

ISE



Industrial and
Systems Engineering

NonOpt: Nonconvex, Nonsmooth Optimizer

FRANK E. CURTIS AND LARA ZEBIANE

Department of Industrial and Systems Engineering, Lehigh University

COR@L Technical Report 25T-005



LEHIGH
UNIVERSITY.

COR@L
COMPUTATIONAL OPTIMIZATION
RESEARCH AT LEHIGH 

NonOpt: Nonconvex, Nonsmooth Optimizer ^{*}

FRANK E. CURTIS^{†1} AND LARA ZEBIANE^{‡1}

¹Department of Industrial and Systems Engineering, Lehigh University

March 28, 2025

Abstract

NonOpt, a C++ software package for minimizing locally Lipschitz objective functions, is presented. The software is intended primarily for minimizing objective functions that are nonconvex and/or nonsmooth. The package has implementations of two main algorithmic strategies: a gradient-sampling and a proximal-bundle method. Each algorithmic strategy can employ quasi-Newton techniques for accelerating convergence in practice. The main computational cost in each iteration is solving a subproblem with a quadratic objective function, a linear equality constraint, and bound constraints. The software contains dual active-set and interior-point subproblem solvers that are designed specifically for solving these subproblems efficiently. The results of numerical experiments with various test problems are provided to demonstrate the speed and reliability of the software.

1 Introduction

This paper provides an introduction to an open-source C++ software package known as NonOpt, which stands for *Nonconvex, Nonsmooth Optimizer* and is available at <https://github.com/frankecurtis/NonOpt>. This software has implementations of a suite of algorithms for solving continuous optimization problems when the objective function is nonconvex and/or nonsmooth. The implemented algorithms are applicable for solving unconstrained problems, which means they could also be employed effectively as subproblem solvers within various algorithmic approaches for solving other types of optimization problems, say, involving discrete decision variables or constraints. The main algorithmic strategies that are implemented in the NonOpt software are a gradient-sampling method and a proximal-bundle method. The software comes equipped with options for employing quasi-Newton techniques—specifically, Davidon-Fletcher-Powell (DFP) [13] and Broyden-Fletcher-Goldfarb-Shanno (BFGS) techniques [2, 16, 17, 35]—within the algorithms. Convergence guarantees can be maintained with such techniques, and importantly their use significantly improves practical performance.

NonOpt builds from earlier Matlab implementations of algorithms that were written and employed for the numerical experiments in [9, 10, 11]. An early C++ implementation of the software (known as SVANO) was written and employed for the experiments in [12]. Further improvements to the software led to the implementation (referred to as NonOpt) written and employed for the experiments in [7]. The present paper provides a thorough description of an updated version of the NonOpt software (version 2.0), which has not appeared previously. It also highlights a significant new feature to the software, namely, an interior-point method based on the strategy in [8] for solving the quadratic optimization (QP) subproblems that arise in gradient-sampling and proximal-bundle methods. The overall algorithmic strategies can make use

^{*}Supported by the U.S. National Science Foundation under award number CCF-2139735 and by the Office of Naval Research under award number N00014-24-1-2703.

[†]E-mail: frank.e.curtis@lehigh.edu

[‡]E-mail: lara.zebiane@lehigh.edu

of inexact subproblem solutions obtained from these solvers by following the procedures in [7], which is particularly effective when solving large-scale problems and the new interior-point subproblem solver is employed. This new interior-point subproblem solver complements a previously implemented dual active-set method. Combined, with these two strategies for solving the arising QP subproblems, the software can solve both small- and large-scale instances efficiently. The numerical experiments that are presented in this paper demonstrate the practical performance of the gradient-sampling and proximal-bundle methods that are implemented in the NonOpt software.

An important feature of the open-source NonOpt software is that it has been written to be extensible, allowing a user to include other search-direction computation schemes, globalization mechanisms, or (quasi-)Newton-based strategies.

1.1 Organization

A formal statement of the optimization problems that the algorithms implemented in NonOpt are designed to solve, as well as preliminary mathematical background on stationarity conditions and theoretical guarantees for the implemented algorithms, are provided in §2. A high-level description of the implemented algorithmic framework, which includes both gradient-sampling and proximal-bundle strategies as special cases, is presented in §3. This section also provides a detailed account of the modules in the code through which one distinguishes between different search-direction computation strategies (e.g., gradient-sampling versus proximal-bundle), globalization mechanisms, quasi-Newton strategies, and others. A complete description of the newly implemented interior-point method for solving the arising QP subproblems is presented in §4. This method has been tailored specifically for the form of the QPs that arise in NonOpt algorithms. Further details of the software implementation are presented in §5 and the results of extensive numerical experiments are presented in §6. Concluding remarks are provided in §7.

2 Problem Formulation and Preliminaries

NonOpt is designed to solve optimization problems of the form

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where the objective function f satisfies the following assumption.

Assumption 2.1 *The objective function $f : \mathbb{R}^n \rightarrow (\mathbb{R} \cup \{\infty\})$ has an effective domain*

$$\text{dom}(f) := \{x \in \mathbb{R}^n : f(x) < \infty\}$$

that is nonempty with a nonempty interior, i.e., $\text{int}(\text{dom}(f)) \neq \emptyset$. In addition, f is locally Lipschitz over $\text{int}(\text{dom}(f))$ in the sense that for every $x \in \text{int}(\text{dom}(f))$ there exist positive real numbers $\epsilon_{f,x} \in \mathbb{R}_{>0}$ and $L_{f,x} \in \mathbb{R}_{>0}$ such that, for all $(\bar{x}, \hat{x}) \in \mathbb{R}^n \times \mathbb{R}^n$ with $\|\bar{x} - x\|_2 \leq \epsilon_{f,x}$ and $\|\hat{x} - x\|_2 \leq \epsilon_{f,x}$, one has that

$$|f(\bar{x}) - f(\hat{x})| \leq L_{f,x} \|\bar{x} - \hat{x}\|_2.$$

It is well known that, by Rademacher's theorem [31], the objective function f being locally Lipschitz over $\text{int}(\text{dom}(f))$ means that it is continuous over $\text{int}(\text{dom}(f))$ and differentiable almost everywhere in $\text{int}(\text{dom}(f))$ in the sense that the subset of points in $\text{int}(\text{dom}(f))$ at which it is not differentiable has a Lebesgue measure of zero [6]. Under Assumption 2.1, the Clarke generalized directional derivative function with respect to f , defined for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^n$ by

$$f^\circ(x, d) = \limsup_{\substack{\bar{x} \rightarrow x \\ \alpha \searrow 0}} \frac{f(\bar{x} + \alpha d) - f(\bar{x})}{\alpha},$$

defines for any $x \in \text{int}(\text{dom}(f))$ a mapping $f^\circ(x, \cdot) : \mathbb{R}^n \rightarrow \mathbb{R}$ that is finite-valued and sublinear with $|f^\circ(x, d)| \leq L_{f,x} \|d\|_2$ for all $d \in \mathbb{R}^n$, where $L_{f,x} \in \mathbb{R}_{>0}$ is a Lipschitz constant for f near x . In addition, for any $x \in \mathbb{R}^n$, the set of Clarke generalized gradients for the objective function f at x is given by

$$\begin{aligned} \partial f(x) &:= \{g \in \mathbb{R}^n : f^\circ(x, d) \geq g^T d \text{ for all } d \in \mathbb{R}^n\} \\ &= \text{conv}(\{g \in \mathbb{R}^n : \{\nabla f(x_k)\} \rightarrow g \text{ for some } \{x_k\} \rightarrow x \text{ with } \{x_k\} \subset \mathcal{D}_{\nabla f}\}), \end{aligned}$$

where $\mathcal{D}_{\nabla f} \subseteq \text{int}(\text{dom}(f))$ is the full-measure subset over which f is differentiable. The latter equation, which states that the Clarke generalized gradient of f at x is equal to the convex hull of limits of gradient sequences, is well known; see [6, 32].

The algorithms that are implemented in NonOpt are well defined under the relatively loose Assumption 2.1 as long as an initial iterate x_1 is provided that lies in $\text{int}(\text{dom}(f))$. However, it should be mentioned that to attain the theoretical convergence guarantees offered for NonOpt's implemented gradient-sampling or proximal-bundle method, the objective function should satisfy additional assumptions as well. First, the objective function should in fact be real-valued over \mathbb{R}^n . NonOpt has safeguards to ensure that no iterate (after x_1) is generated at which the objective function is infinite-valued, although technically speaking the implemented algorithms might not be guaranteed to have convergence guarantees when f can take infinite values. Second, the objective function should satisfy at least one of two additional assumptions depending on whether the gradient-sampling or proximal-bundle method is employed. Specifically, to be consistent with the convergence guarantees offered in, e.g., [3, 4, 21] and/or [27, 34], problem (1) should satisfy the following assumption in addition to Assumption 2.1.

Assumption 2.2 *The (real-valued) objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ satisfies at least one of the following two conditions:*

- (a) *f is continuously differentiable on an open set $\mathcal{D}_{\nabla f}$ with full measure over \mathbb{R}^n ;*
- (b) *f is weakly upper semismooth over \mathbb{R}^n in the sense that, for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^n$, the directional derivative of f at x along d , namely,*

$$f'(x, d) := \lim_{\alpha \searrow 0} \frac{f(x + \alpha d) - f(x)}{\alpha}$$

exists and for all sequences $\{(\alpha_k, g_k)\} \subset \mathbb{R}_{>0} \times \mathbb{R}^n$ satisfying $\{\alpha_k\} \searrow 0$ and $g_k \in \partial f(x + \alpha_k d)$ for all $k \in \mathbb{N}$ the directional derivative satisfies

$$\liminf_{k \rightarrow \infty} g_k^T d \geq f'(x, d).$$

Assumption 2.2(a) ensures that convergence guarantees for the implemented gradient-sampling methods hold. On the other hand, Assumption 2.2(b) ensures guarantees for the implemented proximal-bundle methods.

Given subroutines for evaluating the objective f and an element of $\partial f(x)$ for any $x \in \text{int}(\text{dom}(f))$ in addition to an initial iterate $x_1 \in \text{int}(\text{dom}(f))$, NonOpt generates an iterate sequence $\{x_k\} \subset \text{dom}(f)$ that is intended to converge to stationarity with respect to f . That is, at the very least, the software aims to generate such a sequence $\{x_k\}$ with a limit point \bar{x} satisfying $0 \in \partial f(\bar{x})$.

3 Algorithmic Framework

In this section we provide a high-level description of the algorithmic framework for solving instances of problem (1) that is implemented in NonOpt. The framework encapsulates various instances of gradient-sampling and proximal-bundle methods, where to employ each particular instance a user can select between multiple options related to the line search, quasi-Newton strategy, and other aspects. We begin this section with a mathematical description of the main components of the framework, then discuss the modules of the code that allow a user to switch between different algorithmic strategies within the gradient-sampling and proximal-bundle methods.

3.1 Main algorithmic components

At initialization with a user-provided initial point $x_1 \in \mathbb{R}^n$, the objective-function value $f_1 := f(x_1)$, a generalized gradient $g_1 \in \partial f(x_1)$, and a symmetric positive definite matrix H_1 with inverse $W_1 := H_1^{-1}$ are all used to compute a search direction. More generally, at the beginning of iteration $k \in \mathbb{N}$ and with the notation $[m] := \{1, \dots, m\}$ for any $m \in \mathbb{N}$, the algorithm has accumulated and/or will generate a set of points $\{x_{k,j}\}_{j \in [m]} \subset \mathbb{R}^n$, real numbers $\{f_{k,j}\}_{j \in [m]} \subset \mathbb{R}$, and real vectors $\{g_{k,j}\}_{j \in [m]} \subset \mathbb{R}^n$ that are used along with symmetric positive definite H_k and/or $W_k := H_k^{-1}$ to compute the search direction $d_k \in \mathbb{R}^n$. In a gradient-sampling method, the set of points $\{x_{k,j}\}_{j \in [m]} \subset \mathbb{R}^n$ corresponds to the current iterate x_k , previous iterates and sample points that are contained within a neighborhood of the current iterate, and newly generated sample points within the same neighborhood; the set of real numbers $\{f_{k,j}\}_{j \in [m]} \subset \mathbb{R}^n$ has each element equal to the current objective function value $f(x_k)$; and the real vectors $\{g_{k,j}\}_{j \in [m]} \subset \mathbb{R}^n$ correspond to gradients of f at the points in $\{x_{k,j}\}_{j \in [m]}$. (The strategies for maintaining and generating these points follow the strategies proposed for gradient-sampling methods in [3, 4, 10, 11, 12, 21].) In a proximal-bundle method, the set of points $\{x_{k,j}\}_{j \in [m]}$ contains the current iterate x_k and (a subset of) prior iterates, the set of real numbers $\{f_{k,j}\}_{j \in [m]} \subset \mathbb{R}$ correspond to the objective-function values at these points (potentially modified through *downshifting* with the α -function for an objective function stated on page 81 of [27]), and the set of real vectors correspond to generalized gradients of f at the points $\{x_{k,j}\}_{j \in [m]}$. (The strategy for maintaining and generating these points follows [27, 34].)

Regardless of the source of $\{(x_{k,j}, f_{k,j}, g_{k,j})\}_{j \in [m]}$, the search direction computation employs the piecewise linear model $l_{k,m} : \mathbb{R}^n \rightarrow \mathbb{R}$ defined by

$$l_{k,m}(x) = \max_{j \in [m]} \{f_{k,j} + g_{k,j}^T(x - x_{k,j})\}.$$

Given $H_k \succ 0$, a corresponding piecewise quadratic model is $q_{k,m} : \mathbb{R}^n \rightarrow \mathbb{R}$ with

$$q_{k,m}(x) = l_{k,m}(x) + \frac{1}{2}(x - x_k)^T H_k(x - x_k).$$

A step toward minimizing f is given by the minimizer of $q_{k,m}$ within a trust region with norm $\|\cdot\|_\infty$ and radius $\delta_k \in \mathbb{R}_{>0}$, which leads to the subproblem

$$\min_{x \in \mathbb{R}^n} q_{k,m}(x) \quad \text{s.t.} \quad x \in \mathcal{X}_k := \{x \in \mathbb{R}^n : \|x - x_k\|_\infty \leq \delta_k\}.$$

This subproblem has a convex feasible region and a strongly convex objective function, so it has a unique optimal solution. (The trust-region radius is always chosen to be finite, although depending on the choices of a couple of critical input parameters, it is unlikely to be active at the optimal solution; see §5.) With the change of variables $d \equiv x - x_k$, an equivalent smooth subproblem is

$$\begin{aligned} \min_{(d,z) \in \mathbb{R}^n \times \mathbb{R}} \quad & z + \frac{1}{2}d^T H_k d \\ \text{s.t.} \quad & x_k + d \in \mathcal{X}_k \quad \text{and} \quad f_{k,j} + g_{k,j}^T(x_k + d - x_{k,j}) \leq z \quad \text{for all } j \in [m]. \end{aligned} \tag{2}$$

Let us call this the primal subproblem. Rather than solve it directly, NonOpt has implemented solvers (see §4) for the corresponding dual subproblem given by

$$\begin{aligned} \max_{(\omega, \gamma) \in \mathbb{R}^m \times \mathbb{R}^n} \quad & -\frac{1}{2}(G_k \omega + \gamma)^T W_k (G_k \omega + \gamma) + b_k^T \omega - \delta_k \|\gamma\|_1 \\ \text{s.t.} \quad & \mathbf{1}^T \omega = 1 \quad \text{and} \quad \omega \geq 0, \end{aligned} \tag{3}$$

where $G_k := [g_{k,1} \ \dots \ g_{k,m}]$, $\mathbf{1}$ (here and throughout the paper) is a vector of ones of appropriate length, and, for all $j \in [m]$, the j th element of b_k is

$$b_{k,j} = f_{k,j} + g_{k,j}^T(x_k - x_{k,j}); \tag{4}$$

see [12]. There is not necessarily a unique solution of (3) due, e.g., to the fact that the columns of G_k might not be linearly independent. That being said, any optimal solution (ω_k, γ_k) of (3) yields the same value of $d_k \leftarrow -W_k(G_k\omega_k + \gamma_k)$, which is part of the unique solution (d_k, z_k) of the primal subproblem.

Upon computation of the search direction d_k , the NonOpt framework conducts a line search to compute the subsequent iterate x_{k+1} . The line search at least imposes on the computed step size $\alpha_k \in \mathbb{R}_{>0}$ the sufficient decrease condition

$$f(x_{k+1}) - f(x_k) \leq -\frac{1}{2}c\alpha_k(G_k\omega_k + \gamma_k)^T W_k(G_k\omega_k + \gamma_k) = -\frac{1}{2}c\alpha_k\|d_k\|_{H_k}^2 \quad (5)$$

for some $c \in (0, 1)$. There is also an option for asking the line-search mechanism to attempt to satisfy a curvature condition as well, as in the Armijo-Wolfe-type line-search schemes proposed in [23, 27]. In a gradient-sampling method, in order to ensure theoretical convergence guarantees, the line search technically needs to require that f is differentiable at x_{k+1} . However, this condition is ignored in the software so that the user does not need to provide a mechanism for checking differentiability of f . It is assumed that, at such a point of nondifferentiability, a request for a gradient of f returns a generalized gradient of f at the point.

The other main components of the framework in each iteration $k \in \mathbb{N}$ are to select the *point set* that defines $\{(x_{k+1,j}, f_{k+1,j}, g_{k+1,j})\}$ and to update a quasi-Newton matrix. The NonOpt software is set up to allow other mechanisms for selecting the point set, although at this time the only mechanism that is implemented is to remove points that either (a) fall outside of a ball centered at x_{k+1} or (b) are the “oldest” when the cardinality of the point set would otherwise exceed a prescribed limit. The latter possibility is straightforward to implement, so we do not provide the details here. As for (a), it is also straightforward: Defining

$$\mathbb{B}_{\leq \epsilon_k}^n(x_k) := \{x \in \mathbb{R}^n : \|x - x_k\|_2 \leq \epsilon_k\}, \quad (6)$$

the strategy removes an element $(\bar{x}, \bar{f}, \bar{g})$ from the point set if $\bar{x} \notin \mathbb{B}_{\leq \epsilon_{k+1}}^n(x_{k+1})$. As for the quasi-Newton updating mechanism, we follow the strategy proposed in [12]. For concreteness, we discuss here the strategy in terms of BFGS updating; the strategy for DFP updating is similar [28]. First, in order to ensure the self-correcting properties of BFGS updating, we employ the *damping* strategy from [12], which involves replacing $y_{k+1} \leftarrow g_{k+1} - g_k$ by a convex combination, call it v_k , of $s_k \leftarrow \alpha_k d_k$ and y_k such that the pair of inequalities

$$\eta \leq \frac{s_k^T v_k}{\|s_k\|_2^2} \quad \text{and} \quad \frac{\|v_k\|_2^2}{s_k^T v_k} \leq \psi \quad (7)$$

holds for user-defined $\eta \in \mathbb{R}_{>0}$ and $\psi \in [\eta, \infty)$. Then, depending on whether a *full-memory* or *limited-memory* [29] option is chosen, the algorithm either sets

$$H_{k+1} \leftarrow \left(I - \frac{s_k s_k^T H_k}{s_k^T H_k s_k} \right)^T H_k \left(I - \frac{s_k s_k^T H_k}{s_k^T H_k s_k} \right) + \frac{v_k v_k^T}{s_k^T v_k} \quad (8a)$$

$$\text{and } W_{k+1} \leftarrow \left(I - \frac{v_k s_k^T}{s_k^T v_k} \right)^T W_k \left(I - \frac{v_k s_k^T}{s_k^T v_k} \right) + \frac{s_k s_k^T}{s_k^T v_k} \quad (8b)$$

or updates a set of pairs $\{(s_j, v_j)\}$ that is used to define H_{k+1} and W_{k+1} implicitly in the subsequent iteration, as in a standard L-BFGS approach [5, 24, 28].

A statement of the framework that is implemented in NonOpt is provided as Algorithm 1 below. The software involves numerous other input parameters, all of which are documented in the code itself. The statement of Algorithm 1 includes only a few parameters that, in our experience, are the most important ones in terms of influencing the practical performance of the implemented algorithms. See the next subsection and §5 for further details on the parameter choices.

3.2 Modules defining key algorithmic strategies

The source code for NonOpt involves low-level and high-level components. By low-level components, we are referring to classes for defining and manipulating objects such as vectors and symmetric matrices, which rely

Algorithm 1 NonOpt Algorithm Framework

Require: $c \in (0, 1)$, $\eta \in \mathbb{R}_{>0}$, and $\psi \in [\eta, \infty)$; an initial point $x_1 \in \mathbb{R}^n$; radii $(\epsilon_1, \delta_1) \in \mathbb{R}_{>0} \times \mathbb{R}_{>0}$; an integer $p \in \mathbb{N}$; and symmetric positive definite $H_1 \in \mathbb{R}^{n \times n}$ with $W_1 \leftarrow H_1^{-1}$

1: set $\mathcal{B}_1 \leftarrow \{(x_1, f(x_1), g_1)\}$ for some $g_1 \in \bar{\partial}f(x_1)$

2: **for all** $k \in \mathbb{N}$ **do**

3: (optional) for each $j \in [p]$, sample \bar{x} from a uniform distribution over $\mathbb{B}_{\leq \epsilon_k}^n(x_k)$ and

$$\mathcal{B}_k \leftarrow \mathcal{B}_k \cup \{(\bar{x}, f(\bar{x}), \bar{g})\} \text{ for some } \bar{g} \in \bar{\partial}f(\bar{x})$$

4: letting $\{(x_{k,j}, f_{k,j}, g_{k,j})\}_{j \in [m]}$, G_k , and b_k be defined by \mathcal{B}_k , solve (3) for (ω_k, γ_k)

5: set $d_k \leftarrow -W_k(G_k \omega_k + \gamma_k)$

6: compute α_k to satisfy (5) by a backtracking Armijo or Armijo-Wolfe line search

7: set $x_{k+1} \leftarrow x_k + \alpha_k d_k$ and $s_k \leftarrow \alpha_k d_k$

8: set $\mathcal{B}_{k+1} \leftarrow \{(x_{k+1}, f(x_{k+1}), g_{k+1})\}$ for some $g_{k+1} \in \bar{\partial}f(x_{k+1})$

9: set $\epsilon_{k+1} \in (0, \epsilon_k]$ and $\delta_{k+1} \in \mathbb{R}_{>0}$

10: (optional) remove $(\bar{x}, \bar{f}, \bar{g})$ from \mathcal{B}_{k+1} such that $\bar{x} \notin \mathbb{B}_{\leq \epsilon_{k+1}}(x_{k+1})$

11: (optional) remove from \mathcal{B}_{k+1} its “oldest” elements until $|\mathcal{B}_{k+1}|$ is below a limit

12: set $y_k \leftarrow g_{k+1} - g_k$ and compute β_k as the smallest value in $[0, 1]$ such that

$$v_k \leftarrow \beta_k s_k + (1 - \beta_k) y_k$$

13: yields (7), then set (H_{k+1}, W_{k+1}) by (8) (or update a set of pairs $\{(s_j, v_j)\}$)

14: **end for**

on BLAS [22, 15, 14] and LAPACK [1] routines, as well as classes for defining and manipulating options (i.e., user-modifiable input parameters) and output streams. The low-level components also include container classes for storing iterate information and other quantities that are passed and/or referenced throughout the software. We do not discuss these low-level components in any further detail in this paper.

The high-level components of NonOpt relate to the main algorithmic strategies. These are managed through a `Strategies` object through which a user can select different options as building blocks for an algorithm instance. The following is a list of the strategies in the code along with corresponding options/parameters that, in our experience, have the greatest effect on the performance of the software.

- `approximate_hessian_update`: Strategy for updating the sequences of Hessian approximations $\{H_k\}$ and inverse Hessian approximations $\{W_k\}$ that appear in the subproblems (see (2)–(3)) for computing search directions; current options are BFGS (default) and DFP. These employ standard procedures—see, e.g., [28]—as well as the damping strategy described in [12]. Information is stored for maintaining both a sequence of Hessian approximations and the corresponding sequence of inverse Hessian approximations. (The exact type of information that is stored depends on the `SymmetricMatrix` strategy, described later on.) The most influential options related to this strategy are the following.
 - `BFGS_correction_threshold_1` and `DFP_correction_threshold_1` for BFGS and DFP updates, respectively. These correspond to the parameter η in (7). The default value is 10^{-8} for both options.
 - `BFGS_correction_threshold_2` and `DFP_correction_threshold_2` for BFGS and DFP updates, respectively. These correspond to the parameter ψ in (7). The default value is 10^8 for both options.
- `derivative_checker`: Strategy for checking user-supplied derivatives; current option is only to compare derivatives to finite-difference approximations. (The user should be aware of the fact that such comparisons may be faulty when a given point is one of nondifferentiability of the objective function.) The derivative-checker is off by default, but can be turned on for the purpose of debugging a user-provided problem instance. The most influential option related to this strategy is the following.

- `DEFD_increment` is the increment or spacing used to compute the finite-difference approximation of the partial derivative of the objective function with respect to each coordinate. The default value is 10^{-8} .
- `direction_computation`: Strategy for building a quadratic optimization (QP) subproblem to be solved for computing the search direction in each iteration; current options are gradient, gradient combination (as in a gradient-sampling method), and cutting plane (default, as in a proximal-bundle method). The latter two strategies have been mentioned previously in this section. The former (gradient) strategy constructs the subproblem using only the current iterate, objective function value, and generalized gradient value, as in a method for smooth optimization; it offers no convergence guarantees in general. (The resulting search direction still employs a Hessian approximation, so this gradient strategy could be used, e.g., to employ a straightforward BFGS-based algorithm.) The most influential options related to this strategy are the following.
- `DCGC_try_aggregation` and `DCCP_try_aggregation` for the gradient combination and cutting plane techniques, respectively. These options determine whether a gradient aggregation strategy is employed, as explained in [7]. The default value is `false` for both options, but setting it to `true` can lead to significantly improved performance when solving some problems.
 - `DCGC_try_gradient_step` and `DCCP_try_gradient_step` for the gradient combination and cutting plane techniques, respectively. These options determine whether a gradient-descent-type step (scaled by W_k , namely, $-W_k g_k$) is considered as the step before the full subproblem solution (see (3)) is computed. The default value is `true` for both options.
 - `DCGC_try_shortened_step` and `DCCP_try_shortened_step` for the gradient combination and cutting plane techniques, respectively. These options determine whether a shortened version of the step computed from a subproblem is considered before the data defining the subproblem is augmented to lead to an improved search direction. The default value is `true` for both options.
- `line_search`: Strategy for computing a step size along each search direction; current options are an Armijo/weak-Wolfe line search (default) or backtracking Armijo line search. By default, each subproblem is defined with a trust-region constraint, although through user-modifiable options a user can make the trust-region more likely or not to influence the search direction. (By default, the parameters that influence the trust-region radii are such that the trust region is rarely active in our numerical experiments.) Overall, through user-modifiable parameters, a user can invoke various combinations of line-search and trust-region strategies. The most influential options are the following.
- `LSWW_stepsize_initial`, `LSWW_stepsize_sufficient_decrease_threshold`, and `LSWW_stepsize_curvature_threshold` for the Armijo/weak-Wolfe line search. These correspond to the initial step size, sufficient decrease parameter, and curvature parameter. The default values are 1, 10^{-10} , and 0.9, respectively.
 - `LSB_stepsize_initial` and `LSB_stepsize_sufficient_decrease_threshold` for the backtracking line search scheme. These correspond to the initial step size and sufficient decrease parameter. The default values are 1 and 10^{-10} .
- `point_set_update`: Strategy for updating the set of points (with corresponding function and/or generalized-gradient values) that may be used to construct subproblems in subsequent iterations; current option is only to update set based on proximity (i.e., distance) to the current iterate. The most influential options related to this strategy are the following.
- `PSP_envelope_factor` is the factor—i.e., multiple of ϵ_k —used to define the neighborhood (see (6)) outside of which points are removed from the point set. The default value is 10^2 .
 - `PSP_size_factor` is the factor—i.e., multiple of n —used to define the cardinality limit for the point set. If this limit is exceeded, then old points are removed. The default value is 5×10^{-2} .

- `qp_solver_small_scale` and `qp_solver_large_scale`: Strategies for solving the arising QP subproblems, at least approximately according to the termination tests derived from [7]; current options for each are a dual active-set method (preferable for small-scale problems) and an interior-point method (preferable for large-scale problems). In each iteration, whether the small-scale or large-scale QP solver is employed is determined by the direction computation strategy. (The gradient direction computation strategy always employs the small-scale solver. The gradient combination and cutting plane strategies choose the solver in iteration k based on the number of columns of the matrix G_k .) The newly implemented interior-point solver is presented in detail in the next section. The most influential options related to this strategy are discussed in §5.
- `symmetric_matrix`: Strategy for storing the aforementioned (inverse) Hessian approximation; current options are as a dense matrix (default, based on all prior pairs for quasi-Newton updating) or as a limited-memory matrix (involving only a prescribed number of recent pairs for quasi-Newton updating). The most influential option related to this strategy is the following.
 - `SMLM_history` for the limited-memory technique is the typical history length for limited-memory quasi-Newton techniques. The default value is 20.
- `termination`: Strategy for terminating the overall algorithm that aims to solve problem (1); current options are a basic strategy (default) based on the norm of the most recently computed search direction or a more expensive strategy that involves computing the minimum-norm element of the convex hull of stored gradients evaluated at points near the current iterate. The most influential options related to this strategy are the following.
 - `TB_objective_similarity_tolerance` for the basic termination strategy (and `TS_objective_similarity_tolerance` for the other strategy) determines a relative tolerance for changes in the objective function value. If the objective function value does not change more than this threshold for a sequence of consecutive iterations, then the sampling radius is reduced or the algorithm terminates due to lack of further progress. The default value is 10^{-5} .

4 Subproblem Solvers

Solving subproblem (3) is the most computationally expensive aspect of each iteration of Algorithm 1, at least without considering the computational expense of each (user-defined) function and generalized gradient evaluation. For solving this subproblem in each iteration, a dual active-set algorithm was implemented previously within NonOpt (version 1.0). That dual active-set algorithm is based on the one proposed in [20], although a major difference is that the method in NonOpt employs a more general distance metric, namely, that defined by the matrix W_k in (3). (The method proposed in [20] only considers $W_k = I$.) The implementation of this dual active-set algorithm was used for the experiments in [7, 12].

Like other active-set algorithms for solving quadratic optimization problems, a drawback of this dual active-set approach is that it may require a large number of iterations to reach an exact solution or even an inexact solution that is close to the exact solution in some sense. A recent major addition to the NonOpt software is an interior-point algorithm for solving the arising subproblems. As shown in our numerical experiments in §6, the interior-point algorithm reduces CPU time significantly compared to the dual active-set method when solving large-scale problems.

We now provide a detailed description of the interior-point algorithm implemented in NonOpt for solving subproblems of the form (3). This subproblem solver is based on the approach proposed in [8] along with a Mehrotra predictor-corrector strategy [26]. It has also been tailored explicitly for solving (3). First, due to the nonsmooth ℓ_1 -norm term in the objective function of (3), we split the variable γ into a pair of nonnegative variables that correspond to the positive and negative parts of γ , call them σ and ρ , respectively. The solution of the resulting subproblem can be obtained by solving the minimization problem

$$\min_{\theta \in \mathbb{R}^t} \frac{1}{2} \theta^T Q \theta + c^T \theta \quad \text{s.t.} \quad A \theta = b \quad \text{and} \quad \theta \geq 0, \quad (9)$$

where $\ell := m + 2n$ and the problem variables and data are given respectively by

$$\theta \equiv \begin{bmatrix} \omega \\ \sigma \\ \rho \end{bmatrix}, \quad Q = \begin{bmatrix} G_k^T W_k G_k & G_k^T W_k & -G_k^T W_k \\ W_k G_k & W_k & -W_k \\ -W_k G_k & -W_k & W_k \end{bmatrix}, \quad c = \begin{bmatrix} -b_k \\ \delta_k \mathbf{1} \\ \delta_k \mathbf{1} \end{bmatrix}, \quad A = \begin{bmatrix} \mathbf{1} \\ 0 \\ 0 \end{bmatrix}^T, \quad \text{and } b = 1.$$

Here, for the sake of notational simplicity, we drop the subscript k in (9). For our subsequent discussions in this section, the iterates of our implemented interior-point method for solving (9) will use iteration index j , which the reader should not confuse with the meaning of the index j in, e.g., (4). Let us also note that if the trust region is inactive in the optimal solution of (2), then the solution of (3) is guaranteed to have $\gamma_k = 0$, in which case the solution of (9) is guaranteed to have $\rho = \sigma = 0$. Thus, in this case, the variables (ρ, σ) do not even need to be introduced and (9) only needs to be defined and solved over the variable ω . The remainder of our description of our interior-point method is effectively the same in either case, so for the sake of generality we only present our method for the case when the trust region constraint might be active. In §5, we discuss our approach for trying to avoid the introduction of these additional variables in practice.

Problem (9) is a convex QP, although as previously mentioned it might not have a unique solution. Indeed, the matrix Q is not positive definite. (One straightforward setting in which the solution is not unique is when the matrix G_k contains two columns that are equal to each other and the corresponding components of b_k are also equal.) The KKT conditions for problem (9) are

$$A\theta = b, \tag{10a}$$

$$A^T u + v - Q\theta - c = 0, \tag{10b}$$

$$\Theta v = 0, \tag{10c}$$

$$\text{and } (\theta, v) \geq 0, \tag{10d}$$

where $(u, v) \in \mathbb{R} \times \mathbb{R}^\ell$ are Lagrange multipliers and $\Theta \equiv \text{diag}(\theta)$. For future use, we also introduce the notation $V \equiv \text{diag}(v)$ for the nonnegativity multipliers.

The KKT conditions are difficult to solve directly due to the complementarity conditions represented by (10c)–(10d). Thus, as in a standard primal-dual interior-point method, rather than solve (10) directly, we introduce a centering term that, when employed, promotes the computation of steps toward or along the *central path*. This path, parameterized by the *barrier parameter* $\mu \in (0, \infty)$, is defined by the set of points satisfying (10a), (10b), and (10d) along with $\Theta v = \mu \mathbf{1}$.

Given a solution estimate (θ_j, u_j, v_j) , our method commences by computing a *predictor* step corresponding to $\mu = 0$ as a solution of the linear system

$$\begin{bmatrix} -Q & A^T & I \\ A & 0 & 0 \\ V_j & 0 & \Theta_j \end{bmatrix} \begin{bmatrix} \Delta\theta_j^{\text{pred}} \\ \Delta u_j^{\text{pred}} \\ \Delta v_j^{\text{pred}} \end{bmatrix} = \begin{bmatrix} Q\theta_j + c - A^T u_j - v_j \\ b - A\theta_j \\ -\Theta_j v_j \end{bmatrix} =: \begin{bmatrix} r_j^d \\ r_j^p \\ r_j^c \end{bmatrix}. \tag{11}$$

The residual vectors r^p and r^d correspond to *primal* and *dual* infeasibility, respectively, whereas the residual vector r^c corresponds to the *complementarity* condition. The linear system (11) can be solved directly, or it can be solved by first solving

$$\left(\underbrace{\begin{bmatrix} -Q & A^T \\ A & 0 \end{bmatrix}}_{\text{constant w.r.t. } j} + \begin{bmatrix} -\Theta_j^{-1} V_j & 0 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \Delta\theta_j^{\text{pred}} \\ \Delta u_j^{\text{pred}} \end{bmatrix} = \begin{bmatrix} r_j^d - \Theta_j^{-1} r_j^c \\ r_j^p \end{bmatrix}, \tag{12}$$

then setting the remaining solution component as $\Delta v_j^{\text{pred}} \leftarrow \Theta_j^{-1}(r_j^c - V_j \Delta\theta_j^{\text{pred}})$. Once the predictor step $(\Delta\theta_j^{\text{pred}}, \Delta u_j^{\text{pred}}, \Delta v_j^{\text{pred}})$ has been computed, our method next computes step sizes $(\alpha_{\theta,j}^{\text{pred}}, \alpha_{u,j}^{\text{pred}}, \alpha_{v,j}^{\text{pred}})$

following a strategy based on that in [8], which is discussed further in the next paragraph. Then, given a current barrier parameter value $\mu_j \leftarrow \theta_j^T v_j / \ell$ along with the predictor step and its corresponding step sizes, the method next computes, as in a standard Mehrotra predictor-corrector strategy [26], the scaling factor $\zeta_j \in \mathbb{R}_{>0}$ by

$$\zeta_j \leftarrow \left(\frac{1}{\mu_j} \cdot \frac{(\theta_j + \alpha_{\theta,j}^{\text{pred}} \Delta \theta_j^{\text{pred}})^T (v_j + \alpha_{v,j}^{\text{pred}} \Delta v_j^{\text{pred}})}{\ell} \right)^3. \quad (13)$$

(In our implementation, we safeguard this choice by setting a larger value for ζ_j , if necessary, to ensure that $\zeta_j \mu_j \geq 10^{-12}$.) The method next computes a *corrector* step based on this scaling factor. Equivalently, the method computes the full search direction (i.e., the predictor step plus the corrector step) by solving

$$\begin{bmatrix} -Q & A^T & I \\ A & 0 & 0 \\ V_j & 0 & \Theta_j \end{bmatrix} \begin{bmatrix} \Delta \theta_j \\ \Delta u_j \\ \Delta v_j \end{bmatrix} = \begin{bmatrix} Q\theta_j + c - A^T u_j - v_j \\ b - A\theta_j \\ -\Theta_j v_j + \zeta_j \mu_j \mathbf{1} \end{bmatrix} =: \begin{bmatrix} r_j^d \\ r_j^p \\ r_j^c + \zeta_j \mu_j \mathbf{1} \end{bmatrix}. \quad (14)$$

Like for the predictor step, the linear system (14) can be solved directly, or it can be solved by employing a reduced linear system similar to (12). In any case, our implementation exploits the fact that the predictor and corrector steps are computed through linear systems involving the same matrix, which allows our implementation to re-use the LDL factorization from the computation of the predictor step in order to compute the corrector step. Once the full step has been computed, the method computes corresponding step sizes $(\alpha_{\theta,j}, \alpha_{u,j}, \alpha_{v,j})$, again using the method proposed in [8] (see the next paragraph). Finally, the solution estimate is updated and the method proceeds to the next iteration.

The search-direction-computation strategy described in the previous paragraph has been employed to great effect for solving linear and quadratic optimization problems with interior-point methods. That said, a critical practical aspect of our proposed method is the manner in which step sizes are chosen. For this purpose, we have found impressive performance to be offered by the strategy proposed in [8], tailored for our setting. This approach starts by choosing maximum step sizes such that a *fraction-to-the-boundary* rule is enforced; see line 2 in Algorithm 3. Afterwards, separate primal and dual step sizes are computed in order to optimize the overall reduction in the merit function $\phi : \mathbb{R}^\ell \times \mathbb{R} \times \mathbb{R}^\ell$ defined by

$$\phi(\theta, u, v) = \|A\theta - b\|_2^2 + \|A^T u + v - Q\theta - c\|_2^2 + \theta^T v.$$

This is done by searching along a two-segment path from $(0,0)$ to the maximum step sizes allowed by the fraction-to-the-boundary rule. We refer the reader to [8] for further details. For our purposes here, we simply state that the desired step sizes can be computed by solving two 2-dimensional convex quadratic optimization problems, the necessary computations for which are negligible.

A complete description of our entire interior-point method is stated as Algorithm 2 along with the step-size-selection subroutine stated in Algorithm 3. Besides tailoring the aforementioned strategies and linear algebra subroutines to the specific formulation in (3), the initial point provided to Algorithm 2 has been tuned carefully to obtain improved practical performance. We discuss this initialization further in the next section along with other implementation details of NonOpt.

5 Implementation Details

The performance of a gradient-based algorithm for solving *smooth* nonconvex continuous optimization problems is sensitive to input parameter choices in general. The performance of an algorithm for solving *non-smooth* nonconvex continuous optimization problems is arguably even more sensitive. We have certainly found this to be the case for NonOpt. Fortunately, based on its default input parameter settings, most of the behaviors of the implemented algorithms in NonOpt seem to be similar to that of an unadulterated quasi-Newton scheme with a weak-Wolfe line search, which is generally quite effective even when solving locally Lipschitz problems that involve nonconvexity and/or nonsmoothness [23]. This can be seen in our

Algorithm 2 Interior Point Algorithm for Solving Problem (9)

```

1: choose initial point  $(\theta_1, u_1, v_1) \in \mathbb{R}_{>0}^\ell \times \mathbb{R} \times \mathbb{R}_{>0}^\ell$  and termination tolerance  $\epsilon \in \mathbb{R}_{>0}$ 
2: for all  $j \in \mathbb{N}$  do
3:   set  $(r_j^p, r_j^d, r_j^c)$  as defined in (11)
4:   if  $\max\{\|r_j^p\|_\infty, \|r_j^d\|_\infty, \|r_j^c\|_\infty\} \leq \epsilon$  then
5:     terminate and return  $(\theta_j, u_j, v_j)$ 
6:   end if
7:   compute  $(\Delta\theta_j^{\text{pred}}, \Delta u_j^{\text{pred}}, \Delta v_j^{\text{pred}})$  by solving the linear system in (11)
8:   compute  $(\alpha_{\theta,j}^{\text{pred}}, \alpha_{u,j}^{\text{pred}}, \alpha_{v,j}^{\text{pred}})$  by Algorithm 3 for  $(\Delta\theta_j^{\text{pred}}, \Delta u_j^{\text{pred}}, \Delta v_j^{\text{pred}})$ 
9:   set  $\mu_j \leftarrow \theta_j^T v_j / \ell$ 
10:  set  $\zeta_j$  by (13)
11:  compute  $(\Delta\theta_j, \Delta u_j, \Delta v_j)$  by solving the linear system in (14)
12:  compute  $(\alpha_{\theta,j}, \alpha_{u,j}, \alpha_{v,j})$  by Algorithm 3 for  $(\Delta\theta_j, \Delta u_j, \Delta v_j)$ 
13:  set  $\theta_{j+1} \leftarrow \theta_j + \alpha_{\theta,j} \Delta\theta_j$ ,  $u_{j+1} \leftarrow u_j + \alpha_{u,j} \Delta u_j$ , and  $v_{j+1} \leftarrow v_j + \alpha_{v,j} \Delta v_j$ 
14: end for

```

experiments in the next section, where one finds (see §6.2–§6.3 in particular) that the gradient-based direction computation strategy (indicated by \mathbb{G}) generally performs quite well. (However, it is also clear from our experiments in the next section that, for certain problems, the gradient-based strategy is inferior to the gradient-sampling and/or proximal-bundle strategy.) NonOpt employs the default input parameters of $c \leftarrow 10^{-10}$, $\eta \leftarrow 10^{-8}$, and $\psi \leftarrow 10^8$ so that, for the most part, an employed algorithm follows such an unadulterated line-search-BFGS scheme. That being said, if a user wants to push an algorithm to obtain a higher-accuracy solution when nonsmoothness is present, then one might consider alternative values of these inputs (e.g., setting $(\eta, \psi) \leftarrow (10^{-2}, 10^2)$) to force each local model of the objective (recall (2)) to rely more heavily on the information in the point set $\{(x_{k,j}, f_{k,j}, g_{k,j})\}_{j \in [m]}$ than the information embedded in the Hessian approximation H_k . The user should understand, however, that such a choice often leads to the algorithm requiring a much higher computational cost per iteration. This, at least, is true based on our observations that such a choice leads to shorter steps, more accumulation of points in the point sets, and thus higher subproblem costs.

Any user of NonOpt should also be made aware that the strategy for choosing $\epsilon_1 \in \mathbb{R}_{>0}$ and setting the remaining elements of the generated sequence $\{\epsilon_k\}$ play an absolutely critical role in the behavior of the software. To describe this strategy, let us begin by saying that NonOpt commences any solve by scaling the objective function in order to ensure that $\|g_1\|_\infty$ is at most 10^2 . Letting \tilde{f} denote the pre-scaled objective function and $\tilde{g}_1 \in \partial f(x_1)$, this is accomplished by setting the objective function to be considered in the remainder of the solve as

$$f \leftarrow \text{scale} \cdot \tilde{f}, \quad \text{where } \text{scale} \leftarrow \min\{1, 10^2 \| \tilde{g}_1 \|_\infty^{-1}\}.$$

After this scaling, by default, the software sets $\epsilon_1 \leftarrow \max\{10^{-2}, 10^{-1} \|g_1\|_\infty\}$. As for setting $\epsilon_{k+1} \in (0, \epsilon_k]$ for all $k \in \mathbb{N}$, the ideal case is for the software to set $\epsilon_{k+1} < \epsilon_k$ if and only if (see Theorems 3.3–3.4 in [12])

$$\max\{\|d_k\|_\infty, \|G_k \omega_k\|_\infty, \|G_k \omega_k + \gamma_k\|_\infty\} \leq \epsilon_k.$$

If and when this condition is satisfied, the software sets $\epsilon_{k+1} \leftarrow 10^{-1} \epsilon_k$. However, for practical purposes, the software will also set $\epsilon_{k+1} \leftarrow 10^{-1} \epsilon_k$ whenever recent improvements in the objective function are small. We discuss this further in §6.2.

The trust region in the primal subproblem (2) can be an effective tool for obtaining a high-accuracy solution. Indeed, setting $\{\delta_k\}$ to relatively small values has a similar effect as choosing η and ψ each closer to 1, as discussed earlier in this section. However, a disadvantage of setting δ_k too small is that it causes the subproblem solve to be more expensive in general, which can be especially detrimental when solving large-scale problems. Therefore, by default, the software sets $\delta_1 \leftarrow \max\{10^{-1}, 10^{10} \|g_1\|_\infty\}$, then, for each $k \in \mathbb{N}$,

Algorithm 3 Step-size computation subroutine for Algorithm 2

Require: search direction, call it $(\Delta\theta_j, \Delta u_j, \Delta v_j)$

1: get current iterate (θ_j, u_j, v_j) and residuals (r_j^p, r_j^d) from Algorithm 2

2: set $\bar{\alpha}_{\theta,j} \leftarrow \left[\max_{i \in [\ell]} \left\{ 1, -\frac{\Delta\theta_j^{(i)}}{\beta\theta_j^{(i)}} \right\} \right]^{-1}$ and $\bar{\alpha}_{v,j} \leftarrow \left[\max_{i \in [\ell]} \left\{ 1, -\frac{\Delta v_j^{(i)}}{\beta v_j^{(i)}} \right\} \right]^{-1}$

3: set $\bar{\alpha}_j \leftarrow \min\{\bar{\alpha}_{\theta,j}, \bar{\alpha}_{v,j}\}$

4: set

$$r_j \leftarrow \begin{bmatrix} r_j^p \\ r_j^d \end{bmatrix}, \quad s_j \leftarrow \begin{bmatrix} -A \\ Q \end{bmatrix} \Delta\theta_j, \quad o_j \leftarrow \begin{bmatrix} 0 \\ -A^T \end{bmatrix} \Delta u_j, \quad \text{and } p_j \leftarrow \begin{bmatrix} 0 \\ -I \end{bmatrix} \Delta v_j$$

5: compute $(\alpha_{\theta,j,1}, \alpha_{u,j,1})$ by solving the two-dimensional convex QP given by

$$\begin{aligned} \min_{(\alpha_\theta, \alpha_u) \in \mathbb{R} \times \mathbb{R}} \quad & \frac{1}{2} \begin{bmatrix} \alpha_\theta \\ \alpha_u \end{bmatrix} \begin{bmatrix} (s_j + p_j)^T (s_j + p_j) + \Delta\theta_j^T \Delta v_j & (s_j + p_j)^T o_j \\ (s_j + p_j)^T o_j & o_j^T o_j \end{bmatrix} \begin{bmatrix} \alpha_\theta \\ \alpha_u \end{bmatrix} \\ & + \begin{bmatrix} r_j^T (s_j + p_j) + \frac{1}{2}(\Delta\theta_j^T v_j + \theta_j^T \Delta v_j) \\ r_j^T o_j \end{bmatrix}^T \begin{bmatrix} \alpha_\theta \\ \alpha_u \end{bmatrix} \\ \text{s.t.} \quad & 0 \leq \alpha_\theta \leq \bar{\alpha}_j \end{aligned}$$

6: set $\alpha_{v,j,1} \leftarrow \alpha_{\theta,j,1}$

7: **if** $\bar{\alpha}_{\theta,j} \leq \bar{\alpha}_{v,j}$ **then**

8: set $\alpha_{\theta,j,2} \leftarrow \bar{\alpha}_{\theta,j}$

9: compute $(\alpha_{u,j,2}, \alpha_{v,j,2})$ by solving the two-dimensional convex QP given by

$$\begin{aligned} \min_{(\alpha_u, \alpha_v) \in \mathbb{R} \times \mathbb{R}} \quad & \frac{1}{2} \begin{bmatrix} \alpha_u \\ \alpha_v \end{bmatrix} \begin{bmatrix} o_j^T o_j & o_j^T p_j \\ o_j^T p_j & p_j^T p_j \end{bmatrix} \begin{bmatrix} \alpha_u \\ \alpha_v \end{bmatrix} \\ & + \begin{bmatrix} r_j^T o_j + \bar{\alpha}_{\theta,j} s_j^T o_j \\ r_j^T p_j + \frac{1}{2}\theta_j^T \Delta v_j + \bar{\alpha}_{\theta,j}(s_j^T p_j + \frac{1}{2}\Delta\theta_j^T \Delta v_j) \end{bmatrix}^T \begin{bmatrix} \alpha_u \\ \alpha_v \end{bmatrix} \\ \text{s.t.} \quad & \bar{\alpha}_j \leq \alpha_v \leq \bar{\alpha}_{v,j} \end{aligned}$$

10: **else**

11: compute $(\alpha_{\theta,j,2}, \alpha_{u,j,2})$ by solving the two-dimensional convex QP given by

$$\begin{aligned} \min_{(\alpha_\theta, \alpha_u) \in \mathbb{R} \times \mathbb{R}} \quad & \frac{1}{2} \begin{bmatrix} \alpha_\theta \\ \alpha_u \end{bmatrix} \begin{bmatrix} s_j^T s_j & s_j^T o_j \\ s_j^T o_j & o_j^T o_j \end{bmatrix} \begin{bmatrix} \alpha_\theta \\ \alpha_u \end{bmatrix} \\ & + \begin{bmatrix} r_j^T s_j + \frac{1}{2}\Delta\theta_j^T v_j + \bar{\alpha}_{v,j}(s_j^T p_j + \frac{1}{2}\Delta\theta_j^T \Delta v_j) \\ r_j^T o_j + \bar{\alpha}_{v,j} o_j^T p_j \end{bmatrix}^T \begin{bmatrix} \alpha_\theta \\ \alpha_u \end{bmatrix} \\ \text{s.t.} \quad & \bar{\alpha}_j \leq \alpha_\theta \leq \bar{\alpha}_{\theta,j} \end{aligned}$$

12: set $\alpha_{v,j,2} \leftarrow \bar{\alpha}_{v,j}$

13: **end if**

14: set $\chi \leftarrow \arg \min_{\chi \in \{1,2\}} \phi(\theta_j + \alpha_{\theta,j,\chi} \Delta\theta_j, u_j + \alpha_{u,j,\chi} \Delta u_j, v_j + \alpha_{v,j,\chi} \Delta v_j)$

15: set $\alpha_{\theta,j} \leftarrow \alpha_{\theta,j,\chi}$, $\alpha_{u,j} \leftarrow \alpha_{u,j,\chi}$, and $\alpha_{v,j} \leftarrow \alpha_{v,j,\chi}$

16: **return** $(\alpha_{\theta,j}, \alpha_{u,j}, \alpha_{v,j})$

sets $\delta_{k+1} \leftarrow \delta_k$ unless $\epsilon_{k+1} < \epsilon_k$, in which case it sets $\delta_{k+1} \leftarrow 10^{-1}\delta_k$. These default settings make it unlikely for the trust-region constraint ever to be active. Consequently, for the implemented interior-point solver, the software first attempts to solve (9) without introducing the variables (σ, ρ) . If the resulting solution $\tilde{\omega}$ yields $\|W_k G_k \tilde{\omega}\|_\infty \leq \delta_k$, then the software sets $(\omega_k, \sigma_k, \rho_k) \leftarrow (\tilde{\omega}, 0, 0)$. Otherwise, it re-solves (9) with all variables in order to compute $(\omega_k, \sigma_k, \rho_k)$. If n is very large, then this latter occurrence can unfortunately lead to a very expensive subproblem solve.

As previously mentioned, the subproblem solves are the most computationally expensive aspect of each iteration of the algorithms in NonOpt, except possibly for the function and generalized gradient evaluations. This means, in turn, both the initialization strategies for the subproblem solvers and the linear algebra subroutines employed in the solvers are critically important. The dual active-set subproblem solver for (3) initializes its solution estimate to have $\omega_j = 1$ for some $j \in [m]$ and $\omega_j = 0$ for all other elements of $[m]$, as well as $\gamma = 0$. The index j is chosen as the index yielding the smallest value of $g_{k,j}^T W_k g_{k,j}$ over $j \in [m]$. The interior-point solver is initialized with $\omega_j \leftarrow 1/m$ for all $j \in [m]$, so that it is also initialized with a feasible solution for (3). Moreover, with respect to $\mu_0 \leftarrow 5 \times 10^{-1}$ (a value chosen based on extensive experimentation), it initializes the components of v corresponding to ω , say, represented by a vector v^ω , as μ/ω_j for all $j \in [m]$. This ensures that the initial point is complementary in the sense that $\omega_j v_j^\omega = \mu$ for all $j \in [m]$. As for (σ, ρ) , if they are introduced, their components are initialized based on the values of $W_k G_k \omega$ for the initialized ω . If $[W_k G_k \omega]_i < -\delta_k$, then

$$\sigma_i \leftarrow \max\{10^{-1}, \delta_k - [W_k G_k \omega]_i\}, \rho_i \leftarrow 10^{-1}, v_i^\sigma \leftarrow \mu_0/\sigma_i, \text{ and } v_i^\rho \leftarrow \mu_0/\rho_i,$$

whereas if $[W_k G_k \omega]_i > \delta_k$, then

$$\sigma_i \leftarrow 10^{-1}, \rho_i \leftarrow \max\{10^{-1}, -\delta_k - [W_k G_k \omega]_i\}, v_i^\sigma \leftarrow \mu_0/\sigma_i, \text{ and } v_i^\rho \leftarrow \mu_0/\rho_i.$$

The use of the value 10^{-1} (chosen based on extensive experimentation) ensures that the initial point is an interior point. Careful selection of the initial point led to a significant reduction in CPU time for our experiments in the next section. We regularly saw that the number of iterations was around 25% of those required when the solve was conducted without a carefully selected initial point.

The interior-point subproblem solver in NonOpt relies on the LAPACK subroutines `dsysv` (to factor the matrix for computing the predictor step) and `dsytrs` (for re-using the factorization to compute the full interior-point step). The dual active-set subproblem solver generally computes and updates its own Cholesky factorizations based on iterative additions and deletions; see [20]. In such cases, linear systems are solved using LAPACK's `dtrsv` subroutine. However, if an update of a factorization through an addition is suspected to lead to numerical error, then a Cholesky factorization of a matrix is computed from scratch using LAPACK's `dpotf2` subroutine. An ambitious user could replace these subroutines with ones, say, from the Harwell Subroutine Library¹ or the Pardiso Solver Project.²

6 Numerical Experiments

We ran four sets of experiments to demonstrate various features of the NonOpt software as well as its overall effectiveness in solving challenging locally Lipschitz minimization problems. Previous articles about the developments of algorithmic strategies that have led to the NonOpt software considered comparisons with other software packages, such as LMBM [19]. However, since NonOpt performed favorably in these comparisons, LMBM is no longer under active development, and we are not aware of any other open-source software packages under active development for general-purpose locally Lipschitz minimization, our experiments in this paper focus exclusively on experiments with NonOpt only.

Our first set of experiments involves comparing the performance of the dual active-set QP subproblem solver versus the newly implemented interior-point QP subproblem solver (described in detail in §4). These experiments show that the interior-point QP solver is a very valuable addition to the software.

¹<https://www.hsl.rl.ac.uk/>

²<https://panua.ch/pardiso/>

Our next two sets of experiments aim to demonstrate the versatility of the NonOpt software when solving a diverse set of locally Lipschitz test problems. The two sets of experiments can be motivated as follows. First, in certain settings, a user may desire for the software to be run for speed, say, to obtain a decent solution relatively quickly. Therefore, we have one set of experiments where the user-defined parameters lead to relatively fast termination. On the other hand, in other settings, a user may desire for the software to be run for accuracy, say, to obtain a higher-quality solution even if the computational time required is significantly higher. Therefore, we have another set of experiments (on the same set of test problems as the previous set of experiments) where the user-defined parameters lead to longer solution times, but better final solutions.

Finally, we present the results of experiments with a few large-scale problems from the field of image processing. The purpose of these experiments is to show that NonOpt is capable of solving problems with hundreds-of-thousands of variables in a computationally efficient manner, and that it can transition seamlessly from solving convex to solving nonconvex locally Lipschitz minimization problems.

All of the experimental results that are provided in this section were obtained by running NonOpt version 2.0 on a Macbook Pro running Sequoia 15.3.1 with an Apple M1 Pro chip and 16GB of memory.

6.1 Experiments with the QP subproblem solvers

The purpose of our first set of experiments is to demonstrate the relative performance of the newly implemented interior-point QP subproblem solver as compared to the previously implemented dual active-set QP solver. Let us refer to these as the IPM and DAS solvers, respectively. The DAS solver is computationally efficient when solving relatively small-scale subproblems and in certain instances when a subproblem is solved that is closely related to the most recently solved subproblem, such as when the software has solved an instance of (3), then requests for a solution of a related instance for which the only differences are that the matrix G_k has been augmented with additional columns and, correspondingly, the vector b_k has been augmented with additional components. However, especially when solving large-scale subproblems, the IPM solver can be much faster, as is typical when one compares interior-point methods with active-set-type methods.

For these experiments, we randomly generated a large number of QP subproblems of increasing size. The generation of each QP proceeded as explained in the bullets below, which essentially amounts to generating values to satisfy the KKT conditions in (10) in order to correspond with a prescribed choice of the optimal solution of the primal subproblem (2). In the following description, we refer to the quantities that one needs to define for an instance of subproblem (3), but drop the iteration k index. The generation procedure for each instance is given problem size parameters $n \in \mathbb{N}$ and $m \in \mathbb{N}$. The values for these parameters that we considered are discussed after the following description of the generation procedure.

- First, $W \in \mathbb{R}^{n \times n}$ was set to the identity matrix. This choice is reasonable since, as far as the subproblem solvers are concerned, the positive-definite matrix W essentially only transforms the columns of G . We expect the relative performance of the subproblem solvers to be the same for other choices of W .
- Second, $\delta \in \mathbb{R}_{>0}$ was set to 1 and an option was set to decide whether the optimal solution of the primal QP, call it d_* , would have (a) $d_* = 0$, (b) $[d_*]_i = \delta$ for all $i \in [n/2]$ and $[d_*]_i = 0$ for all $i \in \{n/2 + 1, \dots, n\}$, or (c) $d_* = \delta \mathbf{1}$. (We generated an equal number of each of these types of QPs for each problem size.) Corresponding to these cases, an unconstrained minimizer of the objective function of (2), call it d_{unc} , was set to have (a) $d_{unc} = 0$, (b) $[d_{unc}]_i = 2\delta$ for all $i \in [n/2]$ and $[d_{unc}]_i = 0$ for all $i \in \{n/2 + 1, \dots, n\}$, or (c) $d_{unc} = 2\delta \mathbf{1}$.
- Third, $G \in \mathbb{R}^{n \times m}$ and $\omega_* \in \mathbb{R}^m$ were generated to satisfy $\mathbf{1}^T \omega_* = 1$, $\omega_* \geq 0$, and $G\omega_* = -d_{unc}$. This was done by first randomly generating $\hat{G} \in \mathbb{R}^{n \times (n-1)}$ with each component drawn from a standard normal distribution and randomly generating $\hat{\omega} \in \mathbb{R}^{n-1}$ with each component drawn from a uniform distribution over $[0, 1]$. Then, \hat{G} was padded with the additional column $-d_{unc} - \hat{G}\hat{\omega}$ and $\hat{\omega}$ was padded with an additional component equal to 1. The next step was to compute $\zeta \leftarrow \|\hat{\omega}\|_1$, normalize $\hat{\omega} \leftarrow \hat{\omega}/\zeta$ so that $\mathbf{1}^T \hat{\omega} = 1$, and correspondingly set $\hat{G} \leftarrow \zeta \hat{G}$. Finally, \hat{G} was padded with an additional

$m - n$ columns with components drawn from a standard normal distribution and $\hat{\omega}$ was padded with additional 0s. This procedure guarantees $\mathbf{1}^T \omega_* = 1$, $\omega_* \geq 0$, and $G\omega_* = -d_{unc}$.

- Fourth, $q \leftarrow -(d_* + WG\omega_*)$ was computed in order to set $\rho_* \leftarrow \max\{-q, 0\}$ and $\sigma_* \leftarrow \max\{q, 0\}$, where the max is taken component-wise in each case. (This guarantees that $d_* = -W(G\omega_* + \sigma_* - \rho_*)$, where $\sigma_* \geq 0$ and $\rho_* \geq 0$.)
- Fifth, u was set to an arbitrary number. (For our experiments in this section, without loss of generality we simply set $u \leftarrow 5$.)
- Sixth, all that remains is to set $b \in \mathbb{R}^m$. Toward this end, a vector $v_\omega \in \mathbb{R}^m$ was generated to have $[v_\omega]_i = 0$ for all $i \in [m]$ such that $[\omega_*]_i > 0$, and all other components drawn from a uniform distribution over $[0,1]$. Finally, $b \leftarrow G^T W(G\omega_* + \sigma_* - \rho_*) + u\mathbf{1} - v_\omega$.

One can verify that, with the procedure above, the generated QP is such that the corresponding unique optimal solution of (2) is the prescribed vector d_* . Overall, we generated 900 QP test problems. We generated equal numbers of problems for the values of $n \in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$, and for each n we generated equal numbers of problems for the values of $m \in \{n+1, 1.5n, 2n\}$. As previously mentioned, for each (n, m) we generated equal numbers of problems for the three prescribed choices of d_* , and for each setting of (n, m, d_*) we generated 10 different problems, each using a different seed for the random number generator. This led to the 900 different QP subproblems.

CPU time results for all runs combined are provided in Figure 1. As n increases, it is clear that IPM scales much better than DAS in terms of solution time. That said, the solution time for DAS is highly variable, so it is appropriate to parse the results further to understand if there are scenarios or settings in which DAS may be more competitive for IPM, even when n increases.

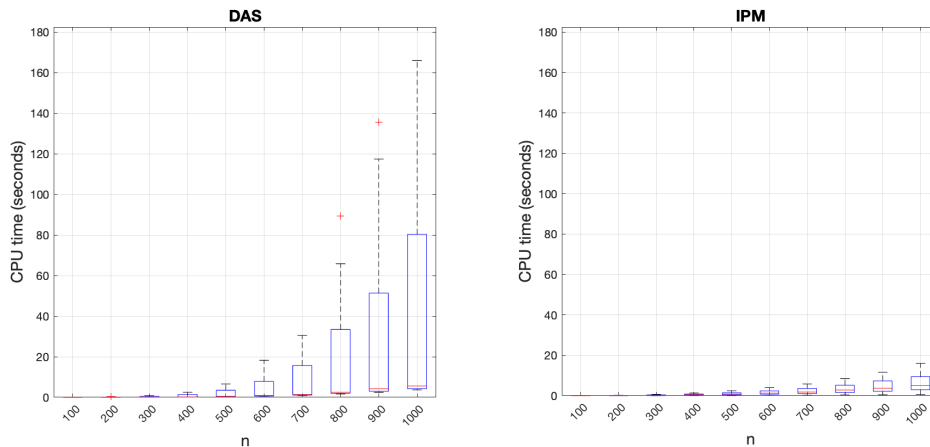


Figure 1: DAS vs. IPM, all problems

Figures 2–4 parse the results based on different d_* values. Through these results, one finds that DAS is more competitive with IPM when $d_* = 0$, which more generally suggests that DAS can be more competitive with IPM when the trust-region constraint is inactive. However, even in this setting, IPM scales better than DAS as n increases. When the trust-region constraint is active, the IPM solution times are typically much faster than the DAS solution times.

Figures 5–7 parse the results based on different m values (i.e., the number of columns of G), namely, $m \in \{n+1, 1.5n, 2n\}$. Here, one finds for $n = m+1$ that the DAS solution times are often competitive with the IPM solution times. However, for DAS there are outliers for which the solution times are much worse, whereas the solution times for IPM are consistently low, even as n increases. For $m \in \{1.5n, 2n\}$, the solution times for DAS are regularly much higher.

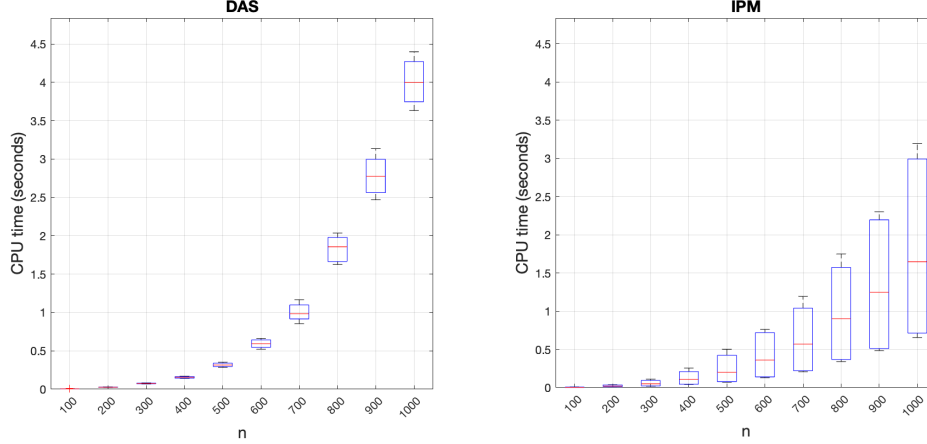


Figure 2: DAS vs. IPM, $d_* = 0$

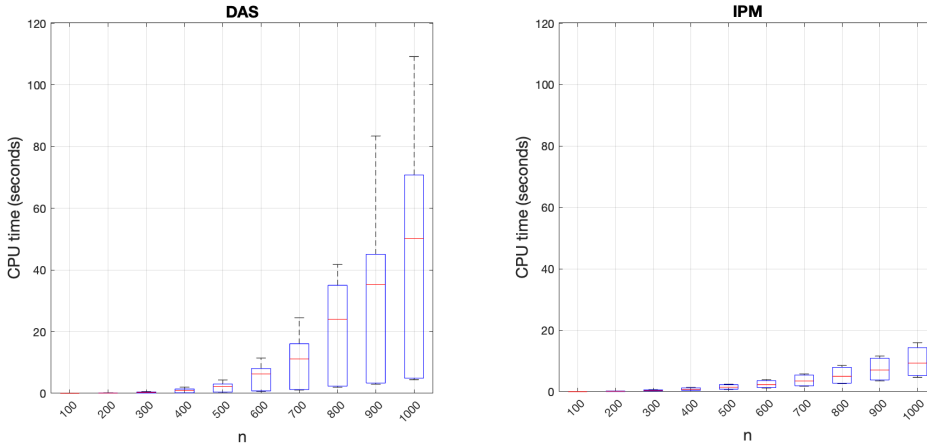


Figure 3: DAS vs. IPM, $[d_*]_i = \delta$ for all $i \in [n/2]$, 0 otherwise

One setting not covered in these experiments is $m \ll n$, which in fact is often the case for subproblems that arise in a run of the NonOpt software with its default settings. Our generation strategy for the QPs in this section—which is based on specifying a particular primal subproblem solution d_* —could be modified to allow for $m \ll n$, although the resulting problems would be more contrived since they would require more hand-picked values for ω_* as opposed to the randomly generated values that are allowed when $m \geq n$. That said, the results in the subsequent sections demonstrate the use of both DAS and IPM within the algorithms implemented in NonOpt. By default, the direction computation strategies in NonOpt employ DAS when the number of columns of G_k is less than or equal to 25, and otherwise IPM is employed. A user can change input options to use only DAS or only IPM, if desired. The threshold of 25 was chosen based on our experiments in the subsequent sections as a value that led to lower overall solution times.

6.2 Experiments designed for speed

Our second set of experiments is designed to demonstrate that with a fixed set of input options, the NonOpt software can generate good solutions of a diverse set of test problems in a computationally efficient manner. Perhaps the most challenging issue when it comes to the implementation of a general-purpose algorithm for locally Lipschitz minimization is selecting effective termination criteria. Theoretically, an ideal stationarity

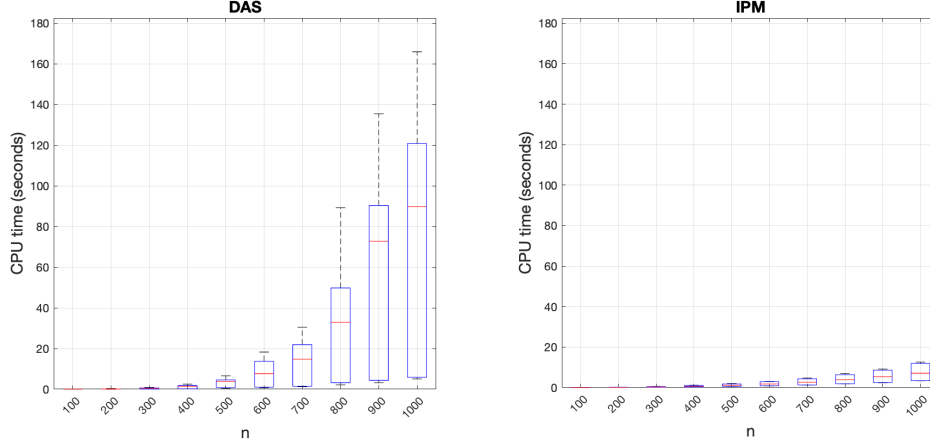


Figure 4: DAS vs. IPM, $d_* = \delta \mathbf{1}$

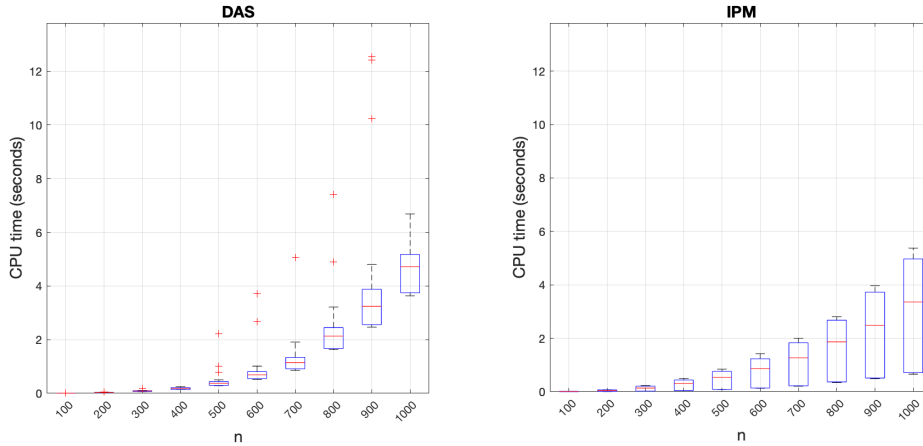


Figure 5: DAS vs. IPM, $m = n + 1$

condition would be to terminate an algorithm in iteration k if and only if $\text{dist}_{\partial f(x_k)}(0) \leq \epsilon$ for some termination tolerance $\epsilon \in \mathbb{R}_{>0}$. However, in general, it is not reasonable to be able to compute the entire set of generalized gradients $\partial f(x_k)$. NonOpt attempts to approximate the minimum-norm element of this set of generalized gradients through the solution ω_k of the dual subproblem (3). Indeed, recall that $\mathbf{1}^T \omega_k = 1$ and $\omega_k \geq 0$, which in turn means that $G_k \omega_k$ is a convex combination of gradients evaluated at points in a small neighborhood of x_k . Therefore, $G_k \omega_k$ can be viewed as an approximation of the minimum-norm element of $\partial f(x_k)$. This means that, in an idealized setting, the algorithm would terminate if and only if $\|G_k \omega_k\|$ is less than a prescribed tolerance.

Practically, however, such an idealized criterion is not reasonable if one wants the software to terminate without excessive computation. After all, to represent the minimum-norm element in the convex hull of a set in \mathbb{R}^n , one may require (by Caratheodory’s theorem) up to $n + 1$ gradients. This means that to enforce such a termination criterion, the algorithm may need to generate G_k with at least $n + 1$ columns. Then, computing the minimum-norm element of the convex hull of the columns of G_k would require a very expensive QP subproblem solve, even when n is only moderately large. Therefore, in order to terminate more quickly—at the possible risk of termination before a more accurate solution could be obtained—NonOpt employs termination conditions related to improvement in the objective function value over recent iterations. (Other software packages for locally Lipschitz minimization also rely heavily on such heuristic termination

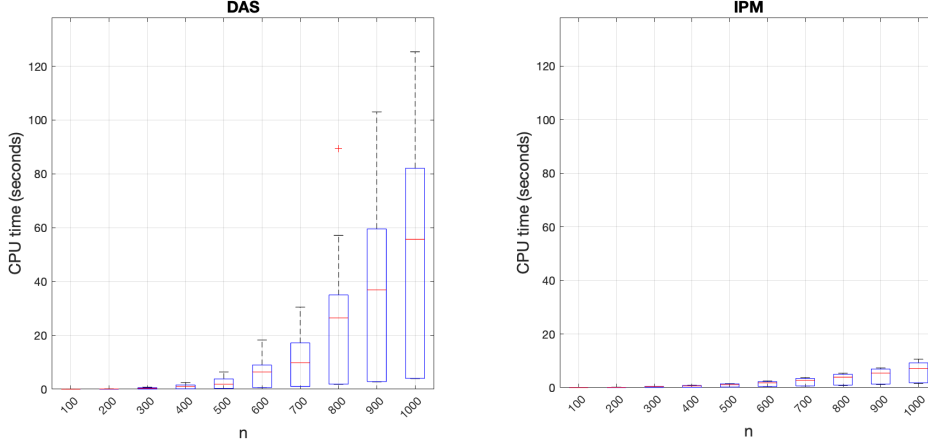


Figure 6: DAS vs. IPM, $m = 1.5n$

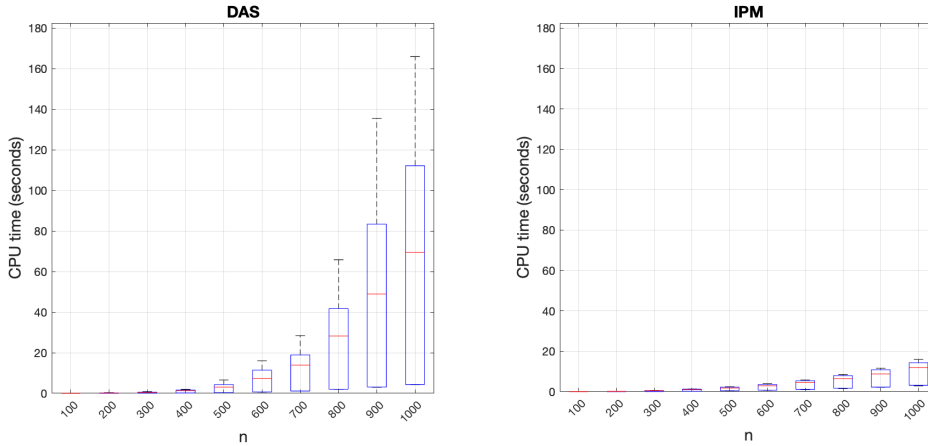


Figure 7: DAS vs. IPM, $m = 2n$

conditions, including LMBM [19] and Hanso [30]; see [11] for data related to this observation.) Specifically, NonOpt maintains a counter of how many of the latest iterations resulted in an objective function decrease that was below a prescribed threshold, call it $\Delta f \in \mathbb{R}_{>0}$. If this counter exceeds a prescribed threshold $n_f \in \mathbb{N}$ (or if $\|G_k \omega_k\|$ is below a threshold proportional to ϵ_k), then the algorithm sets ϵ_{k+1} to a fraction of ϵ_k in line 9 of Algorithm 1; otherwise, it sets $\epsilon_{k+1} \leftarrow \epsilon_k$. Once ϵ_k is below a prescribed threshold and such a condition holds, then the overall algorithm terminates. The parameters Δf and n_f can be controlled easily by a user.

For our experiments to demonstrate the speed of the NonOpt software with multiple direction computation strategies, we considered the default values for the parameters Δf and n_f , which are 10^{-5} and 10, respectively. (The default settings presume that a user would aim for speed when using NonOpt “out of the box.”) We considered a set of 20 test problems. The first ten problems come from [18] and the remainder come from [25]. See Table 1 for a list of the problem names and their corresponding minimal values of the objective function, where known. The problems are all scalable in the sense that they are defined for any $n \in \mathbb{N}$. For our experiments, we chose $n = 1000$ for all problems. An initial point x_0 is given with each problem definition in [18, 25]. We used these initial points in our experiments. Importantly, we note that the first 12 problems are all convex (i.e., all ten problems from [18] and the first two from [25]) and the remainder are nonconvex.

Table 1: Problem names, sizes, and minimal objective function values (f_*), where known.

name	n	f_*
ActiveFaces	1000	0
BrownFunction.2	1000	0
ChainedCB3.1	1000	$2(n - 1) = 1998$
ChainedCB3.2	1000	$2(n - 1) = 1998$
ChainedCrescent.1	1000	0
ChainedCrescent.2	1000	0
ChainedLQ	1000	$-\sqrt{2}(n - 1) \approx 1412.799$
ChainedMifflin.2	1000	-706.55
MaxQ	1000	0
MxHilb	1000	0
Test29.2	1000	0
Test29.5	1000	0
Test29.6	1000	-
Test29.11	1000	-
Test29.13	1000	-
Test29.17	1000	-
Test29.19	1000	-
Test29.20	1000	-
Test29.22	1000	-
Test29.24	1000	-

The results obtained from NonOpt using its three available direction computation strategies (see page 8) are presented in Table 2. The table shows the number of iterations, number of function evaluations, number of gradient evaluations, final objective value, and CPU time required (in seconds) for each run.

A couple of observations are worthwhile to make at this point. (For further observations, see the next subsection.) First, observe that, for the most part, the final objective values obtained using the three different direction computation strategies were relatively consistent. Second, observe that the final objective values obtained when solving the convex problems are typically close to their known optimal values; see Table 1. (For some problems, one might desire additional digits of accuracy, which motivates the experiments in the next subsection.) Third, observe that the time required for each run was at most a few seconds. The total time required to run this entire experiment—i.e., all runs of all problems—was approximately 85 seconds. We comment on this further in the next subsection.

6.3 Experiments designed for accuracy

We ran a second set of experiments using the same setup as in the prior subsection, but with tighter termination tolerances. There are various input parameters available to a user of the software that the user can adjust in order to try to obtain a more accurate solution estimate. For our purposes here, the only adjustment that we made was to set $\Delta f \leftarrow 10^{-8}$ and $n_f \leftarrow 20$ (as opposed to $\Delta f \leftarrow 10^{-5}$ and $n_f \leftarrow 10$ in the previous subsection). The results are provided in Table 3.

A few observations are worthwhile to make, especially when one compares the results in Table 3 to those in Table 2. First, it should be said that for some problems the gains in terms of final objective values are only modest despite the tighter tolerances and (in some cases significantly) increased computational effort. This suggests that the default settings of NonOpt—which are designed for speed—can typically yield good final objective values. Second, observe that in other cases the final solution quality improves somewhat significantly with the tighter tolerances. Third, observe that in some nonconvex instances the direction computation strategy GC appears to find a solution that is substantially better than even those obtained by the other strategies. With respect to this, we draw the reader’s attention to the results for `Test29_6`, `Test29_17`, and `Test29_19`. For these problems, strategy GC with the tighter tolerances led to final objective values that are improved by around an order of magnitude or more. All of the solution quality improvements in these experiments come at a cost, however. Whereas the total time required for the experiments in the previous subsection was around 85 seconds, the total time required for the experiments here was around 580 seconds.

6.4 Experiments on large-scale image-processing problems

Our final set of experiments is designed to demonstrate that NonOpt is able to solve large-scale problems efficiently, and that it can transition seamlessly from solving convex to nonconvex problems. For these

experiments, we considered a set of related image-denoising problems. Suppose that a grayscale image represented by a matrix $X_o \in [256]^{n_r} \times [256]^{n_c}$ has been corrupted by noise so that one only has access to a noisy grayscale image $\bar{X} \in [256]^{n_r} \times [256]^{n_c}$ with $\bar{X} \approx X_o$. (The presence of $[256]$ in these spaces is due to the fact that each pixel value is represented by an integer between 1 and 256. Moreover, $n_r \in \mathbb{N}$ is the number of rows in the image and $n_c \in \mathbb{N}$ is the number of columns.) The goal is to denoise the image by solving an optimization problem. To demonstrate such a procedure, the optimization problems that we consider for our experiments each have the form

$$\begin{aligned} \min_{X \in \mathbb{R}^{n_r \times n_c}} \quad & \sum_{i \in [n_r]} \sum_{j \in [n_c]} (X_{i,j} - \bar{X}_{i,j})^2 \\ & + \lambda \sum_{i \in [n_r-1]} \sum_{j \in [n_c-1]} \phi_\beta(X_{i+1,j} - X_{i,j}) + \phi_\beta(X_{i,j+1} - X_{i,j}), \end{aligned} \quad (15)$$

where $\lambda \in \mathbb{R}_{>0}$ is a regularization parameter and $\phi_\beta : \mathbb{R} \rightarrow \mathbb{R}$ is a regularization function (also known as a regularizer) that is parameterized by $\beta \in \mathbb{R}_{>0}$. (As is common in the literature on image denoising, problem (15) is defined over $\mathbb{R}^{n_r \times n_c}$, although to render the solution as an image we round each pixel value to the nearest integer in $[256]$.) Overall, we consider the following four choices of the regularizer:

- (a) absolute value: $\phi_\beta(t) = \beta|t|$;
- (b) logarithmic: $\phi_\beta(t) = \beta \log(1 + \beta|t|)$;
- (c) fractional: $\phi_\beta(t) = \frac{\beta|t|}{1 + \beta|t|}$; and
- (d) hard thresholding: $\phi_\beta(t) = \frac{\beta}{2} - \frac{\max\{0, \beta - |t|\}^2}{2\beta}$.

The absolute value regularizer can be derived as the anisotropic version of the well known total variation regularizer [33]. The use of this particular regularizer makes the objective of problem (15) convex. Each of the other three choices of the regularizer makes the objective of problem (15) nonconvex; see Figure 8.

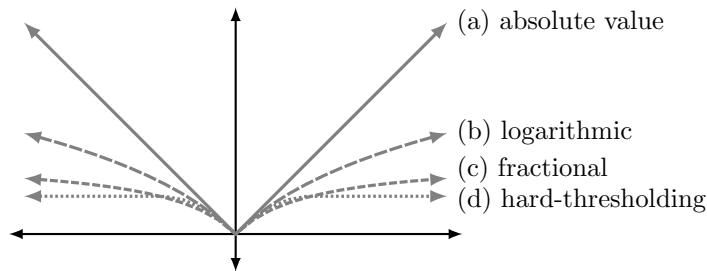


Figure 8: Regularizers for image denoising problem instances, each illustrated for $\beta = 1$.

We remark in passing that there exist numerous specialized algorithms for solving image denoising problems. The aim of our experiments here is not to attempt to compete with the state-of-the-art methods for solving such problems. Rather, our aim is to demonstrate that NonOpt can be employed to solve large-scale problem instances and to highlight the fact that NonOpt can be employed to solve problem (15) regardless of the choice of regularizer—convex or nonconvex.

We considered X_o as a 605-by-807 grayscale image, the original of which is shown on the left-hand side of Figure 9. We generated \bar{X} by adding “salt and pepper” noise to X_o using the default settings of `imnoise` in Matlab’s image processing toolbox. The resulting noisy image is shown on the right-hand side of Figure 9.

We employed NonOpt to solve instances of problem (15) using each of the four aforementioned regularizers. Given the size of the image that we considered, each instance had 488,235 variables. Due to the large scale of these problems, we used the limited memory `SymmetricMatrix` strategy, i.e., L-BFGS



Figure 9: Original (left) and noisy (right) image for denoising experiments.

for the Hessian approximations; see page 9. In order to present results for reasonable scenarios, we tuned $(\lambda, \beta) \in \mathbb{R}_{>0} \times \mathbb{R}_{>0}$ for each instance by searching over pairs of parameter values of the form $(2^i, 2^j)$ for all $(i, j) \in [-20, 25] \times [-20, 25]$ and, for each instance, choosing the values that led to the smallest mean-squared error (MSE) with the original image X_o . The resulting tuned input parameter values, MSE values, and solution time required for each instance are shown in Table 4. (For each instance, we take the final solution obtained by NonOpt, round each value to the nearest integer in $[256]$ to produce a matrix X_* , and compute the resulting MSE with X_o .) The resulting denoised images are shown in Figure 10. Overall, given the large scale of the problem instances, NonOpt is able to find good-quality solutions to both convex and nonconvex instances in relatively little time.

7 Conclusion

We have provided a thorough introduction to an open-source C++ software package for solving locally Lipschitz minimization problems. We have discussed the overall algorithmic framework that is implemented in the software, which includes both gradient-sampling and proximal-bundle methodologies, and provided an overview of the various modules in the software that allow a user to tailor the solver to their particular application(s). We have also discussed the details of a newly implemented interior-point subproblem solver that leads to significantly lower solution times for each subproblem and the overall NonOpt algorithmic strategy in general. The results of our numerical experiments have shown the benefits of the new QP subproblem solver, the speed and reliability of the software, and that NonOpt can solve large-scale (convex and nonconvex) problems efficiently.

References

- [1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
- [2] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6(1):76–90, 1970.
- [3] James V. Burke, Frank E. Curtis, Adrian S. Lewis, Michael L. Overton, and Lucas E. A. Simoes. Gradient Sampling Methods for Nonsmooth Optimization. In *Numerical Nonsmooth Optimization*, chapter 6, pages 201–225. Springer, 2020.

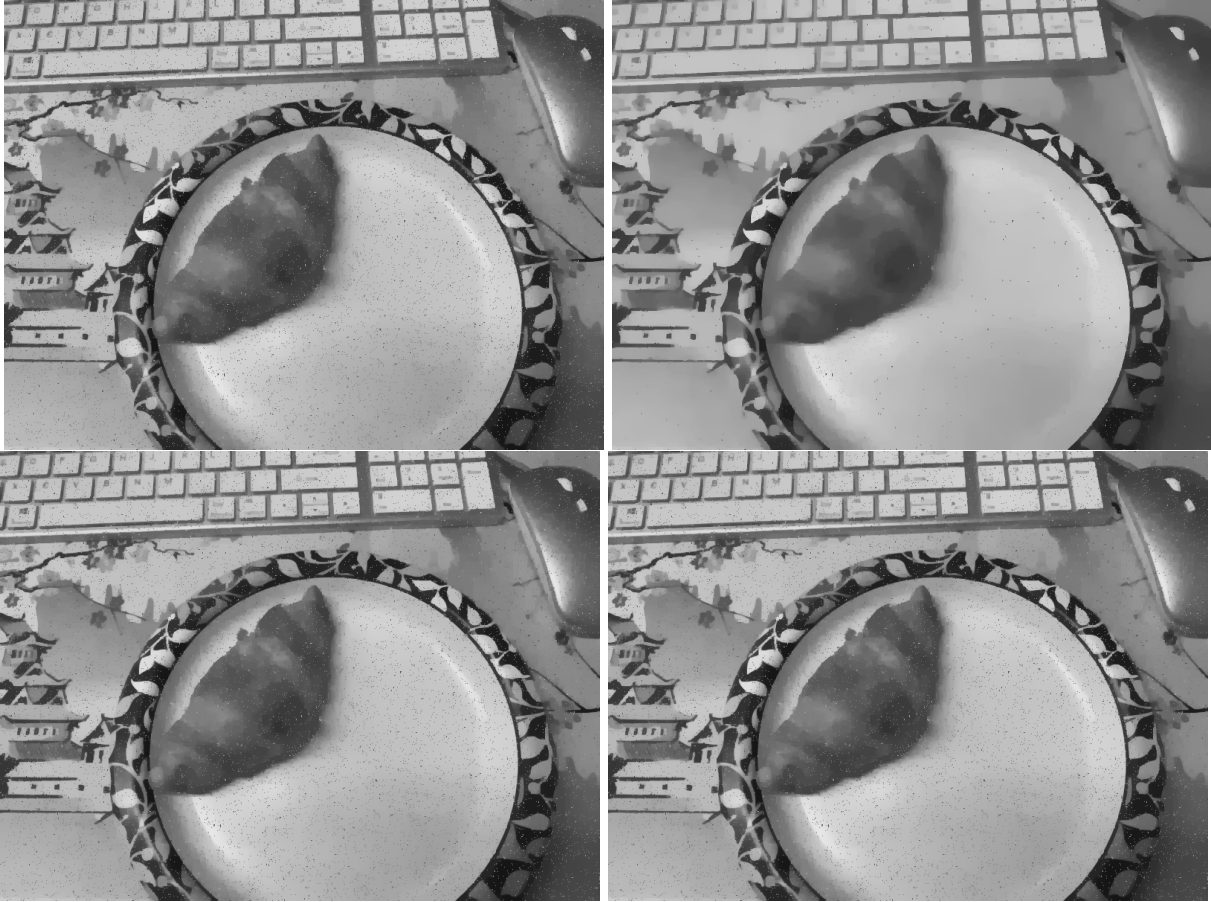


Figure 10: Recovered images obtained with regularizer (a) absolute value (top, left), (b) logarithmic (top, right), (c) fractional (bottom, left), and (d) hard thresholding (bottom, right).

- [4] James V. Burke, Adrian S. Lewis, and Michael L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization*, 15(3):751–779, January 2005.
- [5] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1-3):129–156, January 1994.
- [6] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Canadian Mathematical Society Series of Monographs and Advanced Texts. John Wiley & Sons, New York, NY, USA, 1983.
- [7] Frank E. Curtis and Minhan Li. Gradient Sampling Methods with Inexact Subproblem Solutions and Gradient Aggregation. *INFORMS Journal on Optimization*, 4(4):347–445, 2022.
- [8] Frank E. Curtis and Jorge Nocedal. Steplength Selection in Interior-Point Methods for Quadratic Programming. *Applied Mathematics Letters*, 20(5):516–523, 2007.
- [9] Frank E. Curtis and Michael L. Overton. A Sequential Quadratic Programming Algorithm for Nonconvex, Nonsmooth Constrained Optimization. *SIAM Journal on Optimization*, 22(2):474–500, 2012.
- [10] Frank E. Curtis and Xiaocun Que. An Adaptive Gradient Sampling Algorithm for Nonsmooth Optimization. *Optimization Methods and Software*, 28(6):1302–1324, 2013.

- [11] Frank E. Curtis and Xiaocun Que. A Quasi-Newton Algorithm for Nonconvex, Nonsmooth Optimization with Global Convergence Guarantees. *Mathematical Programming Computation*, 7(4):399–428, 2015.
- [12] Frank E. Curtis, Daniel P. Robinson, and Baoyu Zhou. A Self-Correcting Variable-Metric Algorithm Framework for Nonsmooth Optimization. *IMA Journal of Numerical Analysis*, 40(2):1154–1187, 2020.
- [13] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.
- [14] J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, mar 1990.
- [15] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14(1):1?17, mar 1988.
- [16] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.
- [17] D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.
- [18] N. Haarakala, K. Miettinen, and M. M. Mäkelä. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software*, 19(6):673–692, 2004.
- [19] Napsu Karmitsa. LMBM — FORTRAN Subroutines for Large-Scale Nonsmooth Minimization: User’s Manual. Technical Report TUCS Technical Report, No. 856, Turku Centre for Computer Science, Turku, 2007.
- [20] Krzysztof C. Kiwiel. A method for solving certain quadratic programming problems arising in nonsmooth optimization. *IMA Journal of Numerical Analysis*, 6(2):137–152, 1986.
- [21] Krzysztof C. Kiwiel. Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization*, 18(2):379–388, January 2007.
- [22] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308?323, sep 1979.
- [23] Adrian S. Lewis and Michael L. Overton. Nonsmooth optimization via quasi-Newton methods. *Mathematical Programming*, 141(1-2):135–163, October 2013.
- [24] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, August 1989.
- [25] L. Lukšan, M. Tůma, M. Šiška, J. Vlček, and N. Ramešová. UFO 2002: Interactive System for Universal Functional Optimization. Technical Report 883, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2002.
- [26] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.
- [27] Robert Mifflin. *A modification and an extension of Lemarechal’s algorithm for nonsmooth minimization*, pages 77–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982.
- [28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer New York, New York, NY, USA, Second edition, 2006.
- [29] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

- [30] M. L. Overton. HANSO: Hybrid Algorithm for Non-Smooth Optimization. <http://www.cs.nyu.edu/faculty/overton/software/hanso/>, 2010.
- [31] Hans Rademacher. Über partielle und totale differenzierbarkeit von Funktionen mehrerer Variablen und über die Transformation der Doppelintegrale. *Mathematische Annalen*, 79(4):340–359, December 1919.
- [32] R. Tyrrell Rockafellar and Roger J. B. Wets. *Variational Analysis*, volume 317 of *Grundlehren Der Mathematischen Wissenschaften*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
- [33] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, November 1992.
- [34] Helga Schramm and Jochem Zowe. A Version of the Bundle Idea for Minimizing a Nonsmooth Function: Conceptual Idea, Convergence Analysis, Numerical Results. *SIAM Journal on Optimization*, 2(1):121–152, 1992.
- [35] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

Table 2: Results when running NonOpt *for speed* when solving the test problems summarized in Table 1. Below, **name** is the problem name, **dir** is the direction computation strategy employed (CP for cutting plane, GC for gradient combination, and G for gradient, as explained on page 8), **iters** is the number of iterations required, **funcs** is the number of function evaluations required, **grads** is the number of gradient evaluations required, **f** is the objective function value of the final iterate, and **CPU** is the time required (in seconds).

name	alg	iters	funcs	grads	f	CPU (s)
ActiveFaces	CP	22	414	347	+3.008077e-09	0.107499
	GC	22	414	347	+3.030957e-09	0.084499
	G	34	458	359	+2.114735e-10	0.084612
BrownFunction_2	CP	107	643	178	+3.995409e-08	2.327747
	GC	107	700	434	+1.077945e-04	1.978943
	G	135	681	142	+1.751286e-06	0.398779
ChainedCB3.1	CP	382	2420	383	+2.001144e+03	1.862424
	GC	370	2304	371	+2.001045e+03	1.888899
	G	385	1689	386	+2.000694e+03	1.322669
ChainedCB3.2	CP	73	343	83	+1.998000e+03	0.289017
	GC	73	347	83	+1.998000e+03	0.283544
	G	73	262	80	+1.998000e+03	0.239194
ChainedCrescent.1	CP	41	208	53	+3.250578e-09	0.182489
	GC	42	230	58	+3.773635e-09	0.202950
	G	69	320	81	+2.846356e-08	0.184924
ChainedCrescent.2	CP	109	860	169	+1.452173e-05	1.703804
	GC	94	699	252	+1.075416e-02	1.305967
	G	91	540	96	+8.126163e-03	0.255329
ChainedLQ	CP	88	499	90	-1.412669e+03	0.297360
	GC	88	498	90	-1.412668e+03	0.298095
	G	88	391	90	-1.412672e+03	0.285419
ChainedMifflin.2	CP	82	646	98	-7.063199e+02	0.459288
	GC	80	677	398	-7.063075e+02	1.620687
	G	80	441	89	-7.063076e+02	0.257966
MaxQ	CP	1797	5431	1889	+1.756840e-02	5.637300
	GC	1797	5431	1889	+1.756840e-02	5.524017
	G	1797	3627	1889	+1.756840e-02	5.474417
MxHilb	CP	198	858	275	+2.114027e-04	4.502786
	GC	198	858	275	+2.114027e-04	4.487898
	G	198	659	275	+2.114027e-04	3.796419
Test29_2	CP	1443	7871	1555	+2.024142e-03	5.872923
	GC	1369	7751	1580	+2.744651e-03	4.633045
	G	1486	6676	1557	+1.803336e-03	4.489679
Test29_5	CP	116	332	117	+5.457738e-06	2.212484
	GC	116	332	117	+5.457738e-06	2.207088
	G	116	215	117	+5.457738e-06	1.802998
Test29_6	CP	111	910	164	+2.000000e+00	2.774004
	GC	102	832	292	+2.000211e+00	1.722507
	G	125	643	126	+2.000000e+00	0.384783
Test29_11	CP	93	1110	134	+1.203082e+04	0.617999
	GC	61	598	86	+1.205794e+04	0.513414
	G	60	475	91	+1.207587e+04	0.119095
Test29_13	CP	1082	6813	1087	+5.671352e+02	13.372474
	GC	79	740	183	+5.799427e+02	1.332347
	G	161	960	170	+5.799165e+02	0.945128
Test29_17	CP	44	656	81	+1.091000e-03	0.559470
	GC	38	407	129	+1.092484e-03	0.529614
	G	54	656	66	+1.092500e-03	0.188093
Test29_19	CP	111	954	165	+1.000000e+00	3.042539
	GC	96	835	276	+1.000135e+00	1.859920
	G	118	612	120	+1.000000e+00	0.368168
Test29_20	CP	87	761	173	+5.000018e-01	1.393339
	GC	92	856	414	+5.000235e-01	1.940743
	G	128	861	250	+5.000000e-01	0.402968
Test29_22	CP	74	678	123	+4.531179e-05	0.479912
	GC	39	646	235	+4.533281e-05	0.220620
	G	79	956	170	+2.190039e-05	0.271480
Test29_24	CP	103	920	190	+1.099124e-01	2.947687
	GC	91	849	418	+1.099203e-01	1.933301
	G	102	619	118	+1.099124e-01	0.275045

Table 3: Results when running NonOpt *for accuracy* when solving the test problems summarized in Table 1. Below, **name** is the problem name, **dir** is the direction computation strategy employed (CP for cutting plane, GC for gradient combination, and G for gradient, as explained on page 8), **iters** is the number of iterations required, **funcs** is the number of function evaluations required, **grads** is the number of gradient evaluations required, **f** is the objective function value of the final iterate, and **CPU** is the time required (in seconds).

name	dir	iters	funcs	grads	f	CPU (s)
ActiveFaces	CP	22	414	347	+3.008077e-09	0.083728
	GC	22	414	347	+3.030957e-09	0.078691
	G	62	582	387	+1.156546e-10	0.130724
BrownFunction_2	CP	148	1256	279	+2.280562e-08	4.199453
	GC	493	3920	5077	+1.040409e-04	23.908808
	G	200	991	207	+1.894216e-06	0.508164
ChainedCB3_1	CP	1713	11832	2252	+1.998001e+03	33.155979
	GC	1550	10283	3452	+1.998002e+03	28.769716
	G	730	3530	741	+1.998489e+03	2.446144
ChainedCB3_2	CP	92	487	120	+1.998000e+03	0.554696
	GC	81	441	112	+1.998000e+03	0.430946
	G	135	578	143	+1.998000e+03	0.370261
ChainedCrescent_1	CP	41	208	53	+3.250578e-09	0.187423
	GC	42	230	58	+3.773635e-09	0.213865
	G	116	551	142	+2.849132e-08	0.283621
ChainedCrescent_2	CP	342	3460	631	+9.485031e-06	12.051088
	GC	1177	10165	12800	+9.118619e-03	68.642038
	G	131	712	138	+8.126163e-03	0.329044
ChainedQ	CP	751	5553	1182	-1.412799e+03	21.581284
	GC	1233	8432	1790	-1.412799e+03	24.280076
	G	793	3434	796	-1.412799e+03	2.595213
ChainedMifflin_2	CP	2084	13223	2190	-7.065434e+02	27.780682
	GC	133	1096	944	-7.063075e+02	4.219681
	G	138	769	151	-7.063076e+02	0.390109
MaxQ	CP	1888	5825	1996	+9.846592e-03	5.841492
	GC	1888	5825	1996	+9.846592e-03	5.823572
	G	1888	3930	1996	+9.846592e-03	5.896577
MxHilb	CP	469	1963	642	+2.043153e-06	10.497335
	GC	469	1963	642	+2.043153e-06	10.470230
	G	469	1493	642	+2.043153e-06	8.779602
Test29_2	CP	2899	17619	3471	+1.253118e-06	21.473363
	GC	2440	16181	3217	+1.861335e-05	14.838679
	G	2948	14754	3119	+1.176407e-06	9.079778
Test29_5	CP	202	732	203	+8.370109e-07	4.356180
	GC	202	732	203	+8.370109e-07	4.350333
	G	202	529	203	+8.370109e-07	3.661412
Test29_6	CP	157	1301	255	+2.000000e+00	4.675574
	GC	2458	17606	5311	+6.923490e-02	19.347360
	G	231	1156	232	+2.000000e+00	0.612321
Test29_11	CP	205	1937	295	+1.203129e+04	4.611533
	GC	1436	10319	7261	+1.202995e+04	40.581987
	G	117	760	121	+1.207587e+04	0.214988
Test29_13	CP	2102	12392	2141	+5.661129e+02	23.540974
	GC	1723	10409	2385	+5.661128e+02	22.928162
	G	1962	8684	1971	+5.661131e+02	10.505420
Test29_17	CP	262	4153	504	+1.057500e-03	5.205482
	GC	1196	13746	3294	+3.668662e-05	23.924566
	G	179	1902	215	+1.080500e-03	0.619420
Test29_19	CP	149	1279	241	+1.000000e+00	4.645676
	GC	1033	8520	4538	+2.543375e-01	20.473368
	G	214	1077	216	+1.000000e+00	0.576925
Test29_20	CP	215	2088	584	+5.000000e-01	8.460440
	GC	622	7547	7004	+5.000085e-01	38.719123
	G	284	1742	524	+5.000000e-01	0.832635
Test29_22	CP	421	5124	1232	+8.267981e-06	12.493644
	GC	216	3511	1008	+7.592961e-06	3.728066
	G	574	6037	1289	+1.501467e-06	1.941063
Test29_24	CP	145	1271	273	+1.099124e-01	4.663748
	GC	468	5633	5036	+1.099154e-01	23.860827
	G	166	910	182	+1.099124e-01	0.368457

Table 4: Data for solving instances of problem (15). Below, **regularizer** is the regularizer employed (see page 21), λ is the regularization parameter, β is the regularizer parameter, **MSE** is the final mean squared error of the solution obtained X_* with respect to the true image X_o , and **CPU** is the number of seconds required to compute X_* .

regularizer	λ	β	MSE: $\ X_* - X_o\ _F^2$	CPU (s)
(a)	2^6	2^0	1.08781e+08	145.271
(b)	2^{21}	2^{-7}	1.20072e+08	156.497
(c)	2^{25}	2^{-19}	1.08780e+08	110.409
(d)	2^6	2^{18}	1.08788e+08	134.867