Les Cahiers du GERAD ISSN: 0711–2440

An exact penalty method for large-scale equality-constrained nonlinear optimization

M. Gollier, Y. Diouane, D. Orban

G–2026–53

September 2026

La collection *Les Cahiers du GERAD* est constituée des travaux de recherche menés par nos membres. La plupart de ces documents de travail a été soumis à des revues avec comité de révision. Lorsqu'un document est accepté et publié, le pdf original est retiré si c'est nécessaire et un lien vers l'article publié est ajouté.

Citation suggérée : M. Gollier, Y. Diouane, D. Orban (Septembre 2026). An exact penalty method for large-scale equality-constrained nonlinear optimization, Rapport technique, Les Cahiers du GERAD G– 2026–53, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2026-53) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

The series *Les Cahiers du GERAD* consists of working papers carried out by our members. Most of these pre-prints have been submitted to peer-reviewed journals. When accepted and published, if necessary, the original pdf is removed and a link to the published article is added.

Suggested citation: M. Gollier, Y. Diouane, D. Orban (September 2026). An exact penalty method for large-scale equality-constrained nonlinear optimization, Technical report, Les Cahiers du GERAD G–2026–53, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2026-53) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2026
– Bibliothèque et Archives Canada, 2026

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2026
– Library and Archives Canada, 2026

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

An exact penalty method for large-scale equality-constrained nonlinear optimization

Maxence Gollier
Youssef Diouane
Dominique Orban

GERAD, Montréal (Qc), Canada, H3T 1J4

Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 1J4

maxence-2.gollier@polymtl.ca
youssef.diouane@polymtl.ca
dominique.orban@gerad.ca

September 2026
Les Cahiers du GERAD
G–2026–53

Copyright © 2026 Gollier, Diouane, Orban

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs. Les auteurs conservent leur droit d'auteur et leurs droits moraux sur leurs publications et les utilisateurs s'engagent à reconnaître et respecter les exigences légales associées à ces droits. Ainsi, les utilisateurs:

- Peuvent télécharger et imprimer une copie de toute publication du portail public aux fins d'étude ou de recherche privée;
- Ne peuvent pas distribuer le matériel ou l'utiliser pour une activité à but lucratif ou pour un gain commercial;
- Peuvent distribuer gratuitement l'URL identifiant la publication.

Si vous pensez que ce document enfreint le droit d'auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l'accès au travail et enquêterons sur votre demande.

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

- May download and print one copy of any publication from the public portal for the purpose of private study or research;
- May not further distribute the material or use it for any profit-making activity or commercial gain;
- May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Abstract : We present an exact penalty method for equality-constrained nonlinear optimization. Our approach transforms the smooth, constrained problem into a sequence of nonsmooth, unconstrained problems by penalizing the constraint violation by its Euclidean norm. An outer loop updates the penalty parameter and the subproblem tolerances, while an inner loop solves each penalized subproblem with a quadratic regularization method. The method also includes a local infeasibility detection mechanism, an extrapolation strategy for warm-starting subproblems, and nonmonotone acceptance rules. We establish global convergence to first-order stationary points under a bounded penalty parameter assumption, with a worst-case complexity of $\mathcal{O}(|\log \epsilon| \, \epsilon^{-2})$ inner iterations. We implement the method in Penelopt, an open-source Julia solver for constrained nonlinear optimization. On 224 problems from the CUTEst collection, Penelopt is competitive with IPOPT and more robust than the recent solvers MadNLP and Uno. These results establish exact penalty methods as a practical alternative to conventional methods for solving equality-constrained nonlinear optimization problems. To the best of our knowledge, Penelopt is the first practical implementation of an exact penalty method for large-scale constrained nonlinear optimization.

Keywords : Nonlinear optimization; nonconvex optimization; constrained optimization; exact penalty; complexity analysis; Julia

Acknowledgements: Research of M. G. is partially supported by a PBEEE Scholarship. Research of Y. D and D. O. is partially supported by an NSERC Discovery Grant.

1 Introduction

We present a nonsmooth exact penalty method to solve the general equality-constrained nonlinear optimization problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f(x) \quad \text{subject to} \;\; c(x) = 0, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$, are continuously differentiable and may be nonconvex.

Following the approach of the penalty methods presented in [19], our algorithm computes a sequence of approximate solutions of nonsmooth penalized problems

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \;\; f(x) + \tau\|c(x)\|_2, \tag{2}$$

for a succession of non-decreasing penalty parameters $\tau > 0$.

Our work builds directly upon that of Diouane et al. [19], who propose the first practical implementation of the ℓ_2-penalty method for solving small-scale instances of (1), relying on the proximal methods of Aravkin et al. [4] and Diouane et al. [20] to handle the resulting nonsmooth penalized subproblems (2). The present work significantly extends both the algorithmic design and the practical implementation of Diouane et al. [19]. We emphasize the following contributions:

1. We introduce a new step computation and acceptance rule that substantially reduces the number of linear system solves required by the algorithm compared to [19];
2. We incorporate several practical heuristics that improve performance on a broad range of problems;
3. We provide a thoroughly tested and well-documented open-source Julia [7] implementation named *Penelopt*.

Together, these advances allow us to draw significantly stronger conclusions than those of Diouane et al. [19]. In particular, our numerical results show that Penelopt is competitive with the state-of-the-art solver IPOPT on large-scale problems, establishing exact ℓ_2-norm penalty methods as a practical alternative to SQP methods for solving (1). Moreover, to the best of our knowledge, Penelopt is the first practical implementation of an exact penalty method for large-scale constrained nonlinear optimization.

The paper is organized as follows. In Section 2, we review key concepts and results that underpin our algorithm. In Section 3, we describe the overall algorithm. The convergence properties of the method are then established in Section 4 under mild assumptions, followed by the presentation of the step computation procedure in Section 5. In Section 6, we discuss the practical aspects of the implementation, including nonmonotone strategies and other heuristics designed to improve performance. Section 7 describes the design and testing of our software. Finally, in Section 8, we evaluate the numerical efficiency and robustness of our solver on CUTEst instances [32], and compare its performance with the state-of-the-art solver IPOPT [59] as well as the recent Uno [57, 58] and the pure Julia MadNLP [56] solvers.

Related work

The pioneering work of Pietrzykowski [54] was the first to establish the theoretical advantages of nonsmooth penalty functions, with particular emphasis on the ℓ_1-norm penalty. His foundational results were subsequently extended and strengthened by Charalambous [14], Han and Mangasarian [35], Coleman and Conn [15], and Bazaraa and Goode [5].

More recently, Cartis et al. [12] introduced trust-region and quadratic regularization algorithms for solving general *composite* optimization problems of the form

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\; f(x) + h(c(x)), \tag{3}$$

where h is convex and Lipschitz continuous, encompassing nonsmooth penalty formulations as a special case. Unlike the present work, Cartis et al. [12] do not provide a practical implementation of their methods. Independently, Aravkin et al. [4] and Diouane et al. [20] developed globally convergent proximal methods capable of solving (3), in which steps are computed by considering a model of $h \circ c$. Then, Diouane et al. [19] proposed an ℓ_2-norm penalty approach for (1), in which the resulting penalized subproblems are handled by such proximal methods, as discussed above. In addition, Diouane et al. [19] provide a first practical implementation for small-scale problems and describe, contrary to [12], a practical step computation procedure to solve (2). We now briefly review alternative state-of-the-art methods for solving (1).

The augmented Lagrangian, originally due to Hestenes [37] and Powell [55], is a prominent framework for constrained optimization. It applies a quadratic penalty to the Lagrangian of (1). Its theory is developed in depth by Bertsekas [6]. State-of-the-art implementations include ALGENCAN [3, 8], LANCELOT [16, 17], and MINOS [47, 48].

Sequential quadratic programming (SQP) constitutes another major paradigm for (1). It solves a sequence of quadratic subproblems that approximate (1). The SQP framework was first introduced by Wilson [61] and subsequently popularized by Palomares and Mangasarian [53] and Han [34]. An overview of SQP is provided by Boggs and Tolle [9]. Widely used implementations include SNOPT [27], FilterSQP [25], and, more recently, Uno [57, 58]. In addition, the IPOPT [59], KNITRO [11, 60], and MadNLP [56] solvers reduce to SQP methods when applied to (1).

Notation

The identity matrix of size n is denoted by I_n, or I if the context is clear. We use $J(x)$ to denote the Jacobian of c at x. We call a function $\mathcal{C}^1$ if it is continuously differentiable. For $p \geq 1$, the vector ℓ_p-norm is $\|\cdot\|_p$. For matrices, $\|\cdot\|_p$ denotes the operator norm induced by the vector ℓ_p-norm. We use $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ to denote the smallest and largest eigenvalues of a symmetric matrix A, respectively. We write $A \succ 0$ if A is positive definite, i.e., $\lambda_{\min}(A) > 0$, and $A \succeq 0$ if A is positive semidefinite, i.e., $\lambda_{\min}(A) \geq 0$. If g, h are two positive functions of $\epsilon > 0$, the notation $g = \mathcal{O}(h)$ means that there is $C > 0$ such that $\limsup_{\epsilon\to 0} g(\epsilon)/h(\epsilon) \leq C$. When clear from the context, given $x_k \in \mathbb{R}^n$, we write f_k for $f(x_k)$, ∇f_k for $\nabla f(x_k)$, c_k for $c(x_k)$ and J_k for $J(x_k)$. We refer to ϵ_{mach} as the machine precision for the finite precision arithmetic. All numerical results presented in this paper are performed in double precision with $\epsilon_{\text{mach}} \approx 10^{-16}$.

2 Background

We say that the Mangasarian-Fromovitz constraint qualification (MFCQ) holds at $\bar{x}$ for (1) if $J(\bar{x})$ has full row rank.

Recall that under the MFCQ, every local solution $\bar{x}$ of (1) satisfies the first-order necessary optimality conditions

$$\nabla f(\bar{x}) + J(\bar{x})^T \bar{y} = 0, \tag{4a}$$
$$c(\bar{x}) = 0, \tag{4b}$$

where $\bar{y} \in \mathbb{R}^m$ is the unique vector of Lagrange multipliers associated with $\bar{x}$.

We say that $\bar{x} \in \mathbb{R}^n$ is a strict local minimizer of (1) if $c(\bar{x}) = 0$ and there exists an open set $\mathcal{V}$ with $\bar{x} \in \mathcal{V}$ such that $f(\bar{x}) < f(x)$ for all $x \in \mathcal{V} \setminus \{\bar{x}\}$ satisfying $c(x) = 0$. Exactness of the ℓ_2-norm penalty function means the following.

Theorem 1 (35, Theorem 4.4). *Let $\bar{x}$ be a strict local minimizer of (1) that satisfies the MFCQ. Then, there exists a finite threshold $\bar{\tau} > 0$ such that $\bar{x}$ is a local minimizer of (2) for all $\bar{\tau} \leq \tau < \infty$. Moreover, $\bar{\tau}$ satisfies $\bar{\tau} \geq \|\bar{y}\|_2$, where $\bar{y}$ is the unique vector of Lagrange multipliers associated with $\bar{x}$.*

Theorem 1 motivates the use of the ℓ_2-norm penalty in (2). Contrary to inexact penalty functions, such as the squared ℓ_2-norm, we do not need to drive $\tau \to \infty$ to establish convergence to a local solution of (1).

We now consider the first-order necessary optimality conditions for (2).

Proposition 1. *Let $\bar{x}$ be a local solution of (2) for some $\tau > 0$. Then, there exists $\bar{y} \in \mathbb{R}^m$ such that*

$$\nabla f(\bar{x}) + J(\bar{x})^T \bar{y} = 0. \tag{5}$$

Moreover, there is $\bar{\alpha} \geq 0$ such that

$$c(\bar{x}) - \bar{\alpha}\bar{y} = 0, \quad \bar{\alpha}(\|\bar{y}\|_2^2 - \tau^2) = 0, \quad \|\bar{y}\|_2 \leq \tau. \tag{6}$$

Proof. We rewrite (2) as

$$\min_x f(x) + \tau\|c(x)\|_2 = \min_x \max_{\|y\|_2 \leq \tau} \; f(x) + y^T c(x). \tag{7}$$

Convexity of the maximization problem in (7) implies that its necessary and sufficient optimality conditions are (6). Then, the necessary first-order optimality conditions for the minimization problem in (7) are (5). □

3 Algorithm description

3.1 The exact penalty method

Our algorithm iteratively computes an approximate solution of (2) for a fixed penalty parameter τ_k, then either increases τ_k or improves the accuracy of the current solution. We call this process the "outer loop" of our algorithm and use the index k to denote the iteration counter.

At each outer iteration k, we compute primal and dual variables (x_k, y_k) that approximately satisfy (5). When the pair (x_k, y_k) additionally (approximately) satisfies the first-order necessary optimality conditions of (1), we declare first-order stationarity and terminate the algorithm. In view of (4), our algorithm stops when (x_k, y_k) satisfies

$$\|\nabla f(x_k) + J(x_k)^T y_k\|_\infty \leq \epsilon_D \quad \text{and} \quad \|c(x_k)\|_\infty \leq \epsilon_P, \tag{8}$$

where $\epsilon_D > 0$ and $\epsilon_P > 0$ are user-defined tolerances. Using the infinity norm in (8) is common for nonlinear optimization solvers [27, 59]. To monitor the accuracy of the approximate solution (x_k, y_k) of (2) we use a sequence of primal and dual tolerances $\epsilon_P^k > 0$ and $\epsilon_D^k > 0$. In view of (5), we impose that the pair (x_k, y_k) satisfy

$$\|\nabla f(x_k) + J(x_k)^T y_k\|_\infty \leq \epsilon_D^k. \tag{9}$$

If $\|c(x_k)\|_\infty > \epsilon_P^k$, we increase τ_k according to

$$\tau_{k+1} = \max(\tau_k + \tau_{\min}, \|y_k\|_1), \tag{10}$$

where $\tau_{\min} > 0$ is a minimum increase for the penalty parameter. We choose the 1-norm in (10) to overestimate $\|\bar{y}\|_2$, using the fact that $\|\cdot\|_2 \leq \|\cdot\|_1$.

If instead $\|c(x_k)\|_\infty \leq \epsilon_P^k$, then, since (x_k, y_k) also satisfy (9), (x_k, y_k) satisfy (8) for the tolerances $(\epsilon_D^k, \epsilon_P^k)$. Therefore, we tighten the accuracy on (x_k, y_k) by updating

$$\epsilon_D^{k+1} = \max(\mu_\epsilon \epsilon_D^k, \epsilon_D) \quad \text{and} \quad \epsilon_P^{k+1} = \max(\mu_\epsilon \epsilon_P^k, \epsilon_P), \tag{11}$$

where $\mu_\epsilon \in (0,1)$.

We reset the accuracy for the subproblem, $\epsilon_D^{k+1} = \epsilon_D^1$, each time we increase τ_k, because x_k may not be an accurate estimate of the solution of the new penalized problem (2). In addition, we improve the initial guess for the next penalized problem (2) by applying an extrapolation technique to (x_k, y_k). Our approach is similar to extrapolation techniques used in interior-point methods [31].

Beyond approximately solving (2), the "outer loop" iterations also detect the local infeasibility of (1) at x_k by computing an optimality measure for

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \ \|c(x)\|_2. \tag{12}$$

When we detect a local solution of (12) with a strictly positive objective value, we declare (1) to be *locally* infeasible.

Algorithm 1 summarizes the "outer loop" of our exact penalty method.

Algorithm 1 Outer loop.

Require: Starting points (x_0, y_0); initial penalty parameter $\tau_{\text{init}} > 0$; constants $\epsilon_D^1 \geq \epsilon_D > 0$; $\epsilon_P^1 \geq \epsilon_P > 0$; $\tau_{\min} > 0$; $\mu_\epsilon \in (0,1)$.

Check (8) for (x_0, y_0). If it is satisfied, **stop** and declare **first-order stationarity**.
Set $\tau_1 \coloneqq \max(\tau_{\text{init}}, \|y_0\|_1)$.
for $k = 1, \ldots$ **do**
  Compute (x_k, y_k), an approximate solution of (2) that satisfies (9), started from a well-chosen initial guess.
  if $\|c(x_k)\|_\infty > \epsilon_P^k$ **then** *Increase penalty parameter.*
    Update τ_{k+1} according to (10).
    Update (x_k, y_k) with the extrapolation technique.
    Reset $\epsilon_D^{k+1} = \epsilon_D^1$ and set $\epsilon_P^{k+1} = \epsilon_P^k$.
  else *Tighten tolerance.*
    Set $\tau_{k+1} \coloneqq \tau_k$.
    Update ϵ_D^{k+1} and ϵ_P^{k+1} with (11).
  end if
  Check (8) for (x_k, y_k). If satisfied, **stop** and declare **first-order stationarity**.
  Estimate local infeasibility.
end for

3.2 Extrapolation technique

Let (x_k, y_k) be an approximate solution of (2) for a given penalty parameter τ_k. In Algorithm 1, when increasing the penalty parameter to τ_{k+1}, we use an extrapolation technique to improve the initial guess for the next penalized problem (2). Indeed, (x_k, y_k) may not be an accurate estimate of the solution of (2) for $\tau \coloneqq \tau_{k+1}$. To do so, we apply the implicit function theorem to the first-order optimality conditions of (2) derived in Proposition 1, whose equality conditions are smooth even though (2) is nonsmooth.

We collect the equality conditions in (5) and (6) into the mapping

$$F(x, y, \alpha, \tau) \coloneqq \begin{pmatrix} \nabla f(x) + J(x)^T y \\ c(x) - \alpha y \\ \alpha(\|y\|_2^2 - \tau^2) \end{pmatrix}.$$

Thus, if $z^* \coloneqq (x^*, y^*, \alpha^*)$ satisfies the first-order optimality conditions of (2) at τ^*, then $F(z^*, \tau^*) = 0$. Assume that the Jacobian of F with respect to (x, y, α) is nonsingular at (z^*, τ^*). Then, the implicit function theorem implies existence of a $\mathcal{C}^1$ path $\tau \to z_\tau \coloneqq (x_\tau, y_\tau, \alpha_\tau)$, defined in a neighborhood of τ^*, such that $z_{\tau^*} = z^*$, and such that the derivative of the path $\dot{z}_{\tau^*}$ satisfies

$$D_z F(z^*, \tau^*)\dot{z}_{\tau^*} = -D_\tau F(z^*, \tau^*). \tag{13}$$

For completeness, we expand the Jacobians in (13) as

$$D_z F(z,\tau) = \begin{pmatrix} \nabla^2_{xx}\mathcal{L}(x,y) & J(x)^T & 0 \\ J(x) & -\alpha I & -y \\ 0 & 2\alpha y^T & \|y\|_2^2 - \tau^2 \end{pmatrix}, \quad \text{and} \quad D_\tau F(z,\tau) = \begin{pmatrix} 0 \\ 0 \\ -2\alpha\tau \end{pmatrix}.$$

At Line 6 of Algorithm 1, given a new penalty parameter τ_{k+1}, we compute

$$x_{k,\text{extr}} \coloneqq x_k + (\tau_{k+1} - \tau_k)\dot{x}_k, \quad \text{and} \quad y_{k,\text{extr}} \coloneqq y_k + (\tau_{k+1} - \tau_k)\dot{y}_k,$$

where $\dot{z}_k = (\dot{x}_k, \dot{y}_k, \dot{\alpha}_k)$ is the solution of (13) for $\tau^* = \tau_k$ and $z^* = (x_k, y_k, \alpha_k)$, and α_k is defined at the end of Section 5. We start the "inner loop," described in Section 3.4, with $(x_{k,\text{extr}}, y_{k,\text{extr}})$.

3.3 Infeasibility detection

To detect local infeasibility of (1), we use an optimality measure for (12) adapted from Diouane et al. [19]. Specifically, we use

$$\sqrt{\nu_k^{-1}\theta(x_k)} \coloneqq \sqrt{\nu_k^{-1}}\sqrt{\|c(x_k)\|_2 - \|c(x_k) + J(x_k)s_{k,\text{feas}}\|_2}, \tag{14}$$

where

$$s_{k,\text{feas}} \in \operatorname*{argmin}_{s\in\mathbb{R}^n} \|c(x_k) + J(x_k)s\|_2 + \tfrac{1}{2\nu_k}\|s\|_2^2, \tag{15}$$

and $\nu_k > 0$ is a regularization parameter.

Proposition 2. *For any $\nu_k > 0$, if $\sqrt{\nu_k^{-1}\theta(x_k)} = 0$, then x_k is first-order stationary for (12).*

Proof. This is a consequence of the discussion following equation (4.8) in [19, Section 4]. □

To compute a solution of (15), we use the following result.

Proposition 3. *Assume that $c(x_k) \neq 0$. The unique solution $s_{k,\text{feas}}$ to (15) satisfies*

$$\begin{pmatrix} I & J(x_k)^T \\ J(x_k) & -I \end{pmatrix} \begin{pmatrix} s_{k,\text{feas}} \\ y_{k,\text{feas}} \end{pmatrix} = -\begin{pmatrix} 0 \\ c(x_k) \end{pmatrix}. \tag{16}$$

The regularization parameter $\nu_k > 0$ of (15) corresponding to $s_{k,\text{feas}}$ is $\nu_k \coloneqq \|y_{k,\text{feas}}\|_2$.

Proof. Using the same proof technique as in Proposition 1 with $\tau = 1$, and since (15) is strongly convex, its unique solution is characterized by the necessary and sufficient optimality conditions

$$\nu_k^{-1}s + J(x_k)^T y = 0, \quad c(x_k) + J(x_k)s - \alpha y = 0, \quad \alpha(\|y\|_2^2 - 1) = 0, \quad \|y\|_2 \leq 1, \quad \alpha \geq 0,$$

which are satisfied for $(s, y, \alpha) = (s_{k,\text{feas}}, \nu_k^{-1}y_{k,\text{feas}}, \nu_k)$, with $\nu_k = \|y_{k,\text{feas}}\|_2$. To see this, the first row of (16) yields

$$\nu_k^{-1}s + J(x_k)^T y = \nu_k^{-1}s_{k,\text{feas}} + \nu_k^{-1}J(x_k)^T y_{k,\text{feas}} = 0.$$

Moreover, the second row of (16) gives

$$c(x_k) + J(x_k)s_{k,\text{feas}} - \nu_k\left(\nu_k^{-1}y_{k,\text{feas}}\right) = c(x_k) + J(x_k)s_{k,\text{feas}} - y_{k,\text{feas}} = 0.$$

Finally, since $\nu_k = \|y_{k,\text{feas}}\|_2$,

$$\|\nu_k^{-1}y_{k,\text{feas}}\|_2 = \nu_k^{-1}\|y_{k,\text{feas}}\|_2 = \nu_k^{-1}\nu_k = 1 \leq 1,$$

so that $\alpha(\|y\|_2^2 - 1) = \nu_k(1-1) = 0$ and $\alpha = \nu_k > 0$, which completes the verification that $(s, y, \alpha) = (s_{k,\text{feas}}, \nu_k^{-1}y_{k,\text{feas}}, \nu_k)$ satisfies the optimality conditions of (15).

Note that $\nu_k = \|y_{k,\text{feas}}\|_2 > 0$ because $y_{k,\text{feas}} = 0$ implies from the first row of (16) that $s_{k,\text{feas}} = 0$, which then gives $y_{k,\text{feas}} = c(x_k) \neq 0$, a contradiction. □

In contrast to Diouane et al. [19], where $\nu_k = 1$ is fixed before solving (15) with [19, Corollary 5.3], ν_k is here determined by the solution of (16).

Following Proposition 3, we compute (14) by solving (16) for $(s_{k,\text{feas}}, y_{k,\text{feas}})$ and setting $\nu_k = \|y_{k,\text{feas}}\|_2$. We wish to declare local infeasibility when $\sqrt{\nu_k^{-1}\theta(x_k)}$ is small. Specifically, we impose that $\sqrt{\nu_k^{-1}\theta(x_k)} < \epsilon_I\|c(x_k)\|_2$, where $\epsilon_I > 0$ is an algorithmic constant, to declare local infeasibility. To guarantee that x_k is close to a stationary point of (12), we additionally require $\sqrt{\nu_k^{-1}\theta(x_k)}$ to be smaller than the current primal tolerance ϵ_P^k. In summary, we declare the problem (1) to be locally infeasible when

$$\sqrt{\nu_k^{-1}\theta(x_k)} < \epsilon_I\|c(x_k)\|_2, \quad \|c(x_k)\|_2 > \epsilon_P \quad \text{and} \quad \sqrt{\nu_k^{-1}\theta(x_k)} < \epsilon_P^k,$$

where ϵ_P, ϵ_P^k are the tolerances described in Section 3.1.

3.4 Solution of the penalized problem

To solve (2) for fixed τ_k, we use an iterative algorithm inspired by [20, Algorithm 4.1]. We refer to this process as the "inner loop" of our algorithm. Let l be the iteration counter of the "inner loop." The method computes a step s_l by approximately solving a regularized quadratic model of (2) around the current iterate x_l which is,

$$\underset{s\in\mathbb{R}^n}{\text{minimize}} \; q(s; x_l, \sigma_l) \coloneqq f_l + \nabla f_l{}^T s + \tfrac{1}{2}s^T(H_l + \sigma_l I)s + \tau_k\|c_l + J_l s\|_2, \tag{17}$$

where $\sigma_l > 0$ is a quadratic regularization parameter and H_l is a symmetric matrix.

We accept a step that satisfies

$$\rho_l \coloneqq \frac{f_l + \tau_k\|c_l\|_2 - (f(x_l + s_l) + \tau_k\|c(x_l + s_l)\|_2)}{f_l + \tau_k\|c_l\|_2 - (f_l + \nabla f_l{}^T s_l + \tau_k\|c_l + J_l s_l\|_2)} \geq \eta_1, \tag{18}$$

for some constant $\eta_1 \in (0, 1)$, in which case we set $x_{l+1} \coloneqq x_l + s_l$. If $\rho_l < \eta_1$, then $x_{l+1} \coloneqq x_l$ and the step is rejected. When we reject a step, we increase the regularization parameter $\sigma_{l+1} > \sigma_l$. When $\rho_l \geq \eta_2 \in [\eta_1, 1)$, we decrease $\sigma_{l+1} \leq \sigma_l$.

As we shall see in Section 4, under a Lipschitz assumption on ∇f and J, steps are guaranteed to be accepted when σ_l is sufficiently large. This fact ensures global convergence of the method. On the other hand, though not proven, we observe fast local convergence of our method close to solutions of (2) when $\sigma_l \approx 0$. For this reason, the parameter σ_l plays a crucial role in the performance of the method in practice. The basic update rule that guarantees the convergence is given in Algorithm 2 and accelerating heuristics are provided in Section 6.2.

We assume that the computation of the step s_l yields an estimate y_{l+1} of the Lagrange multipliers, as formalized below.

Step Assumption 3.1. For each $l \in \mathbb{N}$, the step s_l satisfies

$$(H_l + \sigma_l I)s_l = -\nabla f_l - J_l{}^T y_{l+1}, \tag{19}$$

for some Lagrange multiplier estimate $y_{l+1} \in \mathbb{R}^m$.

For the method to converge, we make the following additional assumption on the step s_l,

Step Assumption 3.2. For each $l \in \mathbb{N}$, the step s_l satisfies the simple decrease condition

$$q(s_l; x_l, \sigma_l) \leq q(0; x_l, \sigma_l), \tag{20}$$

and, for some constant $\eta_C > 0$ independent of l, the Cauchy decrease condition

$$f_l + \tau_k\|c_l\|_2 - f_l - \nabla f_l{}^T s_l - \tau_k\|c_l + J_l s_l\|_2 \geq \tfrac{1}{2}\eta_C \min(1, \|\nabla f_l + J_l^T y_{l+1}\|_\infty^2). \tag{21}$$

As a consequence of Step Assumption 3.1, our solver produces iterates (x_l, y_l), which we use to define H_l in (17) as either the Hessian of the Lagrangian $\nabla^2 \mathcal{L}(x_l, y_l)$ or a quasi-Newton approximation thereof. When we present our step computation procedure in Section 5, we will show that our implementation satisfies Step Assumptions 3.1 and 3.2. Step Assumption 3.2 stands as a significant improvement over the penalty method in [19], which assumed that the step s_l is an exact solution of (17).

Algorithm 2 summarizes the "inner loop" of our exact penalty method.

Algorithm 2 Inner loop.

Require: Starting points (x_0, y_0); initial regularization parameter $\sigma_0 > 0$; penalty parameter $\tau_k > 0$; constants $\epsilon_D^k > 0$; $\eta_1 \in (0,1)$; $\eta_2 \in [\eta_1, 1)$; $\gamma_1 > 1$; $\gamma_2 \geq \gamma_1$; $\gamma_3 \in (0,1]$;

1: **for** $l = 0, 1, \ldots$ **do**

2: Choose a symmetric matrix $H_l \approx \nabla^2 \mathcal{L}(x_l, y_l)$.

3: Compute a step s_l and an estimate of the Lagrange multipliers y_{l+1} that satisfy Step Assumptions 3.1 and 3.2.

4: Check (9) for (x_l, y_{l+1}). If satisfied, **stop** and return (x_l, y_{l+1}).

5: Compute the ratio

$$\rho_l := \frac{f_l + \tau_k \|c_l\|_2 - (f(x_l + s_l) + \tau_k \|c(x_l + s_l)\|_2)}{f_l + \tau_k \|c_l\|_2 - (f_l + \nabla f_l{}^T s_l + \tau_k \|c_l + J_l s_l\|_2)}.$$

6: If $\rho_l \geq \eta_1$, set $x_{l+1} := x_l + s_l$. Otherwise, set $x_{l+1} := x_l$.

7: Update the regularization parameter according to

$$\sigma_{l+1} \in \begin{cases} [\gamma_3 \sigma_l, \sigma_l] & \text{if } \rho_l \geq \eta_2, \\ \sigma_l & \text{if } \eta_1 \leq \rho_l < \eta_2, \\ [\gamma_1 \sigma_l, \gamma_2 \sigma_l] & \text{if } \rho_l < \eta_1. \end{cases}$$

8: **end for**

4 Convergence analysis

The aim of this section is to show that Algorithm 1 produces iterates (x_k, y_k) that eventually satisfy (8). Our approach can be summarized as follows. First, we show that Algorithm 2 brings

$$\|\nabla f(x_l) + J(x_l)^T y_{l+1}\|_\infty, \tag{22}$$

below a given tolerance $\epsilon > 0$ in $\mathcal{O}(\epsilon^{-2})$ iterations. Then, we show convergence of Algorithm 1.

4.1 Convergence of the inner loop

We now prove a complexity result for Algorithm 2 with respect to (22). Our working assumptions are the following.

Model Assumption 4.1. There exists $\kappa_m > 0$ such that for all $l \in \mathbb{N}$ and $k \in \mathbb{N}$,

$$|f(x_l + s_l) + \tau_k \|c(x_l + s_l)\|_2 - (f_l + \nabla f_l{}^T s_l + \tau_k \|c_l + J_l s_l\|_2)| \leq \kappa_m \|s_l\|_2^2.$$

Model Assumption 4.2. There exists $\kappa_H > 0$ such that for all $l \in \mathbb{N}$ and $k \in \mathbb{N}$,

$$\|H_l\|_2 \leq \kappa_H.$$

Following the discussion in [19, Section 4], Model Assumption 4.1 is satisfied when ∇f and J are Lipschitz continuous and the penalty parameter τ_k is bounded.

The following lemma is critical in showing global convergence of the method.

Lemma 1. *Let condition* (20) *of Step Assumption 3.2 and Model Assumptions 4.1 and 4.2 be satisfied. Define*

$$\sigma_{\text{succ}} := \kappa_H + \tfrac{2\kappa_m}{(1-\eta_2)}. \tag{23}$$

If, at iteration l of Algorithm 2, $s_l \neq 0$ and $\sigma_l \geq \sigma_{\text{succ}}$, then $\rho_l \geq \eta_2$.

Proof. From (23), $\sigma_l \geq \sigma_{\text{succ}} > \kappa_H$, so that Model Assumption 4.2 implies that $H_l + \sigma_l I$ is positive definite. Additionally, since s_l satisfies (20),

$$\tau_k \|c_l\|_2 - \nabla f_l{}^T s_l - \tau_k \|c_l + J_l s_l\|_2 \geq \tfrac{1}{2} s_l^T (H_l + \sigma_l I) s_l. \tag{24}$$

Then, the definition of ρ_l in Algorithm 2, (24), and Model Assumption 4.1 imply that

$$\begin{aligned} |\rho_l - 1| &= \left| \frac{f(x_l + s_l) + \tau_k \|c(x_l + s_l)\|_2 - f_l - \nabla f_l{}^T s_l - \tau_k \|c_l + J_l s_l\|_2}{\tau_k \|c_l\|_2 - \nabla f_l{}^T s_l - \tau_k \|c_l + J_l s_l\|_2} \right| \\ &\leq \frac{2\kappa_m \|s_l\|_2^2}{s_l{}^T (H_l + \sigma_l I) s_l} \\ &\leq \frac{2\kappa_m}{\lambda_{\min}(H_l + \sigma_l I)}. \end{aligned}$$

Now, from (23),

$$\lambda_{\min}(H_l + \sigma_l I) = \lambda_{\min}(H_l) + \sigma_l \geq \sigma_l - \kappa_H \geq \frac{2\kappa_m}{1 - \eta_2},$$

which gives $|\rho_l - 1| \leq 1 - \eta_2$. □

The mechanism of Algorithm 2 and Lemma 1 imply that

$$\sigma_l \leq \sigma_{\max} \coloneqq \max\{\sigma_0, \gamma_2 \sigma_{\text{succ}}\}, \quad \forall l \in \mathbb{N}. \tag{25}$$

As a consequence of (25), we establish the following connection between the two conditions of Step Assumption 3.2.

Lemma 2. *Let Step Assumption 3.1 and condition (20) of Step Assumption 3.2 be satisfied and Model Assumptions 4.1 and 4.2 be satisfied. Then, for each $l \in \mathbb{N}$, if $H_l + \sigma_l I$ is positive definite, s_l satisfies (21) with $\eta_C = (\kappa_H + \sigma_{\max})^{-1}$.*

Proof. From (20), s_l satisfies (24). Moreover, under (20) and Model Assumptions 4.1 and 4.2, Lemma 1 implies (25). Then, from $H_l + \sigma_l I \succ 0$, Model Assumption 4.2 and (25),

$$\begin{aligned} \tau_k \|c_l\|_2 - \nabla f_l{}^T s_l - \tau_k \|c_l + J_l s_l\|_2 &\geq \tfrac{1}{2} s_l^T (H_l + \sigma_l I) s_l \\ &= \tfrac{1}{2} s_l^T (H_l + \sigma_l I)(H_l + \sigma_l I)^{-1} (H_l + \sigma_l I) s_l \\ &\geq \tfrac{1}{2} \lambda_{\min}((H_l + \sigma_l I)^{-1}) \|(H_l + \sigma_l I) s_l\|_2^2 \\ &\geq \tfrac{1}{2} (\kappa_H + \sigma_{\max})^{-1} \|(H_l + \sigma_l I) s_l\|_2^2. \end{aligned}$$

The conclusion follows by injecting (19) into the last inequality and using the fact that $\|\cdot\|_2 \geq \|\cdot\|_\infty$. □

Moving on to the complexity analysis, given a user-defined tolerance $\epsilon \leq 1$, we seek an upper bound on $l_\epsilon \coloneqq \min\{l \in \mathbb{N} \mid \|\nabla f_l + J_l^T y_{l+1}\|_\infty \leq \epsilon\} = |\mathcal{S}_k(\epsilon)| + |\mathcal{U}_k(\epsilon)| + 1$, where

$$\mathcal{S}_k(\epsilon) \coloneqq \{l \in \mathbb{N} \mid \rho_l \geq \eta_1 \text{ and } l < l_\epsilon\}, \quad \mathcal{U}_k(\epsilon) \coloneqq \{l \in \mathbb{N} \mid \rho_l < \eta_1 \text{ and } l < l_\epsilon\}.$$

We use subscripts k to emphasize that the complexity analysis is for the inner loop of outer loop iteration k of Algorithm 1.

Lemma 3. *Let condition (21) of Step Assumption 3.2, Step Assumption 3.1, Model Assumptions 4.1 and 4.2 be satisfied. Assume that $x \to f(x) + \tau_k \|c(x)\|_2$ is bounded below by $(f + \tau_k \|c\|_2)_{\text{low}}$. Then,*

$$|\mathcal{S}_k(\epsilon)| \leq \frac{f_0 + \tau_k \|c_0\|_2 - (f + \tau_k \|c\|_2)_{\text{low}}}{\frac{1}{2} \eta_1 \eta_C} \epsilon^{-2}. \tag{26}$$

Proof. Let $l \in \mathcal{S}_k(\epsilon)$. From (18) and (21),

$$\begin{aligned} f_l + \tau_k\|c_l\|_2 - f(x_l + s_l) - \tau_k\|c(x_l + s_l)\|_2 &\geq \eta_1(\tau_k\|c_l\|_2 - \nabla f_l{}^T s_l - \tau_k\|c_l + J_l s_l\|_2) \\ &\geq \tfrac{1}{2}\eta_1\eta_C \min(1, \|\nabla f_l + J_l^T y_{l+1}\|_\infty^2). \end{aligned}$$

Because $1 \geq \epsilon^2$ by definition and $\|\nabla f_l + J_l^T y_{l+1}\|_\infty^2 > \epsilon^2$ following that $l < l_\epsilon$, the above simplifies to

$$f_l + \tau_k\|c_l\|_2 - f(x_l + s_l) - \tau_k\|c(x_l + s_l)\|_2 \geq \tfrac{1}{2}\eta_1\eta_C\epsilon^2.$$

The remainder of the proof is classic and identical to, e.g., [4, Lemma 3.6]. □

Lemma 4. *Under the assumptions of Lemmas 1 and 3,*

$$|\mathcal{U}_k(\epsilon)| \leq \log_{\gamma_1}(\sigma_{\max}/\sigma_0) + |\mathcal{S}_k(\epsilon)||\log_{\gamma_1}(\gamma_3)|. \tag{27}$$

Proof. The proof is similar to [4, Lemma 3.7] and is omitted. □

Theorem 2. *Let Step Assumption 3.2, Step Assumption 3.1, and Model Assumptions 4.1 and 4.2 hold and assume that $x \to f(x) + \tau_k\|c(x)\|_2$ is bounded below by $(f + \tau_k\|c\|_2)_{\text{low}}$. Then,*

$$l_\epsilon \leq \left(1 + |\log_{\gamma_1}(\gamma_3)|\right) \frac{f_0 + \tau_k\|c_0\|_2 - (f + \tau_k\|c\|_2)_{\text{low}}}{\frac{1}{2}\eta_1\eta_C}\epsilon^{-2} + \log_{\gamma_1}(\sigma_{\max}/\sigma_0) + 1. \tag{28}$$

In particular, $l_\epsilon = \mathcal{O}(\epsilon^{-2})$.

Proof. By definition, $l_\epsilon = |\mathcal{S}_k(\epsilon)| + |\mathcal{U}_k(\epsilon)| + 1$. The result follows by bounding $|\mathcal{U}_k(\epsilon)|$ with (27) and then $|\mathcal{S}_k(\epsilon)|$ with (26). □

4.2 Convergence of the outer loop

Theorem 2 allows us to establish that the iterates (x_k, y_k) generated by Algorithm 1 are well defined and satisfy (9).

Corollary 1. *Let Step Assumptions 3.1 and 3.2 and Model Assumptions 4.1 and 4.2 hold and assume that $x \to f(x) + \tau_1\|c(x)\|_2$ is bounded below by $(f + \tau_1\|c\|_2)_{\text{low}}$. Then, for each $k \in \mathbb{N}$, the iterates (x_k, y_k) of Algorithm 1 generated by Algorithm 2 are well defined and satisfy (9).*

Proof. If $x \to f(x) + \tau_1\|c(x)\|_2$ is bounded below, then $x \to f(x) + \tau_k\|c(x)\|_2$ is also bounded below for all $k \in \mathbb{N}$ since $\tau_k \geq \tau_1$. Therefore, because Step Assumptions 3.1 and 3.2 and Model Assumptions 4.1 and 4.2 hold, Theorem 2 applies for each $k \in \mathbb{N}$, and Algorithm 2 brings (22) below ϵ_D^k in $\mathcal{O}((\epsilon_D^k)^{-2})$ iterations. Then, since Step Assumption 3.1 holds, the pair (x_k, y_k) satisfies (9). □

The following lemma proves the convergence of Algorithm 1 to first-order stationary points of (1). The approach here is similar to [19, Theorem 4.5] with the exception that we assume that the penalty parameter is bounded. Proving the convergence in the unbounded case is beyond the scope of this paper. For simplicity, we assume that $\epsilon_P = \epsilon_D = \epsilon$. We seek an upper bound on $k_\epsilon := \min\{k \in \mathbb{N} \mid \|\nabla f_k + J_k^T y_k\|_\infty \leq \epsilon \text{ and } \|c_k\|_\infty \leq \epsilon\}$.

Lemma 5. *Let the assumptions of Corollary 1 hold and assume that there is a $\bar{\tau}$, independent of ϵ such that $\|c(x_k)\|_\infty \leq \epsilon$ whenever $\tau_k \geq \bar{\tau}$. Then,*

$$k_\epsilon \leq \left\lceil \frac{\bar{\tau} - \tau_{\text{init}}}{\tau_{\min}} \right\rceil + \left\lceil \log_{\mu_\epsilon}(\epsilon/\epsilon_P^1) \right\rceil + \left(\left\lceil \frac{\bar{\tau} - \tau_{\text{init}}}{\tau_{\min}} \right\rceil + 1 \right) \left\lceil \log_{\mu_\epsilon}(\epsilon/\epsilon_D^1) \right\rceil + 1.$$

Proof. Corollary 1 ensures that the iterates (x_k, y_k) are well defined and satisfy (9). From Algorithm 1 and (11), ϵ_P^k drops below ϵ in at most $\lceil \log(\epsilon/\epsilon_P^1)/\log(\mu_\epsilon)\rceil$ iterations. This implies that τ_k remains constant for at most

$$\lceil \log(\epsilon/\epsilon_P^1)/\log(\mu_\epsilon)\rceil + \lceil \log(\epsilon/\epsilon_D^1)/\log(\mu_\epsilon)\rceil$$

outer loop iterations. Otherwise, we would have both ϵ_P^k and ϵ_D^k below ϵ which implies (8) from (9) and $\|c(x_k)\|_\infty \leq \epsilon_P^k$ so that Algorithm 1 stops.

Therefore, since ϵ_P^k is never reset whereas ϵ_D^k is reset each time τ_k increases, $\tau_k \geq \bar{\tau}$ in at most

$$\left\lceil \frac{\bar{\tau} - \tau_{\text{init}}}{\tau_{\min}} \right\rceil + \lceil \log(\epsilon/\epsilon_P^1)/\log(\mu_\epsilon)\rceil + \left\lceil \frac{\bar{\tau} - \tau_{\text{init}}}{\tau_{\min}} \right\rceil \lceil \log(\epsilon/\epsilon_D^1)/\log(\mu_\epsilon)\rceil$$

outer loops. Once τ_k attains this threshold, $\|c(x_k)\|_\infty \leq \epsilon$. Therefore, besides the iterations that drive ϵ_P^k below ϵ, which are already counted, there are at most

$$\lceil \log(\epsilon/\epsilon_D^1)/\log(\mu_\epsilon)\rceil$$

additional iterations before $\epsilon_D^k \leq \epsilon$. Once this happens, (8) follows from (9) at the next outer iteration. □

In Lemma 5, we have shown that $k_\epsilon = \mathcal{O}(|\log(\epsilon)|)$. The assumption that such a $\bar{\tau}$ exists in Lemma 5 is motivated by the exactness of the penalty function stated in Theorem 1. In general however, we should only expect Algorithm 1 to reach an approximate local solution of (12) rather than an approximate feasible solution when $\tau_k \geq \bar{\tau}$, which is the assumption used in [12, 19]. When this approximate local solution is infeasible for (1), the infeasibility detection mechanism of Section 3.3 would trigger.

Combining the results of Sections 4.1 and 4.2, we can compute a bound on the number of inner and outer loop iterations of Algorithm 1. Each inner loop iteration is called with a tolerance ϵ_D^k, therefore, this number is

$$\sum_{k=1}^{k_\epsilon} |\mathcal{S}_k(\epsilon_D^k)| + |\mathcal{U}_k(\epsilon_D^k)| + 1.$$

Theorem 3. *Let the assumptions of Lemma 5 hold. For each $k \geq 1$, let $x_{k,0}$ denote the starting point of Algorithm 2 at outer iteration k of Algorithm 1, and assume that $\{x_{k,0}\}$ is bounded. Then,*

$$\sum_{k=1}^{k_\epsilon} |\mathcal{S}_k(\epsilon_D^k)| + |\mathcal{U}_k(\epsilon_D^k)| + 1 = \mathcal{O}(|\log(\epsilon)|\epsilon^{-2}). \tag{29}$$

Proof. Since f and c are continuous and the set $\{x_{k,0}\}$ is bounded, $(f + \bar{\tau}\|c\|_2)_{\text{up}} := \sup_{k\geq 1} f(x_{k,0}) + \bar{\tau}\|c(x_{k,0})\|_2$ is finite. Since $\tau_k \in [\tau_1, \bar{\tau}]$, $\epsilon_D^k \geq \epsilon$, (26) implies

$$|\mathcal{S}_k(\epsilon_D^k)| \leq \frac{(f + \bar{\tau}\|c\|_2)_{\text{up}} - (f + \tau_1\|c\|_2)_{\text{low}}}{\frac{1}{2}\eta_1\eta_C}\epsilon^{-2}. \tag{30}$$

The result follows by combining (30) with (27) and Lemma 5 and observing that the right-hand side of (30) is independent of k. □

5 Step computation

Now, we present our approach for solving (17). For future reference, we define the implicit functions $s_l(\alpha)$ and $y_{l+1}(\alpha)$ with

$$\begin{pmatrix} H_l + \sigma_l I & J_l^T \\ J_l & -\alpha I \end{pmatrix} \begin{pmatrix} s_l(\alpha) \\ y_{l+1}(\alpha) \end{pmatrix} = -\begin{pmatrix} \nabla f_l \\ c_l \end{pmatrix}, \quad \text{for } \alpha \geq 0. \tag{31}$$

When (31) has multiple solutions, which may happen for $\alpha = 0$ when J_l is rank deficient or for $\alpha \geq 0$ when $H_l + \sigma_l I$ is singular, $y_{l+1}(\alpha)$ denotes the least-norm y-component among them. If $H_l + \sigma_l I \succ 0$, all solutions share the same $s_l(\alpha)$.

We separate our step computation procedure into two cases. We first assume that $H_l + \sigma_l I$ is positive definite and J_l has full row rank. We call this the *definite case*. Then, we describe the general case with both assumptions lifted.

5.1 Definite case

The following result characterizes the solution of (17). It is an adaptation of [19, Corollary 5.4], which works on the Schur complement of the linear system in (31), whereas we work on the full system. Moreover, [19, Corollary 5.4] uses the notation $\nu_l := 1/\sigma_l$, multiplies the first row of (31) by ν_l, and uses $\tilde{y}_{l+1}(\alpha) := \nu_l y_{l+1}(\alpha)$, which gives the equivalent system

$$\begin{pmatrix} \nu_l H_l + I & J_l^T \\ J_l & -\tilde{\alpha} I \end{pmatrix} \begin{pmatrix} s_l(\tilde{\alpha}) \\ \tilde{y}_{l+1}(\tilde{\alpha}) \end{pmatrix} = - \begin{pmatrix} \nu_l \nabla f_l \\ c_l \end{pmatrix}, \quad \text{for } \tilde{\alpha} \geq 0,$$

where $\tilde{\alpha} := \alpha/\nu_l$.

Proposition 4 (19, Corollary 5.4)**.** *Assume that $H_l + \sigma_l I$ is positive definite. Then the unique solution $s_l^* := s_l(\alpha_l^*)$ of (17) satisfies (31), where $\alpha_l^* \geq 0$ is such that $\alpha_l^*(\|y_{l+1}(\alpha_l^*)\|_2 - \tau_k) = 0$ and $\|y_{l+1}(\alpha_l^*)\|_2 \leq \tau_k$.*

Our approach for computing steps is based on Proposition 4. We first compute $s_l(0)$ and $y_{l+1}(0)$ by solving (31) with $\alpha = 0$. Then, we check whether $\|y_{l+1}(0)\|_2 \leq \tau_k$. If this is not the case, we solve

$$\|y_{l+1}(\alpha)\|_2 = \tau_k, \tag{32}$$

for $\alpha > 0$. To solve the equation, we apply Newton's method to the equivalent *secular* equation $1/\|y_{l+1}(\alpha)\|_2 - 1/\tau_k = 0$, which turns out to be better suited for numerical computations. This approach is motivated by the method of Moré and Sorensen [46], and we refer to [19, Section 5] for more details. For completeness, the following is an adaptation of [18, Lemma 7.3.1] that characterizes the iterates of the Newton method.

Lemma 6. *Let $\bar{\alpha} \geq 0$ be such that the matrix in (31) is nonsingular. Assume that $y_{l+1}(\bar{\alpha}) \neq 0$. Then, the secular function $\xi : \alpha \to 1/\|y_{l+1}(\alpha)\|_2 - 1/\tau_k$ is differentiable at $\bar{\alpha}$. Moreover,*

$$\frac{\xi(\bar{\alpha})}{\xi'(\bar{\alpha})} = -\frac{(1 - \tau_k^{-1}\|y_{l+1}(\bar{\alpha})\|_2)\|y_{l+1}(\bar{\alpha})\|_2^2}{y_{l+1}(\bar{\alpha})^T y'_{l+1}(\bar{\alpha})},$$

where $y'_{l+1}(\bar{\alpha})$ is the solution of (31) with right-hand side $(0, y_{l+1}(\bar{\alpha}))$.

In Lemma 6, the assumption that (31) is nonsingular is satisfied for all $\bar{\alpha}$ when $H_l + \sigma_l I$ is positive definite and J_l has full row rank. The convergence of Newton's method is guaranteed by the following lemma.

Lemma 7 (18, Lemma 7.3.2)**.** *Assume that $H_l + \sigma_l I$ is positive definite. Then, Newton's method applied to the secular equation $\alpha \to 1/\|y_{l+1}(\alpha)\|_2 - 1/\tau_k$ started from any $\alpha \in (-\lambda_{\min}(J_l(H_l + \sigma_l I)^{-1} J_l^T), \alpha_l^*]$ produces a sequence of iterates that monotonically increase towards α_l^*, where α_l^* is the unique non-negative root to the equation.*

Crucially, each iteration of Newton's method applied to the secular equation requires the solution of the symmetric indefinite linear system (31) for different values of α. We use sparse LDL^T factorization algorithms to compute these. In our implementation, the matrix factorizations are computed using the open-source solver MUMPS [1, 2] or the factorization routine MA57 [23] from the Harwell subroutine library. Instead of requiring convergence to the exact solution of $q(\cdot; x_l, \sigma_l)$, we truncate Newton's

method when we find $s_l := s_l(\alpha_l)$ that satisfies (20), which significantly reduces the number of iterations of Newton's method in practice.

In practice, we find that we rarely need to perform an iteration of Newton's method. That is, the choice $\alpha_l = 0$ often satisfies (20) so that we do not need to update it. We summarize our step computation procedure when $H_l + \sigma_l I \succ 0$ and J_l has full row rank in Algorithm 3.

Algorithm 3 Step computation – Definite case.

Require: Symmetric positive definite matrix $H_l + \sigma_l I$; full row rank Jacobian J_l; gradient ∇f_l; constraint values c_l; penalty parameter τ_k.

1: Set $\alpha_l := 0$.

2: Solve

$$\begin{pmatrix} H_l + \sigma_l I & J_l^T \\ J_l & -\alpha_l I \end{pmatrix} \begin{pmatrix} s_l \\ y_{l+1} \end{pmatrix} = - \begin{pmatrix} \nabla f_l \\ c_l \end{pmatrix}, \tag{33}$$

3: **while** $q(s_l; x_l, \sigma_l) > q(0; x_l, \sigma_l)$ **do**

4: Solve

$$\begin{pmatrix} H_l + \sigma_l I & J_l^T \\ J_l & -\alpha_l I \end{pmatrix} \begin{pmatrix} s'_l \\ y'_{l+1} \end{pmatrix} = \begin{pmatrix} 0 \\ y_{l+1} \end{pmatrix}. \tag{34}$$

5: Set *Newton step for the secular equation.*

$$\alpha_l^+ := \alpha_l + \frac{(1 - \tau_k^{-1}\|y_{l+1}\|_2)\|y_{l+1}\|_2^2}{y_{l+1}^T y'_{l+1}}. \tag{35}$$

6: Set $\alpha_l := \alpha_l^+$ and solve (33).

7: **end while**

In the definite case, the step computed by Algorithm 3 satisfies Step Assumptions 3.1 and 3.2. Condition (20) is satisfied by construction, and (21) follows from (20) by Lemma 2, so that Step Assumption 3.2 holds. Moreover, since (s_l, y_{l+1}) solves (33), its first block row is (19), which gives Step Assumption 3.1.

5.2 General case

In general, the positive definiteness of $H_l + \sigma_l I$ and the full-rank condition on J_l may not hold. To detect potential degeneracy, we use the inertia of (33) as an indicator for the properties of both matrices. Specifically, the inertia is the triple (n_+, n_-, n_0), where n_+, n_-, and n_0 denote the numbers of positive, negative, and zero eigenvalues, respectively. When a sparse factorization routine is used to solve (33), the inertia is a by-product of the factorization.

Proposition 5. *Assume that $H_l + \sigma_l I$ is positive definite. Then,*

1. *The inertia of (33) with $\alpha_l = 0$ is $(n, r, m - r)$, where $r = \text{Rank}(J_l)$.*
2. *The inertia of (33) with $\alpha_l > 0$ is $(n, m, 0)$.*

Proof. This is a direct consequence of the Haynsworth inertia additivity formula [36]. □

Our approach for detecting degeneracy in $H_l + \sigma_l I$ and J_l is based on Proposition 5. First, we try to solve (33) with $\alpha_l = 0$ and compute the inertia (n_+, n_-, n_0). If $n_- < m$, then we set α_l to a small positive value $\alpha_{\text{trial}} > 0$ and retry solving (33), which updates (n_+, n_-, n_0). After this first correction step, if $n_+ < n$, we increase σ_l and resolve (33) with the updated value of α_l until $(n_+, n_-, n_0) = (n, m, 0)$.

Following this inertia correction step, two cases may occur: either $H_l + \sigma_l I$ is positive definite or not.

First, if $H_l + \sigma_l I$ is positive definite, the same procedure as Algorithm 3 applies to compute a step satisfying Step Assumptions 3.1 and 3.2, regardless of rank deficiency in J_l, provided that $\alpha_{\text{trial}} > 0$.

Proposition 6. *Assume that $H_l + \sigma_l I$ is positive definite and that J_l is rank deficient. Assume that $s = 0$ is not a solution to (17). Then, there is an $\alpha_{\text{trial}} > 0$ such that Algorithm 3 started with $\alpha_l = \alpha_{\text{trial}}$ computes a step s_l satisfying Step Assumptions 3.1 and 3.2.*

Proof. Step Assumption 3.1 is satisfied by construction, since s_l is the solution of (31) for some $\alpha \geq 0$. Moreover, following Lemma 2, it is sufficient to show Algorithm 3 produces a step s_l satisfying (20).

Since $H_l + \sigma_l I$ is positive definite, Proposition 4 applies unchanged. In Proposition 4, either $\alpha_l^* = 0$ or $\alpha_l^* > 0$.

If $\alpha_l^* = 0$, then the solution of (17) is $s_l^* = s_l(0)$. In particular, since $s_l^* \neq 0$ and s_l^* is the unique solution of (17), by strict convexity of (17),

$$q(s_l(0); x_l, \sigma_l) < q(0; x_l, \sigma_l).$$

Continuity of $\alpha \to q(s(\alpha); x_l, \sigma_l)$ on $[0, \infty)$ then implies that there is an $\alpha_{\text{trial}} > 0$ such that s_l satisfies (20) for all $\alpha_l \in [0, \alpha_{\text{trial}}]$.

If $\alpha_l^* > 0$, then if $\alpha_{\text{trial}} \in (0, \alpha_l^*]$, Lemma 7 implies that the Newton iterates (35) converge to α_l^*. In particular, Algorithm 3 converges to the solution of (17) and produces an s_l satisfying (20). □

The assumption that $s = 0$ is not a solution to (17) in Proposition 6 is not restrictive. Indeed, if the solution of (17) is $s = 0$, following Proposition 4, there is an $\alpha_l^* \geq 0$ such that $(0, y_{l+1}(\alpha_l^*))$ is a solution of (31). The first block row of (31) then implies that $\nabla f_l + J_l^T y_{l+1}(\alpha_l^*) = 0$, which means that x_l is stationary with respect to (4a), in which case, Algorithm 2 terminates and returns x_l.

If after the inertia correction step, $H_l + \sigma_l I$ is not positive definite, a possible approach is to increase σ_l and restart the procedure until it is. However, experience showed that this usually leads to unnecessarily large values in σ_l and slower convergence of the algorithm. Indeed, observe that if $\alpha_l = 0$, then even if $H_l + \sigma_l I$ is indefinite, when the inertia of (31) is $(n, m, 0)$, the solution s_l of (31) is a solution of

$$\min_s f_l + \nabla f_l{}^T s + \tfrac{1}{2} s^T (H_l + \sigma_l I) s \quad \text{s.t.} \quad c_l + J_l s = 0,$$

which often turns out to be a good step for (2) in practice. We found that a better technique is to perform a very limited number $j_{\max}$ of iterations of Newton's method using the update rule of Lemma 6. We stress that neither Proposition 4 nor Lemma 7 apply. Hence, we cannot guarantee that we will satisfy (20). Moreover, Lemma 2 no longer applies, so that (20) does not imply (21). As a consequence, we verify both (20) and (21). If both conditions are satisfied, we return the step; otherwise, we either continue to the next Newton iteration or increase σ_l and restart the procedure. Note that Step Assumption 3.1 is satisfied by construction, since s_l is the solution of (31).

In practice, before committing to the LDL^T factorization of $H_l + \sigma_l I$, we try cheap tests to check whether $H_l + \sigma_l I$ is positive definite. Specifically, we perform the following two tests:

1. We verify whether all diagonal elements of $H_l + \sigma_l I$ are positive. We return that $H_l + \sigma_l I$ is not positive definite if this is not the case.
2. We verify whether $H_l + \sigma_l I$ is strictly diagonally dominant. We return that $H_l + \sigma_l I$ is positive definite if this is the case.

This allows us to save a few LDL^T factorizations in practice.

Note that increasing σ_l in Algorithm 4 does not break the convergence of Algorithm 2. Indeed, the increase of σ_l in Algorithm 4 is proportional to a constant factor $\mu_\sigma > 1$, which is the same mechanism as in Algorithm 2.

Finally, the value α_k used in the extrapolation technique of Section 3.2 is the last value of α_l computed by Algorithm 3 or Algorithm 4 in the final iteration of Algorithm 2 at outer iteration k.

Algorithm 4 Step computation – General case.

Require: Symmetric matrix H_l; regularization parameter $\sigma_l > 0$; Jacobian J_l; gradient ∇f_l; constraint values c_l; penalty parameter τ_k; constants $\alpha^1_{\text{trial}} > 0$; $\alpha^2_{\text{trial}} > \alpha^1_{\text{trial}}$; $\mu_\sigma > 1$; $\mu_\alpha \in (0,1)$; $\sigma_{\text{ub}} > 0$; $j_{\max} \in \mathbb{N}$; $\eta_C > 0$.

1: Set $\alpha_l \coloneqq 0$.
2: Attempt to solve (33), compute the inertia (n_+, n_-, n_0). *Inertia corrections.*
3: If either the factorization (33) fails or $n_- < m$, set $\alpha_l \coloneqq \alpha^1_{\text{trial}}$ and retry solving (33). If the factorization fails again or if $n_- < m$, set $\alpha_l \coloneqq \alpha^2_{\text{trial}}$ and retry solving (33). If it still fails, proceed to Line 4.
4: Until $n_+ = n$ and the factorization (33) succeeds, set $\sigma_l \coloneqq \mu_\sigma \sigma_l$, repeat Line 2. If $\sigma_l > \sigma_{\text{ub}}$, **set** $H_l \coloneqq 0$ and **restart**.
5: Compute the inertia of $H_l + \sigma_l I$ with an LDL^T factorization.
6: If $H_l + \sigma_l I$ is positive definite, compute s_l with Algorithm 3 and **return**.
7: **for** $j = 1, \ldots, j_{\max}$ **do**
8: If s_l satisfies (20) and (21), **return**.
9: Solve (34).
10: Compute α_l^+ with (35).
11: If $\alpha_l^+ < 0$, set $\alpha_l \coloneqq \max(\alpha^1_{\text{trial}}, \mu_\alpha \alpha_l)$, else set $\alpha_l \coloneqq \alpha_l^+$. Solve (33).
12: **end for**
13: Set $\sigma_l \coloneqq \mu_\sigma \sigma_l$ and **restart**.

6 Implementation details

In addition to the core algorithmic components presented in the previous sections, our implementation incorporates several practical strategies to improve performance and robustness. This section describes and justifies these heuristics, which include nonmonotone strategies, initialization techniques, and regularization parameter updates.

6.1 Nonmonotone strategies

A first practical issue encountered in our implementation arises when the penalty parameter τ_k significantly exceeds the threshold $\bar{\tau}$ presented in Theorem 1. For example, this can occur when $\tau_1 = \max(\tau_{\text{init}}, \|y_0\|_1) \gg \bar{\tau}$. In view of the right-hand side of (26), the number of inner loop iterations increases with τ_k. Moreover, similar to the observations of Maratos [40], we find that the algorithm requires smaller steps, that is, larger values of σ_l, when τ_k is too large. To mitigate this effect, we implement two concurrent nonmonotone strategies that relax the step acceptance rule (18) in Algorithm 2.

The first strategy, proposed by Grippo et al. [33], modifies the value of ρ_l in (18). Let $q \in \mathbb{N}$ be a user-defined memory parameter. Let $q_l \coloneqq \min(q, l)$ if $q > 0$ and $q_l \coloneqq 0$ otherwise. Define $\mathcal{S}^+_{q_l}$ as the set of the q_l most recent successful iterations. We define

$$(f + \tau_k \|c\|_2)_{\max,l} \coloneqq \max_{j \in \mathcal{S}^+_{q_l}} f(x_j) + \tau_k \|c(x_j)\|_2. \tag{36}$$

For completeness, we set $\mathcal{S}^+_0 = \{0\}$, and define $(f + \tau_k \|c\|_2)_{\max,0} \coloneqq f(x_0) + \tau_k \|c(x_0)\|_2$. Then, we modify (18) with

$$\rho_l \coloneqq \frac{(f + \tau_k \|c\|_2)_{\max,l} - (f(x_l + s_l) + \tau_k \|c(x_l + s_l)\|_2)}{(f + \tau_k \|c\|_2)_{\max,l} - (f_l + \nabla f_l{}^T s_l + \tau_k \|c_l + J_l s_l\|_2)} \geq \eta_1. \tag{37}$$

To disable this heuristic, it suffices to set $q = 0$.

While the strategy above has proved very effective in practice, we still observed a few instances where Algorithm 2 struggles when the penalty parameter is too large. To address this, we implemented a second nonmonotone strategy. Our approach is inspired by the watchdog technique proposed by Chamberlain et al. [13]. Specifically, it allows a few consecutive iterations to be accepted even if they do not satisfy (37).

The watchdog has two states: *active* and *inactive*. It is *inactive* when Algorithm 2 starts. Suppose that, at some iterate x_{l_0}, the watchdog is *inactive* and the step s_{l_0} satisfies $\rho_{l_0} < \eta_1$ in (37). Instead of rejecting the step, we check whether it sufficiently decreases either f or the penalty term, taken separately. More specifically, we compute

$$\rho^f_{l_0} \coloneqq \frac{f(x_{l_0}) - f(x_{l_0} + s_{l_0})}{-\nabla f(x_{l_0})^T s_{l_0}} \quad \text{and} \quad \rho^c_{l_0} \coloneqq \frac{\|c(x_{l_0})\|_2 - \|c(x_{l_0} + s_{l_0})\|_2}{\|c(x_{l_0})\|_2 - \|c(x_{l_0}) + J(x_{l_0})s_{l_0}\|_2},$$

where we set $\rho^f_{l_0} \coloneqq 0$ if $\nabla f(x_{l_0})^T s_{l_0} > 0$, and $\rho^c_{l_0} \coloneqq 0$ if $\|c(x_{l_0}) + J(x_{l_0})s_{l_0}\|_2 > \|c(x_{l_0})\|_2$, since the corresponding linear model then predicts no decrease. If $\max(\rho^f_{l_0}, \rho^c_{l_0}) \geq \eta_3$, where $\eta_3 \in (0, 1)$ is a user-defined parameter, we activate the watchdog, accept the step s_{l_0}, keep $\sigma_{l_0+1} = \sigma_{l_0}$ and continue with Algorithm 2. Otherwise, we reject the step and proceed to the next iteration of Algorithm 2. While the watchdog is active, we accept steps based on the monotone ratio (18) instead of (37).

Once activated, the watchdog either succeeds or fails. If it remains active for more than $r \in \mathbb{N}$ iterations, it fails. In this case, we deactivate it, reset (x_l, y_l) to (x_{l_0}, y_{l_0}) and increase $\sigma_l \in [\gamma_1^r \sigma_{l_0}, \gamma_2^r \sigma_{l_0}]$. On the other hand, if for some $r_{l_0} \leq r$ and $l \coloneqq l_0 + r_{l_0}$, the step $s_{\text{wd},l} \coloneqq x_l - x_{l_0}$ satisfies either

$$f(x_{l_0}) + \tau_k \|c(x_{l_0})\|_2 - f(x_{l_0} + s_{\text{wd},l}) - \tau_k \|c(x_{l_0} + s_{\text{wd},l})\|_2 \geq \tfrac{1}{2}\eta_0 \min(1, \|\nabla f_l + J_l^T y_{l+1}\|^2_\infty), \tag{38}$$

or

$$\|\nabla f_l + J_l^T y_{l+1}\|_\infty \leq (1 - \eta_0)\|\nabla f_{l_0} + J_{l_0}^T y_{l_0+1}\|_\infty, \tag{39}$$

where $\eta_0 \in (0, 1)$ is a user-defined parameter, then it succeeds and is deactivated. We justify (38) with Step Assumption 3.2. For (39), a uniform decrease in the optimality measure $\|\nabla f_l + J_l^T y_{l+1}\|_\infty$ should eventually lead to $\|\nabla f_l + J_l^T y_{l+1}\|_\infty \to 0$.

6.2 Quadratic regularization parameter heuristics

As explained in Section 3.4, and shown in Section 4.1, the choice of the regularization parameter σ_l is crucial for both the practical performance and the theoretical convergence of Algorithm 2. We now discuss how we update σ_l throughout the inner iterations.

When we first call Algorithm 2, on the first outer iteration, we initialize the regularization parameter as

$$\sigma_0 = \epsilon_{\text{mach}}. \tag{40}$$

In subsequent calls to Algorithm 2, we initialize σ_0 at outer iteration $k + 1$ as the final value of σ_l on outer iteration k. To ensure that the convergence analysis in Section 4.1 remains applicable, we project the former into the interval $[\sigma_{\min}, \sigma_{\text{ub}}]$ with $\sigma_{\min} = \epsilon_{\text{mach}}$ and the same value of σ_{ub} as in Algorithm 4. In particular, the value of $\sigma_{\max}$ in (25) can be bounded independently of k. Then, σ_0 in the upper bound of Lemma 4 can be replaced with $\sigma_{\min}$ which is independent of k too.

The motivation for (40) is to be aggressive near a potential solution. Indeed, we have observed fast local convergence behavior when $\sigma_l \approx 0$. Therefore, when the initial pair (x_0, y_0) is close to a solution of (1), the choice (40) allows us to converge to high levels of accuracy in just a few iterations.

When this is not the case however, we often observe that the aggressive initialization (40) produces poor agreement between the actual and first-order predicted reductions measured in (37). As a result, the ratio ρ_l in (37) is often negative during the first few inner iterations. Applying the standard update strategy in this situation generates a sequence of unsuccessful iterations to reach higher orders of magnitude for σ_l. To avoid this behavior, our implementation allows for a larger increase of σ_l on the first unsuccessful iteration of Algorithm 2. That is, when we encounter the first unsuccessful iteration, we set $\sigma_{l+1} \coloneqq \max(\gamma_2 \sigma_l, \sqrt{\|\nabla f_l + J_l^T y_{l+1}\|_\infty})$. Choosing σ_{l+1} proportional to the square root of a measure of dual infeasibility has recently been suggested in [21, 42]. We observed that this

choice saves a few unsuccessful iterations during the early stages of the algorithm. Moreover, this strategy does not affect the proof mechanism developed in Section 4.1 because it only allows for a larger increase of σ_l once.

6.3 Linear algebra routines

By default, we solve the linear systems (33) using the multifrontal symmetric-indefinite LDL^T factorization implemented in MUMPS [1, 2, 45]. We use a default pivot tolerance of $\epsilon_{\text{piv}} = \epsilon_{\text{mach}}$, which favors sparse factors over pivoting robustness and results in faster factorizations.

We also use iterative refinement, with at most 5 refinement steps. After each solve, we compare the backward error estimate reported by MUMPS with $\sqrt{\epsilon_{\text{mach}}}$. If the estimate exceeds this threshold, we consider the solve unsuccessful, increase the pivot tolerance to 10^{-2}, and force exactly 5 refinement steps for all subsequent solves.

This adaptive strategy exploits the fact that well-conditioned systems can typically be solved efficiently with a small pivot tolerance, while a single ill-conditioned system warrants more conservative factorization parameters. To avoid using an unnecessarily large pivot tolerance indefinitely, we reduce ϵ_{piv} by a factor of 10 whenever the backward error estimate falls below $100\epsilon_{\text{mach}}$ for more than 20 consecutive solves.

Finally, we use MUMPS automatic scaling of (33) to improve the conditioning of the linear systems.

6.4 Initialization

For the initial approximation of the Lagrange multiplier y_0, if no initial guess is provided, we initialize y_0 as the least-squares solution of $J_0^T y = -\nabla f_0$ by solving

$$\begin{pmatrix} I & J_0^T \\ J_0 & 0 \end{pmatrix} \begin{pmatrix} w \\ y_0 \end{pmatrix} = \begin{pmatrix} -\nabla f_0 \\ 0 \end{pmatrix}. \tag{41}$$

We use the same sparse LDL^T factorization routines as in Section 6.3 to solve (41). If the factorization of the matrix in (41) fails, we retry by setting the lower-right block to $-\alpha^1_{\text{trial}} I$ first and then $-\alpha^2_{\text{trial}} I$ as in Algorithm 4.

If $\nabla f_0 = 0$, however, $y_0 = 0$, so that, in view of Line 2 of Algorithm 1, the value of τ_1 is set to τ_{init}. For example, this happens when f is constant. In this case, instead of solving (41), we solve

$$\begin{pmatrix} I & J_0^T \\ J_0 & 0 \end{pmatrix} \begin{pmatrix} w \\ y_0 \end{pmatrix} = \begin{pmatrix} 0 \\ -c_0 \end{pmatrix}. \tag{42}$$

The advantage of (41) and (42) is that if x_0 is a local solution of (1), then our algorithm stops immediately after having computed y_0. However, a problem with the initialization strategy above is that it may produce a very large value of $\|y_0\|_1$, especially when J_0 is ill-conditioned, which in turn produces a very large value of τ_1, in view of Line 2 of Algorithm 1. To avoid this, similarly to IPOPT [59, Section 3.6], we first implemented a simple safeguard that checks whether $\|y_0\|_1 > y_{\max}$ and discards y_0 when this condition is met. This strategy, however, only improved the performance of the algorithm on a few problems, while unnecessarily discarding y_0 on many others. We found that a numerically more robust approach is to detect ill-conditioning of either (41) or (42). Let K be the matrix in (41) or (42), u the right-hand side of the corresponding system and $\|K\|_{\max}$ the maximum absolute value of the entries of K. If K is nonsingular, then its condition number satisfies

$$\kappa_\infty(K) = \|K\|_\infty \|K^{-1}\|_\infty \geq \|K\|_{\max} \|(w, y_0)\|_\infty / \|u\|_\infty.$$

We therefore check whether

$$\|K\|_{\max} \|(w, y_0)\|_\infty / \|u\|_\infty > 1/\epsilon_{\text{mach}}. \tag{43}$$

When (43) is satisfied, we retry the initialization by setting the lower-right block of K to $-\alpha^1_{\text{trial}} I$ first and then $-\alpha^2_{\text{trial}} I$ as in Algorithm 4. If (43) is met for both α^1_{trial} and α^2_{trial}, we set $y_0 = 0$.

6.5 Quasi-Newton approximations

For the definition of H_l in (17), our algorithm allows for quasi-Newton approximations of the Hessian of the Lagrangian. Namely, our algorithm implements a limited memory BFGS method to approximate $H_l \approx \nabla^2 \mathcal{L}(x_l, y_l)$. At each inner iteration l, we compute

$$y_{\text{qn},l} \coloneqq \nabla \mathcal{L}(x_{l+1}, y_{l+1}) - \nabla \mathcal{L}(x_l, y_{l+1}).$$

When we accept the step s_l, we push the pair $(s_l, y_{\text{qn},l})$ into a limited memory BFGS data structure using the compact representation of Byrd et al. [10]. This allows us to write H_l as

$$H_l = \delta_l I + N_l M_l N_l^T.$$

Our implementation defines δ_l as $\delta_l = y_{\text{qn},l}^T y_{\text{qn},l} / s_l^T y_{\text{qn},l}$. To ensure positive definiteness and boundedness of H_l, the update is skipped unless

$$s_l^T y_{\text{qn},l} > \epsilon_{\text{mach}} \|y_{\text{qn},l}\|_2 \max(\|s_l\|_2, \|y_{\text{qn},l}\|_2).$$

The condition $s_l^T y_{\text{qn},l} > \epsilon_{\text{mach}} \|s_l\|_2 \|y_{\text{qn},l}\|_2$ ensures positive definiteness, while the condition $s_l^T y_{\text{qn},l} > \epsilon_{\text{mach}} \|y_{\text{qn},l}\|_2^2$ ensures that H_l is bounded. If more than two consecutive updates are skipped, the approximation is reset.

In Algorithms 3 and 4, when we solve (33) and (34), we use a standard Sherman-Morrison-Woodbury formula. That is, we use an LDL^T factorization of the 2×2 block system with H_l replaced by $\delta_l I$ and then update the inverse by adding the low-rank term $N_l M_l N_l^T$, following the procedure described in [60, Section 3.8].

6.6 Handling small steps

In a few cases, we have observed that the steps s_l computed by Algorithm 4 become very small relative to x_l. For this reason, at each inner iteration l, we check whether

$$|s_l^{(i)}| < \epsilon_{\text{mach}}(1 + |x_l^{(i)}|), \quad \text{for each } i = 1, \ldots, n. \tag{44}$$

When this happens, the solver effectively stalls, therefore we terminate the overall algorithm with a corresponding failure message. To make sure that (44) is not triggered by some numerical error, we return the failure message when (44) is satisfied for $n_{\text{small}} + 1$ consecutive inner iterations.

6.7 Other heuristics

By default, our implementation is not invariant to scaling of the variables, objective function, and constraints. Similar behavior has been reported for other solvers, including IPOPT [59, Section 3.8]. We implemented the scaling strategy described in [59, Section 3.8], but found that it degraded performance of the method, therefore we did not include it in the implementation.

We use two safeguards to adapt the penalty parameter τ_k. First, for some values of τ_k, $x \to f(x) + \tau_k \|c(x)\|_2$ can be unbounded below. In this case, the iterates of Algorithm 2 may satisfy $f_l \to -\infty$. We therefore terminate the inner loop with a failure message when $f_l \leq -1/\epsilon_{\text{mach}} \approx -10^{16}$. The outer loop then increases the penalty parameter according to $\tau_{k+1} = 10\tau_k$ and restarts the inner loop.

Second, τ_k can occasionally be substantially smaller than the penalty threshold discussed in Theorem 1. In this situation, the algorithm may fail to sufficiently prioritize constraint feasibility, as

indicated by $\|c(x_k)\|_2 \geq \|c(x_{k-1})\|_2$. When this occurs, we increase the penalty parameter more aggressively by multiplying the minimum penalty increase $\tau_{\min}$ by a factor of 10. When the penalty parameter becomes larger than a fixed threshold $\tau_{\max} > 0$, we terminate the algorithm with a failure message.

Finally, to compute the tolerances ϵ_D^k and ϵ_P^k used in Algorithm 1 in practice, instead of (11) we use a relative tolerance based on the current residuals

$$\epsilon_D^k \coloneqq \min\left(1, \max\left(\mu_\epsilon \|\nabla f(x_{k-1}) + J(x_{k-1})^T y_{k-1}\|_\infty, \epsilon_D\right)\right) \text{ and } \epsilon_P^k \coloneqq \min\left(1, \max\left(\mu_\epsilon \|c(x_{k-1})\|_\infty, \epsilon_P\right)\right).$$

When either dual or primal infeasibility are already small, this avoids unnecessary calls to Algorithm 2 with a loose tolerance that is satisfied immediately.

6.8 Algorithmic hyperparameters

This section summarizes all default algorithmic hyperparameters that we presented throughout this paper. The default values of the constants in Algorithms 1 and 2 are $\tau_{\text{init}} = 1$; $\epsilon_P^1 = 1$; $\epsilon_D^1 = 1$; $\tau_{\min} = 1$; $\tau_{\max} = 1/\epsilon_{\text{mach}}^{1.5} \approx 10^{23}$; $\mu_\epsilon = 0.01$; $\sigma_0 = \epsilon_{\text{mach}} \approx 10^{-16}$; $\eta_1 = \epsilon_{\text{mach}}^{0.25} \approx 10^{-4}$; $\eta_2 = 0.1$ by default and 0.9 when H_l is a quasi-Newton approximation; $\gamma_1 = \gamma_2 = 3$; $\gamma_3 = 1/3$. For the infeasibility detection described in Section 3.3, we choose $\epsilon_I = 10^{-3}$. In Algorithm 4, the default values of the constants are $\alpha_{\text{trial}}^1 = \epsilon_{\text{mach}}^{0.8} \approx 10^{-13}$; $\alpha_{\text{trial}}^2 = \epsilon_{\text{mach}}^{0.6} \approx 10^{-10}$; $\mu_\alpha = 0.1$; $\mu_\sigma = 10$; $\sigma_{\text{ub}} = 1/\epsilon_{\text{mach}}^{0.8} \approx 10^{12}$; $\eta_C = 10^{-16}$; $j_{\max} = 5$.

For the nonmonotone strategies described in Section 6.1, we choose $q = 12$; $r = 10$; $\eta_0 = \epsilon_{\text{mach}}^{0.5} \approx 10^{-8}$; $\eta_3 = 0.9$. The limited memory BFGS approximation of Section 6.5 uses a default memory of 6. For the small-step test of Section 6.6, we choose $n_{\text{small}} = 2$.

7 Software

Penelopt, whose name derives from *penalty optimization*, is written in Julia [7] and is available as open-source software under the Mozilla Public License 2.0 [29]. The results of this paper were obtained with version 1.0.0 of Penelopt and Julia 1.12.

Penelopt is part of the JSO ecosystem [51], which provides a common interface for nonlinear optimization solvers in Julia. We use the NLPModels.jl API [52] to model the optimization problems arising in our algorithm, and the SolverCore.jl API [41] for the implementation of the optimization algorithm.

Our continuous integration solves a set of small test problems at each commit, on Linux, macOS and Windows. The documentation [30] lists all parameters of the algorithm along with their default values, given in Section 6.8. It also contains tutorials that show how to call Penelopt on JuMP [24, 39], AMPL [26], and CUTEst [32] models, and how to switch between different linear solvers.

For example, to solve the sixth problem from the Hock–Schittkowski collection [38], formulated as

$$\min_{x_1, x_2} (1 - x_1)^2 \quad \text{subject to } 10(x_2 - x_1^2) = 0,$$

we can model it using the automatic differentiation interface of NLPModels.jl implemented in the ADNLPModels.jl package [44].

```
using ADNLPModels

f(x) = (1 - x[1])^2
c(x) = [10 * (x[2] - x[1]^2)]
```

```
lcon = [0.0]
ucon = [0.0]

x0 = [-1.2, 1.0]

nlp = ADNLPModel(f, x0, c, lcon, ucon)
```

Then, we call the solver with the default algorithmic parameters,

```
using Penelopt
stats = L2Penalty(nlp; print_level = 1)
```

which results in the output shown in Listing 1.

```
This is Penelopt.jl v1.0.0.
Running with linear solver MUMPS v5.9.1.
-------------------------------------------------------------------------------------------------------
Iter    sIter  Objective       pfeas       dfeas       τ           ptol        dtol        ‖x‖
-------------------------------------------------------------------------------------------------------
0       0      +4.8400000e+00  4.40e+00    1.56e+00    1.00e+00    1.00e+00    1.56e-02    1.56e+00
1       8      +1.4515287e-04  4.25e-02    9.71e-03    1.00e+00    4.25e-04    9.71e-05    1.39e+00
2       2      +9.9022550e-11  3.91e-06    7.88e-06    1.00e+00    8.05e-08    7.88e-08    1.41e+00
3       1      +2.3024446e-15  9.81e-10    3.85e-08    1.00e+00    8.05e-08    3.82e-08    1.41e+00
4       1      +5.7937134e-21  2.22e-14    6.09e-11    1.00e+00    8.05e-08    3.82e-08    1.41e+00
Number of Iterations: 4
Objective...........: +5.793713424604307e-21
Primal Feasibility..:  2.220446049250313e-14
Dual Feasibility....:  6.090412449872034e-11
EXIT: first_order.
```

Listing 1: Output of the solver on HS6.

The solver reports the version and linear solver in use, followed by the iteration log and the final status. On each iteration, the solver reports in order: the outer iteration number, the number of iterations of Algorithm 2, the objective value, the residuals $\|c(x_k)\|_\infty$ and $\|\nabla f_k + J_k^T y_k\|_\infty$, the penalty parameter τ_k, the primal and dual tolerances ϵ_P^k, ϵ_D^k, and the norm of the current iterate $\|x_k\|_2$.

8 Numerical results

In this section, we present how our algorithm compares with other state-of-the-art solvers on a set of benchmark problems. Additionally, we provide numerical experiments to show the effect of the two nonmonotone strategies described in Section 6.1 on the performance of our algorithm.

We evaluate the performance of our algorithm on the CUTEst [32] benchmark collection. We select all CUTEst instances with equality constraints only, at least one constraint, at most as many constraints as variables, and no bounds other than fixed variables, which are removed in a preprocessing step. This results in a set of 224 problems. The largest problem in terms of variables has $n = 132{,}098$ and $m = 65{,}025$. In terms of constraints, the largest problem has $n = 123{,}200$ and $m = 123{,}200$.

We compare our algorithm with the open-source IPOPT [59] v3.14 solver, which reduces to an SQP method for (1). As additional points of comparison, we also run MadNLP [56] v0.11 and Uno v2.9 [57, 58].

All solvers run on each problem with a time limit of 5 minutes. All benchmarks were run on GitHub-hosted `windows-latest` runners, which are x64 virtual machines running Windows Server with 4 virtual CPUs and 16 GB of memory. Moreover, all solvers declare optimality when (8) is fulfilled with $\epsilon_P = \epsilon_D = 10^{-6}$.

We disable the acceptable-level termination criteria of IPOPT, MadNLP, and Uno. For all three of these, we turn off scaling of the dual feasibility measure [59, (5)] by setting $s_{\max}$ to the maximum representable Float64 value. Although an optimization algorithm might encounter numerical difficulties

when the Jacobian is ill-conditioned near a solution of (1), we do not consider altering the stopping criterion an appropriate strategy. First, this makes it difficult to compare the performance of different solvers. Second, it gives a false impression of the quality of the solution to (1) when the problem is difficult.

All other parameters are set to their default values, and we use the hyperparameters from Section 6.8 for Penelopt. All solvers, including Penelopt, use the MUMPS [1, 2] v5.9 sparse linear solver to solve the linear systems arising in the computations. We run Uno with its default `auto` preset, which selects the IPOPT preset when $n + m \geq 2{,}000$ or when the Jacobian and the Hessian of the Lagrangian have at least 50,000 nonzeros in total, and the FilterSQP preset, with BQPD as the QP solver, otherwise.

Two exceptions apply to the problem set described above. First, when using L-BFGS approximations, the problem FLOSP2TL caused the solvers to crash, hence we do not include it in L-BFGS benchmark runs. Second, MadNLP and Uno repeatedly crashed on the problems CYCLOOCF and CYCLOOCT. We therefore did not run these two solvers on these two problems, and we count both problems as failures for MadNLP and Uno.

CUTEst benchmark

To present the numerical results, we use Dolan and Moré [22] performance profiles. Given a set $\mathcal{P}$ of problems and a set $\mathcal{S}$ of solvers, we compute a performance metric $t_{p,s}$ for each $p \in \mathcal{P}$ and $s \in \mathcal{S}$. We then compute the ratio $r_{p,s} = t_{p,s} / \min_{s' \in \mathcal{S}} t_{p,s'}$ and define the performance profile of solver s as

$$\pi_s(\omega) := \frac{1}{n_p} \sum_{p \in \mathcal{P}} \mathbf{1}_{r_{p,s} \leq \omega},$$

where $n_p = |\mathcal{P}|$, $\omega \geq 1$, and $\mathbf{1}_{r_{p,s} \leq \omega} = 1$ if $r_{p,s} \leq \omega$ and 0 otherwise.

For the time profiles, we measure the wall-clock time t, in seconds, reported by each solver. Timings are recorded with a resolution of one millisecond. To account for this timer resolution, we use $t_{p,s} := \max(t, 10^{-3})$ as the performance metric for time, so that two runs that both complete in under one millisecond are considered tied.

To test the infeasibility detection mechanism of Section 3.3, we trigger an infeasibility certification procedure whenever a solver declares infeasibility. Suppose a solver declares a problem infeasible at a point $\bar{x}$. To certify local infeasibility, we wish to find a local minimum x^* of

$$\min \|c(x)\|_2^2,$$

that satisfies $\|c(x^*)\|_2 > 0$ and $\|J(\bar{x})(\bar{x} - x^*)\|_2 \leq \Delta$. In practice, we solve the equivalent problem

$$\min \|r\|_2^2 \quad \text{subject to } c(x) - r = 0,$$

using IPOPT with all default parameters, a time limit of 5 minutes, and a tolerance of 10^{-9}, started from $x_0 = \bar{x}$. We choose $\Delta = 10$ and verify that $\|c(x^*)\|_2 > 0$ by checking that $\|c(x_{\text{IPOPT}})\|_2 > 10^{-3}$, where x_{IPOPT} is the solution returned by IPOPT. Then, we check that $\|J(\bar{x})(\bar{x} - x^*)\|_2 \leq \Delta$ by verifying that $\|J(\bar{x})(\bar{x} - x_{\text{IPOPT}})\|_2 \leq \Delta$. If IPOPT fails to solve the problem or either condition is not met, we consider the certification procedure a failure.

Although this certification procedure might fail because x_{IPOPT} is an approximation of x^*, we consider it an effective measure of the quality of the infeasibility detection mechanism. When a solver declares a problem infeasible, we consider the run successful if the certification procedure confirms local infeasibility and unsuccessful otherwise. If both solvers declare infeasibility, the certification procedure may confirm only one of them. In this case, we consider only the solver with confirmed infeasibility successful.

The source code to reproduce the numerical results is available in Gollier [28]. Figure 1 reports performance profiles for Penelopt against IPOPT, with the exact Hessian and with an L-BFGS approximation. Figure 2 additionally compares Penelopt, with $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$, against MadNLP and Uno under matching tolerances and time limits.

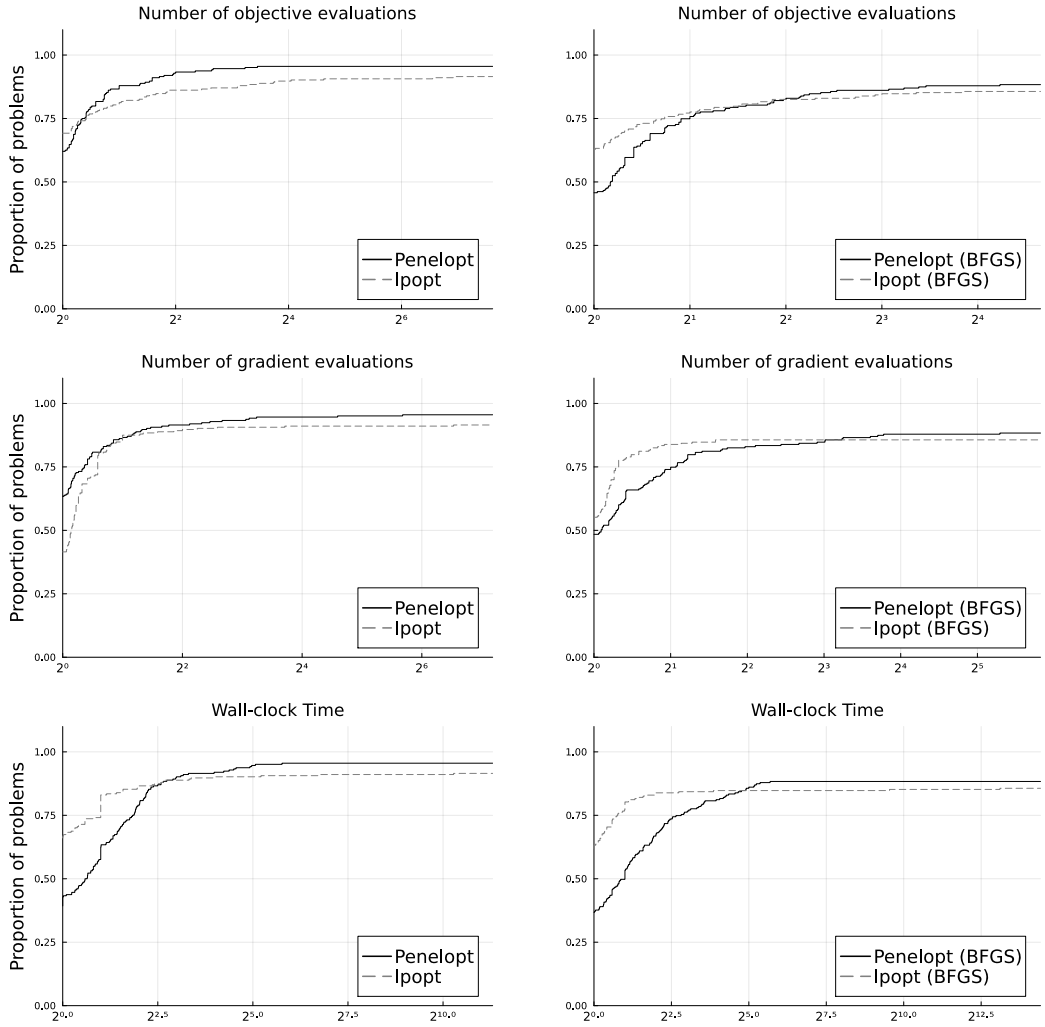

Figure 1: Left: Penelopt against IPOPT with $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$. Right: Penelopt against IPOPT with an L-BFGS quasi-Newton approximation of $\nabla^2 \mathcal{L}(x_l, y_l)$, on 223 problems (FLOSP2TL excluded). Bottom row: wall-clock time, with times below 1 ms set to 1 ms. The higher, the better.

In Figure 1, we observe that Penelopt is competitive with IPOPT in terms of robustness and efficiency, with respect to the number of objective and gradient evaluations. We see in Figure 1 that Penelopt is slower than IPOPT in terms of wall-clock time. We attribute this mainly to our step computation procedure (Algorithm 4), which can require multiple factorizations of the KKT system (33) to compute a step s_l. Then, in Figure 2, we observe that Penelopt is ahead of MadNLP and Uno in terms of robustness.

When $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$, Penelopt timed out on the problems EIGENBCO and DRCAVTY3, while it ended with a small step status on the problem CATENARY. All other problems were either solved or declared infeasible. In Table 1, we report the results of the infeasibility detection procedure described above.

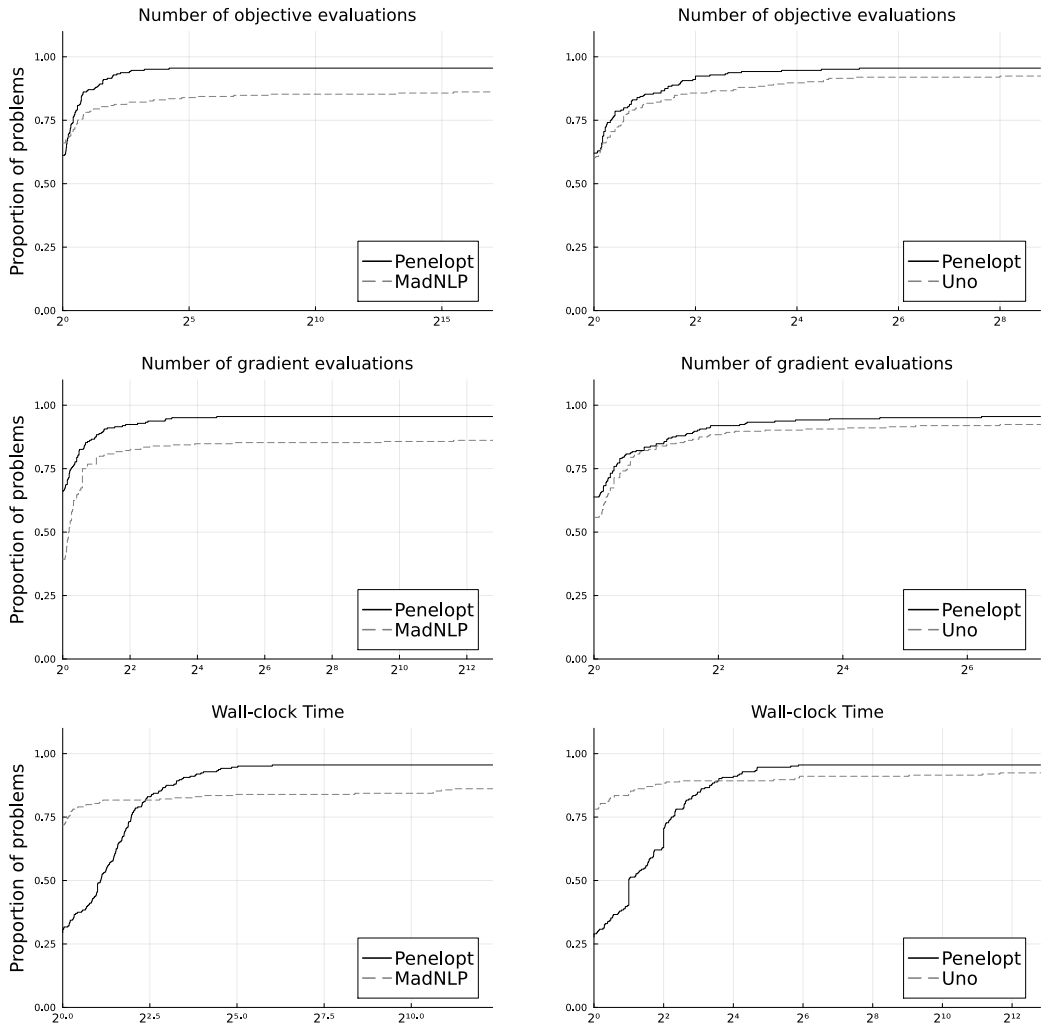

Figure 2: Left: Penelopt against MadNLP with $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$. Right: Penelopt against Uno with $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$. CYCLOOCF and CYCLOOCT count as failures for MadNLP and Uno. Bottom row: wall-clock time, with times below 1 ms set to 1 ms. The higher, the better.

Effect of nonmonotone strategies

We now present the effect of the two strategies of Section 6.1, which, among the heuristics of Section 6, have the largest impact on the performance of Penelopt in our experiments. We compare Penelopt, with $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$ and the default hyperparameters of Section 6.8, with two variants: one in which the watchdog is disabled, and one in which steps are accepted with the monotone ratio (18) instead of (37), the watchdog being kept. All other settings are as above. Figure 3 reports the corresponding performance profiles in terms of objective and gradient evaluations.

We see on Figure 3 that both nonmonotone strategies improve the robustness of Penelopt, with the update rule (37) having a larger impact than the watchdog rule.

Table 1: Infeasibility detection results on the CUTEst benchmark with $H_l \coloneqq \nabla^2 \mathcal{L}(x_l, y_l)$. Columns: problem name, number of variables, number of constraints, Penelopt status, infeasibility certification for the Penelopt solution, IPOPT status, infeasibility certification for the IPOPT solution. False indicates that we were unable to certify infeasibility.

Problem	n	m	Penelopt (default)	Certified	IPOPT (default)	Certified
ARTIF	5002	5000	infeasible	true	infeasible	true
DENSCHNENE	3	3	first_order		infeasible	true
DRCAVTY3	4489	3969	max_time		infeasible	true
FLOSP2HH	2883	2763	infeasible	false	infeasible	false
FLOSP2HL	2883	2763	infeasible	true	infeasible	true
FLOSP2HM	2883	2763	infeasible	true	infeasible	true
FLOSP2TH	2883	2763	infeasible	false	first_order	
FREURONE	2	2	infeasible	true	infeasible	true
HATFLDFLNE	3	3	infeasible	false	first_order	
HIMMELBD	2	2	infeasible	true	infeasible	true
JUNKTURN	10010	7000	first_order		infeasible	false
LINCONT	1257	419	infeasible	true	infeasible	false
LUKVLE15	9997	7497	first_order		infeasible	false
LUKVLE16	9997	7497	first_order		infeasible	false
NONDIANE	5000	5000	infeasible	true	max_time	
NONMSQRTNE	4900	4900	infeasible	false	infeasible	false
OSCIPANE	10	10	infeasible	false	first_order	
POROUS1	5184	4900	infeasible	false	first_order	
POWELLSQ	2	2	infeasible	true	infeasible	true
POWERSUMNE	4	4	first_order		infeasible	false
VANDANIUMS	22	10	infeasible	true	unknown	
WOODSNE	4000	3001	infeasible	true	infeasible	true
YATP2CNE	123200	123200	infeasible	false	first_order	

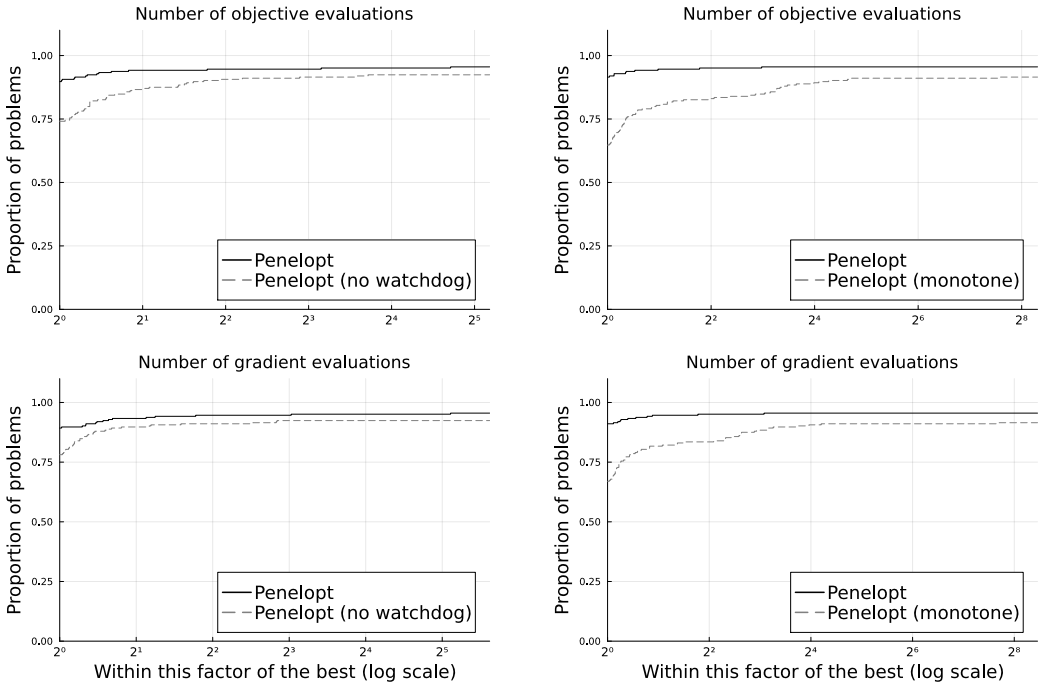

Figure 3: Ablation study with $H_l = \nabla^2 \mathcal{L}(x_l, y_l)$. Left: Penelopt against Penelopt without the watchdog. Right: Penelopt against Penelopt with the monotone acceptance rule (18). The higher, the better.

9 Conclusion

Penelopt is the first large-scale implementation of the exact penalty method. On large-scale problems, our implementation is competitive with IPOPT and places the exact penalty method on par with SQP methods in terms of robustness and efficiency.

The extension of the method to handle bound constraints is the subject of active research. In particular, we are investigating the use of a logarithmic barrier function to handle inequalities.

We are also considering alternative linear algebra routines for solving the linear systems arising in Algorithms 3 and 4. Specifically, we are exploring preconditioned iterative methods to replace the sparse LDL^T factorizations. In addition, we plan to evaluate Penelopt in single and quadruple precision. For this purpose, in addition to MUMPS and MA57, our solver is compatible with LDLFactorizations.jl [50], which is also part of the JSO ecosystem, implements a sparse LDL^T factorization in pure Julia and supports single and quadruple precision. In a similar fashion, we plan to extend our implementation to GPU architectures by using cuDSS [43, 49] in place of MUMPS and MA57 for the linear algebra routines, as in MadNLP.

The superlinear local convergence rate of Algorithm 2 also remains to be established. Additionally, it remains to prove that Algorithm 1 converges to a solution of (1) when the penalty parameter τ_k is unbounded. Finally, a more theoretically grounded strategy for the indefinite $H_l + \sigma_l I$ case of Algorithm 4 is also a subject of ongoing research.

AI usage disclosure

Generative AI tools were used to improve the language and grammar of this paper, to screen the text for mathematical inconsistencies, and to draft and revise parts of the text; all suggested references were verified by the authors. Claude (Anthropic) was used to assist with code refactoring and documentation during the development of Penelopt. The authors reviewed and edited all such output as needed and take full responsibility for the content of this paper.

References

[1] P. Amestoy, I. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods. Appl. Mech. Eng., 184(2):501–520, 2000.

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. MUMPS: A general purpose distributed memory sparse solver. In T. Sørevik, F. Manne, A. H. Gebremedhin, and R. Moe, editors, Applied Parallel Computing. New Paradigms for HPC in Industry and Academia, volume 1947 of Lecture Notes in Computer Science, pages 121–130, Heidelberg, Berlin, New York, 2001. Springer Verlag.

[3] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt. On augmented Lagrangian methods with general lower-level constraints. SIAM J. Optim., 18(4):1286–1309, 2008.

[4] A. Y. Aravkin, R. Baraldi, and D. Orban. A proximal quasi-Newton trust-region method for nonsmooth regularized optimization. SIAM J. Optim., 32:900–929, 2022.

[5] M. S. Bazaraa and J. J. Goode. Sufficient conditions for a globally exact penalty function without convexity. In M. Guignard, editor, Optimality and Stability in Mathematical Programming, volume 19 of Mathematical Programming Studies, pages 1–15. Springer Verlag, Heidelberg, Berlin, New York, 1982.

[6] D. P. Bertsekas. Constrained Optimization and Lagrange Multiplier Methods. Academic Press, New York, 1982.

[7] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. SIAM Rev., 59(1):65–98, 2017.

[8] E. Birgin and J. M. Martinez. Practical Augmented Lagrangian Methods for Constrained Optimization. Number 10 in Fundamentals of Algorithms. SIAM, Philadelphia, USA, 2014.

[9] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. Acta Numer., 4:1–51, 1995.

[10] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. Math. Program., 63(1):129–156, 1994.

[11] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. SIAM J. Optim., 9(4):877–900, 1999.

[12] C. Cartis, N. I. M. Gould, and Ph. L. Toint. On the evaluation complexity of composite function minimization with applications to nonconvex nonlinear programming. SIAM J. Optim., 21:1721–1739, 2011.

[13] R. Chamberlain, M. J. D. Powell, C. Lemaréchal, and H. C. Pedersen. The watchdog technique for forcing convergence in algorithms for constrained optimization. Math. Program. Stud., 16:1–17, 1982.

[14] C. Charalambous. A lower bound for the controlling parameters of the exact penalty functions. Math. Program., 15:278–290, 1978.

[15] T. F. Coleman and A. R. Conn. Second-order conditions for an exact penalty function. Math. Program., 19:178–185, 1980.

[16] A. R. Conn, N. I. M. Gould, and Ph. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. SIAM J. Numer. Anal., 28(2):545–572, 1991.

[17] A. R. Conn, N. I. M. Gould, and Ph. Toint. LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A). Springer Verlag, Heidelberg, Berlin, New York, 1992.

[18] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Trust-Region Methods. Number 1 in MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[19] Y. Diouane, M. Gollier, and D. Orban. Nonsmooth exact penalty methods for equality-constrained optimization: Complexity and implementation. SIAM J. Optim., 36(2):626–650, 2026.

[20] Y. Diouane, M. L. Habiboullah, and D. Orban. A proximal modified quasi-Newton method for nonsmooth regularized optimization. SIAM J. Optim., 36(2):534–563, 2026.

[21] N. Doikov and Y. Nesterov. Gradient regularization of Newton method with Bregman distances. Math. Program., 204(1):1–25, 2024.

[22] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. Math. Program., 91:201–213, 2002.

[23] I. S. Duff. MA57—a code for the solution of sparse symmetric definite and indefinite systems. ACM Trans. Math. Software, 30(2):118–144, 2004.

[24] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. SIAM Rev., 59(2):295–320, 2017.

[25] R. Fletcher and S. Leyffer. User manual for filterSQP. Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Dundee, Scotland, 1998.

[26] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. Management Sci., 36(5):519–554, 1990.

[27] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM J. Optim., 12(4):979–1006, 2002.

[28] M. Gollier. Benchmark-runners: Benchmark scripts for Penelopt.jl. `https://github.com/MaxenceGollier/Benchmark-runners`, 2026. Version 1.0.0.

[29] M. Gollier. Penelopt.jl: A large-scale equality-constrained optimization solver. `https://github.com/JuliaSmoothOptimizers/Penelopt.jl`, 2026. Version 1.0.0.

[30] M. Gollier. Penelopt.jl documentation. `https://jso.dev/Penelopt.jl/stable`, 2026.

[31] N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. SIAM J. Optim., 11(4):974–1002, 2001.

[32] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. Comput. Optim. Appl., 60:545–557, 2015.

[33] L. Grippo, F. Lampariello, and S. Lucidi. A truncated Newton method with nonmonotone line search for unconstrained optimization. J. Optim. Theory and Applics., 60(3):401–419, 1989.

[34] S. P. Han. A globally convergent method for nonlinear programming. J. Optim. Theory and Applics., 22 (3):297–309, 1977.

[35] S. P. Han and O. L. Mangasarian. Exact penalty functions in nonlinear programming. Math. Program., 17:251–269, 1979.

[36] E. V. Haynsworth. Determination of the inertia of a partitioned Hermitian matrix. Linear Algebra Appl., 1(1):73–81, 1968.

[37] M. R. Hestenes. Multiplier and gradient methods. J. Optim. Theory and Applics., 4(5):303–320, 1969.

[38] W. Hock and K. Schittkowski. Test Examples for Nonlinear Programming Codes, volume 187 of Lecture Notes in Economics and Mathematical Systems. Springer Verlag, Heidelberg, Berlin, New York, 1981.

[39] M. Lubin, O. Dowson, J. Dias Garcia, J. Huchette, B. Legat, and J. P. Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. Math. Program. Comp., 15:581–589, 2023.

[40] N. Maratos. Exact penalty function algorithms for finite dimensional and control optimization problems. PhD thesis, University of London, 1978.

[41] T. Migot, D. Orban, A. Soares Siqueira, and contributors. SolverCore.jl: Core package to build novel solvers in Julia, 2023. DOI: 10.5281/zenodo.4758376.

[42] K. Mishchenko. Regularized Newton method with global $\mathcal{O}(1/k^2)$ convergence. SIAM J. Optim., 33(3): 1440–1462, 2023.

[43] A. Montoison and contributors. CUDSS.jl: Julia interface for NVIDIA cuDSS, 2024.

[44] A. Montoison, T. Migot, D. Orban, and A. S. Siqueira. ADNLPModels.jl: Automatic differentiation models implementing the nlpmodels api, 2021. DOI: 10.5281/zenodo.4605982.

[45] A. Montoison, D. Orban, W. R. Sweeney, and contributors. MUMPS.jl: A Julia Interface to MUMPS, 2024. DOI: 10.5281/zenodo.3271646.

[46] J. J. Moré and D. C. Sorensen. Computing a trust region step. SIAM J. Sci. and Statist. Comput., 4: 553–572, 1983.

[47] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. Math. Program., 14: 41–72, 1978.

[48] B. A. Murtagh and M. A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. In A. G. Buckley and J.-L. Goffin, editors, Algorithms for Constrained Minimization of Smooth Nonlinear Functions, volume 16 of Mathematical Programming Studies, pages 84–117. Springer Verlag, Heidelberg, Berlin, New York, 1982.

[49] NVIDIA Corporation. NVIDIA cuDSS: A High-Performance CUDA Library for Direct Sparse Solvers, 2023. `https://developer.nvidia.com/cudss`.

[50] D. Orban and contributors. LDLFactorizations.jl: Factorization of Symmetric Matrices, 2023. DOI: 10.5281/zenodo.3900668.

[51] D. Orban and A. S. Siqueira. Infrastructure and solvers for continuous optimization in Julia, 2019. DOI: 10.5281/zenodo.2655082.

[52] D. Orban, A. Soares Siqueira, and contributors. NLPModels.jl: Data Structures for Optimization Models, 2023. DOI: 10.5281/zenodo.2558627.

[53] U. M. G. Palomares and O. L. Mangasarian. Superlinearly convergent quasi-Newton algorithms for nonlinearly constrained optimization problems. Math. Program., 11(1):1–13, 1976.

[54] T. Pietrzykowski. An exact potential method for constrained maxima. SIAM J. Numer. Anal., 6(2): 299–304, 1969.

[55] M. J. D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, Optimization, pages 283–298. Academic Press, London, 1969.

[56] S. Shin, M. Anitescu, and F. Pacaud. Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods. Electric Power Systems Research, 236: 110651, 2024.

[57] C. Vanaret and S. Leyffer. Implementing a unified solver for nonlinearly constrained optimization. Math. Program. Comp., 2026.

[58] C. Vanaret and A. Montoison. Uno: a composable framework for nonlinearly constrained optimization. J. Open Source Softw., 11(123):10229, 2026.

[59] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program., 106:25–57, 2006.

[60] R. Waltz, J. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. Math. Program., 107(3):391–408, 2006.

[61] R. Wilson. A simplicial algorithm for concave programming. PhD thesis, Graduate School of Business Administration, Harvard University, 1963.