



Research article

A truncated hyperbolic smoothing Newton method for finite minimax problems

Wenzhe Zhao and Liping Tang*

National Center for Applied Mathematics in Chongqing, Chongqing Normal University, Chongqing 401331, China

* **Correspondence:** Email: tanglipings@163.com, tanglipings@cqnu.edu.cn.

Abstract: Based on the hyperbolic smoothing function and truncation strategy, this paper proposes a truncated hyperbolic smoothing Newton method for finite minimax problems with a large number of component functions. Gaps between the original and truncated hyperbolic smoothing functions are established in terms of the function values, gradients, and Hessian matrices, respectively, which in turn lead to a rule for selecting the truncation parameter. Based on this rule, the global convergence and the local quadratic convergence rate of the method are established. Numerical experiments demonstrate that, under the same accuracy requirement, the proposed method substantially reduces CPU time and is generally more efficient than both the exponential smoothing method and the untruncated smoothing method.

Keywords: finite minimax problems; unconstrained optimization; hyperbolic smoothing; truncated Newton method

Mathematics Subject Classification: 90C47, 90C30, 65K05

1. Introduction

In this paper, we consider the following unconstrained finite minimax problem:

$$\min_{x \in \mathbb{R}^n} f(x) := \max_{i \in I} f_i(x), \quad (1.1)$$

where each function $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable, and $I := \{1, 2, \dots, m\}$. From a theoretical point of view, minimax optimization problems are closely related to many optimization problems, including nonlinear programming [1], systems of nonlinear equations [2], linear complementarity problems [3], and multiobjective optimization problems [4]. Such problems also arise in a wide range of applications, such as structural optimization [5], data fitting [6], location problems [7–9], and resource allocation [10].

Existing methods for finite minimax problems can be broadly divided into two classes. The first class consists of nonsmooth optimization methods, including subgradient methods [11] and bundle methods [12, 13]. These methods usually employ negative subgradient directions as descent directions. However, directly solving the nonsmooth problem may fail to guarantee convergence to a stationary point in practical computations. To overcome this limitation, smoothing methods have been developed. The basic idea is to approximate f by a family of smooth functions and thereby transform (1.1) into a sequence of smooth parameterized optimization problems. Because the smoothed functions provide first- and second-order information, mature smooth optimization techniques can be applied, leading to stronger convergence guarantees and faster convergence rates than direct nonsmooth methods. In recent years, several strategies have also been investigated for large-scale or related minimax optimization, including active-set smoothing methods [14] and smoothing schemes combined with quasi-Newton solvers [15]. Their common idea is to avoid full-scale computations at each iteration by selecting relevant component functions or by approximating curvature information without forming exact Hessians, respectively.

According to the smoothing strategy, smoothing methods for finite minimax problems can be divided into local smoothing methods and global smoothing methods. Local smoothing methods [15–18] smooth the objective function only near kink points, namely points at which the objective function is nondifferentiable. Nevertheless, local smoothing may generate spurious local minimizers caused by combinations of adjacent shallow local minimizers. To avoid this drawback, global smoothing methods have received considerable attention.

Among global smoothing methods, the exponential smoothing function has been widely studied [19–23]. Based on the maximum-entropy principle, Li [19] proposed the aggregate smoothing function

$$\mu \log \left(\sum_{i=1}^m e^{f_i(x)/\mu} \right),$$

where μ is the smoothing parameter. Polak et al. [20] proposed an adaptive smoothing rule for finite minimax problems, and Polak et al. [21] incorporated an active-index strategy into the smoothing framework. Guo and Wan [23] proposed a smoothing three-term conjugate gradient algorithm for large-scale minimax problems. However, the exponential smoothing function may suffer from numerical overflow, especially when the number of component functions is large. Xiao and Yu [22] proposed a truncated aggregate smoothing Newton–Armijo method that smooths only the component functions close to f at the current iteration, thereby reducing the computational cost. Nevertheless, this truncation does not eliminate the overflow risk inherent in the exponential function.

Another important global smoothing tool is the hyperbolic smoothing function [24, 25]

$$\varphi_{\tau}(x) = \frac{x + \sqrt{x^2 + \tau^2}}{2}. \quad (1.2)$$

It is a smooth approximation of the plus function $\max\{0, x\} = (x + |x|)/2$. Bagirov et al. [24, 25] introduced this function for minimax problems and compared the resulting method with local smoothing, exponential smoothing, and nonlinear programming approaches. The numerical results indicate that the hyperbolic smoothing approach is robust for nonconvex small-scale problems and is more numerically stable than exponential smoothing. In recent years, hyperbolic smoothing techniques have also been applied to a wider class of nonsmooth optimization models.

Although exponential smoothing functions generally provide tighter approximations to the original nonsmooth function, hyperbolic smoothing functions do not involve exponential terms and therefore avoid exponential overflow in practical computations. Therefore, it is worthwhile to investigate hyperbolic smoothing methods for finite minimax problems.

When the number of component functions is large, owing to the aggregate term in the original hyperbolic smoothing function (3.6), the main computational burden of the original hyperbolic smoothing methods [24, 25] for finite minimax problems arises from the repeated evaluations of the gradients and Hessian matrices of all component functions at each iteration. Consequently, the computational cost of the original methods increases significantly as the number of component functions grows. A similar difficulty arises in exponential smoothing methods. To reduce this cost, a truncation strategy was adopted in [22]. However, exponential smoothing methods require exponential evaluations and may suffer from numerical overflow in practical computations. To overcome the computational limitations of conventional hyperbolic smoothing methods, we incorporate the truncation strategy proposed in [22] into the hyperbolic smoothing framework and develop a truncated hyperbolic smoothing Newton method for finite minimax problems. Compared with the existing nontruncated hyperbolic smoothing methods, the proposed method has three distinctive features: At the modeling level, only the component functions which satisfy the truncation criterion are retained in the construction of the truncated hyperbolic smoothing model; at the theoretical level, gaps between the gradients and Hessian matrices of the original and truncated smoothing functions are established, and these estimates yield a rule for selecting the truncation parameter; and at the algorithmic level, the proposed method reduces the cost of component function evaluations at each iteration and is proved to be globally convergent and to have a local quadratic convergence rate.

The main contributions of this paper are summarized as follows.

- A truncated hyperbolic smoothing Newton method is proposed for solving finite minimax problems with a large number of component functions. Under the rule of choosing the truncation parameter, the global convergence and the local quadratic convergence rate of the method are proved.
- Numerical experiments show that the proposed method can reduce CPU time while maintaining comparable accuracy. By introducing a truncation strategy into the hyperbolic smoothing method, the proposed method reduces the number of component functions, which in turn lowers the cost of computing gradients and Hessian matrices.

The remainder of this paper is organized as follows. Section 2 introduces the notation and basic properties of the hyperbolic smoothing function. Section 3 gives an equivalent reformulation of (1.1) and studies the associated hyperbolic smoothing approximation. Section 4 introduces the truncated smoothing function and presents the corresponding truncation analysis. Section 5 presents the truncated hyperbolic smoothing Newton method and establishes convergence results. Section 6 reports numerical experiments. Section 7 presents a sensitivity analysis of the proposed method with respect to the truncation parameter. Section 8 concludes the paper.

2. Preliminaries

Throughout this paper, $\mathbb{N}$ denotes the set of non-negative integers, and $\mathbb{R}^n$ denotes the n -dimensional Euclidean space. The vector 0_n is the zero vector in $\mathbb{R}^n$. The inner product and Euclidean norm are

denoted by $\langle \cdot, \cdot \rangle$ and $\| \cdot \|$, respectively. For $x^* \in \mathbb{R}^n$ and $\varepsilon > 0$, let

$$O_\varepsilon(x^*) = \{x \in \mathbb{R}^n \mid \|x - x^*\| < \varepsilon\}.$$

For a set $A \subset \mathbb{R}^n$, $\text{co}(A)$ denotes its convex hull, $|A|$ denotes its cardinality, and $\emptyset$ denotes the empty set. For component functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$, define

$$\gamma(x) := \max_{i \in I} \|\nabla f_i(x)\|, \quad \omega(x) := \max_{i \in I} \|\nabla^2 f_i(x)\|.$$

For a square matrix $\mathbf{B}$, $\sigma_{\min}(\mathbf{B})$, $\sigma_{\max}(\mathbf{B})$, and $\text{cond}(\mathbf{B})$ denote its smallest eigenvalue, largest eigenvalue, and condition number, respectively. The active index set of (1.1) at x is

$$I(x) := \{j \in I \mid f_j(x) = f(x)\}. \quad (2.1)$$

For a scalar c , the lower level set of f is

$$\mathcal{L}_{f,c} = \{x \in \mathbb{R}^n \mid f(x) \leq c\}.$$

Definition 1. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The function f is said to be locally Lipschitz continuous at x^* if there exist constants $\varepsilon > 0$ and $L > 0$ such that

$$|f(x) - f(y)| \leq L\|x - y\|, \quad \forall x, y \in O_\varepsilon(x^*).$$

If f is locally Lipschitz continuous at every point of $\mathbb{R}^n$, then f is called a locally Lipschitz function.

Definition 2. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$. A point $x^* \in \mathbb{R}^n$ is called a local minimizer of f if there exists $\varepsilon > 0$ such that

$$f(x^*) \leq f(x), \quad \forall x \in O_\varepsilon(x^*).$$

Definition 3. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be locally Lipschitz continuous. A point $x^* \in \mathbb{R}^n$ is called a Clarke stationary point of f if

$$0_n \in \partial f(x^*),$$

where $\partial f(x^*)$ denotes the Clarke subdifferential. If f is differentiable at x^* , then $\partial f(x^*) = \{\nabla f(x^*)\}$.

For the plus function

$$\theta(x) = \max\{0, x\}, \quad x \in \mathbb{R}, \quad (2.2)$$

we consider the hyperbolic smoothing function proposed in [24, 25]:

$$\varphi_\tau(x) = \frac{x + \sqrt{x^2 + \tau^2}}{2}, \quad x \in \mathbb{R}, \quad (2.3)$$

where $\tau > 0$.

Proposition 1 ([25]). The function φ_τ has the following properties:

- (i) φ_τ is monotonically increasing, convex, and infinitely differentiable;
- (ii) for any $x \in \mathbb{R}$, $\theta(x) < \varphi_\tau(x) \leq \theta(x) + \tau/2$.

3. Hyperbolic smoothing

In this section, we present a reformulation of the minimax problem and prove, by a new approach, several results established in [25] for the reformulated problem.

Problem (1.1) can be equivalently written as

$$\min_{(x,t) \in \mathbb{R}^{n+1}} \{t \mid f_i(x) - t \leq 0, i \in I\}. \quad (3.1)$$

Using an exact penalty formulation, (3.1) is related to

$$\min_{(x,t) \in \mathbb{R}^{n+1}} F_\xi(x, t) = t + \xi \sum_{i \in I} \max\{0, f_i(x) - t\}. \quad (3.2)$$

In what follows, it is sufficient to consider $\xi = 1$, namely

$$\min_{(x,t) \in \mathbb{R}^{n+1}} F(x, t) := t + \sum_{i \in I} \max\{0, f_i(x) - t\}. \quad (3.3)$$

Proposition 2 ([25]). *Let f be defined by (1.1). Then,*

$$f(x) = \min_{t \in \mathbb{R}} F(x, t).$$

Remark 1. *Proposition 2 states that minimizing f is equivalent to minimizing $F(x, t)$ with $t = f(x)$.*

For a given $(x, t) \in \mathbb{R}^{n+1}$, decompose I as

$$I = I_1 \cup I_2 \cup I_3,$$

where

$$I_1 = \{i \in I \mid f_i(x) < t\}, \quad I_2 = \{i \in I \mid f_i(x) = t\}, \quad I_3 = \{i \in I \mid f_i(x) > t\}.$$

Let $H_i(x, t) = \max\{0, f_i(x) - t\}$. Then,

$$\partial H_i(x, t) = \begin{cases} \{0_{n+1}\}, & i \in I_1, \\ \text{co}\{0_{n+1}, (\nabla f_i(x), -1)\}, & i \in I_2, \\ \{(\nabla f_i(x), -1)\}, & i \in I_3. \end{cases}$$

Hence,

$$\partial F(x, t) = \{(0_n, 1)\} + \sum_{i \in I_2} \text{co}\{0_{n+1}, (\nabla f_i(x), -1)\} + \sum_{i \in I_3} (\nabla f_i(x), -1). \quad (3.4)$$

Proposition 3. *The following statements hold.*

- (i) *If $x^* \in \mathbb{R}^n$ is a stationary point of f , then $(x^*, f(x^*))$ is a stationary point of F .*
- (ii) *If (x^*, t^*) is a stationary point of F , then x^* is a stationary point of f .*

Proof. (i) Let x^* be a stationary point of f . Set

$$t^* = f(x^*) = \max_{i \in I} f_i(x^*).$$

Then, $t^* \geq f_i(x^*)$ for all $i \in I$, and

$$I_3^* = I_3(x^*, t^*) = \emptyset.$$

Moreover, $I_2^* = I_2(x^*, t^*) \neq \emptyset$ because at least one component function is active at x^* ; otherwise, $f_i(x^*) < t^*$ for all $i \in I$, contradicting $t^* = f(x^*)$. Therefore, the subdifferential of F at (x^*, t^*) is

$$\partial F(x^*, t^*) = \{(0_n, 1)\} + \sum_{i \in I_2^*} \text{co}\{0_{n+1}, (\nabla f_i(x^*), -1)\}.$$

For the point (x^*, t^*) , we have $I(x^*) = I_2^*$. Because x^* is a stationary point of f , it follows that

$$0_n \in \text{co}\{\nabla f_i(x^*) \mid i \in I_2^*\},$$

which implies that there exist $\lambda_i \geq 0$ ($i \in I_2^*$) and $i \in I_2^*$ such that

$$\sum_{i \in I_2^*} \lambda_i = 1, \quad \sum_{i \in I_2^*} \lambda_i \nabla f_i(x^*) = 0_n. \quad (3.5)$$

It follows that

$$\begin{aligned} \partial F(x^*, t^*) &\ni (0_n, 1) + \sum_{i \in I_2^*} (1 - \lambda_i) 0_{n+1} + \sum_{i \in I_2^*} \lambda_i (\nabla f_i(x^*), -1) \\ &= \left(\sum_{i \in I_2^*} \lambda_i \nabla f_i(x^*), 1 - \sum_{i \in I_2^*} \lambda_i \right) = 0_{n+1}. \end{aligned}$$

Thus, (x^*, t^*) is a stationary point of F .

- (ii) Let (x^*, t^*) be a stationary point of F . According to the definitions of $I_1^* = I_1(x^*, t^*)$, $I_2^* = I_2(x^*, t^*)$, and $I_3^* = I_3(x^*, t^*)$, we consider the following three cases.

Case 1. Suppose that $t^* > f(x^*) = \max_{i \in I} f_i(x^*)$. Then, $t^* > f_i(x^*)$ for all $i \in I$, so that $I_2^* = I_3^* = \emptyset$. By (3.4),

$$\partial F(x^*, t^*) = \{(0_n, 1)\}.$$

Thus, $0_{n+1} \notin \partial F(x^*, t^*)$, contradicting the stationarity of (x^*, t^*) .

Case 2. Suppose that $t^* = f(x^*)$. Then, $t^* \geq f_i(x^*)$ for all $i \in I$. Hence, $I_2^* \neq \emptyset$, and $I_3^* = \emptyset$. From (3.4), it follows that

$$\partial F(x^*, t^*) = \{(0_n, 1)\} + \sum_{i \in I_2^*} \text{co}\{0_{n+1}, (\nabla f_i(x^*), -1)\}.$$

Because $0_{n+1} \in \partial F(x^*, t^*)$, there exist $\lambda_i \in [0, 1]$, $i \in I_2^*$, such that

$$0_{n+1} = (0_n, 1) + \sum_{i \in I_2^*} (1 - \lambda_i) 0_{n+1} + \sum_{i \in I_2^*} \lambda_i (\nabla f_i(x^*), -1),$$

which yields

$$\sum_{i \in I_2^*} \lambda_i \nabla f_i(x^*) = 0_n, \quad \sum_{i \in I_2^*} \lambda_i = 1.$$

Therefore, $0_n \in \text{co}\{\nabla f_i(x^*) \mid i \in I_2^*\} = \partial f(x^*)$, and x^* is a stationary point of f .

Case 3. Suppose that $t^* < f(x^*)$. Then, $I_3^* \neq \emptyset$, and $|I_3^*| \geq 1$. In this case, (3.4) gives

$$\partial F(x^*, t^*) = \left\{ \left(\sum_{i \in I_3^*} \nabla f_i(x^*), 1 - |I_3^*| \right) \right\} + \sum_{i \in I_2^*} \text{co}\{0_{n+1}, (\nabla f_i(x^*), -1)\}.$$

Because (x^*, t^*) is stationary for F , there exist $\lambda_i \in [0, 1]$, $i \in I_2^*$ such that

$$0_{n+1} = \left(\sum_{i \in I_3^*} \nabla f_i(x^*) + \sum_{i \in I_2^*} \lambda_i \nabla f_i(x^*), 1 - |I_3^*| - \sum_{i \in I_2^*} \lambda_i \right).$$

Hence,

$$0 = 1 - |I_3^*| - \sum_{i \in I_2^*} \lambda_i \leq 1 - |I_3^*| \leq 0.$$

It follows that $|I_3^*| = 1$, and $\sum_{i \in I_2^*} \lambda_i = 0$. Thus, I_3^* is a singleton. Let $I_3^* = \{j\}$. Then, the first component of the above equality gives

$$\nabla f_j(x^*) = 0_n.$$

Because $I(x^*) = I_3^* = \{j\}$ in this case, f is differentiable at x^* , and $0_n \in \partial f(x^*)$. Therefore, x^* is a stationary point of f .

□

Remark 2. The proof of Proposition 3 shows that if $(x^*, t^*) \in \mathbb{R}^{n+1}$ is a stationary point of F , then only the cases $t^* = f(x^*)$ and $t^* < f(x^*)$ can occur. In the second case, the set I_3^* degenerates into a singleton, and hence, $I(x^*)$ is also a singleton. Thus, f is differentiable at x^* . In minimax problems, most stationary points are kink points, that is, points where f is nondifferentiable. This indicates that for most stationary points of minimax problems, one usually has $t^* = f(x^*)$. Consequently, almost all stationary points (x^*, t^*) of F can be represented as $(x^*, f(x^*))$.

Proposition 4 ([25]). (i) If $x^* \in \mathbb{R}^n$ is a local minimizer of f , then $(x^*, f(x^*))$ is a local minimizer of F .

(ii) If (x^*, t^*) is a local minimizer of F , then x^* is a local minimizer of f .

Applying (2.3) to (3.3), define the hyperbolic smoothing approximation

$$\Phi_\tau(x, t) = t + \sum_{i \in I} \frac{f_i(x) - t + \sqrt{(f_i(x) - t)^2 + \tau^2}}{2}, \quad \tau > 0. \quad (3.6)$$

Its gradient is

$$\nabla \Phi_\tau(x, t) = (G_{1\tau}(x, t), G_{2\tau}(x, t)), \quad (3.7)$$

where

$$G_{1\tau}(x, t) = \frac{1}{2} \sum_{i \in I} (1 + \beta_{i\tau}(x, t)) \nabla f_i(x), \quad (3.8)$$

$$G_{2\tau}(x, t) = 1 - \frac{|I|}{2} - \frac{1}{2} \sum_{i \in I} \beta_{i\tau}(x, t), \quad (3.9)$$

with

$$\beta_{i\tau}(x, t) = \frac{f_i(x) - t}{\sqrt{(f_i(x) - t)^2 + \tau^2}}. \quad (3.10)$$

Remark 3. Although the truncated aggregate exponential smoothing method has a certain theoretical advantage in approximation accuracy, the hyperbolic smoothing function has a simpler computational structure and requires no exponential evaluations. Moreover, the truncation strategy reduces the number of component gradients and Hessian matrices evaluated at each iteration. These computational advantages are reflected in the numerical results in Section 6: Tables 1–3 show that the truncated hyperbolic smoothing Newton method proposed in this paper generally achieves objective values comparable to those obtained by the truncated aggregate exponential smoothing method proposed in [22] while requiring less CPU time, especially when the number of component functions is large. Tables 4 and 5 further indicate that this advantage is generally maintained as the number of variables or component functions increases.

Proposition 5 ([25]). For any $x \in \mathbb{R}^n$ and $t \in \mathbb{R}$,

$$0 < \Phi_\tau(x, t) - F(x, t) \leq \frac{|I|\tau}{2} = \frac{m\tau}{2}.$$

Proposition 6 ([25]). If $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$, $i \in I$ are continuously differentiable, and ∇f_i are locally Lipschitz continuous, then for each fixed $\tau > 0$, $\nabla \Phi_\tau$ is locally Lipschitz continuous.

Proposition 7 ([25]). Assume that f_i , $i \in I$ are twice continuously differentiable, and let $Q \subset \mathbb{R}^{n+1}$ be bounded. Then, for any $(x, t) \in Q$ and $u = (y, s) \in \mathbb{R}^{n+1}$, there exists $L < \infty$ such that

$$\langle u, \nabla^2 \Phi_\tau(x, t) u \rangle \leq L \|u\|^2.$$

Proposition 8 ([25]). If $v = \lim_{\tau \rightarrow 0} \nabla \Phi_\tau(x, t)$ exists, then $v \in \partial F(x, t)$.

Remark 4. Proposition 7 implies that Φ_τ is L -smooth on bounded sets. Similarly, by replacing Φ_τ with $\bar{\Phi}_\tau$ in Proposition 7, one can show that $\bar{\Phi}_\tau$ is also L -smooth with the same Lipschitz constant L .

4. Truncated hyperbolic smoothing

For $x \in \mathbb{R}^n$ and $\mu > 0$, define the truncated index set

$$\bar{I} = \{i \in I \mid f_i(x) - f(x) \leq \mu\}. \quad (4.1)$$

Because $\bar{I} \subset I$, the truncated hyperbolic smoothing function is

$$\bar{\Phi}_\tau(x, t) = t + \sum_{i \in \bar{I}} \frac{f_i(x) - t + \sqrt{(f_i(x) - t)^2 + \tau^2}}{2}. \quad (4.2)$$

Proposition 9. Let $\bar{\Phi}_\tau$ and Φ_τ be defined in (4.2) and (3.6), respectively. Then, at $(\bar{x}, f(\bar{x}))$, the following estimates hold:

(i) *Function value gap:*

$$0 \leq |\Phi_\tau(\bar{x}, f(\bar{x})) - \bar{\Phi}_\tau(\bar{x}, f(\bar{x}))| \leq \frac{(m-1)(\sqrt{\mu^2 + \tau^2} - \mu)}{2}.$$

(ii) *Gradient gap:*

$$0 \leq \|\nabla \Phi_\tau(\bar{x}, f(\bar{x})) - \nabla \bar{\Phi}_\tau(\bar{x}, f(\bar{x}))\| \leq \frac{1}{2}(m-1) \left(1 - \frac{\mu}{\sqrt{\mu^2 + \tau^2}} \right) (\gamma(\bar{x}) + 1).$$

(iii) *Hessian gap:*

$$\begin{aligned} 0 &\leq \|\nabla^2 \Phi_\tau(\bar{x}, f(\bar{x})) - \nabla^2 \bar{\Phi}_\tau(\bar{x}, f(\bar{x}))\| \\ &\leq \frac{1}{2}(m-1) \left(1 - \frac{\mu}{\sqrt{\mu^2 + \tau^2}} \right) \omega(\bar{x}) + \frac{1}{2}(m-1) \frac{\tau^2}{(\mu^2 + \tau^2)^{3/2}} (\gamma(\bar{x})^2 + 1). \end{aligned}$$

Proof. (i) Clearly, $|I \setminus \bar{I}| \leq |I| - 1 = m - 1$. Because $t = f(\bar{x})$ and $i \in I \setminus \bar{I}$ imply $f(\bar{x}) - f_i(\bar{x}) > \mu$, we have

$$\begin{aligned} &|\Phi_\tau(\bar{x}, f(\bar{x})) - \bar{\Phi}_\tau(\bar{x}, f(\bar{x}))| \\ &= \sum_{i \in I \setminus \bar{I}} \frac{f_i(\bar{x}) - f(\bar{x}) + \sqrt{(f_i(\bar{x}) - f(\bar{x}))^2 + \tau^2}}{2} \\ &\leq (m-1) \frac{\sqrt{\mu^2 + \tau^2} - \mu}{2}. \end{aligned}$$

(ii) From (3.7)–(3.10),

$$\begin{aligned} &\|\nabla \Phi_\tau(\bar{x}, f(\bar{x})) - \nabla \bar{\Phi}_\tau(\bar{x}, f(\bar{x}))\| \\ &= \left\| \left(\frac{1}{2} \sum_{i \in I \setminus \bar{I}} (1 + \beta_{i\tau}(\bar{x}, f(\bar{x}))) \nabla f_i(\bar{x}), -\frac{1}{2} |I \setminus \bar{I}| - \frac{1}{2} \sum_{i \in I \setminus \bar{I}} \beta_{i\tau}(\bar{x}, f(\bar{x})) \right) \right\| \\ &\leq \frac{1}{2} \sum_{i \in I \setminus \bar{I}} (1 + \beta_{i\tau}(\bar{x}, f(\bar{x}))) \|\nabla f_i(\bar{x}), -1\| \\ &\leq \frac{1}{2} (m-1) \left(1 - \frac{\mu}{\sqrt{\mu^2 + \tau^2}} \right) (\gamma(\bar{x}) + 1). \end{aligned}$$

(iii) By the definition of the Hessian matrix, the difference between the full and truncated Hessians

is contributed only by indices in $I \setminus \bar{I}$. Hence,

$$\begin{aligned}
 & \|\nabla^2 \Phi_\tau(\bar{x}, f(\bar{x})) - \nabla^2 \bar{\Phi}_\tau(\bar{x}, f(\bar{x}))\| \\
 &= \left\| \frac{1}{2} \sum_{i \in I \setminus \bar{I}} (1 + \beta_{i\tau}(\bar{x}, f(\bar{x}))) \begin{pmatrix} \nabla^2 f_i(\bar{x}) & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{2} \sum_{i \in I \setminus \bar{I}} \mu_{i\tau}(\bar{x}, f(\bar{x})) (\nabla f_i(\bar{x}), -1)^T (\nabla f_i(\bar{x}), -1) \right\| \\
 &\leq \frac{1}{2} \sum_{i \in I \setminus \bar{I}} (1 + \beta_{i\tau}(\bar{x}, f(\bar{x}))) \omega(\bar{x}) + \frac{1}{2} \sum_{i \in I \setminus \bar{I}} \mu_{i\tau}(\bar{x}, f(\bar{x})) (\gamma(\bar{x})^2 + 1) \\
 &\leq \frac{1}{2} (m-1) \left(1 - \frac{\mu}{\sqrt{\mu^2 + \tau^2}} \right) \omega(\bar{x}) + \frac{1}{2} (m-1) \frac{\tau^2}{(\mu^2 + \tau^2)^{3/2}} (\gamma(\bar{x})^2 + 1),
 \end{aligned}$$

where

$$\mu_{i\tau}(x, t) = \frac{\tau^2}{((f_i(x) - t)^2 + \tau^2)^{3/2}}.$$

□

Motivated by the preceding propositions, we now present the following selection rule for the truncation parameter.

Corollary 1. Suppose that $\gamma(x)$ and $\omega(x)$ are bounded on Q , where $Q \subset \mathbb{R}^n$ is bounded; namely, there exist constants γ and ω such that

$$\gamma \geq \max_{x \in Q} \gamma(x), \quad \omega \geq \max_{x \in Q} \omega(x).$$

For any $x \in Q$, $\tau > 0$, $\varepsilon_1 > 0$, and $\varepsilon_2 > 0$, if μ satisfies

$$\mu \geq \max \left\{ \tau \left(\frac{(m-1)(\gamma+1) - 2\varepsilon_1}{2\varepsilon_1} \right), \sqrt{\frac{\tau A(m-1)}{(\sqrt{2\varepsilon_2} - \tau A(m-1) - \sqrt{\tau A(m-1)})^2} - \tau^2} \right\}, \quad (4.3)$$

where $A = \max\{\omega/2, \gamma^2 + 1\}$, then

$$\|\nabla \Phi_\tau(x, f(x)) - \nabla \bar{\Phi}_\tau(x, f(x))\| \leq \varepsilon_1,$$

$$\|\nabla^2 \Phi_\tau(x, f(x)) - \nabla^2 \bar{\Phi}_\tau(x, f(x))\| \leq \varepsilon_2.$$

Proof. Because $\mu + \tau \geq \sqrt{\mu^2 + \tau^2}$, by Proposition 9, we get

$$\begin{aligned}
 & \|\nabla \Phi_\tau(x, f(x)) - \nabla \bar{\Phi}_\tau(x, f(x))\| \\
 &\leq \frac{(m-1)(\gamma+1)}{2} \left(1 - \frac{\mu}{\sqrt{\mu^2 + \tau^2}} \right) \leq \frac{(m-1)(\gamma+1)}{2} \left(1 - \frac{\mu}{\mu + \tau} \right) \\
 &= \frac{(m-1)(\gamma+1)\tau}{2(\mu + \tau)} \leq \varepsilon_1,
 \end{aligned}$$

which implies that

$$\mu \geq \tau \left(\frac{(m-1)(\gamma+1) - 2\varepsilon_1}{2\varepsilon_1} \right).$$

Similarly,

$$\begin{aligned}
 & \|\nabla^2 \Phi_\tau(x, f(x)) - \nabla^2 \bar{\Phi}_\tau(x, f(x))\| \\
 & \leq \frac{1}{2}(m-1) \left(1 - \frac{\mu}{\sqrt{\mu^2 + \tau^2}} \right) \omega + \frac{1}{2}(m-1) \frac{\tau^2}{(\mu^2 + \tau^2)^{3/2}} (\gamma^2 + 1) \\
 & \leq \frac{1}{2}(m-1) \frac{\tau\omega}{\sqrt{\mu^2 + \tau^2}} + \frac{1}{2}(m-1) \frac{\tau}{\mu^2 + \tau^2} (\gamma^2 + 1) \\
 & \leq \frac{1}{2}(m-1) \tau A \left(\frac{2}{\sqrt{\mu^2 + \tau^2}} + \frac{1}{\mu^2 + \tau^2} \right).
 \end{aligned}$$

Thus, it is sufficient that

$$\frac{2}{\sqrt{\mu^2 + \tau^2}} + \frac{1}{\mu^2 + \tau^2} \leq \frac{2\varepsilon_2}{(m-1)\tau A}.$$

Solving this inequality yields

$$\mu^2 + \tau^2 \geq \frac{\tau A(m-1)}{\left(\sqrt{2\varepsilon_2 - \tau A(m-1)} - \sqrt{\tau A(m-1)} \right)^2}.$$

Because $\mu > 0$, it follows that

$$\mu \geq \sqrt{\frac{\tau A(m-1)}{\left(\sqrt{2\varepsilon_2 - \tau A(m-1)} - \sqrt{\tau A(m-1)} \right)^2} - \tau^2}.$$

Combining the two lower bounds gives (4.3). The proof is complete. $\square$

5. A truncated hyperbolic smoothing Newton method

From Section 3 and Proposition 5, solving (1.1) is closely related to solving the smoothed minimization problem

$$\min_{x \in \mathbb{R}^n} \Phi_\tau(x, f(x)), \tag{5.1}$$

where

$$\Phi_\tau(x, f(x)) = f(x) + \sum_{i \in I} \frac{f_i(x) - f(x) + \sqrt{(f_i(x) - f(x))^2 + \tau^2}}{2}.$$

Choose an initial point $x_0 \in \mathbb{R}^n$ and parameters $p_0 > 0$, $\hat{p} \gg 1$, $r_k > 0$, $\alpha \in (0, 0.5)$, $\beta, \kappa_1 \in (0, 1)$, $\eta \in (0, (1 - \alpha)\kappa_1^2)$, and $\delta > 0$. Let $\gamma \geq \max_{x \in Q} \gamma(x)$, and $\omega \geq \max_{x \in Q} \omega(x)$. Let $\varepsilon_1(p), \varepsilon_2(p), \sigma(p) : (0, \infty) \rightarrow (0, \infty)$ satisfy $\lim_{p \rightarrow +\infty} \sigma(p) = 0$, $\varepsilon_1(p) = \eta\sigma(p)$, $\varepsilon_2(p) > 0$, and let $\lim_{k \rightarrow +\infty} r_k = 0$. Define

$$\theta(x, t) = \max\{0, \delta - e(x, t)\},$$

where $e(x, t)$ denotes the smallest eigenvalue of $\nabla^2 \bar{\Phi}_\tau(x, t)$. Set

$$\mathbf{B}(x, t) = \theta(x, t)\mathbf{I} + \nabla^2 \bar{\Phi}_\tau(x, t). \tag{5.2}$$

Algorithm 1 Truncated hyperbolic smoothing Newton method

1: **Step 1.** Set $k = \ell = 0$, $s = 1$, $x_k^\ell = x_0$, and $\tau_0 = 1/p_0$.
 2: **Step 2.** Compute μ and $\bar{l}$ according to (4.3) and (4.1), respectively. Compute

$$t_k^\ell = f(x_k^\ell) = \max_{j \in I} f_j(x_k^\ell), \quad \tau_k = \frac{1}{p_k}.$$

3: **if** $\|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, t_k^\ell)\| > \sigma(p_k)$ **then**
 4: **Step 3.** Compute $\mathbf{B}(x_k^\ell, t_k^\ell)$ according to (5.2). Perform the Cholesky factorization

$$\mathbf{B}(x_k^\ell, t_k^\ell) = \mathbf{R}\mathbf{R}^T,$$

and compute the reciprocal condition number κ_0 of $\mathbf{R}$.
 5: **if** $\kappa_0 \geq \kappa_1$ **then**
 6: Go to **Step 4**.
 7: **else**
 8: Go to **Step 5**.
 9: **end if**
 10: **Step 4.** Compute

$$h_k^\ell = -\mathbf{B}(x_k^\ell, t_k^\ell)^{-1} \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, t_k^\ell),$$

and go to **Step 6**.
 11: **Step 5.** Compute

$$h_k^\ell = -\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, t_k^\ell),$$

and go to **Step 6**.
 12: **Step 6.** Use a backtracking procedure to compute the largest stepsize $\lambda_k^\ell = \beta^q$, $q \in \mathbb{N}$, satisfying

$$\Phi_{\tau_k}((x_k^\ell, t_k^\ell) + \lambda_k^\ell h_k^\ell) - \Phi_{\tau_k}(x_k^\ell, t_k^\ell) \leq \alpha \lambda_k^\ell \langle \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, t_k^\ell), h_k^\ell \rangle.$$

13: **Step 7.** Compute

$$(x_k^{\ell+1}, t_k^{\ell+1}) = (x_k^\ell, t_k^\ell) + \lambda_k^\ell h_k^\ell,$$

set $\ell = \ell + 1$, and update

$$t_k^{\ell+1} = f(x_k^{\ell+1}) = \max_{j \in I} f_j(x_k^{\ell+1}).$$

14: **if**

$$|\Phi_{\tau_k}(x_k^{\ell+1}, t_k^{\ell+1}) - \Phi_{\tau_k}(x_k^\ell, t_k^\ell)| \leq r_k$$

or

$$\|\nabla \bar{\Phi}_{\tau_k}(x_k^{\ell+1}, t_k^{\ell+1})\| \leq \sigma(p_k),$$

then
 15: **break**.
 16: **else**
 17: Go to **Step 3**.
 18: **end if**
 19: **else**
 20: Go to **Step 8**.
 21: **end if**
 22: **Step 8.**
 23: **if** $s \neq 1$ **then**
 24: Update $p_{k+1} = s(k+2)$, set $k = k+1$ and $\ell = 0$, and go to **Step 2**.
 25: **else**
 26: Go to **Step 9**.
 27: **end if**
 28: **Step 9.**
 29: **if** $p_k \leq \hat{p}$ **then**
 30: Update $p_{k+1} = p_k + 1000$, set $k = k+1$ and $\ell = 0$, and go to **Step 2**.
 31: **else**
 32: Update

$$s = \max \left\{ 2, \frac{\hat{p} + 1}{k + 1} \right\}, \quad p_{k+1} = \max\{p_k + 1, s(k+2)\},$$

set $k = k+1$ and $\ell = 0$, and go to **Step 2**.
 33: **end if**

Remark 5. For some problem instances, the choices of $\{\tau_k\}$ and $\sigma(p_k)$ are important. If τ_k tends to zero too rapidly, the intrinsic ill-conditioning of the smoothed subproblem may become severe, and more inner iterations may be required. At the same time, τ_k must be sufficiently small relative to $\sigma^2(p_k)$ to ensure the desired descent properties. This trade-off is also reflected in Lemma 1, particularly in Assertion (iii).

Assumption 1. For any given initial point $x_0 \in \mathbb{R}^n$, the level set $\mathcal{L}_{f,c}$ is bounded, where $c = \Phi_{\tau_0}(x_0, f(x_0))$.

Assumption 2. For each $i \in I$, the function f_i is twice continuously differentiable.

Remark 6. Assumption 1 is mild and can be easily verified in many practical problems. In particular, it holds whenever f is coercive. For $f(x) = \max_{i \in I} f_i(x)$, it is sufficient that at least one component function f_i be coercive.

Lemma 1. Suppose that Assumption 1 holds. Let the sequence $\{(x_k^\ell, f(x_k^\ell))\}$ be generated by Algorithm 1. For any $x_k^\ell \in \mathcal{L}_{f,c}$ satisfying

$$\|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| > \sigma(p_k),$$

the following assertions hold:

(i) The gradient gap satisfies

$$-\eta \|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| \leq \|\nabla \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| \leq \eta \|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\|.$$

(ii) The line search terminates after finitely many reductions, and the stepsize $\lambda_k^\ell \in (\lambda^* \beta, \lambda^*)$, where $\lambda^* = \min \{2\kappa_2((1 - \alpha)\kappa_1^2 - \eta)/L, 1\}$ and $\kappa_2 = \min\{\delta, 1\}$.

(iii) If $\sigma^2(p_k) > m\tau_k(L + \delta)/(\alpha\lambda^*\beta)$, then $x_k^{\ell+1} \in \mathcal{L}_{f,c}$.

Proof. (i) The left-hand inequality follows immediately from the non-negativity of the norm. For the right-hand inequality, by Corollary 1 and the choice of parameters in the method, we have

$$\|\nabla \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| \leq \varepsilon_1(p_k) = \eta\sigma(p_k).$$

Moreover,

$$\|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| > \sigma(p_k).$$

Because $\eta > 0$, it follows that

$$\|\nabla \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| \leq \eta \|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\|.$$

Therefore, the desired inequality follows.

(ii) The proof is similar to that of Lemma 3.2 in [22], using the local Lipschitz smoothness of Φ_{τ_k} . In particular, the Armijo condition is satisfied whenever the trial step size is smaller than λ^* . Therefore, the backtracking line search terminates after finitely many reductions, and the accepted step size satisfies

$$\lambda_k^\ell \in (\lambda^* \beta, \lambda^*).$$

(iii) By part (ii), the accepted step size satisfies

$$\lambda_k^\ell > \lambda^* \beta.$$

Moreover, because

$$\|\nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell))\| > \sigma(p_k)$$

holds throughout the inner iteration, it follows that

$$\begin{aligned}
 & - \left\langle \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell)), h_k^\ell \right\rangle \\
 & > \frac{\sigma^2(p_k)}{\sigma_{\max}(\mathbf{B}(x_k^\ell, f(x_k^\ell)))} > \frac{\sigma^2(p_k)}{L + \theta(x_k^\ell, f(x_k^\ell)) - \varepsilon_2(p_k)} \\
 & > \frac{\sigma^2(p_k)}{L + \theta(x_k^\ell, f(x_k^\ell))} = \frac{\sigma^2(p_k)}{L + \max\{\delta, e(x_k^\ell, f(x_k^\ell))\} - e(x_k^\ell, f(x_k^\ell))} \\
 & > \frac{\sigma^2(p_k)}{L + \delta + e(x_k^\ell, f(x_k^\ell)) - e(x_k^\ell, f(x_k^\ell))} = \frac{\sigma^2(p_k)}{L + \delta}.
 \end{aligned}$$

Combining the above estimate with the Armijo line-search condition, it can be obtained that

$$\Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \Phi_{\tau_k}(x_k^{\ell+1}, t_k^{\ell+1}) > -\alpha\lambda_k^\ell \left\langle \nabla \bar{\Phi}_{\tau_k}(x_k^\ell, f(x_k^\ell)), h_k^\ell \right\rangle > \frac{\alpha\lambda_k^\ell \sigma^2(p_k)}{L + \delta} > \frac{\alpha\lambda^* \beta \sigma^2(p_k)}{L + \delta}.$$

Furthermore,

$$\begin{aligned}
 & \Phi_{\tau_k}(x_k^{\ell+1}, t_k^{\ell+1}) - \Phi_{\tau_k}(x_k^{\ell+1}, f(x_k^{\ell+1})) \\
 & = \Phi_{\tau_k}(x_k^{\ell+1}, t_k^{\ell+1}) - F(x_k^{\ell+1}, t_k^{\ell+1}) + F(x_k^{\ell+1}, t_k^{\ell+1}) - F(x_k^{\ell+1}, f(x_k^{\ell+1})) \\
 & \quad + F(x_k^{\ell+1}, f(x_k^{\ell+1})) - \Phi_{\tau_k}(x_k^{\ell+1}, f(x_k^{\ell+1})) \\
 & > 0 + 0 - \frac{m\tau_k}{2} = -\frac{m\tau_k}{2}.
 \end{aligned}$$

Combining the preceding inequalities, we obtain

$$\Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \Phi_{\tau_k}(x_k^{\ell+1}, f(x_k^{\ell+1})) > \frac{\alpha\lambda^* \beta \sigma^2(p_k)}{L + \delta} - \frac{m\tau_k}{2}. \quad (5.3)$$

Because

$$\sigma^2(p_k) > \frac{m\tau_k(L + \delta)}{\alpha\lambda^* \beta},$$

we have

$$\frac{\alpha\lambda^* \beta \sigma^2(p_k)}{L + \delta} - m\tau_k > 0.$$

Consequently,

$$c > f(x_k^\ell) > \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \frac{m\tau_k}{2} > \Phi_{\tau_k}(x_k^{\ell+1}, f(x_k^{\ell+1})) - m\tau_k + \frac{\alpha\lambda^* \beta \sigma^2(p_k)}{L + \delta} > f(x_k^{\ell+1}).$$

Therefore,

$$x_k^{\ell+1} \in \mathcal{L}_{f,c}.$$

This completes the proof. □

Lemma 2. Let the sequences $\{(x_k^\ell, f(x_k^\ell))\}$ and $\{\tau_k\}$ be generated by Algorithm 1. Under Assumptions 1 and 2, if $\sigma^2(p_k) > m\tau_k(L + \delta)/(\alpha\lambda^*\beta)$ for every k , then the following statements hold:

- (i) for every k , the inner sequence $\{(x_k^\ell, f(x_k^\ell))\}$ is finite;
- (ii) the sequence $\{\tau_k\}$ is strictly decreasing, and $\lim_{k \rightarrow \infty} \tau_k = 0$.

Proof. (i) By (5.3) and the condition in Lemma 1(iii), the decrease at each inner iteration satisfies

$$\Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - \Phi_{\tau_k}(x_k^{\ell+1}, f(x_k^{\ell+1})) > \frac{\alpha\lambda^*\beta\sigma^2(p_k)}{L + \delta} - \frac{m\tau_k}{2} > \frac{m\tau_k}{2}.$$

Thus, for any positive integer ℓ , it follows that

$$\frac{m\ell\tau_k}{2} < \Phi_{\tau_k}(x_k^0, f(x_k^0)) - \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)).$$

Moreover, by Proposition 5 and the identity $F(x, f(x)) = f(x)$, it follows that

$$\begin{aligned} \frac{m\ell\tau_k}{2} &< \Phi_{\tau_k}(x_k^0, f(x_k^0)) - \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) \\ &= \Phi_{\tau_k}(x_k^0, f(x_k^0)) - F(x_k^0, f(x_k^0)) + F(x_k^0, f(x_k^0)) - F(x_k^\ell, f(x_k^\ell)) \\ &\quad + F(x_k^\ell, f(x_k^\ell)) - \Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) \\ &< \frac{m\tau_k}{2} + f(x_k^0) - f(x_k^\ell). \end{aligned}$$

Therefore,

$$\frac{m(\ell - 1)\tau_k}{2} < f(x_k^0) - f(x_k^\ell).$$

Because $f(x_k^0) \leq f(x_0)$ and $f(x_k^\ell) \geq f(x^*)$, one further obtains

$$\frac{m(\ell - 1)\tau_k}{2} < f(x_k^0) - f(x_k^\ell) \leq f(x_0) - f(x^*).$$

It follows that

$$\ell < \frac{2(f(x_0) - f(x^*))}{m\tau_k} + 1.$$

Because $f(x^*)$ and $\tau_k > 0$ are finite for each fixed k , the number of inner iterations is finite.

- (ii) The second conclusion follows from the update rule for p_k , which is monotone increasing and satisfies $p_k \rightarrow \infty$; because $\tau_k = 1/p_k$, we have $\tau_k \rightarrow 0$.

□

Theorem 1. Let $\{(x_k^{\ell_k}, f(x_k^{\ell_k}))\}_{k=1}^\infty$ be generated by Algorithm 1, where $\{(x_k^{\ell_k}, f(x_k^{\ell_k}))\}$ is finite for a fixed k . If Assumptions 1 and 2 hold and that $\sigma^2(p_k) > m\tau_k(L + \delta)/(\alpha\lambda^*\beta)$ for every k ; then, there exists a subsequence $\mathcal{N} \subset \mathbb{N}$ such that

$$(x_k^{\ell_k}, f(x_k^{\ell_k})) \rightarrow (x^*, f(x^*)) \quad (k \rightarrow \infty, k \in \mathcal{N}),$$

and $0_n \in \partial f(x^*)$.

Proof. By Lemmas 1 and 2, because $\mathcal{L}_{f,c}$ is bounded, and $(x_k^{\ell_k}, f(x_k^{\ell_k})) \in \mathcal{L}_{f,c}$ for all $k \in \mathbb{N}$, there exists a subsequence $\mathcal{N} \subset \mathbb{N}$ such that

$$(x_k^{\ell_k}, f(x_k^{\ell_k})) \rightarrow (x^*, f(x^*)) \quad (k \rightarrow \infty, k \in \mathcal{N}).$$

Let

$$I_2^* = \bigcup_{k \in \mathcal{N}} (I_1(x_k^{\ell_k}, f(x_k^{\ell_k})) \cap I(x^*)) \cup I_2(x_k^{\ell_k}, f(x_k^{\ell_k})).$$

Then,

$$\begin{aligned} \lim_{k \rightarrow \infty} \beta_{j\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k})) &= \lim_{k \rightarrow \infty} \frac{f_j(x_k^{\ell_k}) - f(x_k^{\ell_k})}{\sqrt{(f_j(x_k^{\ell_k}) - f(x_k^{\ell_k}))^2 + \tau_k^2}} \\ &= \begin{cases} -1, & j \in I^1 := \bigcup_{k \in \mathcal{N}} I_1(x_k^{\ell_k}, f(x_k^{\ell_k})) \setminus I_2^*, \\ -\alpha_j, & \alpha_j \in [0, 1], \quad j \in I^2 := \bigcup_{k \in \mathcal{N}} I_1(x_k^{\ell_k}, f(x_k^{\ell_k})) \cap I(x^*), \\ 0, & j \in I^3 := \bigcup_{k \in \mathcal{N}} I_2(x_k^{\ell_k}, f(x_k^{\ell_k})). \end{cases} \end{aligned}$$

From the definition of $\nabla \Phi_{\tau_k}$ and $I = I^1 \cup I^2 \cup I^3$, it follows that

$$\lim_{k \rightarrow \infty} \nabla \Phi_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k})) = \left[\frac{1}{2} \left(\sum_{j \in I^2} (1 - \alpha_j) \nabla f_j(x^*) + \sum_{j \in I^3} \nabla f_j(x^*) \right), 1 - \frac{1}{2} \left(\sum_{j \in I^2} (1 - \alpha_j) + \sum_{j \in I^3} 1 \right) \right]. \quad (5.4)$$

On the other hand, by (3.4), we obtain

$$\partial F(x^*, t^*) = \{(0_n, 1)\} + \sum_{j \in I_2^*} \text{co}\{0_{n+1}, (\nabla f_j(x^*), -1)\} = \left\{ \left(\sum_{j \in I_2^*} \rho_j \nabla f_j(x^*), 1 - \sum_{j \in I_2^*} \rho_j \right) : \rho_j \in [0, 1] \right\}.$$

Take

$$\rho_j = \begin{cases} (1 - \alpha_j)/2, & j \in I^2, \\ 1/2, & j \in I^3 \end{cases}$$

in the above equation. Then, combining this with (5.4), we get

$$\lim_{k \rightarrow \infty} \nabla \Phi_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k})) \in \partial F(x^*, t^*).$$

We now consider two cases.

Case 1. If

$$\|\nabla \bar{\Phi}_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\| \leq \sigma(p_k),$$

then

$$\lim_{k \rightarrow \infty} \|\nabla \bar{\Phi}_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\| = 0, \quad (5.5)$$

since $\sigma(p_k) \rightarrow 0$ as $k \rightarrow \infty$.

Case 2. If

$$\|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\| > \sigma(p_k),$$

then by the stopping criterion based on the function value difference,

$$\begin{aligned} r_k &= |\Phi_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k})) - \Phi_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))| \\ &\geq -\alpha\lambda_k^{\ell_k-1} \langle \nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1})), h_k^{\ell_k-1} \rangle \\ &\geq \alpha\lambda_k^{\ell_k-1} \|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))\|_{\mathbf{B}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))}^2. \end{aligned}$$

Following the proof of Lemma 1, one has

$$\lambda^*\beta \leq \lambda_k^{\ell_k-1} \leq \lambda^*, \quad \lambda^* = \min \left\{ \frac{2\kappa_2((1-\alpha)\kappa_1^2 - \eta)}{L}, 1 \right\},$$

where $\kappa_2 = \min\{\delta, 1\}$. Then,

$$\|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))\|_{\mathbf{B}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))}^2 \leq \frac{r_k}{\alpha\lambda^*\beta}.$$

Because $r_k \rightarrow 0$, and $\mathbf{B}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))$ is positive definite, we have

$$\lim_{k \rightarrow \infty} \|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))\| = 0. \quad (5.6)$$

Moreover,

$$\begin{aligned} &\|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1})) - \nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\| \\ &= \left\| \left(\int_0^1 \nabla^2\bar{\Phi}_{\tau_k}((x_k^{\ell_k-1}, f(x_k^{\ell_k-1})) + \theta(x_k^{\ell_k}, f(x_k^{\ell_k}))) d\theta \right) \lambda_k^{\ell_k-1} h_k^{\ell_k-1} \right\| \\ &\leq \lambda^* L \|h_k^{\ell_k-1}\| \leq \lambda^* L \|\mathbf{B}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))^{-1}\| \|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))\| \\ &\leq \frac{\lambda^* L}{\kappa_1} \|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k-1}, f(x_k^{\ell_k-1}))\|. \end{aligned}$$

Combining this with (5.6) yields (5.5).

By Lemma 1, we derive that

$$\|\nabla\Phi_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k})) - \nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\| \leq \eta \|\nabla\bar{\Phi}_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\|.$$

Together with (5.5), this yields

$$\lim_{k \rightarrow \infty} \|\nabla\Phi_{\tau_k}(x_k^{\ell_k}, f(x_k^{\ell_k}))\| = 0.$$

Therefore, Proposition 8 implies that $0_{n+1} \in \partial F(x^*, t^*)$. It then follows from Proposition 3 that $0_n \in \partial f(x^*)$. $\square$

Theorem 2. For a fixed $\tau > 0$, let x^* be a local minimizer of $\Phi_\tau(x, f(x))$. Suppose that there exist constants $a_1 > 1$ and $a_2 < 1$ such that

$$\sigma_{\min}(\nabla^2\bar{\Phi}_\tau(x^*, f(x^*))) > a_1\delta, \quad \text{cond}(\nabla^2\bar{\Phi}_\tau(x^*, f(x^*))) < \frac{a_2}{\kappa_1^2}.$$

Assume also that f_j , $j \in I$, are twice continuously differentiable and that Assumption 1 holds. For this fixed τ , suppose that the sequence $\{x^\ell\}$ is generated by Algorithm 1. Then, for all $\ell > \ell_0$ where ℓ_0 is a fixed positive integer, the following assertions hold:

(i) the search direction is

$$h^\ell = -\nabla^2 \bar{\Phi}_\tau(x^\ell, f(x^\ell))^{-1} \nabla \bar{\Phi}_\tau(x^\ell, f(x^\ell)),$$

and $\lambda^\ell = 1$;

(ii)

$$\|x^{\ell+1} - x^*\| = O(\|x^\ell - x^*\|^2).$$

Proof. From Lemma 2(i), the inner sequence $\{x^\ell\}$ is finite. Let its length be ℓ_k . Because

$$\sigma_{\min}(\nabla^2 \bar{\Phi}_\tau(x^*, f(x^*))) > a_1 \delta,$$

there exists a fixed positive integer ℓ_0 such that for all $\ell > \ell_0$,

$$h^\ell = -\nabla^2 \bar{\Phi}_\tau(x^\ell, f(x^\ell))^{-1} \nabla \bar{\Phi}_\tau(x^\ell, f(x^\ell)).$$

Then,

$$\begin{aligned} & \Phi_\tau((x^\ell, t^\ell) + h^\ell) - \Phi_\tau(x^\ell, t^\ell) - \alpha \langle \nabla \bar{\Phi}_\tau(x^\ell, t^\ell), h^\ell \rangle \\ &= \langle \nabla \Phi_\tau(x^\ell, t^\ell) - \nabla \bar{\Phi}_\tau(x^\ell, t^\ell), h^\ell \rangle + (1 - \alpha) \langle \nabla \bar{\Phi}_\tau(x^\ell, t^\ell), h^\ell \rangle + \frac{1}{2} \langle h^\ell, \nabla^2 \Phi_\tau(x^\ell, t^\ell) h^\ell \rangle + o(\|h^\ell\|^2) \\ &= \langle \nabla \Phi_\tau(x^\ell, t^\ell) - \nabla \bar{\Phi}_\tau(x^\ell, t^\ell), h^\ell \rangle + \frac{1}{2} \langle h^\ell, (\nabla^2 \Phi_\tau(x^\ell, t^\ell) - \nabla^2 \bar{\Phi}_\tau(x^\ell, t^\ell)) h^\ell \rangle \\ & \quad + \left(\alpha - \frac{1}{2} \right) \langle \nabla \bar{\Phi}_\tau(x^\ell, t^\ell), \nabla^2 \bar{\Phi}_\tau(x^\ell, t^\ell)^{-1} \nabla \bar{\Phi}_\tau(x^\ell, t^\ell) \rangle. \end{aligned}$$

Let

$$\varepsilon_1 = \min\{\eta\sigma(p), O(\sigma^2(p))\}, \quad \varepsilon_2 = O(\sigma(p)).$$

Under the rule given in Corollary 1, Proposition 9 yields

$$\|\nabla \Phi_\tau(x^\ell, t^\ell) - \nabla \bar{\Phi}_\tau(x^\ell, t^\ell)\| \leq \varepsilon_1 \leq O(\sigma^2(p)),$$

$$\|\nabla^2 \Phi_\tau(x^\ell, t^\ell) - \nabla^2 \bar{\Phi}_\tau(x^\ell, t^\ell)\| \leq \varepsilon_2 = O(\sigma(p)).$$

The remaining argument is similar to that in the proof of Theorem 3.5 of [22], with Lemma 2 used to guarantee the required finite termination of the inner loop. Therefore, the unit step size is accepted for all sufficiently large ℓ , and the standard Newton error estimate yields

$$\|x^{\ell+1} - x^*\| = O(\|x^\ell - x^*\|^2).$$

□

6. Numerical experiments

This section compares the following algorithms:

- THSNM: the truncated hyperbolic smoothing Newton method proposed in this paper;

- HSNM: the same framework without truncation; specifically, the full gradient $\nabla\Phi_{\tau_k}$ and Hessian $\nabla^2\Phi_{\tau_k}$ are used;
- TASNМ: the truncated aggregate smoothing Newton method in [22].

Unless otherwise stated, the parameters used in THSNM are

$$\begin{aligned}\gamma &= 3, \quad \omega = 2.5, \quad p_0 = 100 \ln(m)/m, \quad \alpha = 0.49, \quad \beta = 0.4, \quad \delta = 0.1, \\ \hat{p} &= \ln(m)/\text{tol}, \quad \kappa_1 = 10^{-7}, \quad \varepsilon_1 = 10^{-2}, \quad \varepsilon_2 = 10^{-1}, \quad r_k = 10^{-5}, \\ \sigma^2(p) &= \min\{0.1, \frac{1000}{p}\}.\end{aligned}$$

When $\tau > 10^{-6}$, we set $\mu = 10^{-2}$. The exceptions are Example 1, where $\beta = 0.6$; Examples 2 and 7, where $\mu = 10^{-4}$; and Example 4, where $\delta = 1.7$, and $\mu = 5 \times 10^{-2}$. For HSNM, we set $\mu = +\infty$; that is, no truncation is used. The parameter settings of TASNМ are the same as those in [22]. The termination criterion is

$$|\Phi_{\tau_k}(x_k^\ell, f(x_k^\ell)) - f(\hat{x})| \leq \text{tol},$$

where $f(\hat{x})$ is the known optimal value, and $\text{tol} = 10^{-4}$. All computations were performed in MATLAB R2020b on a laptop with an AMD Ryzen 7 4800H CPU at 2.90 GHz and 16 GB RAM running a 64-bit operating system. Only the MATLAB functions `chol`, `rcond`, and `eig` were used to compute the Cholesky factorization, the reciprocal condition number, and the smallest eigenvalue, respectively. The results denoted by `Fminimax` were obtained by directly calling the MATLAB function `fminimax`.

The test problems are listed below. In each example, x_0 denotes the initial point.

Example 1 (Rotated paraboloid data set, [22]). *This example considers a class of data sets generated by rotated paraboloids. The problem arises from the measurement of paraboloidal reflectors in computational metrology [26]. The data set $\{(z_{j1}, z_{j2}, z_{j3})\}_{j=1}^m$ is generated according to the procedure in [26].*

First, points $(\tilde{z}_{j1}, \tilde{z}_{j2}, \tilde{z}_{j3})$ are sampled from an unrotated paraboloid satisfying

$$\tilde{z}_{j3} = 5(\tilde{z}_{j1}^2 + \tilde{z}_{j2}^2).$$

Then, a noise term $r \sim \mathcal{N}(0, 0.2)$ is added to the third coordinate. Finally, the noisy points are rotated and translated to obtain the observed points (z_{j1}, z_{j2}, z_{j3}) :

$$(z_1^j, z_2^j, z_3^j, 1) = (\tilde{z}_1^j, \tilde{z}_2^j, \tilde{z}_3^j, 1) \bar{\mathbf{C}} \left(\frac{\pi}{25} \right) \bar{\mathbf{B}} \left(\frac{\pi}{20} \right) \bar{\mathbf{A}}(2.1, -1.4, 1.3),$$

where

$$\begin{aligned}\bar{\mathbf{A}}(x_3, x_4, x_5) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_3 & x_4 & x_5 & 1 \end{bmatrix}, \quad \bar{\mathbf{B}}(x_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(x_1) & \sin(x_1) & 0 \\ 0 & -\sin(x_1) & \cos(x_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\ \bar{\mathbf{C}}(x_2) &= \begin{bmatrix} \cos(x_2) & 0 & -\sin(x_2) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(x_2) & 0 & \cos(x_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.\end{aligned}$$

Thus, the objective function is defined by

$$F(x) = \max_{1 \leq j \leq m} \left| \bar{z}_3^j - x_6 \left((\bar{z}_1^j)^2 + (\bar{z}_2^j)^2 \right) \right|,$$

where

$$(\bar{z}_1^j, \bar{z}_2^j, \bar{z}_3^j, 1) = (z_1^j, z_2^j, z_3^j, 1) \bar{\mathbf{A}}(x_3, x_4, x_5) \bar{\mathbf{B}}(x_1) \bar{\mathbf{C}}(x_2).$$

Here, $n = 6$, and $x_0 = (0, 0, -2, 2, -2, 5)$.

Example 2 ([20]).

$$F(x) = \max_{1 \leq j \leq m} |f^j(x)|, \quad f^j(x) = \sin y_j - (x_3 y_j^2 + x_2 y_j + x_1), \quad y_j = \frac{j-1}{m-1}.$$

Here, $n = 3$, and $x_0 = (1, 1, 1)$.

Example 3 ([28]).

$$F(x) = \max_{1 \leq j \leq m} f^j(x),$$

where

$$f_j(x) = \left(x_1 + x_2 y_j - e^{y_j} \right)^2 + \left(x_3 + x_4 \sin y_j - \cos y_j \right)^2, \quad y_j = \frac{4(j-1)}{m-1}.$$

Here, $n = 4$, and $x_0 = (25, 5, -5, 1)$.

Example 4 ([22]).

$$F(x) = \max_{1 \leq j \leq m} |f^j(x)|,$$

where

$$\begin{aligned} g^j(x) &= \left(\sum_{1 \leq k \leq n/2} \cos(2x_k t^j) + \sin(2x_{k+n/2} t^j) \right) \left(\sum_{1 \leq k \leq n/2} x_k \cos(2k t^j) + x_{k+n/2} \sin(2k t^j) \right), \\ f^j(x) &= g^j(x) - y^j, \quad t^j = \frac{2\pi(j-1)}{m}, \\ y^j &= g^j(s) + \sigma^j, \\ s &= (0.5, 0.5, \dots, 0.5), \quad \sigma \sim \mathcal{N}(0, 0.3). \end{aligned}$$

The initial point is $x_0 = (0.5, 0.5, \dots, 0.5) \in \mathbb{R}^n$.

Example 5 ([22]).

$$F(x) = \max_{1 \leq j \leq m} |f^j(x)|,$$

where

$$\begin{aligned} f^j(x) &= \frac{x_3}{x_2} e^{-t^j x_1} \sin(t^j x_2) + x_1 e^{-x_2 t^j} \cos(x_3 t^j + x_4) + x_5 e^{-x_6 t^j} - y^j, \\ y^j &= \frac{3}{20} e^{-t^j} + \frac{1}{52} e^{-5t^j} - \frac{1}{65} e^{-2t^j} (3 \sin(2t^j) + 11 \cos(2t^j)) \\ &\quad + \frac{1}{2} e^{-t^j} - e^{-2t^j} + \frac{1}{2} e^{-3t^j} + \frac{3}{2} e^{-3t^j/2} \sin(7t^j) + e^{-5t^j/2} \sin(5t^j), \\ t^j &= \frac{10(j-1)}{m-1}, \quad j = 1, \dots, m. \end{aligned}$$

Here, $n = 6$ and $x_0 = (2, 2, 7, 0, -2, 1)$.

Example 6 ([27]).

$$F(x) = \max_{1 \leq j \leq m} f^j(x), \quad f^j(x) = \sum_{i=1}^{11} \frac{1}{i+j-1} \exp\left((x_i + \sin(j-1+2(i-1)))^2\right),$$

where $n = 11$, and $x_0 = (1, 1, \dots, 1)$.

Example 7 ([20]).

$$F(x) = \max_{1 \leq j \leq m} |f^j(x)|, \quad f_j(x) = \sqrt{y_j} - (x_4 - (x_1 y_j^2 + x_2 y_j + x_3)^2),$$

where $y_j = 0.25 + 0.75(j-1)/(m-1)$. Here, $n = 4$, and $x_0 = (1, 1, 1, 1)$.

Example 8 ([20]).

$$F(x) = \max_{1 \leq j \leq m} |f^j(x)|, \quad f_j(x) = (2y_j^2 - 1)x + y_j(1 - y_j)(1 - x), \quad y_j = \frac{j-1}{m-1}.$$

Here, $n = 1$, and $x_0 = 5$.

The numerical results are summarized in Tables 1–9. In the tables, x^* denotes the final local minimizer, $f^* = f(x^*)$, and the CPU time is the average over ten independent runs.

Table 1. Numerical results for Example 1.

Method	$m = 1000$		$m = 5000$		$m = 10,000$		$m = 20,000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	0.7406	0.548863	2.0484	0.686132	1.5219	0.718192	3.1500	0.744105
THSNM	0.0484	0.548961	0.3156	0.686228	0.2734	0.718289	0.3438	0.744205
HSNM	1.3328	0.549799	18.2906	0.686965	78.2422	0.719072	353.2609	0.745074
TASNM	0.1938	0.548959	1.0391	0.686220	1.7281	0.718288	2.6016	0.744196

Table 2. Numerical results for Example 2.

Method	$m = 500$		$m = 1000$		$m = 5000$		$m = 10,000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	0.7312	0.004505	0.7594	0.004505	7.0234	0.004505	19.6172	0.004505
THSNM	0.0313	0.004603	0.0672	0.004599	1.0875	0.004604	2.7719	0.004604
HSNM	0.6969	0.004603	2.4344	0.004603	169.9859	0.004604	1258.6625	0.004604
TASNM	0.4813	0.004604	1.6938	0.004605	7.8219	0.004605	16.1672	0.004604

Table 3. Numerical results for Example 3.

Method	$m = 1000$		$m = 5000$		$m = 10,000$		$m = 20,000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	0.5953	125.618586	3.2250	125.618592	7.6219	125.618590	21.1141	125.618592
THSNM	0.0828	125.618683	0.4656	125.618690	0.6687	125.618675	2.9000	125.618675
HSNM	2.7687	125.618683	23.1141	125.618683	107.1031	125.618683	995.1484	125.618690
TASNM	0.2797	125.618678	1.6594	125.618688	2.8844	125.618688	5.3297	125.618689

Table 4. Numerical results for Example 4 with changeable n , and $m = 1000$.

Method	$n = 20$		$n = 50$		$n = 70$		$n = 100$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	1.7281	0.828343	4.3359	0.637887	12.6891	0.616768	31.8609	0.605588
THSNM	0.2656	0.827738	3.9937	0.637986	6.5750	0.616868	12.4844	0.605686
HSNM	24.1391	0.828441	11.1391	0.638732	15.5953	0.617685	27.9672	0.606441
TASNM	1.0344	0.827734	4.4063	0.637983	8.6609	0.616867	16.9391	0.605684

Table 5. Numerical results for Example 4 with changeable m , and $n = 50$.

Method	$m = 2000$		$m = 3000$		$m = 4000$		$m = 5000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	8.0641	0.797087	2.3609	0.895293	5.7047	0.970455	5.4266	0.910089
THSNM	3.7766	0.797187	7.2219	0.895392	7.2172	0.970554	4.3594	0.910183
HSNM	27.7437	0.797960	143.8672	0.896278	70.3375	0.971404	40.9516	0.911072
TASNM	5.5469	0.797184	10.1125	0.895393	11.2656	0.970555	10.5406	0.910170

Table 6. Numerical results for Example 5.

Method	$m = 2000$		$m = 4000$		$m = 10,000$		$m = 20,000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	1.0250	0.163266	1.7219	0.163262	4.3703	0.163265	9.6453	0.163265
THSNM	0.1094	0.163364	1.1687	0.163362	1.6059	0.163365	2.9531	0.163364
HSNM	2.1500	0.163364	7.2750	0.163359	59.0406	0.163364	382.6422	0.163364
TASNM	0.9812	0.163359	1.1844	0.163340	1.6750	0.163353	3.8375	0.163351

Table 7. Numerical results for Example 6.

Method	$m = 1000$		$m = 5000$		$m = 10,000$		$m = 20,000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	1.4406	3.703483	1.7953	3.703483	2.0875	3.703483	5.4281	3.703483
THSNM	0.0437	3.703575	0.1625	3.703576	0.3859	3.703574	0.9703	3.703577
HSNM	0.9875	3.703576	7.4641	3.703576	25.8422	3.703575	114.6672	3.703575
TASNM	0.2375	3.703581	0.3484	3.703581	0.6094	3.703581	0.9266	3.703581

Table 8. Numerical results for Example 7.

Method	$m = 1000$		$m = 5000$		$m = 10,000$		$m = 20,000$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	1.4141	0.002701	16.4469	0.002673	41.9438	0.002662	82.1453	0.002657
THSNM	0.2172	0.002798	1.5219	0.002799	3.5156	0.002799	7.1641	0.002801
HSNM	3.0922	0.002801	34.0047	0.002800	183.6016	0.002800	1157.7484	0.002801
TASNM	0.6687	0.002794	2.3453	0.002798	4.1672	0.002798	7.6547	0.002799

Table 9. Numerical results for Example 8.

Method	$m = 1500$		$m = 2500$		$m = 3500$		$m = 4500$	
	CPU	f^*	CPU	f^*	CPU	f^*	CPU	f^*
Fminimax	0.8859	0.178395	0.6266	0.178395	1.3141	0.178395	1.4312	0.178395
THSNM	0.0609	0.178494	0.3438	0.178495	0.8234	0.178495	1.9563	0.178495
HSNM	0.8938	0.178495	3.2172	0.178494	7.9375	0.178495	16.0641	0.178493
TASNM	0.3734	0.178402	0.6422	0.178421	2.1828	0.178419	0.1031	0.178439

From the numerical results reported in Examples 1–8, it can be observed that the proposed truncated hyperbolic smoothing Newton method (THSNM) is effective for solving finite minimax optimization problems in most of the tested cases.

- **Compared with the `fminimax` solver in MATLAB:**

Tables 1–3 and 6–9 show that THSNM generally attains objective function values comparable to those obtained by the MATLAB's `fminimax` solver. Moreover, for problems with a large number of component functions m , THSNM requires significantly less CPU time. Nevertheless, Table 5 indicates that the computational advantage of THSNM over `fminimax` is problem-dependent; in certain instances, `fminimax` may be more efficient, although the objective function values remain comparable.

- **Compared with HSNM:**

Tables 1–3 and 6–9 indicate that as the number of component functions m increases, the CPU time of HSNM grows rapidly, whereas the CPU time of THSNM increases much more moderately. These results demonstrate that the truncation strategy effectively reduces the computational burden associated with evaluating gradients and Hessian matrices.

- **Compared with TASNM proposed in [22]:**

Tables 1, 3, and 6 show that THSNM generally achieves objective function values comparable to those obtained by the truncated aggregate smoothing Newton method (TASNM) proposed in [22] while requiring less CPU time under the same accuracy requirements. Furthermore, Tables 4 and 5 show that as the number of variables n increases, the CPU time of THSNM grows moderately and remains overall smaller than that of TASNM. This suggests that the proposed method is less sensitive to increases in problem dimension than the exponential smoothing approach.

The numerical results demonstrate that the proposed THSNM algorithm can substantially reduce the computational cost of solving finite minimax problems while maintaining comparable solution accuracy. This advantage becomes more pronounced when the number of component functions is large, which indicates the practical potential of the proposed method.

7. Sensitivity analysis with respect to the truncation parameter μ

In this section, we investigate the sensitivity of THSNM to the truncation parameter μ in terms of attained accuracy, computational efficiency, and iteration behavior. According to (4.1), the value of μ determines the number of component functions retained in the truncated smoothing model and therefore affects both the per-iteration computational cost and the approximation quality of the truncated function.

Corollary 1 provides a theoretical selection rule for μ by controlling the differences between the full and truncated gradients and Hessian matrices. To examine the practical influence of this parameter, we test

$$\mu \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}.$$

These fixed values are used only when $\tau_k > 10^{-6}$, whereas the original implementation is retained when $\tau_k \leq 10^{-6}$. All other algorithmic parameters and stopping criteria remain unchanged.

In Tables 10–17, q denotes the average number of backtracking reductions per inner iteration, and ℓ and k denote the total numbers of inner and outer iterations, respectively. The entries in the column $|f - f^*|$ are reported in units of 10^{-5} . The influence of μ is assessed through the relative objective value error, CPU time, the average number of backtracking reductions, and the inner and outer iteration counts.

Table 10. Numerical results for Example 1 with varying μ and m .

μ	$m = 1000$					$m = 5000$					$m = 10,000$					$m = 20,000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	0.0969	9.6732	3.8427	89	25	0.1797	9.9434	4.7436	78	23	0.2891	9.9879	5.1081	111	23	0.3766	9.8508	5.7143	91	23
10^{-2}	0.0516	9.8282	6.1186	194	26	0.1219	9.5858	5.5575	113	23	0.2609	9.7196	5.9710	138	23	0.3219	9.9938	5.7600	100	21
10^{-3}	0.1109	8.5169	7.8883	376	38	0.4172	9.7765	8.1161	336	27	0.4516	9.9879	7.9190	210	26	0.6953	9.9640	8.7664	197	22
10^{-4}	0.1844	1.9305	8.7537	670	167	0.6016	7.3924	10.6995	376	89	1.7125	2.0664	7.9407	708	417	1.5984	7.7706	10.7401	377	104

Table 11. Numerical results for Example 2 with varying μ and m .

μ	$m = 500$					$m = 1000$					$m = 5000$					$m = 10,000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	0.5813	9.9339	4.8710	124	82	1.7984	9.8954	6.4972	179	112	28.6453	9.9713	5.5891	275	241	161.8500	9.9992	6.0902	388	337
10^{-2}	0.5094	9.8563	4.7239	134	82	1.4453	9.8944	5.3816	152	113	27.0594	9.9995	5.5893	280	241	161.7656	9.9646	6.4065	401	338
10^{-3}	0.0281	9.9463	6.0672	119	81	0.3609	9.9858	6.6928	166	112	5.1375	9.9827	8.2646	291	240	20.0234	9.9769	8.6447	394	337
10^{-4}	0.0313	9.8545	13.1339	127	81	0.0703	9.4627	14.9778	180	128	1.0188	9.9844	17.8693	283	240	2.7687	9.9983	16.5902	388	338

Table 12. Numerical results for Example 3 with varying μ and m .

μ	$m = 1000$					$m = 5000$					$m = 10,000$					$m = 20,000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	0.0891	9.6861	3.1193	176	16	0.3859	9.8491	3.0971	206	19	0.7734	8.5232	3.1163	258	65	6.3484	9.8491	5.6498	434	39
10^{-2}	0.0750	9.6861	4.7441	211	17	0.3031	9.8491	3.3578	218	18	0.5922	8.5232	3.7733	225	34	2.5609	8.3232	5.0839	477	43
10^{-3}	0.1047	9.6861	6.1143	280	16	0.5250	9.8491	5.2375	299	18	0.8063	9.2861	4.8373	295	28	1.9812	9.8491	5.0138	363	37
10^{-4}	10.2922	8.1603	13.3703	15209	13859	1.5672	6.0344	7.8676	740	152	3.5547	6.2344	9.1089	845	290	7.1156	9.8491	8.4303	839	227

Table 13. Numerical results for Example 4 with varying μ and m , where $n = 50$.

μ	$m = 2000$					$m = 3000$					$m = 4000$					$m = 5000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	4.1000	9.9572	2.9310	87	42	16.9203	9.7031	10.5714	210	111	5.9484	9.8239	6.0000	80	39	3.2172	9.3663	3.3333	51	17
10^{-2}	4.8828	9.9572	4.1103	145	42	24.2000	9.9813	9.7295	366	155	30.6516	9.9583	3.0719	584	366	6.9672	8.9369	4.8917	120	21
10^{-3}	10.0875	9.8139	6.0764	314	44	25.8203	9.6291	7.7893	503	127	78.8031	9.8803	13.5536	784	483	25.3984	9.9560	5.2231	399	183
10^{-4}	39.3047	9.6278	8.9004	984	458	63.6188	9.7507	8.0851	1199	652	143.5828	9.9400	13.3072	1351	1037	386.5703	9.9808	14.1535	1577	1182

Table 14. Numerical results for Example 5 with varying μ and m .

μ	$m = 2000$					$m = 4000$					$m = 10,000$					$m = 20,000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	0.6828	9.9378	4.5882	68	36	1.4969	9.9296	5.5581	86	60	3.9594	9.9902	5.4403	134	75	12.4297	9.9470	4.6076	158	104
10^{-2}	0.1125	9.8321	4.6092	87	36	1.1547	9.9669	4.7480	127	67	1.7797	9.9708	5.5692	130	77	3.2078	9.8770	5.1448	145	93
10^{-3}	0.1250	9.5192	6.5111	90	35	0.2750	9.4885	6.0180	111	61	1.2578	9.8859	6.7308	182	94	5.5938	9.8993	8.0480	271	155
10^{-4}	0.2671	9.2361	8.8354	158	77	0.4781	9.4245	9.4626	147	63	1.7172	9.9902	12.3125	176	108	5.9141	9.6579	11.7009	331	212

Table 15. Numerical results for Example 6 with varying μ and m .

μ	$m = 1000$					$m = 5000$					$m = 10,000$					$m = 20,000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	0.0922	9.1181	4.3256	43	12	0.2516	9.1657	4.4419	43	12	0.4703	9.2373	5.0909	44	12	0.9234	9.4042	5.2857	42	12
10^{-2}	0.0437	9.2134	4.7234	47	12	0.1797	9.2611	4.5263	38	12	0.4063	9.1419	4.6667	45	12	0.9187	9.3803	4.6327	49	12
10^{-3}	0.0859	9.2850	5.9815	54	12	0.3125	9.4280	5.8868	53	12	0.6656	9.2373	5.9333	60	12	1.0797	9.2134	5.2778	54	12
10^{-4}	0.1109	6.9961	8.8857	70	15	0.7531	9.8572	9.5922	103	11	1.1578	7.7114	8.9080	87	14	1.9391	9.3088	8.8657	67	13

Table 16. Numerical results for Example 7 with varying μ and m .

μ	$m = 1000$					$m = 5000$					$m = 10,000$					$m = 20,000$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	2.3766	9.8950	3.8533	184	103	32.2906	9.9121	4.3981	314	209	184.7937	9.9762	5.5165	424	292	919.7406	9.9702	6.6555	566	409
10^{-2}	2.4531	9.9871	4.2578	225	103	30.1844	9.9172	4.8266	323	212	154.3422	9.9771	5.0614	391	289	763.2437	9.9577	6.8822	501	400
10^{-3}	0.5828	9.9090	6.1143	204	103	7.0250	9.9290	6.4771	327	212	27.0000	9.9631	7.1476	420	293	142.5687	9.9941	8.3254	547	408
10^{-4}	0.1375	9.6927	10.1019	324	156	1.3078	9.8316	12.2123	424	284	2.9859	9.8224	13.2218	559	390	7.0391	9.9980	14.7874	696	527

Table 17. Numerical results for Example 8 with varying μ and m .

μ	$m = 1500$					$m = 2500$					$m = 3500$					$m = 4500$				
	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k	CPU	$ f - f^* $	q	ℓ	k
10^{-1}	0.5172	9.9126	6.4112	197	133	1.7328	9.9856	6.6117	291	212	4.2828	9.9543	8.1921	380	290	8.9812	9.7650	8.0409	514	376
10^{-2}	0.0625	9.8634	6.6439	205	134	0.3000	9.9900	6.8472	301	211	0.7406	9.9900	7.8477	407	293	1.6641	9.9945	8.4015	523	386
10^{-3}	0.0406	9.9304	5.9645	197	134	0.0938	9.9900	7.4983	287	213	0.4047	9.9513	8.3668	398	293	0.5125	9.8261	8.3452	478	370
10^{-4}	0.0437	9.9692	6.5528	199	131	0.0844	9.9945	7.3860	285	212	0.3656	9.8112	8.4948	386	296	0.4906	9.8842	8.5524	487	373

From the numerical results reported in Tables 10–17, the sensitivity of THSNM to the truncation parameter μ can be summarized from the following three aspects.

- **Attained accuracy:**

Overall, the errors remain relatively stable as μ varies, and most of them are close to the stopping tolerance. When μ is reduced to 10^{-4} , lower objective value errors can be observed in some cases, particularly in Tables 10, 12, and 15. However, this decrease is not consistent across all the test problems, and the errors do not exhibit a uniform monotonic dependence on μ .

- **Computational efficiency:**

The CPU time exhibits different trends for different test problems. When the component functions have favorable local smoothness properties and relatively simple forms with respect to the decision variables, such as low-degree polynomial forms, a smaller μ generally leads to a greater reduction in CPU time. Moreover, this reduction becomes more evident as the number of component functions increases, as shown in Tables 11, 16, and 17.

By contrast, when the component functions have less favorable local smoothness properties, an excessively small μ may adversely affect the computational efficiency. For example, in Examples 1, and 4–6, the gradients and Hessian matrices of the component functions may have relatively large local Lipschitz constants, and the component functions also exhibit a certain degree of oscillation. In this case, when μ is small, the additional computational cost of backtracking reductions and iterations caused by the truncation errors may exceed the saving in the per-iteration computational cost achieved by truncation, as observed in Tables 10 and 13–15.

- **Iteration behavior:**

As shown in Tables 10–17, the average number of backtracking reductions q generally tends to increase as μ decreases. A similar but problem-dependent tendency can be observed for the inner and outer iteration counts ℓ and k . In Tables 11, 16, and 17, although the average number of backtracking reductions q may increase as μ decreases, the inner and outer iteration counts ℓ and k remain comparable or increase only moderately. This indicates that when the problem dimension is relatively low, and the component functions have favorable local smoothness properties, a smaller μ does not lead to large truncation errors in the gradients and Hessian matrices, so the truncation strategy can still generate effective Newton directions, and the local convergence performance of the algorithm is not significantly affected, although additional backtracking reductions may be required.

By contrast, when the gradients or Hessian matrices of the component functions have relatively large local Lipschitz constants, or when the component functions exhibit a certain degree of oscillation, such as those involving exponential or trigonometric terms, a smaller μ may have a more pronounced influence on the iteration counts. In Examples 1, 4, and 6, the smaller μ may enlarge the errors between the truncated and full gradients and Hessian matrices, thereby leading to a noticeable or substantial increase in the numbers of backtracking reductions and iterations, as shown in Tables 10, 13, and 15. A similar increase in the iteration counts is also observed in Table 12, indicating that the iteration process of this problem is likewise sensitive to the gradient and Hessian truncation errors.

Overall, the numerical results suggest that the practical choice of μ should take the structure of the problem into account. For problems whose component functions have less favorable local smoothness properties, for example, when their gradients or Hessian matrices have relatively large local Lipschitz constants or when the component functions are highly oscillatory, an excessively small μ may lead to relatively large truncation errors in the gradients and Hessian matrices, thereby resulting in more backtracking reductions or iterations. For such problems, relatively large values such as $\mu = 10^{-1}$ or $\mu = 10^{-2}$ are generally preferable. By contrast, for low-dimensional problems whose component functions have more favorable local smoothness properties, smaller values such as $\mu = 10^{-3}$ or $\mu = 10^{-4}$ may be more effective. This advantage becomes more evident as the number of component functions

increases. In practical computations, $\mu = 10^{-2}$ may first be used as a reasonable initial choice, and Corollary 1 can then be used to control the subsequent truncation errors in the gradients and Hessian matrices.

8. Conclusion

This paper proposes a truncated hyperbolic smoothing Newton method for finite minimax problems with multiple component functions. From a theoretical perspective, gaps are established for the differences in the function values, gradients, and Hessian matrices between the original and truncated hyperbolic smoothing functions. Based on the gaps, this paper provides a rule for choosing the truncation parameter. Under the rule, this paper proves the global convergence of the proposed method and its local quadratic convergence rate. In terms of numerical performance, the proposed method demonstrates clear advantages over both the exponential smoothing method and the untruncated smoothing method, requiring less CPU time under the same stopping criterion. The sensitivity analysis further indicates that the attained accuracy is relatively insensitive to the truncation parameter μ , whereas the computational efficiency and iteration behavior exhibit problem structure-dependent sensitivity to its choice.

In future work, we plan to investigate efficient Hessian approximation techniques and develop quasi-Newton methods for large-scale minimax optimization problems. The goal is to reduce the computational and storage costs associated with Hessian evaluations while maintaining the favorable convergence properties of Newton-type methods.

Author contributions

These authors contributed equally to this work.

Use of Generative-AI tools declaration

During the preparation of this manuscript, the authors used ChatGPT for language polishing and expression improvement.

Acknowledgments

This research was supported by the National Key Research and Development Program of China (No. 2023YFA1011504), the National Natural Science Foundation of China (Nos. 12171060, 12431010), the Chongqing Natural Science Foundation (No. CSTB2024NSCQ-LZX0140), the Science and Technology Research Program of Chongqing Municipal Education Commission (No. KJZD-M202300504), the Bayu Scholars Program of Chongqing Municipal Education Commission, “Bowang Scholars” Young Top Talents Cultivation Program of Chongqing Normal University, and the Foundation of Chongqing Normal University (No. 22XLB006).

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. X. Jiang, Y. Yang, Y. Lu, M. Cao, Flattened aggregate function method for nonlinear programming with many complicated constraints, *Numer. Algor.*, **86** (2021), 103–122. <https://doi.org/10.1007/s11075-020-00881-1>
2. H. Y. Liu, J. Hu, Y. F. Li, Z. W. Wen, *Optimization Modeling, Algorithms and Theory*, Higher Education Press, Beijing, 2020.
3. S. Wang, X. Yang, A power penalty method for linear complementarity problems, *Oper. Res. Lett.*, **36** (2008), 211–214. <https://doi.org/10.1016/j.orl.2007.06.006>
4. T. M. Nogueira Pires, R. P. Cardoso Coelho, F. M. Andrade Pires, Composite bayesian optimisation for multi-objective problems with smooth Tchebycheff scalarisation, *Struct. Multidiscip. Optim.*, **69** (2026), 34. <https://doi.org/10.1007/s00158-025-04229-y>
5. A. Nishioka, Y. Kanno, Smoothing inertial method for worst-case robust topology optimization under load uncertainty, *Struct. Multidiscip. Optim.*, **66** (2023), 82. <https://doi.org/10.1007/s00158-023-03543-7>
6. Q. Ploussard, Piecewise linear approximation with minimum number of linear segments and minimum error: A fast approach to tighten and warm start the hierarchical mixed integer formulation, *Eur. J. Oper. Res.*, **315** (2024), 50–62. <https://doi.org/10.1016/j.ejor.2023.11.017>
7. X. Li, T. Zhang, L. Wang, H. Ma, X. Zhao, A minimax regret model for the leader–follower facility location problem, *Ann. Oper. Res.*, **309** (2022), 861–882. <https://doi.org/10.1007/s10479-020-03826-y>
8. S. Seyyedi Ghomi, F. Baroughi, Robust vertex centdian facility location problem on tree networks, *Ann. Oper. Res.*, **341** (2024), 1135–1149. <https://doi.org/10.1007/s10479-024-06096-0>
9. Z. Drezner, D. Zerom, Competitive facility location under attrition, *Comput. Manag. Sci.*, **20** (2023), 39. <https://doi.org/10.1007/s10287-023-00473-z>
10. J. Park, H. El-Amine, N. Mutlu, An exact algorithm for large-scale continuous nonlinear resource allocation problems with minimax regret objectives, *INFORMS J. Comput.*, **33** (2021), 1213–1228. <https://doi.org/10.1287/ijoc.2020.0999>
11. J. A. Chatelon, D. W. Hearn, T. J. Lowe, A subgradient algorithm for certain minimax and minisum problems, *Math. Program.*, **15** (1978), 130–145. <https://doi.org/10.1007/BF01609012>
12. C. Tang, J. Jian, G. Li, A proximal-projection partial bundle method for convex constrained minimax problems, *J. Ind. Manag. Optim.*, **15** (2019), 757–774. <https://doi.org/10.3934/jimo.2018069>
13. C. Tang, H. Chen, J. Jian, An improved partial bundle method for linearly constrained minimax problems, *Stat. Optim. Inf. Comput.*, **4** (2016), 84–98. <https://doi.org/10.19139/soic.v4i1.205>
14. Z. Zhou, Q. Yang, An active set smoothing method for solving unconstrained minimax problems, *Math. Probl. Eng.*, **2020** (2020), 9108150. <https://doi.org/10.1155/2020/9108150>
15. N. Yilmaz, A new smoothing function technique for solving minimax problems, *J. Appl. Anal. Comput.*, **15** (2025), 1703–1718. <https://doi.org/10.11948/20240361>

16. I. Zang, A smoothing-out technique for min–max optimization, *Math. Program.*, **19** (1980), 61–77. <https://doi.org/10.1007/BF01581628>
17. F. Ye, H. Liu, S. Zhou, S. Liu, A smoothing trust-region Newton-CG method for minimax problem, *Appl. Math. Comput.*, **199** (2008), 581–589. <https://doi.org/10.1016/j.amc.2007.10.070>
18. N. Yilmaz, A new approach for solving minimax problems using new generation smoothing techniques, *Dolomites Res. Notes Approx.*, **18** (2025), 97–111. <https://doi.org/10.25430/pupj-DRNA-2025-2-12>
19. X. Li, An entropy-based aggregate method for minimax optimization, *Eng. Optim.*, **18** (1992), 277–285. <https://doi.org/10.1080/03052159208941026>
20. E. Polak, J. O. Royset, R. S. Womersley, Algorithms with adaptive smoothing for finite minimax problems, *J. Optim. Theory Appl.*, **119** (2003), 459–484. <https://doi.org/10.1023/B:JOTA.00000006685.60019.3e>
21. E. Polak, R. S. Womersley, H. X. Yin, An algorithm based on active sets and smoothing for discretized semi-infinite minimax problems, *J. Optim. Theory Appl.*, **138** (2008), 311–328. <https://doi.org/10.1007/s10957-008-9355-9>
22. Y. Xiao, B. Yu, A truncated aggregate smoothing Newton method for minimax problems, *Appl. Math. Comput.*, **216** (2010), 1868–1879. <https://doi.org/10.1016/j.amc.2009.11.034>
23. J. Guo, Z. Wan, A smoothing three-term conjugate gradient algorithm for solving finite minimax problems, *Math. Numer. Sin.*, **44** (2022), 324–338. <https://doi.org/10.12286/jssx.j2020-0714>
24. A. M. Bagirov, N. Sultanova, A. Al Nuaimat, S. Taheri, Solving minimax problems: Local smoothing versus global smoothing, in *Numerical Analysis and Optimization* (eds. M. Al-Baali, L. Grandinetti, A. Purnama), Springer, Cham, (2018), 23–43. https://doi.org/10.1007/978-3-319-90026-1_2
25. A. M. Bagirov, A. Al Nuaimat, N. Sultanova, Hyperbolic smoothing function method for minimax problems, *Optimization*, **62** (2013), 759–782. <https://doi.org/10.1080/02331934.2012.675335>
26. H. Späth, Least squares fitting with rotated paraboloids, *Math. Commun.*, **6** (2001), 173–179.
27. E. Polak, D. Q. Mayne, J. E. Higgins, Superlinearly convergent algorithm for min-max problems, *J. Optim. Theory Appl.*, **69** (1991), 407–439. <https://doi.org/10.1007/BF00940683>
28. K. M. Brown, J. E. Dennis, New computational algorithms for minimizing a sum of squares of nonlinear functions, Report 71-6, Department of Computer Science, Yale University, 1971.



AIMS Press

©2026 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (<https://creativecommons.org/licenses/by/4.0>)