Les Cahiers du GERAD ISSN: 0711–2440

A regularization method for constrained nonlinear least squares

D. Orban,
A. Soares Siqueira

G–2019–17

February 2019

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

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

A regularization method for constrained nonlinear least squares

Dominique Orban [a,b]

Abel Soares Siqueira [c]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

[c] *Federal University of Paraná, Department of Mathematics, 80060-000, Curitiba, Paraná, Brazil*

dominique.orban@gerad.ca
abelsiqueira@ufpr.br

February 2019
Les Cahiers du GERAD
G–2019–17

Abstract: We propose a regularization method for nonlinear least-squares problems with equality constraints. Our approach is modeled after those of Arreckx and Orban (2018) and Dehghani et al. (2019), and applies a selective regularization scheme that may be viewed as a reformulation of an augmented Lagrangian. Our formulation avoids the occurrence of the operator $A(x)^T A(x)$, where A is the Jacobian of the nonlinear residual, which typically contributes to the density and ill conditioning of subproblems. Under boundedness of the derivatives, we establish global convergence to a KKT point or a stationary point of an infeasibility measure. If second derivatives are Lipschitz continuous and a second-order sufficient condition is satisfied, we establish superlinear convergence without requiring a constraint qualification to hold. The convergence rate is determined by a Dennis-Moré-type condition. We describe our implementation in the Julia language, which supports multiple floating-point systems. We illustrate a simple progressive scheme to obtain solutions in quadruple precision. Because our approach is similar to applying an SQP method with an exact merit function on a related problem, we show that our implementation compares favorably to IPOPT in IEEE double precision.

Keywords: Factorization-free implementation, least-squares problem, nonlinear least-squares, constrained nonlinear least-squares, Julia

Résumé : Nous proposons une méthode de régularisation pour les problèmes aux moindres carrés non linéaires avec contraintes d'égalité. Notre appliquons les approches de Arreckx and Orban (2018) et Dehghani et al. (2019) et appliquons une régularisation sélective qui peut s'interpréter en termes d'un lagrangien augmenté. Notre formulation évite de faire apparaître l'opérateur $A(x)^T A(x)$, où A est le jacobien du résidu non linéaire, lequel contribue typiquement à la densité et au mauvais conditionnement des sous-problèmes. En supposant que les dérivées sont bornées, nous établissons la convergence vers un point de KKT ou un point stationnaire de la mesure de violation des contraintes. Si les dérivées secondes sont Lipschitz continues et si une condition suffisante du second ordre est satisfaite, la convegence superlinéaire a lieu sans hypothèse de qualification des contraintes. Le taux de convergence est déterminé par une condition de type Dennis et Moré. Nous décrivons notre implémentation dans le langage Julia, lequel permet d'utiliser plusieurs systèmes en virgule flottante. Un schéma progressif simple permet d'obtenir une solution en quadruple précision à un coût inférieur au coût de la méthode appliquée entièrement en quadruple précision. Notre approche est semblable à une méthode SQP avec fonction de mérite exacte appliquée à un problème connexe. Pour cette raison, nous montrons que notre implémentation se compare avantageusement à IPOPT en IEEE double précision.

Acknowledgments: Research partially supported by an NSERC Discovery Grant.

1 Introduction

We present an algorithm for the constrained nonlinear least-squares problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ f(x) := \tfrac{1}{2}\|F(x)\|^2 \quad \text{subject to } c(x) = 0, \tag{1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^p$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable. We make no assumptions on the relations between n, p and m. Our approach follows closely Arreckx and Orban (2018), who devise an algorithm for equality-constrained optimization, and Dehghani et al. (2019), who focus on linearly-constrained linear least-squares problems. Likewise, our method features primal and dual regularization, which helps overcome difficulties with ill-conditioned and rank-deficient Jacobians of either F and c. Our convergence analysis relies heavily on (Armand et al., 2013). Our formulation avoids the occurrence of the operator $A(x)^T A(x)$, where A is the Jacobian of the nonlinear residual, which typically contributes to dense and ill conditioned subproblems. Under boundedness of the derivatives, we establish global convergence to a KKT point or a stationary point of an infeasibility measure. If second derivatives are Lipschitz continuous and a second-order sufficient condition is satisfied, we establish superlinear convergence without requiring a constraint qualification to hold. The convergence rate is determined by a Dennis-Moré-type condition.

We implement our approach in the Julia[1] language, building on the JuliaSmoothOptimizers[2] infrastructure for optimization. A special feature of our implementation is multiprecision support, which allows to run the solver in any floating-point precision supported by Julia, including half, single, double and quadruple precision, as well as the variable-precision floats implemented in the GNU MPFR Library[3]. We illustrate how this feature may be exploited in a simple warm-start cascade using increasing levels of accuracy to obtain a solution in quadruple precision at a fraction of the cost of running the entire solver in quadruple precision.

Conceptually, our approach is related to an SQP method with exact penalty function applied to

$$\underset{(x,r)\in\mathbb{R}^{n+p}}{\text{minimize}}\ \tfrac{1}{2}\|r\|^2 \quad \text{subject to } F(x) - r = 0,\ c(x) = 0, \tag{2}$$

which is equivalent to (1). For this reason, we compare our approach in double precision with IPOPT on a collection of problems with and without equality constraints and show that our method is competitive. A theoretical advantage over SQP is that our approach does not suffer from the Maratos effect.

Related research

Schittkowski (1988, 2007) also employs the transformation (2), which he solves by way of a general SQP solver using BFGS approximations to the Hessian of the Lagrangian. Li (2000) uses a limited memory BFGS approximation in the same SQP framework.

Mahdavi-Amiri and Bartels (1989), Mahdavi-Amiri and Bidabadi (2012), Mahdavi-Amiri and Ansari (2012) and Ansari and Mahdavi-Amiri (2014) use an exact nonsmooth penalty with a two-step SQP algorithm for constrained nonlinear least squares. Mahdavi-Amiri and Bartels (1989), Mahdavi-Amiri and Ansari (2012) and Mahdavi-Amiri and Bidabadi (2012) use projected gradient steps with linesearch, and Ansari and Mahdavi-Amiri (2014) uses a combined trust-region and linesearch approach. Li et al. (2002) also uses a secant approximation in a two-step SQP framework. Bidabadi and Mahdavi-Amiri (2013) uses exact penalty and projected structured secant approximation, and Bidabadi (2017) uses a spectral scaling structured BFGS approach for constrained nonlinear least squares.

Fernanda et al. (2005) use a primal-dual interior-point algorithm for constrained nonlinear least squares with a quasi-Newton approximation of the Hessian in factored form.

[1] julialang.org
[2] juliasmoothoptimizers.github.io
[3] www.mpfr.org

Wright (1998) introduces stabilized SQP, a local strategy that regularizes the Newton system, allowing for ill-conditioned or rank-deficient Jacobians while retaining superlinear convergence properties. Izmailov et al. (2015) combine stabilized SQP with augmented Lagrangian, using outer and inner iterations. Izmailov et al. (2016) use an exact two-parameter penalty function with linesearch for equality-constrained problems.

The survey of stabilized SQP of Fernández and Solodov (2014) provides motivation and numerous recent references.

Fernández et al. (2012) describe an inexact restoration scheme for stabilized SQP methods.

Armand and Omheni (2015), Gill et al. (2016b) and Gill et al. (2016a) propose globally convergent primal-dual augmented Lagrangian approaches for stabilized SQP with fast local convergence properties.

Our contribution builds on the framework of Armand et al. (2013) to ensure a smooth transition between a globally convergent strategy and stabilized SQP in the local regime.

Notation

We use $\|\cdot\|$ to denote the Euclidean norm. The Jacobian of F and c at x is denoted $A(x)$ and $B(x)$, respectively. For two nongegative scalar sequences a_k and b_k converging to zero, we use the Landau symbols $a_k = o(b_k)$ if $\lim_{k\to+\infty} a_k/b_k = 0$ and $a_k = \omega(b_k)$ if $b_k = o(a_k)$. We write $a_k = O(b_k)$ if there exists a constant $C > 0$ such that $a_k \leq Cb_k$ for all sufficiently large k, and similarly, $a_k = \Omega(b_k)$ if there exists a constant $C > 0$ such that $a_k \geq Cb_k$, and $\Theta(b_k)$ if both $a_k = O(b_k)$ and $b_k = O(a_k)$.

2 Regularized constrained nonlinear least squares

Following the strategy of Friedlander and Orban (2012) and later Arreckx and Orban (2018) and Dehghani et al. (2019), we consider the regularization of (1) about the current iterate x_k and Lagrange multiplier estimate y_k given by

$$\begin{array}{ll} \underset{(x,u)\in\mathbb{R}^{n+m}}{\text{minimize}} & \frac{1}{2}\|F(x)\|^2 + \frac{1}{2}\rho_k\|x - x_k\|^2 + \frac{1}{2}\delta_k\|u + y_k\|^2 \\ \text{subject to} & c(x) + \delta_k u = 0, \end{array} \tag{3}$$

where $\rho_k, \delta_k > 0$ are primal and dual regularization parameters. We introduce $r = F(x)$ as in (2) and obtain the equivalent problem

$$\begin{array}{ll} \underset{(x,u,r)\in\mathbb{R}^{n+m+p}}{\text{minimize}} & \frac{1}{2}\|r\|^2 + \frac{1}{2}\rho_k\|x - x_k\|^2 + \frac{1}{2}\delta_k\|u + y_k\|^2 \\ \text{subject to} & F(x) - r = 0, \\ & c(x) + \delta_k u = 0. \end{array} \tag{4}$$

For any $\delta_k > 0$, (4) satisfies the linear independence constraint qualification at all feasible (x, u, r). The Karush-Kuhn-Tucker conditions for (4) say that there exist s and y such that

$$\begin{aligned} \rho_k(x - x_k) - A(x)^T s - B(x)^T y &= 0 \\ r + s &= 0 \\ \delta_k(u + y_k) - \delta_k y &= 0 \\ F(x) - r &= 0 \\ c(x) + \delta_k u &= 0. \end{aligned} \tag{5}$$

Friedlander and Orban (2012) call the regularization *exact*, given the relationship between the KKT points of (1) and (4) in the sense of the following result, which parallels (Arreckx and Orban, 2018, Theorem 1) and follows from the KKT conditions of (1) and (4).

Theorem 1 *Suppose $(x_k, r_k, u_k, s_k, y_k)$ is a KKT point of (4) for certain $\rho_k \geq 0$ and $\delta_k \geq 0$. Then (x_k, y_k) is a KKT point of (1).*

Alternatively, let $\rho_k = 0$, $\delta_k > 0$ *and assume* $(\overline{x}, \overline{r}, \overline{u}, \overline{s}, \overline{y})$ *is a KKT point of* (4) *with* $\overline{x}$ *feasible for* (1). *Then* $\overline{u} = 0$, $\overline{y} = y_k$, $\overline{s} = -\overline{r} = -F(\overline{x})$ *and* $(\overline{x}, \overline{y})$ *is a KKT point of* (1).

Conversely, suppose (x^*, y^*) *is a KKT point of* (1). *If we choose* $(x_k, y_k) = (x^*, y^*)$ *in* (4), *then* $(x, r, u, s, y) = (x^*, F(x^*), 0, -F(x^*), y^*)$ *is a KKT point of* (4) *for any* $\rho, \delta > 0$.

We eliminate $s = -r$ and $u = y - y_k$ in (5) and obtain

$$\begin{aligned} \rho_k(x - x_k) + A(x)^T r - B(x)^T y &= 0 \\ F(x) - r &= 0 \\ c(x) + \delta_k(y - y_k) &= 0. \end{aligned}$$

Let us now denote $w := (x, r, y)$ and define

$$G(w; w_k, \rho_k, \delta_k) := \begin{bmatrix} \rho_k(x - x_k) + A(x)^T r - B(x)^T y \\ F(x) - r \\ c(x) + \delta_k(y - y_k) \end{bmatrix}.$$

For simplicity of notation, we let $A_k := A(x_k)$ and $B_k := B(x_k)$, as well as

$$G_k := G(w_k; w_k, \rho_k, \delta_k) = \begin{bmatrix} A_k^T r_k - B_k^T y_k \\ F(x_k) - r_k \\ c(x_k) \end{bmatrix},$$

and $\nabla G_k := \nabla G(w_k; w_k, \rho_k, \delta_k)$, where ∇G is the transposed Jacobian of G. Our method computes a step $\Delta w = (\Delta x, \Delta r, \Delta y)$ solution of $K_k \Delta w = -G_k$, where K_k is obtained by allowing a symmetric Hessian approximation $H_k \approx H(x_k, r_k, y_k)$, where

$$H(x, r, y) = \sum_{i=1}^{p} r_i \nabla^2 F_i(x) - \sum_{i=1}^{m} y_i \nabla^2 c_i(x) \tag{6}$$

is the (1,1) block of ∇G_k. The system $K_k \Delta w = -G_k$ can be written as

$$\begin{bmatrix} H_k + \rho_k I & A_k^T & B_k^T \\ A_k & -I & \\ B_k & & -\delta_k I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta r \\ -\Delta y \end{bmatrix} = \begin{bmatrix} -A_k^T r_k + B_k^T y_k \\ r_k - F(x_k) \\ -c(x_k) \end{bmatrix}, \tag{7a}$$
$$\tag{7b}$$
$$\tag{7c}$$

which we may solve (7) using a symmetric indefinite factorization such as that of Duff (2004).

2.1 Algorithm

We base our algorithm on the framework of Armand et al. (2013), also used by Arreckx and Orban (2018). We measure the KKT error with

$$\|G_k\|_* = \|A_k^T r_k - B_k^T y_k\| + \|F(x_k) - r_k\| + \|c(x_k)\|.$$

At the beginning of each outer iteration, we compute a search direction (7) and test if a full step $w_{k+1} = w_k + \Delta w$ is acceptable in the sense that

$$\|G_{k+1}\|_* \leq \theta \|G_k\|_* + \epsilon_k, \tag{8}$$

where $\theta \in (0, 1)$ and $\epsilon_k > 0$ is monotonically decreasing. This is called an extrapolation step. If the extrapolation step is not accepted, we fall back on an inner iteration during which we compute search directions (7) and perform a linesearch to enforce decrease of a merit function in order to find w_{k+1} satisfying (8).

The Lagrangian of (1) is

$$L(x, y) = \tfrac{1}{2}\|F(x)\|^2 - y^T c(x). \tag{9}$$

The inner iterations are a series of line search steps on the augmented Lagrangian merit function

$$\phi_k(x;\delta) = L(x,y_k) + \tfrac{1}{2}\delta^{-1}\|c(x)\|^2.$$

Algorithm 2.1 outlines the k-th outer iteration and Algorithm 2.2 details the inner iteration. The stopping condition used at line 1 of Algorithm 2.1 is based on $\|G_k\|_*$ and is described in Section 5.

From $x_{k,j}$, given a step Δx, we seek $\alpha_j \in (0,1]$ satisfying the Armijo condition at line 3 of Algorithm 2.2.

Let Δw be the solution of (7). We can write from (7b) and (7c)

$$\Delta r = A_{k,j}\Delta x - r_{k,j} + F(x_{k,j}), \quad \text{and} \quad \Delta y = -\delta_{k,j}^{-1}\big(c(x_{k,j}) + B_{k,j}\Delta x\big),$$

which we substitute into (7a) to obtain

$$W_{k,j}\Delta x = -A_{k,j}^T F(x_{k,j}) + B_{k,j}^T\left(y_k - \delta_{k,j}^{-1}c(x_{k,j})\right) = -\nabla\phi_k(x_{k,j};\delta_{k,j}), \tag{10}$$

where

$$W_{k,j} := H_{k,j} + \rho_{k,j}I + A_{k,j}^T A_{k,j} + \delta_{k,j}^{-1}B_{k,j}^T B_{k,j}. \tag{11}$$

Therefore if $W_{k,j}$ is positive definite, Δx is a descent direction for $\phi_k(\cdot;\delta_{k,j})$ from $x_{k,j}$. As is well known, this condition can be imposed by controlling $\rho_{k,j}$.

Algorithm 2.1 k-th outer iteration

Require: $w_k = (x_k, r_k, y_k)$
1: **if** a stopping condition is satisfied **then**
2: terminate with w_k as approximate KKT point
3: **end if**
4: choose parameters $\rho_k \geq 0$ and $\delta_k > 0$, and compute Δw solution of (7)
5: set $\hat{w}_k = w_k + \Delta w$
6: choose $\epsilon_k > 0$
7: **if** $\|G(\hat{w}_k;\hat{w}_k,\rho_k,\delta_k)\|_* \leq \theta\|G_k\|_* + \epsilon_k$ **then**
8: $w_{k+1} = \hat{w}_k$
9: **else**
10: use Algorithm 2.2 to identify w_{k+1} satisfying (8).
11: **end if**

Algorithm 2.2 Inner iterations corresponding to outer iteration k

Require: $\gamma_A \in (0,1)$, $j = 0$, $\delta_{k,0} = \delta_k$ and $w_{k,0} := w_k$
1: **while** $\|G(w_{k,j};w_{k,j},\rho_{k,j},\delta_{k,j})\|_* > \theta\|G_k\|_* + \epsilon_k$ **do**
2: choose $\rho_{k,j} \geq 0$ and solve (7) evaluated at $w_{k,j}$ with parameters $\rho_{k,j}$ and $\delta_{k,j}$
3: perform a line search to find $\alpha_j \in (0,1]$ such that

$$\phi_k(x_{k,j} + \alpha_j\Delta x;\delta_{k,j}) \leq \phi_k(x_{k,j};\delta_{k,j}) + \alpha_j\gamma_A\nabla\phi_k(x_{k,j};\delta_{k,j})^T\Delta x$$

4: set $x_{k,j+1} = x_{k,j} + \alpha_j\Delta x$
5: set $r_{k,j+1} = F(x_{k,j+1})$
6: **if** $\|\nabla_x L(x_{k,j+1}, y_k - \delta_{k,j}^{-1}c(x_{k,j+1}))\| \leq \theta\|\nabla_x L(x_{,k}, y_k)\| + \frac{1}{2}\epsilon_k$ and
7: $\|c(x_{k,j+1})\| > \theta\|c(x_k)\| + \frac{1}{2}\epsilon_k$ **then**
8: $\delta_{k,j+1} = \delta_{k,j}/10$
9: **else**
10: $\delta_{k,j+1} = \delta_{k,j}$
11: **end if**
12: $j \leftarrow j+1$
13: **end while**
14: set $x_{k+1} = x_{k,j}$, $y_{k+1} = y_k - \delta_{k,j}^{-1}c(x_{k,j})$, and $r_{k+1} = r_{k,j}$.

3 Global convergence

The global convergence of our method is heavily based on the framework of Armand et al. (2013) and on the global convergence of Arreckx and Orban (2018). Only a few modifications are needed to take the constraint $F(x) = r$ into account.

We consider the following assumptions.

Assumption 1 *The sequences $\{H_{k,j}\}$, $\{\rho_{k,j}\}$, $\{A_{k,j}\}$ and $\{B_{k,j}\}$ are bounded.*

Assumption 2 *The matrices $W_{k,j}$ are uniformly positive definite for k, $j \in \mathbb{N}$.*

Assumption 1 is standard. In Assumption 2, $W_{k,j}$ can be made sufficiently positive definite by selecting $\rho_{k,j}$ sufficiently large, which can be done by monitoring the inertia of the matrix in (7).

Note that the choice $H_k = -\sum_{i=1}(y_k)_i \nabla^2 c_i(x_k)$ amounts to a Gauss-Newton approach. Assumption 2 will be satisfied if either each $A_{k,j}$ has full column rank or $\rho_{k,j} > 0$ for all k and j. In particular, in the under-determined case, if $A_{k,j}^T A_{k,j}$ is not positive definite on $\text{Null}(B_{k,j})$, we must choose $\rho_{k,j} > 0$, which amounts to a Levenberg-Marquardt approach.

Our first result concerns the convergence of the inner iterations, and is inspired by (Arreckx and Orban, 2018, Theorem 2).

Theorem 2 *Let Assumptions 1 and 2 hold and $\{x_{k,j}\}$ be generated by Algorithm 2.2. Then, either*

1. *we find $\|G_{k+1}\|_* \leq \theta \|G_k\|_* + \epsilon_k$, or*
2. *$\liminf_{j\to\infty} \|B_{k,j+1}^T c(x_{k,j+1})\| = 0$.*

Proof. First, assume that $\delta_{k,j}$ is decreased an infinite number of times in line 8 of Algorithm 2.2. From the condition on line 6, there is an infinite subset $\mathcal{J} \subseteq \mathbb{N}$ such that, for all $j \in \mathcal{J}$,

$$\|A_{k,j+1}^T F_{k,j+1} - B_{k,j+1}^T (y_k - \delta_{k,j}^{-1} c(x_{k,j+1}))\| \leq \theta \|\nabla_x L(x_k, y_k)\| + \tfrac{1}{2}\epsilon_k,$$

but $\|c(x_{k,j+1})\| > \theta \|c(x_k)\| + \frac{1}{2}\epsilon_k > 0$. Because $\{\delta_{k,j}\}_{j\in\mathcal{J}} \to 0$, we must have $\{B_{k,j+1}^T c(x_{k,j+1})\}_{j\in\mathcal{J}} \to 0$.

If, on the other hand, there exists $\delta_{\min} > 0$ such that $\delta_{k,j} \geq \delta_{\min}$ for all j, Assumption 1 implies that $W_{k,j}$ is uniformly bounded for all j. Because $W_{k,j}$ is also uniformly positive definite from Assumption 2, then there exists $\sigma \in (0, 1)$ such that

$$\sigma \|d\|^2 \leq d^T W_{k,j} d \leq \tfrac{1}{\sigma}\|d\|^2 \quad \text{for all } d \neq 0 \text{ and all } j \geq 0.$$

Consequently (10) yields

$$\|\nabla \phi(x_{k,j})\| = \|W_{k,j}\Delta x\| \leq \tfrac{1}{\sigma}\|\Delta x\|,$$

and

$$\nabla \phi(x_{k,j})^T \Delta x = -\Delta x^T W_{k,j} \Delta x \leq -\sigma \|\Delta x\|^2 \leq -\sigma^2 \|\nabla \phi(x_{k,j})\| \|\Delta x\|.$$

We then have from (Birgin and Martínez, 2014, Theorem 8.2) that there exists an infinite subset $\mathcal{J} \subseteq \mathbb{N}$ such that $\{\nabla \phi(x_{k,j})\}_{j\in\mathcal{J}} \to 0$. Thus, the first condition on line 6 of Algorithm 2.2 is satisfied for all $j \in \mathcal{J}$ sufficiently large. Since line 8 is only accessed a finite number of times, at some point we must have $\|c(x_{k,j+1})\| \leq \theta \|c(x_k)\| + \frac{1}{2}\epsilon_k$. Because $r_{k,j+1} = F(x_{k,j+1})$, we obtain $w_{k+1} := w_{k,j+1}$ such that

$$\begin{aligned} \|G_{k+1}\|_* &= \|G(w_{k,j+1}; w_{k,j+1}, \rho_{k,j}, \delta_{k,j})\|_* \\ &= \|\nabla \phi(x_{k,j+1}, \delta_{k,j})\| + \|c(x_{k,j+1})\| \leq \|G_k\|_* + \epsilon_k. \end{aligned}$$

□

Note that in the first scenario of the proof of Theorem 2, any limit point of $\{x_{k,j}\}_{j\in\mathcal{J}}$ is stationary for the infeasibility measure $\|c(x)\|$.

We now turn to convergence of the outer iterations. The proof of the following result is the same as that of (Arreckx and Orban, 2018, Theorem 6), which is in turn inspired from the proof of (Armand et al., 2013, Theorem 1).

Theorem 3 *Suppose Assumptions 1 and 2 hold, that Algorithm 2.2 always succeeds when called by Algorithm 2.1, and that* $\{\epsilon_k\} \to 0$. *Then, Algorithm 2.1 generates* $\{w_k\}$ *such that* $\{G_k\} \to 0$.

Theorem 3 implies that if $w^* = (x^*, r^*, y^*)$ is an accumulation point of $\{w_k\}$, then (x^*, y^*) is a KKT point for (1).

4 Local convergence

Our local convergence is inherited from Arreckx and Orban (2018), with some small modifications due to the initial transformation made by introducing $r = F(x)$ and the use of exact methods for solving the linear systems. Crucially, our analysis assumes that there exist Lagrange multipliers at a solution of (1), but we do not assume that the constraints satisfy a qualification condition.

Consider the KKT measure of (2)

$$G(w) := G(w; w, 0, 0) = \begin{bmatrix} A(x)^T r - B(x)^T y \\ F(x) - r \\ c(x) \end{bmatrix}, \tag{12}$$

and note that the $(1,1)$ block of its Jacobian is (6). The Hessian of (9) is

$$\nabla^2_{17} L(x, y) = H(x, F(x), y) + A(x)^T A(x).$$

Because H_k is an approximation to $H(x_k, r_k, y_k)$, $H_k + A_k^T A_k$ can be interpreted as an approximation to $\nabla^2_{xx} L(x_k, y_k)$. Furthermore, (7) can be rewritten as

$$\begin{bmatrix} H_k + \rho_k I + A_k^T A_k & B_k^T \\ B_k & -\delta_k I \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = \begin{bmatrix} -A_k^T F(x_k) + B_k^T y_k \\ -c(x_k) \end{bmatrix},$$

which eliminates r and Δr, keeping most of the analysis essentially the same as that of Arreckx and Orban (2018).

Let $\{(x_k, y_k)\}$ be a convergent sequence generated by Algorithm 2.1, converging to a KKT point x^* of (1). Define $\mathcal{Y}$ the set of Lagrange multipliers associated to x^*, i.e.,

$$\mathcal{Y} := \{y^* \in \mathbb{R}^m \mid (x^*, y^*) \text{ satisfies the KKT conditions for (1)}\},$$

and let $\mathcal{S} := \{x^*\} \times \mathcal{Y}$.

Our working assumptions are as follows.

Assumption 3 *The sequence* $\{w_k\}$ *generated by Algorithm 2.1 converges to* $w^* = (x^*, r^*, y^*)$ *for a certain* $y^* \in \mathcal{Y}$.

Note that $r^* = F(x^*)$.

Assumption 4 *The functions* F *and* c *are twice continuously differentiable with locally Lipschitz second derivatives on* $\mathbb{R}^n$.

Assumption 5 *For all sufficiently large* k, δ_k *is chosen as* $\|G_k\|_*$.

Assumption 6 *H_k is uniformly bounded, i.e., there exists $\kappa > 0$ such that $\|H_k\| \leq \kappa$ for all $k \geq 0$.*

In particular, Assumption 6 is satisfied if $H_k = H(x_k, r_k, y_k)$,

Assumption 7 *The approximation $H_k + A_k^T A_k$ is sufficiently positive definite on the nullspace of $B(x^*)$ for all sufficiently large k, i.e., there exists $\eta > 0$, such that $z^T(H_k + A_k^T A_k)z \geq \eta\|z\|^2$ for all $z \in \text{Null}(B(x^*))$.*

Assumption 7 implies that $H_k + A_k^T A_k + \delta_k^{-1} B_k^T B_k$ is positive definite for all k sufficiently large, and therefore we can eventually always choose $\rho_k = 0$.

The following assumption on the quality of the approximation H_k yields asymptotic superlinear convergence.

Assumption 8 *The approximation H_k satisfies*

$$\| \left(H(x_k, r_k, y_k) - H_k\right) \Delta x\| = o(\|\Delta x\|),$$

for all k sufficiently large.

Note that Assumption 8 is equivalent to assuming that $\|(H(x^*, r^*, y^*) - H_k)\Delta x\| = o(\|\Delta x\|)$ for all sufficiently large k, which similar to the Dennis and Moré (1977) condition for superlinear convergence of quasi-Newton methods. In particular, Assumption 8 is satisfied if $(H(x_k, r_k, y_k) - H_k) \to 0$.

A stronger assumption provides us with a specific convergence rate.

Assumption 9 *There exists $0 < \beta \leq 1$ such that*

$$\| \left(H(x_k, r_k, y_k) - H_k\right) \Delta x\| = O(\|\Delta x\|^{1+\beta}),$$

for all k sufficiently large.

In particular, Assumption 9 is satisfied with $\beta = 1$ if $H_k = H(x_k, r_k, y_k)$ for all k sufficiently large.

Our first result follows from (Arreckx and Orban, 2018, Theorem 10) and takes Δr into account, in addition to Δx and Δy.

Theorem 4 *Let Assumptions 3 to 7 hold. For all sufficiently large k,*

$$\Delta x = O(\delta_k), \quad \Delta r = O(\delta_k), \quad \text{and} \quad \Delta y = O(\delta_k).$$

Proof. Arreckx and Orban (2018, Theorem 10) show $\Delta x = O(\delta_k)$ and $\Delta y = O(\delta_k)$. Using (7b) and Assumption 5 we have

$$\begin{aligned} \|\Delta r\| &= \|A_k \Delta x + (F(x_k) - r_k)\| \\ &\leq \|A_k\|\|\Delta x\| + \|F(x_k) - r_k\| \\ &= O(\Delta x) + O(\|G_k\|_*) = O(\delta_k). \end{aligned}$$

□

The next theorem parallels (Arreckx and Orban, 2018, Theorem 11) and lays down the basis for fast local convergence based on the quality of the Hessian approximation.

Theorem 5 *Let $\{w_k\}$ be generated by Algorithm 2.1, Assumptions 3 to 7 hold, $\rho_k = 0$ for all sufficiently large k, and*

$$\delta_k^+ := \|G(w_k + \Delta w; w_k + \Delta w, \rho_k, \delta_k)\|_*.$$

Then,

1. *if Assumption 8 is satisfied, then $\delta_k^+ = o(\delta_k)$;*

2. *if Assumption 9 is satisfied, then* $\delta_k^+ = O(\delta_k^{1+\beta})$;
3. *if* $H_k = H(x_k, r_k, y_k)$ *for all sufficiently large* k, *then* $\delta_k^+ = O(\delta_k^2)$.

Proof. Using the Taylor expansion of G defined in (12) at $w_k + \Delta w$ about w_k and (7) written as $K_k \Delta w = -G_k$, we obtain

$$\begin{aligned} G(w_k + \Delta w) &= G(w_k) + \nabla G(w_k)^T \Delta w + O(\|\Delta w\|^2) \\ &= G_k + K_k \Delta w + (\nabla G(w_k)^T - K_k)\Delta w + O(\|\Delta w\|^2) \\ &= (\nabla G(w_k)^T - K_k)\Delta w + O(\|\Delta w\|^2) \\ &= \begin{bmatrix} (H(x_k, r_k, y_k) - H_k)\Delta x \\ 0 \\ -\delta_k \Delta y \end{bmatrix} + O(\|\Delta w\|^2), \end{aligned}$$

where we use Assumption 4. Because (12) also implies that $G(w) = G(w; w, \rho, \delta)$ for any ρ and δ, we have

$$\begin{aligned} \delta_k^+ &= \|G(w_k + \Delta w; w_k + \Delta w, \rho_k, \delta_k)\|_* = \|G(w_k + \Delta w)\|_* \\ &= \| \left(H(x_k, r_k, y_k) - H_k\right) \Delta x\| + \delta_k \|\Delta y\| + O(\|\Delta w\|^2) \\ &= \| \left(H(x_k, r_k, y_k) - H_k\right) \Delta x\| + O(\delta_k^2). \end{aligned}$$

where we use Theorem 4. If $H_k = H(x_k, r_k, y_k)$ for all sufficiently large k, we therefore obtain $\delta_k^+ = O(\delta_k^2)$. If on the other hand Assumption 9 is satisfied, we obtain $\delta_k^+ = O(\delta_k^{1+\beta})$. Finally, if Assumption 8 is satisfied, $\delta_k^+ = o(\delta_k)$. □

The following theorem summarizes Corollaries 12 and 14, and Theorem 13 of Arreckx and Orban (2018).

Theorem 6 *Let* $\{w_k\}$ *be generated by Algorithm 2.1, Assumptions 3 to 7 hold,* $\rho_k = 0$ *for all sufficiently large* k. *If either*

1. *Assumption 8 is satisfied and* $\epsilon_k = \Omega(\delta_k)$, *or*
2. *Assumption 9 is satisfied and* $\epsilon_k = \omega(\delta_k^{1+\beta})$, *or*
3. $H_k = H(x_k, r_k, y_k)$ *for all sufficiently large* k, *and* $\epsilon_k = \omega(\delta_k^2)$,

then

i. *for sufficiently large* k, $w_{k+1} = \hat{w}_k$ *at line 5 of Algorithm 2.1;*
ii. $dist(w_k, \mathcal{S}) = \Theta(\delta_k)$,
iii. $\|w_k - w^*\| = \Theta(\delta_k)$.

Theorems 5 and 6 imply that Algorithm 2.1 performs no inner iterations asymptotically, and that the sequences $\{\text{dist}(w_k, \mathcal{S})\} \to 0$ and $\{w_k\} \to w^*$ at the same rate as $\{\delta_k\} \to 0$.

5 Implementation and numerical results

5.1 Implementation

We implemented Algorithm 2.1 in the Julia language, an interpreted language with just-in-time compilation, version 1.1 We build upon the infrastructure for optimization provided by the JuliaSmoothOptimizers suite of packages:

- the NLPModels.jl package provides facilities for modeling least-squares problems in the form (1), while retaining the ability to evaluate F and its derivatives;

- NLSProblems.jl is a collection of least-squares problems using the JuMP modeling language (Dunning et al., 2017) containing, among others, the least-squares problems from Moré et al. (1981), the constrained least-squares problems of Hock and Schittkowski (1981), and some of the equality-constrained least-squares problems from (Lukšan and Vlček, 1999, Section 5);
- CUTEst.jl interfaces Julia with the CUTEst problem collection and tools (Gould et al., 2015), which provides a number of least-squares problems in the form of feasibility problems;
- HSL.jl interfaces Julia with the symmetric indefinite factorization subroutines of MA57 (Duff, 2004; HSL, 2007);
- LDLFactorizations.jl is a translation of the concise LDL factorization of Tim Davis (2006), included in SuiteSparse;
- Krylov.jl is a collection of iterative methods for linear systems, linear least-squares and linear least-norm problems;
- SolverBenchmark.jl provides optimization solvers and benchmarking facilities;
- BenchmarkProfiles.jl produces Dolan and Moré (2002) performance profiles;
- NLPModelsIpopt.jl allows us to ask IPOPT (Wächter and Biegler, 2006) to solve an NLPModel.

We solve (7) using a symmetric indefinite factorization, and adjust the regularization parameter ρ to control the inertia, similarly to Wächter and Biegler (2006). The strategy is based on the following result

Proposition 1 (Forsgren, 2002, Proposition 2) *Let*

$$K = \begin{bmatrix} H & A^T \\ A & -D \end{bmatrix},$$

where H is an $n \times n$ symmetric matrix, D is an $m \times m$ symmetric positive definite matrix, and A is an $m \times n$ matrix. The inertia of K is $(n, m, 0)$ if and only if $H + A^T D^{-1} A$ is positive definite.

By Proposition 1, if ρ_k is chosen so that (7) has inertia $(n, m + p, 0)$, the Schur complement

$$H_k + \rho_k I + \begin{bmatrix} A_k^T & B_k^T \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \delta_k^{-1} I \end{bmatrix} \begin{bmatrix} A_k \\ B_k \end{bmatrix} = H_k + \rho_k I + A_k^T A_k + \delta_k^{-1} B_k^T B_k$$

is positive definite. Algorithm 5.1 summarizes the schedule of ρ_k, and is inspired by (Wächter and Biegler, 2006, Algorithm IC).

Algorithm 5.1 ρ update at iteration k

Require: $0 < \rho_{\min} < \rho^0$, $\kappa^- \in (0, 1)$, $\kappa^{++} > \kappa^+ > 1$, ρ_{k-1} (with $\rho_{-1} := 0$)
set $\rho_k \leftarrow 0$ and attempt to factorize the matrix in (7) *no regularization*
if factorization is successful and the inertia is correct **then**
  solve (7) and exit
end if
if $\rho_{k-1} = 0$ **then**
  $\rho_k \leftarrow \rho^0$
else
  $\rho_k \leftarrow \max\{\rho_{\min}, \kappa^- \rho_{k-1}\}$ *try decreasing ρ*
end if
attempt to factorize the matrix in (7)
while factorization is not successful or the inertia is not correct **do**
  if $\rho_{k-1} = 0$ **then**
    $\rho_k \leftarrow \kappa^{++} \rho_k$
  else
    $\rho_k \leftarrow \kappa^+ \rho_k$
  end if
  attempt to factorize the matrix in (7)
end while
solve (7) and exit.

The Lagrange multipliers are initialized to the least-squares approximation

$$y_0 = \underset{y}{\operatorname{argmin}} \ \tfrac{1}{2}\|B_0^T y - A_0^T F(x_0)\|^2,$$

which is computed using the CGLS implementation of the Krylov.jl package. CGLS is also used if we find x_k such that $\|F(x_k)\| \leq \epsilon_F$ and $\|c(x)\| \leq \epsilon_c$, for certain tolerances ϵ_F, $\epsilon_c > 0$. In such a situation, we define $r_k := F(x_k)$ and compute least-squares multipliers in hope of terminating the iterations early.

As explained below, our implementation accommodates multiple floating-point systems, which we exploit to define the parameters used in Algorithm 2.1, Algorithm 2.2 and Algorithm 5.1. Let ε_M denote the machine epsilon of a given floating-point system, and b_M the number of bits used to store real numbers. We use $\gamma_A = \varepsilon_M^{1/4}$, $\rho^0 = \varepsilon_M^{1/3}$, $\rho_{\min} = \varepsilon_M^{1/2}$, $\kappa^- = 1/3$, $\kappa^+ = 8$, $\kappa^{++} = \min(100, 2b_M)$, $\epsilon_{-1} = 10^3$, $\epsilon_{k+1} = \max(\min(10^3\delta_k, 0.99\epsilon_k), 0.9\epsilon_k)$, $\theta = 0.99$, and $\epsilon_F = \epsilon_c = \epsilon_{\text{tol}}^{1/2}$, where $\epsilon_{\text{tol}} > 0$ is the stopping tolerance defined below. We define $\delta_{-1} = 1.0$ and update

$$\delta_k = \max\left\{\varepsilon_M^{1/2}, \min\left(0.1\delta_{k-1}, \|G_k\|_*\right)\right\} \quad (k \geq 0),$$

at each outer iteration.

We name our implementation CaNNOLeS (**C**onstrained **a**nd **N**o**N**linear **O**ptimizer of **Le**ast **S**quares).

5.2 Numerical results against IPOPT

When applied to (2), IPOPT (Wächter and Biegler, 2006) shares a number of similarities with our solver. IPOPT computes steps using (7) with $\delta_k = 0$, and may select ad-hoc values of $\delta_k > 0$ if it encounters numerical difficulties.

Both our solver and IPOPT use MA57 (Duff, 2004; HSL, 2007) to factorize the matrix of (7). We call IPOPT from Julia using our NLPModelsIpopt.jl[4] package. In order for stopping conditions to be comparable in both solvers, we set the IPOPT parameters `dual_inf_tol=Inf`, `constr_viol_tol=Inf` and `compl_inf_tol=Inf` to disable additional stopping conditions related to those tolerances, `acceptable_iter=0` to disable the search for an acceptable point, and `nlp_scaling_method="none"` to disable problem scaling, as our implementation does not scale problems at this stage. Both methods use a maximum run time of 300 seconds. The stopping condition used in both solvers is that given by Wächter and Biegler (2006), i.e.,

$$\max\left\{\frac{\|\nabla f(x) + B(x)^T\lambda\|_\infty}{s_d}, \|c(x)\|_\infty\right\} \leq \epsilon_{\text{tol}},$$

where

$$s_d = \max\left\{100, \frac{\|\lambda\|_1}{m}\right\}/100,$$

and $\epsilon_{\text{tol}} = \varepsilon_M^{1/2}$ by default, Our comparison ran on a computer with an i7-7500U CPU and 16GB of RAM.

We compare the two solvers on 303 problems, where 215 are unconstrained and 88 have equality constraints. Part of the test set consists of the 126 CUTEst problems with no objective and equality constraints only, with up to 10,000 variables and constraints. In those problems, the residual is defined as the constraint functions, so that they are treated as unconstrained least squares. The other problems are from NLSProblems.jl. Among them, 15 equality-constrained problems are available in different sizes. We use each of them four times by setting the number of variables to 100, 500, 1,000 and 5,000.

Our results are presented in the form of Dolan and Moré (2002) performance profiles, using elapsed time and number of function evaluations, i.e., the sum of the number of evaluations of F, c, and their first and

[4] github.com/JuliaSmoothOptimizers/NLPModelsIpopt.jl

second derivatives, as performance metrics. Second derivatives of the residual are treated similarly to second derivatives of the constraints, i.e., each sum

$$\sum_{i=1}^{p} r_i \nabla^2 F_i(x) \qquad \text{and} \qquad \sum_{i=1}^{m} y_i \nabla^2 c_i(x)$$

counts as one Hessian evaluation, so that one evaluation of $H(x, r, y)$ counts as two Hessian evaluations.

If f_1 and f_2 are the final objective value identified by each solver on a given problem, we consider that $f_1 = f_2$ if both solvers converged and

$$\max\{f_1, f_2\} \leq \min\{f_1, f_2\} + \epsilon_a + \epsilon_r \max\{|f_1|, |f_2|\},$$

(Wächter and Biegler, 2006). We use $\epsilon_a = \epsilon_r = 0.1$. We exclude the 18 problems for which CaNNOLeS and IPOPT found different objective values from the performance profiles, but report them in the tables of results found in the electronic supplement. We further remove DMN15102, DMN15103, DMN15332, DMN15333, DMN37142, DMN37143 and EIGENB, on which both solvers attain the time limit and fail. IPOPT reported a zero CPU time on certain problems. Their CPU time was reset to half of the smallest nonzero time reported by IPOPT so that they could participate in the profiles.

Figure 1 shows the performance profiles for the remaining 278 problems. The profiles on the left compare the two solvers in terms of elapsed time, while those on the right compare in terms of total number of residual and constraint evaluations. In order to make sure that the IPOPT run time is not contaminated by the interface with Julia, we extract it from the IPOPT output. The top two profiles cover the entire test set. The remaining profiles consider subsets of problems: unconstrained problems, constrained problems, and problems with fewer and more than 100 variables. The profiles show that CaNNOLeS and IPOPT have similar robustness in all cases. IPOPT requires fewer function evaluations overall but CaNNOLeS is a close second. IPOPT is unexpectedly slower on equality-constrained problems. NSLCON is promising overall in terms of elapsed time.

The complete results can be found in the electronic supplement.

5.3 Numerical results with multiple precision

Certain applications, including metabolic expression models in systems biology (Ma and Saunders, 2014), require solutions to higher accuracy than double precision can offer. By way of a mechanism called *multiple dispatch*, the Julia language lets us implement our solver in a generic fashion so that the just-in-time compiler specializes the machine code to the data type of the inputs. This means that we implement each function once and it may be used with multiple input data types, with no extra programming effort, so that our solver may be used in various floating-point systems.

In the current state of our implementation, automatic differentiation in multiple floating-point systems can only be performed in forward mode using the ForwardDiff.jl[5] package, which restricts the size of the problems that we can solve in reasonable time. However, it is sufficient to illustrate the behavior of our solver when using multiple levels of precision. We consider problems 5.1 and 5.4 from Lukšan and Vlček (1999), namely

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n-1} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \\ \text{subject to} \quad & 3x_{k+1}^3 + 2x_{k+2} + \sin(x_{k+1} - x_{k+2}) \sin(x_{k+1} + x_{k+2}) + \\ & 4x_{k+1} - x_k e^{x_k - x_{k+1}} = 8, \quad k = 1, \ldots, n-2, \end{aligned} \tag{13}$$

[5] github.com/JuliaDiff/ForwardDiff.jl

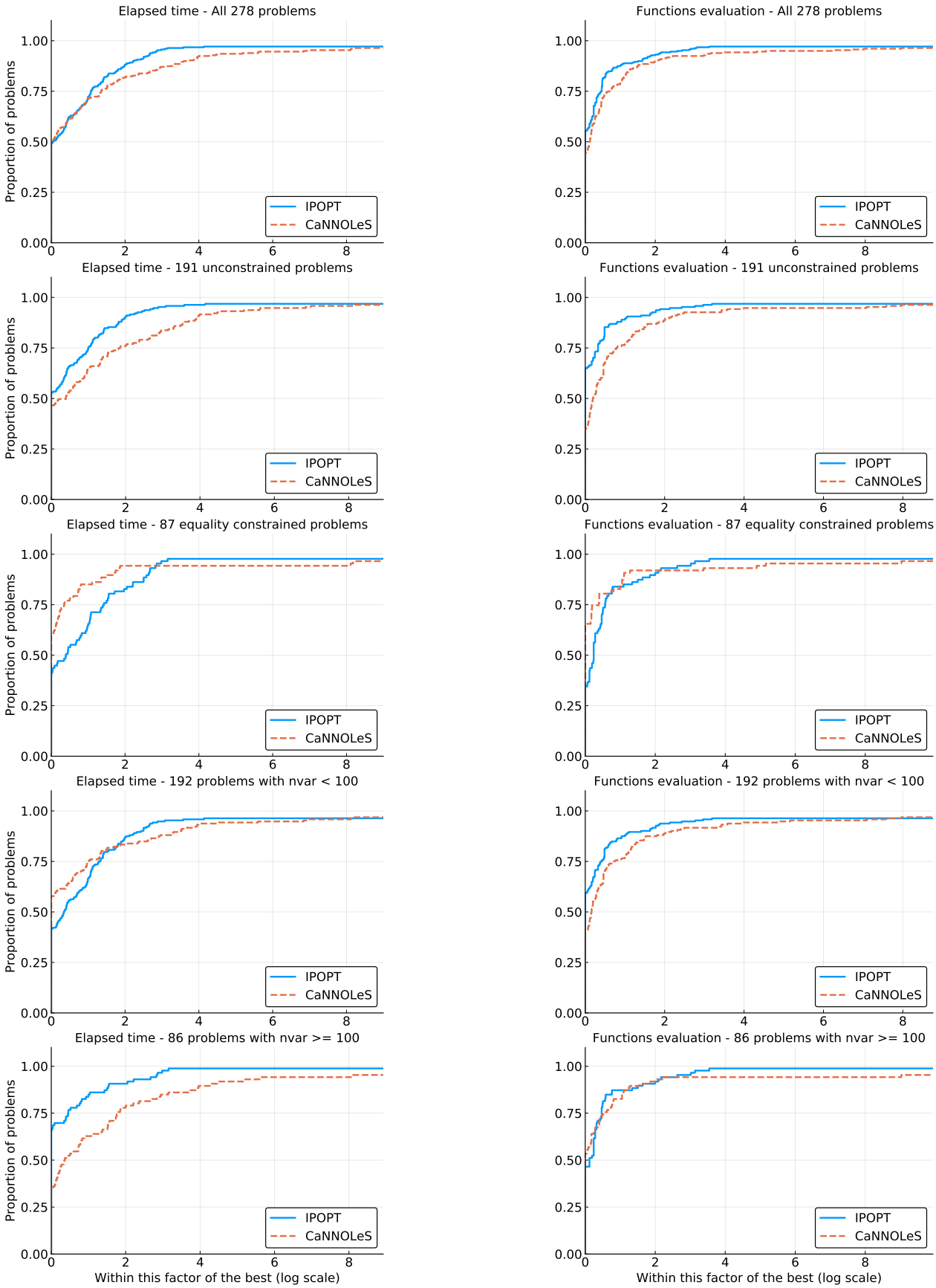

Figure 1: Performance profile on 278 problems in elapsed time (left) and function evaluations (right) comparing CaNNOLeS (dashed red curve) with IPOPT (continuous blue curve).

with starting point $x_0 = (-1.2, 1.0, -1.2, \ldots, 1.0)$, and

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n/2-1} (\exp(x_{2i-1}) - x_{2i})^4 + 100(x_{2i} - x_{2i+1})^6 + \\ & + \tan^4(x_{2i+1} - x_{2i+2}) + x_{2i-1}^8 + (x_{2i+2} - 1)^2 \\ \text{subject to} \quad & 8x_{k+1}(x_{k+1}^2 - x_k) - 2(1 - x_{k+1}) + 4(x_{k+1} - x_{k+2}^2), \\ & k = 1, \ldots, n-2, \end{aligned} \tag{14}$$

with $(x_0)_i = 1$ if $\mod(i, 4) = 1$ and $(x_0)_i = 2$ otherwise. We choose $n = 25$ variables.

We consider the following strategy to solve each of these problems: starting from x_0, we solve the problem in Float32 (single precision) up to tolerance $\epsilon > 0$, and we use the solution obtained to warm start the solution in Float64 (double precision), up to a tolerance of 10^{-12}. When our implementation runs with Float32 data, (7) is factorized by calling the single precision subroutines of MA57.

Tables 1 and 2 report the results. The column ϵ_{32} is the base-10 logarithm of the tolerance used to solve the problem in Float32. The second and third columns are the number of functions evaluations and the optimality residual at the solution of the Float32 execution. The fourth and fifth columns are the same statistics for the Float64 execution. The sixth column is the weighted sum of functions evaluations, where we follow Gratton et al. (2018) and give each evaluation in Float64 four times the cost of an evaluation in Float32. The last column is the total elapsed time. The first row of each table corresponds to solving the problem in Float64 only.

Tables 1 and 2 show that there may be much to gain by initiating the minimization in a lower precision and gradually transitioning to higher-precision floating-point systems. The elapsed time decreases slightly on (13) but drops by a factor of two on (14). Although the total number of function evaluations remains constant for all choices of ϵ_{32}, the weighted effort measure decreases by a factor of almost 2 for (13) and almost 5 for (14) with $\epsilon_{32} = 10^{-3}$. On typical computers, where Float32 and Float64 calculations are both implemented at the hardware level, this has important implications in terms of energy savings.

Table 1: Multiple precision results for (13).

ϵ_{32}	f_{32}	$\|G_{32}\|_*$	f_{64}	$\|G_{64}\|_*$	W	Time
			60	1.3e−14	240	0.04
−1	44	9.6e−05	20	1.3e−14	124	0.03
−2	44	9.6e−05	20	1.3e−14	124	0.03
−3	44	9.6e−05	20	1.3e−14	124	0.03

Table 2: Multiple precision results for (14).

ϵ_{32}	f_{32}	$\|G_{32}\|_*$	f_{64}	$\|G_{64}\|_*$	W	Time
			300	1.8e−14	1200	0.12
−1	162	5.6e−02	28	2.4e−13	274	0.07
−2	170	7.0e−04	20	5.5e−14	250	0.06
−3	170	7.0e−04	20	5.5e−14	250	0.06

One point of consideration is that Julia supports additional floating-point types. Float16 is available in the standard Julia library although calculations in Float16 are performed at the software level. An interface to the arbitrary precision library GNU MPFR provides a BigFloat data type, for which the number of bits used to store real numbers can be specified by the user. Again, calculations in BigFloat are performed at the software level. In particular, BigFloat allows us to simulate calculations in quadruple precision. Various external Julia packages implement or interface specialized implementations of quadruple precision reals, and those may perform better than BigFloat although they are also implemented at the software level. For instance, DecFP.jl[6] interfaces the Intel Decimal Floating-Point Math Library[7], which implements the IEEE

[6] https://github.com/JuliaMath/DecFP.jl

[7] https://software.intel.com/en-us/articles/intel-decimal-floating-point-math-library

754-2008 Decimal Floating-Point Arithmetic specification. DoubleDouble.jl[8] implements accuracy extension as described by Dekker (1971), which predates the IEEE-754 standard.

Thanks to multiple dispatch, our implementation is able to run in any of those floating-point types without extra programming effort. However, we may no longer use MA57, which is limited to Float32 and Float64. Our package LDLFactorizations.jl is a translation of the concise indefinite LDL factorization of Tim Davis (2006). Though LDL may not be as robust as MA57, it allows us to use various floating-point systems. On specialized platforms, one would use optimized linear algebra kernels resting on floating-point calculations occurring at the hardware level.

For illustration, we report results using the BigFloat data type with 128 bits reals, and use LDLFactorizations.jl as the linear solver. Our stopping tolerance in BigFloat128 is set to 10^{-20} and we consider that an evaluation in BigFloat128 costs 16 evaluations in Float32.

Tables 3 and 4 document our results. The ϵ_{32} and ϵ_{64} values are the base-10 logarithm of the tolerance used in the Float32 solution and the Float64 solution. The largest savings occur for $\epsilon_{32} = 10^{-3}$ and $\epsilon_{64} = 10^{-12}$ on (13), for which the elapsed time decreases by a factor of 6.5 and the weighted effort by 3 compared to a solve entirely performed in BigFloat128. The final optimality residual also improves by 6 orders of magnitude. Note that 12 function evaluations correspond to 2 or 3 iterations of Algorithm 2.1, so that superlinear convergence is taking place. On (14), the largest savings also occur for $\epsilon_{32} = 10^{-1}$ and $\epsilon_{64} = 10^{-12}$ with a decrease in elapsed time of a factor of almost 30, in weighted effort of a factor of almost 9. Although computations in BigFloat are by no means efficient, these results illustrate the benefits of working with multiple floating-point systems simultaneously.

Table 3: Multiple precision results in Float32, Float64, and BigFloat128 on (13).

ϵ_{32}	ϵ_{64}	f_{32}	$\|G_{32}\|_*$	f_{64}	$\|G_{64}\|_*$	f_{128}	$\|G_{128}\|_*$	W	Time
						60	1.6e−22	960	8.60
−1	−6	44	9.5e−05	12	3.0e−09	20	2.2e−34	412	2.48
−1	−8	44	9.5e−05	12	3.0e−09	20	2.2e−34	412	2.75
−1	−10	44	9.5e−05	20	1.2e−14	12	4.0e−28	316	1.54
−1	−12	44	9.5e−05	20	1.2e−14	12	4.0e−28	316	1.34
−3	−6	44	9.5e−05	12	3.0e−09	20	2.2e−34	412	2.51
−3	−8	44	9.5e−05	12	3.0e−09	20	2.2e−34	412	2.57
−3	−10	44	9.5e−05	20	1.2e−14	12	4.0e−28	316	1.37
−3	−12	44	9.5e−05	20	1.2e−14	12	4.0e−28	316	1.31

Table 4: Multiple precision results in Float32, Float64, and BigFloat128 on (14).

ϵ_{32}	ϵ_{64}	f_{32}	$\|G_{32}\|_*$	f_{64}	$\|G_{64}\|_*$	f_{128}	$\|G_{128}\|_*$	W	Time
						246	2.4e−22	3936	20.47
−1	−6	162	5.6e−02	20	1.0e−06	20	1.8e−26	562	1.41
−1	−8	162	5.6e−02	28	2.5e−13	12	1.3e−26	466	0.71
−1	−10	162	5.6e−02	28	2.5e−13	12	1.3e−26	466	0.71
−1	−12	162	5.6e−02	28	2.5e−13	12	1.3e−26	466	0.70
−3	−6	170	7.0e−04	12	5.4e−07	20	1.8e−27	538	1.35
−3	−8	170	7.0e−04	20	5.6e−14	12	8.9e−28	442	0.73
−3	−10	170	7.0e−04	20	5.6e−14	12	8.9e−28	442	0.72
−3	−12	170	7.0e−04	20	5.6e−14	12	8.9e−28	442	0.69

6 Discussion and future work

We presented a new algorithm for constrained nonlinear least squares, with promising performance.

Multiprecision support will have increasing importance in large-scale applications in coming years. For instance, training deep learning networks is essentially constrained by memory use and computation time,

[8] https://github.com/JuliaMath/DoubleDouble.jl

and low-precision and mixed-precision support are paramount (Gupta et al., 2015; Micikevicius et al., 2017). At the other end of the spectrum, high-accuracy solutions are desirable in certain applications, including metabolic expression models in systems biology (Ma and Saunders, 2014).

Our implementation can accommodate problems in different floating-point systems, a property that can be exploited for solution on GPUs or heterogeneous architectures. Although Algorithm 2.1 does not feature anything related to using several floating-point systems, the simple warmstart strategy that we illustrate shows the potential benefits. More sophisticated strategies are possible, for instance by making provision for dynamic accuracy in the objective function value and/or the gradient (Bellavia et al., 2018; Gratton and Toint, 2018), and those are the subject of ongoing research.

Several extensions of the present research are of interest. Firstly, minimizing a composite objective of the form $f(x) = g(x) + \frac{1}{2}\|F(x)\|^2$ is possible by merging the techniques developed in the present research with those of Arreckx and Orban (2018). Secondly, the solution of bound-constrained problems, and, by extension, problems with inequality constraints, is relevant in numerous applications. Finally, our method can be extended to a factorization-free implementation, making it suitable for large-scale problems by following the strategy of Arreckx and Orban (2018). The key property to be explored is that the matrix of (7) can be made symmetric and quasi-definite (Vanderbei, 1995) if ρ_k is large enough, or more practically, by using a positive definite H_k, and this would allow using one of the iterative methods described by Orban and Arioli (2017). In particular, a limited-memory variant of the approximation given by Dennis et al. (1981) should satisfy Assumption 8, (Dennis and Walker, 1981).

References

Ansari MR, Mahdavi-Amiri N (2014) A robust combined trust region–line search exact penalty projected structured scheme for constrained nonlinear least squares. Optim Methods Softw 30(1):162–190, DOI: 10.1080/10556788.2014.909970

Armand P, Omheni R (2015) A globally and quadratically convergent primal–dual augmented lagrangian algorithm for equality constrained optimization. Optimization Methods and Software 32(1):1–21, DOI: 10.1080/10556788.2015.1025401, URL https://doi.org/10.1080/10556788.2015.1025401

Armand P, Benoist J, Orban D (2013) From global to local convergence of interior methods for nonlinear optimization. Optim Methods Softw 28(5):1051–1080, DOI: 10.1080/10556788.2012.668905

Arreckx S, Orban D (2018) A regularized factorization-free method for equality-constrained optimization. SIAM J Optim 28(2):1613–1639, DOI: 10.1137/16M1088570

Bellavia S, Gurioli G, Morini B, Toint PhL (2018) Deterministic and stochastic inexact regularization algorithms for nonconvex optimization with optimal complexity. Tech. rep., University of Namur, URL http://www.optimization-online.org/DB_HTML/2018/11/6919.html

Bidabadi N (2017) Using a spectral scaling structured BFGS method for constrained nonlinear least squares. Optim Methods Softw pp 1–14, DOI: 10.1080/10556788.2017.1385073

Bidabadi N, Mahdavi-Amiri N (2013) Superlinearly convergent exact penalty methods with projected structured secant updates for constrained nonlinear least squares. J Optim Theory and Applics 162(1):154–190, DOI: 10.1007/s10957-013-0438-x

Birgin E, Martínez J (2014) Practical Augmented Lagrangian Methods for Constrained Optimization. SIAM, Philadelphia, USA, DOI: 10.1137/1.9781611973365

Davis TA (2005) Algorithm 849: A concise sparse Cholesky factorization package. ACM Trans Math Software 31(4):587–591, DOI: 10.1145/1114268.1114277

Dehghani M, Lambe A, Orban D (2019) A regularized interior-point method for constrained linear least squares. INFOR: Information Systems and Operational Research DOI: 10.1080/03155986.2018.1559428, published online: 19 Feb 2019

Dekker TJ (1971) A floating-point technique for extending the available precision. Numer Math 18(3):224–242, DOI: 10.1007/BF01397083

Dennis JE, Moré JJ (1977) Quasi-Newton methods, motivation and theory. SIAM Rev 19(1):46–89, DOI: 10.1137/1019005

Dennis JE, Walker HF (1981) Convergence theorems for least-change secant update methods. SIAM J Numer Anal 18(6):949–987

Dennis JE Jr, Gay DM, Welsch RE (1981) An adaptive nonlinear least-squares algorithm. ACM Trans Math Software 7(3):348–368, DOI: 10.1145/355958.355965

Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profile. Math Program, Series A 91(2):201–213, DOI: 10.1007/s101070100263

Duff IS (2004) MA57—a code for the solution of sparse symmetric definite and indefinite systems. ACM Trans Math Software 30(2):118–144, DOI: 10.1145/992200.992202

Dunning I, Huchette J, Lubin M (2017) JuMP: A modeling language for mathematical optimization. SIAM Rev 59(2):295–320, DOI: 10.1137/15M1020575

Fernanda M, Costa P, Fernandes EMGP (2005) A primal-dual interior-point algorithm for nonlinear least squares constrained problems. Top 13(1):145–166, DOI: 10.1007/bf02578992

Fernández D, Solodov M (2014) STABILIZED SEQUENTIAL QUADRATIC PROGRAMMING: A SURVEY. Pesquisa Operacional 34(3):463–479, DOI: 10.1590/0101-7438.2014.034.03.0463

Fernández D, Pilotta EA, Torres GA (2012) An inexact restoration strategy for the globalization of the sSQP method. Comput Optim Appl 54(3):595–617, DOI: 10.1007/s10589-012-9502-y

Forsgren A (2002) Inertia-controlling factorizations for optimization algorithms. Appl Numer Math 43(1-2):91–107, DOI: 10.1016/s0168-9274(02)00119-8

Friedlander MP, Orban D (2012) A primal-dual regularized interior-point method for convex quadratic programs. Math Program Comp 4(1):71–107, DOI: 10.1007/s12532-012-0035-2

Gill PE, Kungurtsev V, Robinson DP (2016a) A stabilized SQP method: global convergence. IMA Journal of Numerical Analysis 37(1):407–443, DOI: 10.1093/imanum/drw004

Gill PE, Kungurtsev V, Robinson DP (2016b) A stabilized SQP method: superlinear convergence. Math Program 163(1-2):369–410, DOI: 10.1007/s10107-016-1066-7

Gould NI, Orban D, Toint PhL (2015) CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput Optim Appl 60(3):545–557, DOI: 10.1007/s10589-014-9687-3

Gratton S, Toint PhL (2018) A note on solving nonlinear optimization problems in variable precision. Tech. rep., University of Namur, URL http://www.optimization-online.org/DB_HTML/2018/12/6982.html

Gratton S, Simon E, Toint PhL (2018) Minimizing convex quadratic with variable precision Krylov methods. Tech. rep., University of Namur, URL http://arxiv.org/abs/1807.07476

Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P (2015) Deep learning with limited numerical precision. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning, JMLR.org, ICML'15, vol 37, pp 1737–1746, URL http://dl.acm.org/citation.cfm?id=3045118.3045303

Hock W, Schittkowski K (1981) Test Examples for Nonlinear Programming Codes. Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-48320-2

HSL (2007) The HSL Mathematical Software Library. STFC Rutherford Appleton Laboratory, http://www.hsl.rl.ac.uk

Izmailov AF, Solodov MV, Uskov EI (2015) Combining stabilized SQP with the augmented lagrangian algorithm. Comput Optim Appl 62(2):405–429, DOI: 10.1007/s10589-015-9744-6

Izmailov AF, Solodov MV, Uskov EI (2016) Globalizing stabilized sequential quadratic programming method by smooth primal-dual exact penalty function. J Optim Theory and Applics 169(1):148–178, DOI: 10.1007/s10957-016-0889-y

Li Z, Osborne M, Prvan T (2002) Adaptive algorithm for constrained least-squares problems. J Optim Theory and Applics 114(2):423–441, DOI: 10.1023/A:1016043919978

Li ZF (2000) On limited memory SQP methods for large scale constrained nonlinear least squares problems. ANZIAM Journal 42:900, DOI: 10.21914/anziamj.v42i0.627

Lukšan L, Vlček J (1999) Sparse and partially separable test problems for unconstrained and equality constrained optimization. Tech. Rep. V-767, Prague: ICS AS CR

Ma D, Saunders MA (2014) Solution of multiscale linear programs using quad precision. SOL Technical Report 2014-1, Stanford University

Mahdavi-Amiri N, Ansari MR (2012) A superlinearly convergent penalty method with nonsmooth line search for constrained nonlinear least squares. Sultan Qaboos University Journal for Science [SQUJS] 17(1):103, DOI: 10.24200/squjs.vol17iss1pp103-124

Mahdavi-Amiri N, Bartels RH (1989) Constrained nonlinear least squares: an exact penalty approach with projected structured quasi-Newton updates. ACM Trans Math Software 15(3):220–242, DOI: 10.1145/66888.66891

Mahdavi-Amiri N, Bidabadi N (2012) Constrained nonlinear least squares: A superlinearly convergent projected structured secant method. International Journal of Electrical and Computer Systems 1(1), DOI: 10.11159/ijecs.2012.001

Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, Wu H (2017) Mixed precision training. ICLR 2018, URL https://arxiv.org/abs/1710.03740v3, arXiv e-Print, 1710.03740

Moré JJ, Garbow BS, Hillstrom KE (1981) Testing unconstrained optimization software. ACM Trans Math Software 7(1):17–41, DOI: 10.1145/355934.355936

Orban D, Arioli M (2017) Iterative Solution of Symmetric Quasi-Definite Linear Systems. SIAM Spotlights, Society for Industrial and Applied Mathematics Philadelphia, DOI: 10.1137/1.9781611974737

Schittkowski K (1988) Solving constrained nonlinear least squares problems by a general purpose SQP-method. In: Hoffmann KH, Zowe J, Hiriart-Urruty JB, Lemaréchal C (eds) Trends in Mathematical Optimization: 4th French-German Conference on Optimization, Birkhäuser Basel, Basel, pp 295–309, DOI: 10.1007/978-3-0348-9297-1˙19

Schittkowski K (2007) NLPLSQ: A Fortran implementation of an SQP-Gauss-Newton algorithm for least squares optimization. Tech. rep., Department of Computer Science, University of Bayreuth

Vanderbei RJ (1995) Symmetric quasidefinite matrices. SIAM J Optim 5(1):100–113, DOI: 10.1137/0805005

Wächter A, Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math Program 106(1):25–57, DOI: 10.1007/s10107-004-0559-y

Wright SJ (1998) Superlinear Convergence of a Stabilized SQP Method to a Degenerate Solution. Comput Optim Appl 11(3):253–275, DOI: 10.1023/A:1018665102534