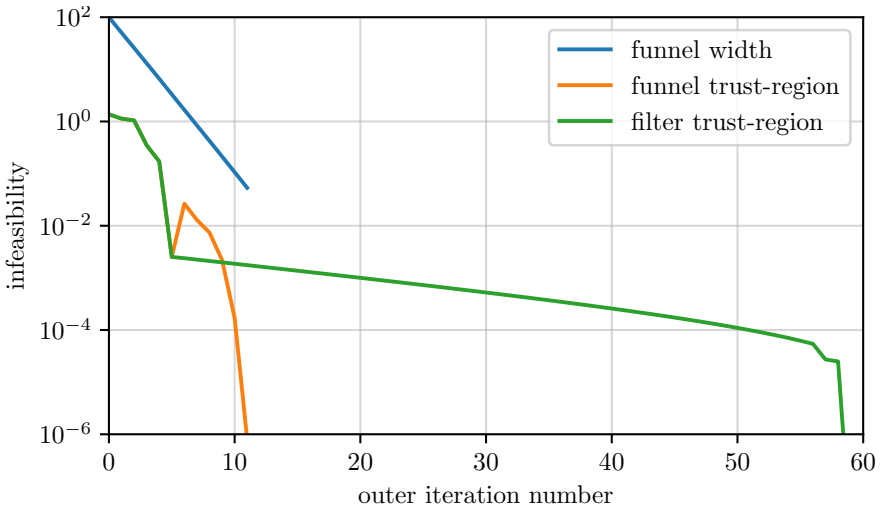


Figure 7: Infeasibility and funnel width as functions of the outer iterations for trust-region funnel and filter algorithms on the `powellbs` problem.

5.3.2 `maratos` Example

Filter (or funnel) methods do not fail for the standard `maratos` and can be shown to take unit steps arbitarily close to a feasible point. However, in [15] another example was introduced proving that, in general, filter methods (including `filterSQP`) do suffer from the Maratos effect:

$$\begin{aligned} \min_{x\in\mathbb{R}^2} \quad & 2\left(x_1^2 + x_2^2 - 1\right) - x_1 \\ \text{s.t.} \quad & x_1^2 + x_1^2 - 1 = 0. \end{aligned}$$

If the starting point is chosen as $x = (\cos(t), \sin(t))$ for $t > 0$ small and $\lambda = \frac{3}{2}$ ($-\frac{3}{2}$ with our notation), the quadratic model of the objective predicts a decrease although the objective increases, which violates the Armijo condition. Thus, the full step is rejected.

The iterations of the trust-region funnel SQP and line-search funnel SQP methods starting from $(x_1, x_2, \lambda) = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -\frac{3}{2})$ are given in Tables 2 and 3, respectively. We see in both cases that in the first iteration, the full step increases the objective and the constraint violation, which yields a step rejection. This demonstrates that the funnel method also suffers from the Maratos effect. Therefore, appropriate measures such as second-order corrections, a watchdog strategy, or non-monotone techniques are required for fast local convergence.

Table 2: Iterations of trust-region funnel SQP method on `maratos` example.

k	l	$\Delta_{\text{TR}}^{(k,l)}$	$\tau^{(k)}$	$\|d^{(k,l)}\|_\infty$	$f^{(k)}$	$h^{(k)}$	$\|\nabla\mathcal{L}\|$	status
0	–	$1.00e+01$	$1.00e+02$	–	-0.707	$5.28e-10$	$7.65e-01$	`initial point`
1	1	$1.00e+01$	$1.00e+02$	$5.00e-01$	-0.207	$5.00e-01$	–	`rejected (Armijo)`
–	2	$2.50e-01$	$1.00e+02$	$2.50e-01$	-0.707	$1.25e-01$	–	`rejected (Armijo)`
–	3	$1.25e-01$	$1.00e+02$	$1.25e-01$	-0.770	$3.13e-02$	$8.59e-01$	f-`type step`
2	1	$2.50e-01$	$1.00e+02$	$2.50e-01$	-0.814	$8.69e-02$	$4.18e-01$	f-`type step`
3	1	$5.00e-01$	$1.00e+02$	$2.71e-01$	-0.883	$7.60e-02$	$5.86e-02$	f-`type step`
4	1	$5.00e-01$	$1.00e+02$	$6.30e-02$	-0.992	$5.06e-03$	$2.86e-03$	f-`type step`
5	1	$5.00e-01$	$1.00e+02$	$2.55e-03$	-1.000	$1.28e-05$	$1.37e-05$	f-`type step`
6	1	$5.00e-01$	$1.00e+02$	$9.77e-06$	-1.000	$1.37e-10$	$1.88e-10$	ε-`optimal`

Table 3: Iterations of line-search funnel SQP method on `maratos` example.

k	l	$\alpha^{(l)}$	regulariz.	$\tau^{(k)}$	$\|d^{(k,l)}\|_\infty$	$f^{(k)}$	$h^{(k)}$	$\|\nabla\mathcal{L}\|$	status
0	–	–	–	$1.00e+02$	–	-0.707	$5.28e-10$	$7.65e-01$	`initial point`
1	1	1	$1.00e-04$	$1.00e+02$	$5.00e-01$	-0.207	$5.00e-01$	–	`rejected (Armijo)`
–	2	0.5	-	$1.00e+02$	$2.50e-01$	-0.707	$1.25e-01$	$4.31e-01$	f-`type step`
2	1	1	$1.00e-04$	$1.00e+02$	$4.81e-01$	-0.605	$2.58e-01$	–	`rejected (Armijo)`
–	2	0.5	–	$1.00e+02$	$2.40e-01$	-0.785	$1.27e-01$	$2.10e-01$	f-`type step`
3	1	1	$1.00e-04$	$1.00e+02$	$2.40e-01$	-0.913	$5.79e-02$	$2.30e-02$	f-`type step`
4	1	1	$1.00e-04$	$1.00e+02$	$2.76e-02$	-0.998	$1.35e-03$	$9.91e-04$	f-`type step`
5	1	1	$1.00e-04$	$1.00e+02$	$6.74e-04$	-1.000	$8.86e-07$	$1.24e-06$	f-`type step`
6	1	1	$1.00e-04$	$1.00e+02$	$8.65e-07$	-1.000	$9.44e-13$	$9.59e-11$	ε-`optimal`

6 Conclusion

We consider a generic double-loop framework for solving nonlinearly constrained optimization problems that has been implemented in the open-source solver `Uno`. To illustrate the framework, we studied several variants of a restoration SQP method for nonlinearly constrained optimization problems, with an emphasis on a new globalization strategy called the funnel method. The method was presented through the lens of filter methods, whose theory served as a starting point for a theoretical analysis. We derived the global convergence for the trust-region funnel SQP method and outlined the main differences and similarities in the different proofs. An implementation of the funnel strategy in the `Uno` solver proved to slightly outperform its filter counterpart with respect to constraint evaluations for a subset of `CUTEst` instances, while being also easier to implement.

References

[1] H. Antil, D. P. Kouri, M.-D. Lacasse, and D. Ridzal. *Frontiers in PDE-constrained Optimization*, volume 163. Springer, 2018.

[2] M. P. Bendsoe and O. Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[3] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[4] L. T. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders. *Real-time PDE-constrained Optimization*. SIAM, 2007.

[5] R. H. Byrd, J. Nocedal, and R. A. Waltz. Steering exact penalty methods for nonlinear programming. *Optimization Methods and Software*, 23(2):197–213, 2008.

[6] F. E. Curtis, N. Gould, D. Robinson, and P. Toint. An interior-point trust-funnel algorithm for nonlinear optimization. *Math. Program.*, 161(1–2):73—134, jan 2017.

[7] E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series B*, 91, 03 2001.

[8] R. Fletcher. An optimal positive definite update for sparse Hessian matrices. *SIAM Journal on Optimization*, 5(1):192–218, 1995.

[9] R. Fletcher. Stable reduced Hessian updates for indefinite quadratic programming. *Mathematical Programming, Series B*, 87:251–264, 04 2000.

[10] R. Fletcher, N. I. M. Gould, S. Leyffer, P. Toint, and A. Wächter. Global convergence of a trust-region SQP-filter algorithm for general nonlinear programming. *SIAM Journal on Optimization*, 13(3):635–659, 2002.

[11] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–269, 02 1999.

[12] R. Fletcher* and S. Leyffer. Solving mathematical programs with complementarity constraints as nonlinear programs. *Optimization Methods and Software*, 19(1):15–40, 2004.

[13] R. Fletcher, S. Leyffer, D. Ralph, and S. Scholtes. Local convergence of [sqp] methods for mathematical programs with equilibrium constraints. *SIAM Journal on Optimization*, 17(1):259–286, 2006.

[14] R. Fletcher, S. Leyffer, and P. Toint. On the global convergence of a filter–SQP algorithm. *SIAM Journal on Optimization*, 13(1):44–59, 2002.

[15] R. Fletcher, S. Leyffer, and P. Toint. A brief history of filter methods. *SIAG/OPT Views-and-News*, 18(1):2–12, 10 2006.

[16] N. Gould, Y. Loh, and D. Robinson. A nonmonotone filter SQP method: Local convergence and numerical results. *SIAM Journal on Optimization*, 25(3):1885–1911, 2015.

[17] N. Gould, D. Orban, and P. Toint. CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60:545–557, 07 2014.

[18] N. Gould, D. Robinson, and P. Toint. Corrigendum: Nonlinear programming without a penalty function or a filter. *Technical Report naXys-07-2011, Namur Centre for Complex Systems (naXys), FUNDP-University of Namur, Namur, Belgium*, 2011.

[19] N. Gould and P. Toint. Nonlinear programming without a penalty function or a filter. *Mathematical Programming*, 122:155–196, 03 2010.

[20] R. Herzog and K. Kunisch. Algorithms for PDE-constrained optimization. *GAMM-Mitteilungen*, 33(2):163–176, 2010.

[21] HSL. A collection of Fortran codes for large scale scientific computation., 2011.

[22] J. Lee and S. Leyffer. *Mixed integer nonlinear programming*, volume 154. Springer Science & Business Media, 2011.

[23] S. Leyffer, M. Menickelly, T. Munson, C. Vanaret, and S. M. Wild. A survey of nonlinear robust optimization. *INFOR: Information Systems and Operational Research*, 58(2):342–373, 2020.

[24] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2 edition, 2006.

[25] J. Outrata, M. Kocvara, and J. Zowe. *Nonsmooth approach to optimization problems with equilibrium constraints: theory, applications and numerical results*, volume 28. Springer Science & Business Media, 1998.

[26] M. Samadi. *Efficient Trust Region Methods for Nonconvex Optimization*. PhD thesis, Lehigh University, 2018.

[27] F. Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*, volume 112. American Mathematical Soc., 2010.

[28] C. Vanaret. Uno (Unifying Nonconvex Optimization). Available at `https://github.com/cvanaret/Uno` under the MIT license.

[29] C. Vanaret and S. Leyffer. Unifying nonlinearly constrained nonconvex optimization. Submitted to Mathematical Programming Computation, 2024.

[30] A. Wächter and L. Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005.

[31] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[32] C. Zoppke-Donaldson. *A tolerance-tube approach to sequential quadratic programming with applications*. PhD thesis, University of Dundee, 1995.

A Detailed Results on CUTEst problems: Number of Function and Derivative Evaluations for Trust-Region Algorithms

	trust-region funnel method						*trust-region filter method*					
instance	*status*	f	c	∇f	∇c	$\nabla^2\mathcal{L}$	*status*	f	c	∇f	∇c	$\nabla^2\mathcal{L}$
aircrfta	*KKT*	4	4	4	4	3	*KKT*	4	4	4	4	3
airport	*KKT*	13	13	13	13	12	*KKT*	13	13	13	13	12
aljazzaf	*KKT*	17	26	23	23	23	*KKT*	12	22	19	19	19
allinitc	*KKT*	29	29	29	29	28	*KKT*	29	29	29	29	28
alsotame	*KKT*	5	5	5	5	4	*KKT*	5	5	5	5	4
argauss	*infeasible*	1	3	3	3	3	*infeasible*	1	3	3	3	3
avgasa	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
avgasb	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
avion2	*KKT*	23	23	11	11	10	*KKT*	23	23	11	11	10
batch	*KKT*	9	9	9	9	8	*KKT*	9	9	9	9	8
biggsc4	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
booth	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
bt1	*infeasible*	2	2	2	2	2	*infeasible*	2	2	2	2	2
bt10	*KKT*	7	7	7	7	6	*KKT*	7	7	7	7	6
bt11	*KKT*	7	7	7	7	6	*KKT*	7	7	7	7	6
bt12	*KKT*	5	5	5	5	4	*KKT*	5	5	5	5	4
bt13	*KKT*	105	110	73	73	75	*KKT*	58	71	42	42	46
bt2	*KKT*	13	13	13	13	12	*KKT*	13	13	13	13	12
bt3	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
bt4	*KKT*	8	8	7	7	6	*KKT*	8	8	7	7	6
bt5	*KKT*	9	9	8	8	7	*KKT*	9	9	8	8	7
bt6	*KKT*	11	11	10	10	9	*KKT*	11	11	10	10	9
bt7	*KKT*	23	23	18	18	17	*KKT*	17	21	13	13	14
bt8	*KKT*	12	12	12	12	11	*KKT*	12	12	12	12	11
bt9	*KKT*	19	19	15	15	14	*KKT*	19	25	18	18	18
byrdsphr	*KKT*	8	23	14	14	14	*KKT*	8	23	14	14	14
cantilvr	*KKT*	13	13	11	11	10	*KKT*	13	13	11	11	10
catena	*KKT*	12	12	11	11	10	*KKT*	12	12	11	11	10
cb2	*KKT*	7	7	7	7	6	*KKT*	7	7	7	7	6
cb3	*KKT*	7	7	7	7	6	*KKT*	7	7	7	7	6
chaconn1	*KKT*	5	5	5	5	4	*KKT*	5	5	5	5	4
chaconn2	*KKT*	5	5	5	5	4	*KKT*	5	5	5	5	4
cluster	*KKT*	10	10	10	10	9	*KKT*	10	10	9	9	8
concon	*KKT*	5	11	10	10	10	*KKT*	7	16	12	12	16
congigmz	*KKT*	5	6	6	6	6	*KKT*	5	6	6	6	6
coolhans	*KKT*	3	3	3	3	2	*KKT*	3	3	3	3	2
core1	*KKT*	6	238	123	123	123	*KKT*	6	238	123	123	123
coshfun	*unbounded*	86	86	75	75	74	*unbounded*	86	86	75	75	74
cresc4	*KKT*	131	161	84	84	85	*KKT*	84	99	53	53	60
csfi1	*KKT*	11	12	10	10	10	*KKT*	20	21	15	15	15
csfi2	*KKT*	7	12	12	12	12	*KKT*	13	38	24	24	28
dallass	*KKT*	18	18	16	16	15	*KKT*	18	18	16	16	15
deconvc	*KKT*	24	24	15	15	14	*KKT*	24	24	15	15	14
degenlpa	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
degenlpb	*KKT*	2	2	2	2	1	*KKT*	2	2	2	2	1
demymalo	*KKT*	8	8	8	8	7	*KKT*	8	8	8	8	7
dipigri	*KKT*	12	12	9	9	8	*KKT*	12	12	9	9	8

disc2	*infeasible*	*9*	*25*	*18*	*18*	*18*	*infeasible*	*5*	*17*	*13*	*13*	*15*
discs	*error*	–	–	–	–	–	*error*	–	–	–	–	–
dixchlng	*KKT*	*10*	*10*	*10*	*10*	*9*	*KKT*	*10*	*10*	*10*	*10*	*9*
dnieper	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
dual1	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dual2	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dual4	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dualc1	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dualc2	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dualc5	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dualc8	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
eigencco	*KKT*	*34*	*34*	*18*	*18*	*17*	*KKT*	*20*	*20*	*12*	*12*	*11*
eigmaxc	*KKT*	*5*	*6*	*6*	*6*	*6*	*KKT*	*5*	*6*	*6*	*6*	*6*
eigminc	*KKT*	*5*	*6*	*6*	*6*	*6*	*KKT*	*5*	*6*	*6*	*6*	*6*
expfita	*KKT*	*13*	*13*	*13*	*13*	*12*	*KKT*	*13*	*13*	*13*	*13*	*12*
extrasim	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
fccu	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
fletcher	*infeasible*	*2*	*2*	*2*	*2*	*2*	*infeasible*	*2*	*2*	*2*	*2*	*2*
genhs28	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
gigomez1	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
goffin	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
gottfr	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*14*	*30*	*15*	*15*	*35*
gridnetg	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
gridneth	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
gridneti	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
haifas	*KKT*	*12*	*12*	*9*	*9*	*8*	*KKT*	*12*	*12*	*9*	*9*	*8*
haldmads	*KKT*	*40*	*40*	*22*	*22*	*21*	*KKT*	*25*	*26*	*16*	*16*	*16*
hatfldf	*KKT*	*8*	*28*	*14*	*14*	*22*	*KKT*	*8*	*28*	*14*	*14*	*22*
hatfldg	*KKT*	*5*	*18*	*10*	*10*	*10*	*KKT*	*5*	*18*	*10*	*10*	*10*
hatfldh	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
heart6	*KKT*	*4*	*197*	*99*	*99*	*99*	*KKT*	*5*	*199*	*100*	*100*	*102*
heart8	*KKT*	*4*	*144*	*72*	*72*	*72*	*KKT*	*5*	*146*	*73*	*73*	*75*
himmelba	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
himmelbc	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*5*	*6*	*5*	*5*	*5*
himmelbd	*infeasible*	*4*	*10*	*7*	*7*	*7*	*infeasible*	*4*	*10*	*7*	*7*	*7*
himmelbe	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
himmelbk	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
himmelp2	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
himmelp3	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
himmelp4	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
himmelp5	*KKT*	*12*	*12*	*10*	*10*	*9*	*KKT*	*12*	*12*	*10*	*10*	*9*
himmelp6	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hong	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs006	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs007	*KKT*	*12*	*12*	*9*	*9*	*8*	*KKT*	*12*	*12*	*9*	*9*	*8*
hs008	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs009	*KKT*	*5*	*5*	*4*	*4*	*3*	*KKT*	*5*	*5*	*4*	*4*	*3*
hs010	*KKT*	*10*	*10*	*10*	*10*	*9*	*KKT*	*10*	*10*	*10*	*10*	*9*
hs011	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs012	*KKT*	*8*	*8*	*6*	*6*	*5*	*KKT*	*8*	*8*	*6*	*6*	*5*
hs013	*KKT*	*25*	*25*	*25*	*25*	*24*	*KKT*	*25*	*25*	*25*	*25*	*24*
hs014	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs015	*KKT*	*4*	*7*	*6*	*6*	*6*	*KKT*	*4*	*7*	*6*	*6*	*6*

hs016	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs017	*KKT*	*8*	*8*	*7*	*7*	*6*	*KKT*	*8*	*8*	*7*	*7*	*6*
hs018	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs019	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs020	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs021	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs022	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs023	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs024	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs026	*KKT*	*18*	*18*	*18*	*18*	*17*	*KKT*	*18*	*18*	*18*	*18*	*17*
hs027	*KKT*	*17*	*17*	*16*	*16*	*15*	*KKT*	*21*	*22*	*16*	*16*	*16*
hs028	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs029	*KKT*	*8*	*8*	*7*	*7*	*6*	*KKT*	*8*	*8*	*7*	*7*	*6*
hs030	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs031	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs032	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs033	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs034	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
hs035	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs036	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs037	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs039	*KKT*	*19*	*19*	*15*	*15*	*14*	*KKT*	*19*	*25*	*18*	*18*	*18*
hs040	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs041	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs042	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs043	*KKT*	*9*	*9*	*8*	*8*	*7*	*KKT*	*9*	*9*	*8*	*8*	*7*
hs044	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs046	*KKT*	*19*	*19*	*19*	*19*	*18*	*KKT*	*19*	*19*	*19*	*19*	*18*
hs047	*KKT*	*21*	*21*	*18*	*18*	*17*	*KKT*	*21*	*21*	*18*	*18*	*17*
hs048	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs049	*KKT*	*17*	*17*	*17*	*17*	*16*	*KKT*	*17*	*17*	*17*	*17*	*16*
hs050	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
hs051	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs052	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs053	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs054	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs055	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs056	*KKT*	*15*	*16*	*16*	*16*	*16*	*KKT*	*15*	*16*	*16*	*16*	*16*
hs057	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs059	*KKT*	*11*	*12*	*10*	*10*	*10*	*KKT*	*11*	*12*	*10*	*10*	*10*
hs060	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs061	*infeasible*	*1*	*2*	*2*	*2*	*2*	*infeasible*	*1*	*2*	*2*	*2*	*2*
hs062	*KKT*	*8*	*8*	*7*	*7*	*6*	*KKT*	*8*	*8*	*7*	*7*	*6*
hs063	*KKT*	*9*	*11*	*10*	*10*	*10*	*KKT*	*8*	*10*	*8*	*8*	*9*
hs064	*KKT*	*12*	*17*	*17*	*17*	*17*	*KKT*	*12*	*17*	*17*	*17*	*17*
hs065	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs066	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
hs067	*KKT*	*12*	*12*	*12*	*12*	*11*	*KKT*	*12*	*12*	*12*	*12*	*11*
hs070	*KKT*	*40*	*40*	*28*	*28*	*27*	*KKT*	*40*	*40*	*28*	*28*	*27*
hs071	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs072	*KKT*	*15*	*18*	*16*	*16*	*16*	*KKT*	*15*	*18*	*16*	*16*	*16*
hs073	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
hs074	*KKT*	*6*	*26*	*19*	*19*	*19*	*KKT*	*6*	*26*	*19*	*19*	*19*

hs075	*KKT*	*5*	*25*	*18*	*18*	*18*	*KKT*	*5*	*25*	*18*	*18*	*18*
hs076	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs077	*KKT*	*11*	*11*	*10*	*10*	*9*	*KKT*	*11*	*11*	*10*	*10*	*9*
hs078	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs079	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs080	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
hs081	*KKT*	*111*	*113*	*58*	*58*	*59*	*KKT*	*29*	*29*	*18*	*18*	*17*
hs083	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs084	*KKT*	*11*	*11*	*8*	*8*	*7*	*KKT*	*11*	*11*	*8*	*8*	*7*
hs085	*error*	–	–	–	–	–	*error*	–	–	–	–	–
hs086	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs087	*KKT*	*7*	*10*	*10*	*10*	*10*	*KKT*	*7*	*10*	*10*	*10*	*10*
hs088	*KKT*	*22*	*29*	*20*	*20*	*21*	*KKT*	*21*	*24*	*21*	*21*	*22*
hs089	*KKT*	*23*	*24*	*17*	*17*	*17*	*KKT*	*23*	*24*	*17*	*17*	*17*
hs090	*KKT*	*81*	*89*	*50*	*50*	*51*	*KKT*	*74*	*82*	*50*	*50*	*51*
hs091	*KKT*	*125*	*126*	*59*	*59*	*59*	*KKT*	*51*	*52*	*29*	*29*	*29*
hs092	*KKT*	*96*	*106*	*50*	*50*	*51*	*KKT*	*40*	*51*	*28*	*28*	*30*
hs093	*infeasible*	*3*	*3*	*3*	*3*	*3*	*infeasible*	*3*	*3*	*3*	*3*	*3*
hs095	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs096	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs097	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs098	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs099	*small feasible*	*14*	*63*	*41*	*41*	*41*	*small feasible*	*14*	*63*	*41*	*41*	*41*
hs100	*KKT*	*12*	*12*	*9*	*9*	*8*	*KKT*	*12*	*12*	*9*	*9*	*8*
hs100lnp	*KKT*	*14*	*14*	*12*	*12*	*11*	*KKT*	*12*	*12*	*9*	*9*	*8*
hs100mod	*KKT*	*12*	*12*	*9*	*9*	*8*	*KKT*	*12*	*12*	*9*	*9*	*8*
hs101	*KKT*	*22*	*27*	*18*	*18*	*18*	*KKT*	*22*	*27*	*18*	*18*	*18*
hs102	*KKT*	*17*	*18*	*15*	*15*	*15*	*KKT*	*17*	*18*	*15*	*15*	*15*
hs103	*KKT*	*16*	*17*	*14*	*14*	*14*	*KKT*	*37*	*1678*	*843*	*843*	*848*
hs104	*KKT*	*31*	*36*	*25*	*25*	*25*	*KKT*	*17*	*23*	*15*	*15*	*15*
hs106	*KKT*	*771*	*771*	*386*	*386*	*385*	*KKT*	*16*	*16*	*15*	*15*	*14*
hs107	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs108	*KKT*	*21*	*21*	*14*	*14*	*13*	*KKT*	*25*	*25*	*19*	*19*	*18*
hs109	*KKT*	*6*	*24*	*18*	*18*	*18*	*KKT*	*6*	*24*	*18*	*18*	*18*
hs111	*KKT*	*35*	*35*	*21*	*21*	*20*	*KKT*	*40*	*40*	*25*	*25*	*24*
hs111lnp	*KKT*	*35*	*35*	*21*	*21*	*20*	*KKT*	*40*	*40*	*25*	*25*	*24*
hs112	*KKT*	*12*	*12*	*12*	*12*	*11*	*KKT*	*12*	*12*	*12*	*12*	*11*
hs113	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs114	*infeasible*	*1*	*197*	*102*	*102*	*102*	*infeasible*	*1*	*197*	*102*	*102*	*102*
hs116	*KKT*	*12*	*12*	*12*	*12*	*11*	*KKT*	*12*	*12*	*12*	*12*	*11*
hs117	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs118	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs119	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs21mod	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs268	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs35mod	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs44new	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs99exp	*KKT*	*12*	*42*	*32*	*32*	*32*	*KKT*	*12*	*42*	*32*	*32*	*32*
hubfit	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hypcir	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*7*	*6*	*6*	*6*
kiwcresc	*KKT*	*11*	*11*	*9*	*9*	*8*	*KKT*	*11*	*11*	*9*	*9*	*8*
lakes	*KKT*	*12*	*5456*	*2737*	*2737*	*2737*	*KKT*	*12*	*5456*	*2737*	*2737*	*2737*
launch	*error*	–	–	–	–	–	*infeasible*	*1*	*1523*	*735*	*735*	*735*

lewispol	*infeasible*	*1*	*4*	*4*	*4*	*4*	*infeasible*	*1*	*4*	*4*	*4*	*4*
linspanh	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
loadbal	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
lootsma	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
lotschd	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
lsnnodoc	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
lsqfit	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
madsen	*KKT*	*14*	*14*	*11*	*11*	*10*	*KKT*	*14*	*14*	*11*	*11*	*10*
makela1	*KKT*	*12*	*12*	*10*	*10*	*9*	*KKT*	*12*	*12*	*10*	*10*	*9*
makela2	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
makela3	*KKT*	*22*	*22*	*20*	*20*	*19*	*KKT*	*22*	*22*	*20*	*20*	*19*
makela4	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
maratos	*KKT*	*10*	*10*	*9*	*9*	*8*	*KKT*	*10*	*10*	*9*	*9*	*8*
matrix2	*KKT*	*12*	*12*	*12*	*12*	*11*	*KKT*	*12*	*12*	*12*	*12*	*11*
mconcon	*KKT*	*5*	*11*	*10*	*10*	*10*	*KKT*	*7*	*16*	*12*	*12*	*16*
mifflin1	*KKT*	*10*	*10*	*9*	*9*	*8*	*KKT*	*10*	*10*	*9*	*9*	*8*
mifflin2	*KKT*	*11*	*11*	*8*	*8*	*7*	*KKT*	*10*	*10*	*8*	*8*	*7*
minmaxbd	*KKT*	*32*	*45*	*25*	*25*	*27*	*KKT*	*22*	*25*	*18*	*18*	*18*
minmaxrb	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
mistake	*KKT*	*27*	*27*	*17*	*17*	*16*	*KKT*	*22*	*22*	*19*	*19*	*18*
model	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
mwright	*KKT*	*9*	*9*	*7*	*7*	*6*	*KKT*	*9*	*9*	*7*	*7*	*6*
nuffield[1]	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
odfits	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
optcntrl	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
optmass	*KKT*	*9*	*9*	*7*	*7*	*6*	*KKT*	*9*	*9*	*7*	*7*	*6*
optprloc	*KKT*	*18*	*18*	*10*	*10*	*9*	*KKT*	*16*	*16*	*9*	*9*	*8*
orthregb	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
orthrege	*KKT*	*2334*	*2364*	*1183*	*1183*	*1185*	*KKT*	*2896*	*2898*	*1450*	*1450*	*1450*
pentagon	*KKT*	*8*	*8*	*6*	*6*	*5*	*KKT*	*8*	*8*	*6*	*6*	*5*
polak1	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
polak2	*KKT*	*22*	*31*	*20*	*20*	*23*	*KKT*	*46*	*55*	*32*	*32*	*35*
polak3	*KKT*	*21*	*26*	*15*	*15*	*15*	*KKT*	*21*	*26*	*15*	*15*	*15*
polak4	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
polak5	*KKT*	*43*	*57*	*45*	*45*	*45*	*KKT*	*82*	*96*	*47*	*47*	*47*
polak6	*KKT*	*31*	*34*	*17*	*17*	*18*	*KKT*	*43*	*48*	*23*	*23*	*25*
portfl1	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
portfl2	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
portfl3	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
portfl4	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
portfl6	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
powellbs	*KKT*	*12*	*12*	*12*	*12*	*11*	*KKT*	*10*	*115*	*60*	*60*	*62*
powellsq	*KKT*	*21*	*35*	*26*	*26*	*26*	*KKT*	*80*	*277*	*135*	*135*	*216*
prodpl0	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
prodpl1	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
recipe	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
res	*KKT*	*2*	*2*	*2*	*2*	*1*	*small feasible*	*3*	*3*	*2*	*2*	*1*
rk23	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*7*	*9*	*7*	*7*	*7*
robot	*KKT*	*14*	*21*	*12*	*12*	*12*	*KKT*	*14*	*21*	*12*	*12*	*12*
rosenmmx	*KKT*	*24*	*27*	*14*	*14*	*14*	*KKT*	*36*	*40*	*19*	*19*	*20*
s365mod	*KKT*	*32*	*48*	*27*	*27*	*32*	*KKT*	*23*	*29*	*18*	*18*	*18*
simpllpa	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*

[1]full name: nuffield_continuum, which this table is too narrow to contain.

simpllpb	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
snake	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
spanhyd	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
spiral	*KKT*	*26*	*26*	*21*	*21*	*20*	*KKT*	*113*	*117*	*88*	*88*	*90*
ssnlbeam	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
stancmin	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
supersim	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
swopf	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
synthes1	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
tame	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
try-b	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
twobars	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
vanderm4	*infeasible*	*1*	*17*	*17*	*17*	*17*	*infeasible*	*1*	*17*	*17*	*17*	*17*
womflet	*KKT*	*33*	*41*	*27*	*27*	*27*	*KKT*	*42*	*46*	*24*	*24*	*25*
zangwil3	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
zecevic2	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
zecevic3	*KKT*	*10*	*15*	*10*	*10*	*10*	*KKT*	*10*	*15*	*10*	*10*	*10*
zecevic4	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
zigzag	*KKT*	*11*	*11*	*11*	*11*	*10*	*KKT*	*11*	*11*	*11*	*11*	*10*
zy2	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*

B Detailed Results on `CUTEst` problems: Number of Function and Derivative Evaluations for Line-Search Algorithms

	line-search funnel method						*line-search filter method*					
instance	*status*	f	c	∇f	∇c	$\nabla^2\mathcal{L}$	*status*	f	c	∇f	∇c	$\nabla^2\mathcal{L}$
aircrfta	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
airport	*KKT*	*13*	*13*	*13*	*13*	*12*	*KKT*	*13*	*13*	*13*	*13*	*12*
aljazzaf	*KKT*	*58*	*58*	*34*	*34*	*33*	*KKT*	*131*	*171*	*40*	*40*	*40*
allinitc	*KKT*	*29*	*29*	*29*	*29*	*28*	*KKT*	*29*	*29*	*29*	*29*	*28*
alsotame	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
argauss	*infeasible*	*1*	*3*	*3*	*3*	*3*	*infeasible*	*1*	*3*	*3*	*3*	*3*
avgasa	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
avgasb	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
avion2	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
batch	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
biggsc4	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
booth	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
bt1	*infeasible*	*2*	*2*	*2*	*2*	*2*	*infeasible*	*2*	*2*	*2*	*2*	*2*
bt10	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
bt11	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
bt12	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
bt13	*KKT*	*24*	*24*	*21*	*21*	*20*	*KKT*	*24*	*24*	*21*	*21*	*20*
bt2	*KKT*	*13*	*13*	*13*	*13*	*12*	*KKT*	*13*	*13*	*13*	*13*	*12*
bt3	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
bt4	*KKT*	*50*	*50*	*50*	*50*	*49*	*KKT*	*50*	*50*	*50*	*50*	*49*
bt5	*KKT*	*37*	*37*	*37*	*37*	*36*	*KKT*	*37*	*37*	*37*	*37*	*36*
bt6	*KKT*	*30*	*30*	*30*	*30*	*29*	*KKT*	*30*	*30*	*30*	*30*	*29*
bt7	*KKT*	*87*	*87*	*53*	*53*	*52*	*KKT*	*109*	*109*	*52*	*52*	*52*
bt8	*KKT*	*28*	*28*	*28*	*28*	*27*	*KKT*	*28*	*28*	*28*	*28*	*27*
bt9	*KKT*	*63*	*63*	*22*	*22*	*21*	*KKT*	*52*	*52*	*20*	*20*	*19*

byrdsphr	*error*	–	–	–	–	–	*KKT*	*5495*	*5589*	*453*	*453*	*498*
cantilvr	*KKT*	*21*	*21*	*14*	*14*	*13*	*KKT*	*24*	*24*	*14*	*14*	*13*
catena	*KKT*	*340*	*340*	*31*	*31*	*32*	*KKT*	*244*	*244*	*25*	*25*	*25*
cb2	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
cb3	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
chaconn1	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
chaconn2	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
cluster	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
concon	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
congigmz	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
coolhans	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*57*	*65*	*11*	*11*	*11*
core1	*KKT*	*254*	*254*	*254*	*254*	*253*	*KKT*	*254*	*254*	*254*	*254*	*253*
coshfun	*error*	–	–	–	–	–	*error*	–	–	–	–	–
cresc4	*error*	–	–	–	–	–	*error*	–	–	–	–	–
csfi1	*KKT*	*14*	*14*	*11*	*11*	*10*	*KKT*	*47*	*47*	*27*	*27*	*26*
csfi2	*KKT*	*72*	*72*	*31*	*31*	*30*	*KKT*	*88*	*88*	*31*	*31*	*30*
dallass	*KKT*	*17*	*17*	*14*	*14*	*13*	*KKT*	*17*	*17*	*14*	*14*	*13*
deconvc	*KKT*	*82*	*82*	*82*	*82*	*81*	*KKT*	*82*	*82*	*82*	*82*	*81*
degenlpa	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
degenlpb	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
demymalo	*KKT*	*17*	*17*	*8*	*8*	*7*	*KKT*	*17*	*17*	*8*	*8*	*7*
dipigri	*KKT*	*17*	*17*	*8*	*8*	*7*	*KKT*	*17*	*17*	*8*	*8*	*7*
disc2	*infeasible*	*2*	*198*	*198*	*198*	*198*	*infeasible*	*2*	*198*	*198*	*198*	*198*
discs	*KKT*	*2758*	*2767*	*2767*	*2767*	*2767*	*KKT*	*2758*	*2767*	*2767*	*2767*	*2767*
dixchlng	*KKT*	*224*	*224*	*208*	*208*	*207*	*error*	–	–	–	–	–
dnieper	*KKT*	*577*	*577*	*577*	*577*	*576*	*KKT*	*577*	*577*	*577*	*577*	*576*
dual1	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
dual2	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
dual4	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
dualc1	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dualc2	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
dualc5	*error*	–	–	–	–	–	*error*	–	–	–	–	–
dualc8	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
eigencco	*KKT*	*13*	*13*	*13*	*13*	*12*	*KKT*	*13*	*13*	*13*	*13*	*12*
eigmaxc	*KKT*	*6*	*7*	*7*	*7*	*7*	*KKT*	*6*	*7*	*7*	*7*	*7*
eigminc	*KKT*	*6*	*7*	*7*	*7*	*7*	*KKT*	*6*	*7*	*7*	*7*	*7*
expfita	*KKT*	*14*	*14*	*14*	*14*	*13*	*KKT*	*14*	*14*	*14*	*14*	*13*
extrasim	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
fccu	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
fletcher	*infeasible*	*2*	*2*	*2*	*2*	*2*	*infeasible*	*2*	*2*	*2*	*2*	*2*
genhs28	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
gigomez1	*KKT*	*17*	*17*	*8*	*8*	*7*	*KKT*	*17*	*17*	*8*	*8*	*7*
goffin	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
gottfr	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*6*	*6*	*5*
gridnetg	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
gridneth	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
gridneti	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
haifas	*KKT*	*31*	*31*	*9*	*9*	*8*	*KKT*	*31*	*31*	*9*	*9*	*8*
haldmads	*KKT*	*143*	*143*	*136*	*136*	*135*	*error*	–	–	–	–	–
hatfldf	*KKT*	*9*	*9*	*8*	*8*	*7*	*infeasible*	*47*	*3267*	*1524*	*1524*	*1524*
hatfldg	*KKT*	*17*	*17*	*9*	*9*	*8*	*KKT*	*21*	*21*	*9*	*9*	*8*
hatfldh	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
heart6	*KKT*	*3*	*2846*	*2843*	*2843*	*2843*	*KKT*	*3*	*2846*	*2843*	*2843*	*2843*

heart8	KKT	5	26	19	19	19	KKT	5	26	19	19	19
himmelba	KKT	2	2	2	2	1	KKT	2	2	2	2	1
himmelbc	KKT	6	6	6	6	5	KKT	7	7	6	6	5
himmelbd	infeasible	134	181	37	37	37	infeasible	57	109	21	21	21
himmelbe	KKT	3	3	3	3	2	KKT	3	3	3	3	2
himmelbk	KKT	708	708	708	708	707	KKT	708	708	708	708	707
himmelp2	KKT	10	10	10	10	9	KKT	10	10	10	10	9
himmelp3	KKT	5	5	5	5	4	KKT	5	5	5	5	4
himmelp4	KKT	6	6	6	6	5	KKT	6	6	6	6	5
himmelp5	KKT	16	16	16	16	15	KKT	16	16	16	16	15
himmelp6	KKT	2	2	2	2	1	KKT	2	2	2	2	1
hong	KKT	5	5	5	5	4	KKT	5	5	5	5	4
hs006	KKT	5	5	5	5	4	KKT	5	5	5	5	4
hs007	KKT	16	16	11	11	10	KKT	16	16	11	11	10
hs008	KKT	6	6	6	6	5	KKT	6	6	6	6	5
hs009	KKT	5	5	4	4	3	KKT	5	5	4	4	3
hs010	KKT	39	39	15	15	14	KKT	52	52	18	18	17
hs011	KKT	6	6	6	6	5	KKT	6	6	6	6	5
hs012	KKT	9	9	7	7	6	KKT	9	9	7	7	6
hs013	KKT	26	26	26	26	25	KKT	26	26	26	26	25
hs014	KKT	6	6	6	6	5	KKT	6	6	6	6	5
hs015	KKT	4	9	9	9	9	KKT	4	9	9	9	9
hs016	KKT	6	6	6	6	5	KKT	6	6	6	6	5
hs017	KKT	10	10	9	9	8	KKT	10	10	9	9	8
hs018	KKT	7	7	7	7	6	KKT	7	7	7	7	6
hs019	KKT	7	7	7	7	6	KKT	7	7	7	7	6
hs020	KKT	5	5	5	5	4	KKT	5	5	5	5	4
hs021	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs022	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs023	KKT	7	7	7	7	6	KKT	7	7	7	7	6
hs024	KKT	16	16	16	16	15	KKT	16	16	16	16	15
hs026	KKT	19	19	19	19	18	KKT	19	19	19	19	18
hs027	KKT	14	14	14	14	13	KKT	38	38	16	16	15
hs028	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs029	KKT	25	25	25	25	24	KKT	25	25	25	25	24
hs030	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs031	KKT	7	7	7	7	6	KKT	7	7	7	7	6
hs032	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs033	KKT	16	16	16	16	15	KKT	16	16	16	16	15
hs034	KKT	8	8	8	8	7	KKT	8	8	8	8	7
hs035	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs036	KKT	7	7	7	7	6	KKT	7	7	7	7	6
hs037	KKT	138	138	132	132	131	KKT	138	138	132	132	131
hs039	KKT	63	63	22	22	21	KKT	52	52	20	20	19
hs040	KKT	19	19	19	19	18	KKT	19	19	19	19	18
hs041	KKT	2	2	2	2	1	KKT	2	2	2	2	1
hs042	KKT	6	6	6	6	5	KKT	6	6	6	6	5
hs043	KKT	8	8	7	7	6	KKT	8	8	7	7	6
hs044	KKT	10	10	10	10	9	KKT	10	10	10	10	9
hs046	KKT	18	18	18	18	17	KKT	18	18	18	18	17
hs047	KKT	18	18	18	18	17	KKT	18	18	18	18	17
hs048	KKT	3	3	3	3	2	KKT	3	3	3	3	2
hs049	KKT	18	18	18	18	17	KKT	18	18	18	18	17

hs050	*KKT*	*10*	*10*	*10*	*10*	*9*	*KKT*	*10*	*10*	*10*	*10*	*9*
hs051	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs052	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs053	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs054	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs055	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
hs056	*KKT*	*102*	*102*	*102*	*102*	*101*	*KKT*	*102*	*102*	*102*	*102*	*101*
hs057	*KKT*	*26*	*26*	*26*	*26*	*25*	*KKT*	*26*	*26*	*26*	*26*	*25*
hs059	*KKT*	*18*	*18*	*12*	*12*	*11*	*KKT*	*18*	*18*	*12*	*12*	*11*
hs060	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs061	*infeasible*	*2*	*3*	*3*	*3*	*3*	*infeasible*	*2*	*3*	*3*	*3*	*3*
hs062	*KKT*	*7*	*7*	*6*	*6*	*5*	*KKT*	*7*	*7*	*6*	*6*	*5*
hs063	*KKT*	*42*	*44*	*43*	*43*	*43*	*KKT*	*42*	*44*	*43*	*43*	*43*
hs064	*KKT*	*16*	*16*	*16*	*16*	*15*	*KKT*	*16*	*16*	*16*	*16*	*15*
hs065	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs066	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
hs067	*error*	–	–	–	–	–	*error*	–	–	–	–	–
hs070	*KKT*	*25*	*25*	*19*	*19*	*18*	*KKT*	*25*	*25*	*19*	*19*	*18*
hs071	*KKT*	*47*	*47*	*47*	*47*	*46*	*KKT*	*47*	*47*	*47*	*47*	*46*
hs072	*KKT*	*17*	*17*	*15*	*15*	*14*	*KKT*	*18*	*18*	*15*	*15*	*14*
hs073	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
hs074	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs075	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs076	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs077	*KKT*	*12*	*12*	*11*	*11*	*10*	*KKT*	*12*	*12*	*11*	*11*	*10*
hs078	*KKT*	*35*	*35*	*35*	*35*	*34*	*KKT*	*35*	*35*	*35*	*35*	*34*
hs079	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs080	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
hs081	*error*	–	–	–	–	–	*error*	–	–	–	–	–
hs083	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs084	*KKT*	*2859*	*2859*	*2859*	*2859*	*2858*	*KKT*	*2859*	*2859*	*2859*	*2859*	*2858*
hs085	*infeasible*	*1*	*34*	*23*	*23*	*23*	*infeasible*	*1*	*34*	*23*	*23*	*23*
hs086	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
hs087	*KKT*	*20*	*20*	*20*	*20*	*19*	*KKT*	*20*	*20*	*20*	*20*	*19*
hs088	*infeasible*	*6*	*6*	*6*	*6*	*6*	*KKT*	*19*	*19*	*17*	*17*	*16*
hs089	*KKT*	*24*	*24*	*19*	*19*	*18*	*KKT*	*24*	*24*	*19*	*19*	*18*
hs090	*infeasible*	*4*	*4*	*4*	*4*	*4*	*KKT*	*18*	*18*	*12*	*12*	*11*
hs091	*KKT*	*32*	*32*	*29*	*29*	*28*	*KKT*	*39*	*39*	*25*	*25*	*24*
hs092	*infeasible*	*4*	*4*	*4*	*4*	*4*	*KKT*	*31*	*31*	*15*	*15*	*14*
hs093	*KKT*	*1558*	*1558*	*1558*	*1558*	*1557*	*KKT*	*1558*	*1558*	*1558*	*1558*	*1557*
hs095	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs096	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs097	*KKT*	*63*	*63*	*63*	*63*	*62*	*KKT*	*63*	*63*	*63*	*63*	*62*
hs098	*KKT*	*63*	*63*	*63*	*63*	*62*	*KKT*	*63*	*63*	*63*	*63*	*62*
hs099	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs100	*KKT*	*17*	*17*	*8*	*8*	*7*	*KKT*	*17*	*17*	*8*	*8*	*7*
hs100lnp	*KKT*	*27*	*27*	*21*	*21*	*20*	*KKT*	*24*	*24*	*17*	*17*	*16*
hs100mod	*KKT*	*25*	*25*	*10*	*10*	*9*	*KKT*	*25*	*25*	*10*	*10*	*9*
hs101	*KKT*	*24*	*24*	*18*	*18*	*17*	*KKT*	*24*	*24*	*18*	*18*	*17*
hs102	*KKT*	*21*	*21*	*19*	*19*	*18*	*KKT*	*87*	*96*	*26*	*26*	*26*
hs103	*KKT*	*14*	*14*	*12*	*12*	*11*	*KKT*	*35*	*35*	*27*	*27*	*26*
hs104	*KKT*	*41*	*41*	*41*	*41*	*40*	*KKT*	*126*	*130*	*51*	*51*	*51*
hs106	*KKT*	*748*	*748*	*744*	*744*	*743*	*KKT*	*724*	*724*	*721*	*721*	*720*

hs107	*KKT*	*11*	*11*	*11*	*11*	*10*	*KKT*	*11*	*11*	*11*	*11*	*10*
hs108	*KKT*	*11*	*11*	*11*	*11*	*10*	*KKT*	*11*	*11*	*11*	*11*	*10*
hs109	*KKT*	*45*	*46*	*46*	*46*	*46*	*KKT*	*45*	*46*	*46*	*46*	*46*
hs111	*KKT*	*17*	*17*	*17*	*17*	*16*	*KKT*	*17*	*17*	*17*	*17*	*16*
hs111lnp	*KKT*	*17*	*17*	*17*	*17*	*16*	*KKT*	*17*	*17*	*17*	*17*	*16*
hs112	*KKT*	*12*	*12*	*12*	*12*	*11*	*KKT*	*12*	*12*	*12*	*12*	*11*
hs113	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
hs114	*error*	–	–	–	–	–	*error*	–	–	–	–	–
hs116	*error*	–	–	–	–	–	*error*	–	–	–	–	–
hs117	*KKT*	*8*	*8*	*7*	*7*	*6*	*KKT*	*8*	*8*	*7*	*7*	*6*
hs118	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs119	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs21mod	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs268	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
hs35mod	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hs44new	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
hs99exp	*error*	–	–	–	–	–	*error*	–	–	–	–	–
hubfit	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
hypcir	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*5*	*5*	*4*
kiwcresc	*KKT*	*215*	*215*	*23*	*23*	*24*	*KKT*	*152*	*154*	*22*	*22*	*22*
lakes	*error*	–	–	–	–	–	*error*	–	–	–	–	–
launch	*error*	–	–	–	–	–	*error*	–	–	–	–	–
lewispol	*infeasible*	*1*	*4*	*4*	*4*	*4*	*infeasible*	*1*	*4*	*4*	*4*	*4*
linspanh	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
loadbal	*KKT*	*14*	*14*	*14*	*14*	*13*	*KKT*	*14*	*14*	*14*	*14*	*13*
lootsma	*KKT*	*16*	*16*	*16*	*16*	*15*	*KKT*	*16*	*16*	*16*	*16*	*15*
lotschd	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
lsnnodoc	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
lsqfit	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
madsen	*KKT*	*10*	*10*	*10*	*10*	*9*	*KKT*	*10*	*10*	*10*	*10*	*9*
makela1	*KKT*	*18*	*18*	*9*	*9*	*8*	*KKT*	*18*	*18*	*9*	*9*	*8*
makela2	*KKT*	*14*	*14*	*6*	*6*	*5*	*KKT*	*14*	*14*	*6*	*6*	*5*
makela3	*KKT*	*83*	*83*	*28*	*28*	*27*	*KKT*	*71*	*71*	*23*	*23*	*22*
makela4	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
maratos	*KKT*	*37*	*37*	*15*	*15*	*14*	*KKT*	*32*	*32*	*14*	*14*	*13*
matrix2	*KKT*	*22*	*22*	*22*	*22*	*21*	*KKT*	*22*	*22*	*22*	*22*	*21*
mconcon	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
mifflin1	*KKT*	*48*	*48*	*15*	*15*	*14*	*KKT*	*39*	*39*	*13*	*13*	*12*
mifflin2	*KKT*	*56*	*56*	*14*	*14*	*13*	*KKT*	*60*	*60*	*14*	*14*	*13*
minmaxbd	*KKT*	*59*	*59*	*15*	*15*	*14*	*KKT*	*74*	*74*	*19*	*19*	*18*
minmaxrb	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
mistake	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
model	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
mwright	*KKT*	*23*	*23*	*22*	*22*	*21*	*KKT*	*23*	*23*	*22*	*22*	*21*
nuffield	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
odfits	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
optcntrl	*KKT*	*4*	*4*	*4*	*4*	*3*	*KKT*	*4*	*4*	*4*	*4*	*3*
optmass	*KKT*	*808*	*808*	*808*	*808*	*807*	*KKT*	*808*	*808*	*808*	*808*	*807*
optprloc	*KKT*	*8*	*8*	*6*	*6*	*5*	*KKT*	*8*	*8*	*6*	*6*	*5*
orthregb	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
orthrege	*error*	–	–	–	–	–	*error*	–	–	–	–	–
pentagon	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
polak1	*KKT*	*93*	*93*	*15*	*15*	*15*	*KKT*	*69*	*69*	*17*	*17*	*16*

polak2	*KKT*	*593*	*641*	*454*	*454*	*455*	*KKT*	*553*	*587*	*451*	*451*	*451*
polak3	*KKT*	*53*	*53*	*15*	*15*	*14*	*KKT*	*87*	*96*	*18*	*18*	*18*
polak4	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
polak5	*KKT*	*30*	*30*	*25*	*25*	*24*	*KKT*	*66*	*66*	*42*	*42*	*41*
polak6	*KKT*	*73*	*73*	*16*	*16*	*15*	*KKT*	*67*	*67*	*15*	*15*	*14*
portfl1	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
portfl2	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
portfl3	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
portfl4	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
portfl6	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
powellbs	*KKT*	*15*	*15*	*15*	*15*	*14*	*KKT*	*223*	*223*	*57*	*57*	*56*
powellsq	*KKT*	*41*	*41*	*34*	*34*	*33*	*error*	–	–	–	–	–
prodpl0	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
prodpl1	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
recipe	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
res	*KKT*	*2*	*2*	*2*	*2*	*1*	*error*	–	–	–	–	–
rk23	*KKT*	*10*	*10*	*10*	*10*	*9*	*KKT*	*13*	*13*	*12*	*12*	*11*
robot	*KKT*	*19*	*23*	*20*	*20*	*20*	*KKT*	*21*	*25*	*21*	*21*	*21*
rosenmmx	*KKT*	*75*	*75*	*14*	*14*	*13*	*KKT*	*79*	*79*	*15*	*15*	*14*
s365mod	*KKT*	*1333*	*1333*	*1333*	*1333*	*1332*	*KKT*	*1339*	*1339*	*1337*	*1337*	*1336*
simpllpa	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
simpllpb	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
snake	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
spanhyd	*KKT*	*6*	*6*	*6*	*6*	*5*	*KKT*	*6*	*6*	*6*	*6*	*5*
spiral	*KKT*	*215*	*215*	*107*	*107*	*106*	*KKT*	*99*	*99*	*72*	*72*	*71*
ssnlbeam	*error*	–	–	–	–	–	*error*	–	–	–	–	–
stancmin	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
supersim	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
swopf	*error*	–	–	–	–	–	*error*	–	–	–	–	–
synthes1	*KKT*	*5*	*5*	*5*	*5*	*4*	*KKT*	*5*	*5*	*5*	*5*	*4*
tame	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
try-b	*KKT*	*9*	*9*	*9*	*9*	*8*	*KKT*	*9*	*9*	*9*	*9*	*8*
twobars	*KKT*	*8*	*8*	*8*	*8*	*7*	*KKT*	*8*	*8*	*8*	*8*	*7*
vanderm4	*infeasible*	*1*	*22*	*22*	*22*	*22*	*infeasible*	*1*	*22*	*22*	*22*	*22*
womflet	*error*	–	–	–	–	–	*error*	–	–	–	–	–
zangwil3	*KKT*	*2*	*2*	*2*	*2*	*1*	*KKT*	*2*	*2*	*2*	*2*	*1*
zecevic2	*KKT*	*3*	*3*	*3*	*3*	*2*	*KKT*	*3*	*3*	*3*	*3*	*2*
zecevic3	*KKT*	*9*	*9*	*8*	*8*	*7*	*KKT*	*9*	*9*	*8*	*8*	*7*
zecevic4	*KKT*	*7*	*7*	*7*	*7*	*6*	*KKT*	*7*	*7*	*7*	*7*	*6*
zigzag	*KKT*	*131*	*131*	*131*	*131*	*130*	*KKT*	*131*	*131*	*131*	*131*	*130*
zy2	*KKT*	*16*	*16*	*16*	*16*	*15*	*KKT*	*16*	*16*	*16*	*16*	*15*