**Electronic Transactions on Numerical Analysis.**
**Volume 60, pp. 618–635, 2024.**
**Copyright © 2024, Kent State University.**
**ISSN 1068–9613.**
**DOI: 10.1553/etna_vol60s618**

**ETNA**
Kent State University **and**
Johann Radon Institute (RICAM)

# LSEMINK: A MODIFIED NEWTON–KRYLOV METHOD FOR LOG-SUM-EXP MINIMIZATION[*]

KELVIN KAN[†], JAMES G. NAGY[‡], AND LARS RUTHOTTO[‡]

**Abstract.** This paper introduces LSEMINK, an effective modified Newton–Krylov algorithm geared toward minimizing the log-sum-exp function for a linear model. Problems of this kind arise commonly, for example, in geometric programming and multinomial logistic regression. Although the log-sum-exp function is smooth and convex, standard line-search Newton-type methods can become inefficient because the quadratic approximation of the objective function can be unbounded from below. To circumvent this, LSEMINK modifies the Hessian by adding a shift in the row space of the linear model. We show that the shift renders the quadratic approximation to be bounded from below and that the overall scheme converges to a global minimizer under mild assumptions. Our convergence proof also shows that all iterates are in the row space of the linear model, which can be attractive when the model parameters do not have an intuitive meaning, as is common in machine learning. Since LSEMINK uses a Krylov subspace method to compute the search direction, it only requires matrix-vector products with the linear model, which is critical for large-scale problems. Our numerical experiments on image classification and geometric programming illustrate that LSEMINK considerably reduces the time-to-solution and increases the scalability compared to geometric programming and natural gradient descent approaches. It has significantly faster initial convergence than standard Newton–Krylov methods, which is particularly attractive in applications like machine learning. In addition, LSEMINK is more robust to ill-conditioning arising from the nonsmoothness of the problem. We share our MATLAB implementation at a GitHub repository (https://github.com/KelvinKan/LSEMINK).

**Key words.** log-sum-exp minimization, Newton–Krylov method, modified Newton method, machine learning, geometric programming

**AMS subject classifications.** 65K10

## 1. Introduction.

We consider minimization problems of the form

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \sum_{k=1}^{N} w^{(k)} \left[ g^{(k)}(\mathbf{x}) - {\mathbf{c}^{(k)}}^\top \mathbf{J}^{(k)}\mathbf{x} \right], \tag{1.1}$$

where

$$g^{(k)}(\mathbf{x}) := \log\left(\mathbf{1}_m^\top \exp(\mathbf{J}^{(k)}\mathbf{x} + \mathbf{b}^{(k)})\right)$$

is the log-sum-exp function for a linear model defined by $\mathbf{J}^{(k)} \in \mathbb{R}^{m\times n}$ and $\mathbf{b}^{(k)} \in \mathbb{R}^m$, $\mathbf{c}^{(k)} \in \mathbb{R}^m$, $\mathbf{1}_m \in \mathbb{R}^m$ is a vector of all ones, $w^{(k)}$'s are weights, and $N$ is the number of linear models. Problem (1.1) arises commonly in machine learning and optimization. For example, multinomial logistic regression (MLR) in classification problems [20, 37, 50] is formulated as (1.1). In geometric programming [46, 51, 52], a non-convex problem can be convexified through a reformulation to the form (1.1). The log-sum-exp function itself also has extensive applications in machine learning. For instance, it can serve as a smooth approximation to the element-wise maximum function [11, 38], where smoothness is desirable in model design since gradient-based optimizers are commonly used. Moreover, the log-sum-exp function is closely related to widely used softmax and entropy functions. For instance, the dual to an entropy maximization problem is a log-sum-exp minimization problem [3, Example 5.5], and the gradient of the log-sum-exp function is the softmax function [10].

[*]Received July 10, 2023. Accepted August 21, 2024. Published online on December 19, 2024. Recommended by Roland Herzog.

[†]Department of Mathematics, University of California, Los Angeles, USA (`kelvin.kan@math.ucla.edu`).
[‡]Departments of Mathematics and Computer Science, Emory University, USA (`{jnagy, lruthotto}@emory.edu`).

Despite the smoothness and convexity of the log-sum-exp function, a standard implementation of line-search Newton-type methods can be problematic. To realize this, note that the gradient and Hessian of the log-sum-exp function are given by

$$\nabla f(\mathbf{x}) = \sum_{k=1}^{N} w^{(k)} {\mathbf{J}^{(k)}}^\top (\mathbf{p}^{(k)} - \mathbf{c}^{(k)}) \qquad \text{and} \qquad \nabla^2 f(\mathbf{x}) = \sum_{k=1}^{N} w^{(k)} {\mathbf{J}^{(k)}}^\top \mathbf{H}^{(k)} \mathbf{J}^{(k)},$$

$$\text{with} \quad \mathbf{p}^{(k)} = \frac{\exp(\mathbf{J}^{(k)}\mathbf{x} + \mathbf{b}^{(k)})}{\mathbf{1}_m^\top \exp(\mathbf{J}^{(k)}\mathbf{x} + \mathbf{b}^{(k)})} \qquad \text{and} \qquad \mathbf{H}^{(k)} = \operatorname{diag}(\mathbf{p}^{(k)}) - \mathbf{p}^{(k)} {\mathbf{p}^{(k)}}^\top.$$

The Hessian is positive semi-definite and rank-deficient because the null space of the $\mathbf{H}^{(k)}$'s contains $\mathbf{1}_m$. Even more problematic is that when the $\mathbf{p}^{(k)}$'s are close to a standard basis vector (which, for example, commonly occurs in MLR), the Hessian is close to the zero matrix even when the gradient is non-zero. In Newton's method, this means that the local quadratic approximation can be unbounded from below. To be precise, it is unbounded from below if and only if the gradient is not in the column space of the Hessian [1, Exercise 2.19].

Disciplined convex programming (DCP) packages (e.g., CVX [15]) can reliably solve the log-sum-exp minimization problem through a reformulation. For instance, CVX first formulates the problem using exponential cones [34, Section 5.2.6] and applies backend solvers to solve the resulting problem directly (e.g., MOSEK [33]) or through successive polynomial approximation (e.g., SPDT3 [47] and SeDuMi [45]). However, this approach can be computationally demanding as the number of conic constraints scales with the product of the number of rows in the linear models and the number of linear models. For instance, CVX did not complete the image classification experiments for the whole dataset in Section 4.2 on a standard laptop in thirty minutes, while LSEMINK finishes on the same hardware in thirty seconds. Furthermore, the formulation relies on access to the elements of the $\mathbf{J}^{(k)}$'s, i.e., this approach is not applicable in a matrix-free setting where the $\mathbf{J}^{(k)}$'s are not built explicitly and only routines for performing matrix-vector products are provided.

Tikhonov regularization [9, 14, 17], which adds $\frac{\alpha}{2}\|\mathbf{x}\|_2^2$ with $\alpha > 0$ to the objective function, avoids the cost of reformulation and alleviates the convergence issues with Newton-type methods. The regularization shifts the Hessian by $\alpha\mathbf{I}$ and renders it positive definite, where $\mathbf{I}$ is the identity matrix. Nonetheless, Tikhonov regularization introduces a bias and consequently changes the optimal solution. The regularization parameter $\alpha$ has to be chosen judiciously—a large $\alpha$ renders the problem easier to solve and produces a more regular solution but introduces more bias. In addition, one cannot use effective parameter selection algorithms [4, 5, 13, 19, 49] for linear problems due to the nonlinearity of the log-sum-exp function. On the other hand, first-order methods like gradient descent [3, 39] or AdaGrad [7], which do not use the Hessian matrix, can avoid the problem. However, their convergence is inferior to methods that utilize curvature information [8].

Modified Newton-type methods effectively tackle problems with rank-deficient or indefinite Hessians and do not introduce bias. The idea is to add a shift to the Hessian so that at the $i$th iteration, the scheme solves

$$\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^\top (\nabla^2 f(\mathbf{x}_i) + \beta_i \mathbf{M}_i)(\mathbf{x} - \mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top (\mathbf{x} - \mathbf{x}_i), \tag{1.2}$$

where $\beta_i$ is a parameter and the shift $\mathbf{M}_i$ renders the Hessian to be sufficiently positive definite. The quadratic approximation is bounded from below since the modified Hessian is positive definite. Hence, the convergence issues are avoided. The effect of the Hessian shift is reminiscent of the Tikhonov regularization approach. Indeed, the scheme is sometimes called a Tikhonov-regularized Newton update [42, Chapter 3.3]. However, the key conceptual difference between (1.2) and Tikhonov regularization is that the former does not introduce any bias to the problem [48], i.e., the optimal solution to the problem is independent of the $\beta_i$'s. There are different ways of defining $\mathbf{M}_i$. For instance, $\mathbf{M}_i$ is spanned by some of the eigenvectors of the Hessian [16, 39] or is a modification to the factorization of the Hessian [12, 32, 35]. However, the computations needed for these approaches are intractable for large-scale problems commonly arising in machine learning. A simple and computationally feasible approach is to set $\mathbf{M}_i$ as the identity matrix [29, 30, 42], which will be used as a comparing method in our numerical experiments.

In this paper, we propose LSEMINK, a novel modified Newton–Krylov method that circumvents the drawbacks outlined above. The main novelty in our method is the Hessian shift $\mathbf{M}_i = \sum_{k=1}^N w^{(k)} {\mathbf{J}^{(k)}}^\top \mathbf{J}^{(k)}$. This generates an update in the row space of the linear model, as compared to the aforementioned modified Newton-type methods, which returns an update in the parameter space of the linear model (i.e., the $\mathbf{x}$-space). This property is preferable in machine learning applications since model parameters often do not have an intuitive meaning, while the row space of the linear model contains interpretable data features. Note that standard convergence guarantees (e.g., [39, Chapter 6.2]), which often require positive definiteness of the modified Hessian, do not apply to our method since our modified Hessian can be rank-deficient. We show that the quadratic approximation is bounded from below, and the overall scheme provably converges to a global minimum. Since a Krylov subspace method is applied to approximately solve (1.2) to obtain the next iterate, LSEMINK is suitable for large-scale problems where the linear models are expensive to build and are only available through matrix-vector multiplications. Our numerical experiments on image classification and geometric programming illustrate that LSEMINK considerably reduces the time-to-solution, increases the scalability compared to DCP and natural gradient descent, and has significantly faster initial convergence than standard Newton–Krylov methods.

This paper is organized as follows. In Section 2, we describe the proposed LSEMINK algorithm. In Section 3, we provide a global convergence guarantee. In Section 4, we demonstrate the effectiveness of LSEMINK using two numerical experiments motivated by geometric programming and image classification, respectively. We finally conclude the paper in Section 5.

## 2. LSEMINK.

We propose LSEMINK, a modified Newton–Krylov method geared toward log-sum-exp minimization problems of the form (1.1). At the $i$th iteration, we first consider the quadratic approximation (1.2) with $\mathbf{M}_i = \sum_{k=1}^N w^{(k)} {\mathbf{J}^{(k)}}^\top \mathbf{J}^{(k)}$. That is,

$$
\begin{aligned}
\min_{\mathbf{x}} q_i(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^\top \left( \nabla^2 f(\mathbf{x}_i) + \beta_i \sum_{k=1}^N w^{(k)} {\mathbf{J}^{(k)}}^\top \mathbf{J}^{(k)} \right)(\mathbf{x}-\mathbf{x}_i) \\
&\qquad + \nabla f(\mathbf{x}_i)^\top (\mathbf{x}-\mathbf{x}_i) \\
&= \frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^\top \left[ \sum_{k=1}^N w^{(k)} \left( {\mathbf{J}^{(k)}}^\top (\mathbf{H}_i^{(k)} + \beta_i \mathbf{I}) \mathbf{J}^{(k)} \right) \right] (\mathbf{x}-\mathbf{x}_i) \\
&\qquad + \nabla f(\mathbf{x}_i)^\top (\mathbf{x}-\mathbf{x}_i),
\end{aligned} \tag{2.1}
$$

whose minimizer is given by $\mathbf{x}_i + \Delta\mathbf{x}_i$, where $\Delta\mathbf{x}_i$ solves the Newton equation

$$
\nabla^2 q_i(\mathbf{x}_i)\Delta\mathbf{x}_i = -\nabla q_i(\mathbf{x}_i), \tag{2.2}
$$

and $\mathbf{H}_i^{(k)}$ is $\mathbf{H}^{(k)}$ evaluated at $\mathbf{x}_i$. It is important to note that the Hessian shift in (2.1) is different from the typical modified Newton approaches (e.g., eigenvalue modification [16, 39], identity matrix [29, 30, 42], or modification to the factorization of the Hessian [12, 32, 35]) which seek to obtain a positive definite Hessian and lead to an update in the parameter space of the linear model (i.e., the $\mathbf{x}$-space). Instead, it generates an update direction in the row space of the linear models. This is preferable, especially in machine learning applications, because model parameters often do not have an intuitive meaning, while the row space of the linear models contains data features and is explicable. Although the Hessian of (2.1) is rank-deficient especially when the linear models are over-parametrized (i.e., the $\mathbf{J}^{(k)}$'s have more columns than rows), it is positive definite in the row space of the linear model. Consequently, the quadratic approximation is bounded from below, and the overall scheme provably converges to a global minimum; see Section 3 for a detailed derivation.

An alternative formulation for (2.1) is

$$\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^\top \nabla^2 f(\mathbf{x}_i)(\mathbf{x}-\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(\mathbf{x}-\mathbf{x}_i) + \frac{\beta_i}{2}\sum_{k=1}^{N} w^{(k)} \|\mathbf{J}^{(k)}(\mathbf{x}-\mathbf{x}_i)\|_2^2,$$

which can be interpreted as a Newton scheme with a proximal term acting on the row space of the $\mathbf{J}^{(k)}$'s. This formulation shows that $\beta_i$ controls the step size in a nonlinear line-search arc. To be precise, $\beta_i = 0$ and $\infty$ correspond to a Newton update with step size $1$ and $0$, respectively, and the update is given nonlinearly for $0 < \beta_i < \infty$. The formulation also shows that our proposed scheme bears similarity to $L^2$-natural gradient descent (NGD) methods [40, 43], which use the same proximal term. Nonetheless, unlike our approach, $L^2$-NGD methods generally do not directly incorporate Hessian information into its search direction and approximate curvature information using only the linear model.

The crucial difference between the proximal term and Tikhonov regularization is that the former does not introduce any bias [42, 48], i.e., the optimal solution is independent of $\beta_i$. Another advantage is that Tikhonov regularization requires parameter tuning, which is commonly done using a grid search for nonlinear problems like (1.1). In our proposed method, the $\beta_i$'s are automatically selected by a backtracking Armijo line-search scheme. The proposed scheme can also be perceived as a proximal-point algorithm acting on the second-order approximation [42].

We compute the update direction $\Delta\mathbf{x}_i$ by approximately solving the Newton equation (2.2) using a Krylov subspace method (e.g., a conjugate gradient method [3, 39]) and obtain the next iterate $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i$. In particular, the Krylov subspace is given by

$$\begin{aligned} &\mathcal{K}_r(\nabla^2 q_i(\mathbf{x}_i), \nabla q_i(\mathbf{x}_i)) \\ &= \mathcal{K}_r\left(\sum_{k=1}^{N} w^{(k)}\left({\mathbf{J}^{(k)}}^\top(\mathbf{H}_i^{(k)} + \beta_i\mathbf{I})\mathbf{J}^{(k)}\right), \sum_{k=1}^{N} w^{(k)}{\mathbf{J}^{(k)}}^\top(\mathbf{p}_i^{(k)} - \mathbf{c}^{(k)})\right), \end{aligned} \tag{2.3}$$

where $r$ is the dimension of the Krylov subspace and $\mathbf{p}_i^{(k)}$ is $\mathbf{p}^{(k)}$ evaluated at $\mathbf{x}_i$. Since the Krylov subspace method only requires routines to perform Hessian-vector multiplications, LSEMINK is applicable to large-scale problems commonly arising in machine learning applications where the linear models are only available through matrix-vector products. An outline of the implementation of LSEMINK is presented in Algorithm 1.

LSEMINK has significantly faster initial convergence compared with standard Newton–Krylov solvers. This is particularly attractive in applications that do not require high accuracy, e.g., image classification. LSEMINK also considerably reduces the time-to-solution and has better scalability compared to geometric programming and natural gradient descent approaches. It avoids the respective drawbacks of the solvers outlined in Section 1. Moreover, it is more robust to ill-conditioning arising from the nonsmoothness of the problem; see Section 4 for

**Algorithm 1** Outline of LSEMINK for solving (1.1).

```
1:  Inputs: Linear models $\mathbf{x} \mapsto \mathbf{J}^{(k)}\mathbf{x}$, $\mathbf{x} \mapsto {\mathbf{J}^{(k)}}^\top \mathbf{x}$, $\mathbf{b}^{(k)}$, $\mathbf{c}^{(k)}$, and weights $w^{(k)}$ for
    $k = 1, 2, \ldots, N$. Initial guess $\mathbf{x}_0$, initial $\beta_0$.
2:  Inputs: Tolerances xtol, gtol for Newton iterations. Tolerances ktol and kmaxiter for the
    Krylov subspace method. Line search parameter $\gamma \in (0, 1)$.
3:  for $i = 0, 1, 2, \ldots$ do
4:    compute $f(\mathbf{x}_i)$, $\nabla f(\mathbf{x}_i)$ and build routines for performing $\mathbf{v} \mapsto \nabla^2 f(\mathbf{x}_i)\mathbf{v}$
5:    for $j = 0, 1, 2, \ldots$ do
6:      compute $\Delta\mathbf{x}_i$ by applying Krylov-subspace methods to approximately solve
        $\nabla^2 q_i(\mathbf{x}_i)\Delta\mathbf{x}_i = -\nabla q_i(\mathbf{x}_i)$ with the current $\beta_i$ and the Krylov subspace
        $\mathcal{K}_r(\nabla^2 q_i(\mathbf{x}_i), \nabla q_i(\mathbf{x}_i))$ until the relative residue drops below ktol or the number of
        iterations exceeds kmaxiter
7:      if $f(\mathbf{x}_i + \Delta\mathbf{x}_i) < f(\mathbf{x}_i) + \gamma \nabla f(\mathbf{x}_i)^\top \Delta\mathbf{x}_i$ then
8:        set $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i$ and break
9:      else
10:       set $\beta_i = 2\beta_i$
11:     end if
12:   end for
13:   if $\|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2 / \|\mathbf{x}_i\|_2 <$ xtol or $\|\nabla f(\mathbf{x}_{i+1})\|_2 <$ gtol then
14:     break
15:   end if
16:   if $j = 0$ then
17:     set $\beta_{i+1} = 0.5 * \beta_i$
18:   else
19:     set $\beta_{i+1} = \beta_i$
20:   end if
21: end for
22: Output: approximate solution $\mathbf{x}_{i+1}$.
```

numerical experiments. The implementation[1] is easy to experiment with, as it only requires minimal knowledge and input from the user.

## 3. Proof of global convergence.

In this section, we prove the global convergence of the proposed LSEMINK algorithm. It is noteworthy that existing convergence results cannot be directly applied due to the rank-deficiency of our modified Hessian. For instance, it is assumed in [39, Chapter 6.2] that the modified Hessian is positive definite and has a bounded condition number. Our proof is modified from the approach in [28], which studies proximal Newton-type methods for composite functions. We first state the main theorem.

THEOREM 3.1. *Assume that $f$ is defined in* (1.1) *and* $\inf_{\mathbf{x}} f(\mathbf{x})$ *is attained in* $\mathbb{R}$. *Then the sequence* $\{\mathbf{x}_i\}_i$ *generated by LSEMINK converges to a global minimum regardless of the choice of the initial guess* $\mathbf{x}_0$.

We note that Theorem 3.1 also applies to the case where the Newton equation (2.2) is solved exactly. In the following, we will first discuss some properties of LSEMINK. We will then state and prove four lemmas, which will aid the proof of Theorem 3.1.

For simplicity of exposition and without loss of generality, in this section, we drop the superscript and focus on the case with only one linear model defined by $\mathbf{J}$, $\mathbf{b}$, and $\mathbf{c}$, and the

[1] We provide a MATLAB implementation at `https://github.com/KelvinKan/LSEMINK`.

weight $w = 1$. In this case, the Krylov subspace in (2.3) becomes

$$\mathcal{K}_r(\nabla^2 q_i(\mathbf{x}_i), \nabla q_i(\mathbf{x}_i)) = \mathcal{K}_r(\mathbf{J}^\top(\mathbf{H}_i + \beta_i \mathbf{I})\mathbf{J}, \mathbf{J}^\top(\mathbf{p}_i - \mathbf{c})). \tag{3.1}$$

We note that our proof can straightforwardly be extended to the general case by setting

$$\mathbf{J} = [\mathbf{J}^{(1)}; \ldots; \mathbf{J}^{(N)}], \qquad \mathbf{c} = [w^{(1)}\mathbf{c}^{(1)}; \ldots; w^{(N)}\mathbf{c}^{(N)}],$$
$$\mathbf{p}_i = [w^{(1)}\mathbf{p}_i^{(1)}; \ldots; w^{(N)}\mathbf{p}_i^{(N)}], \quad \text{and} \quad \mathbf{H}_i = \operatorname{blkdiag}(w^{(1)}\mathbf{H}_i^{(1)}, \ldots, w^{(N)}\mathbf{H}_i^{(N)}),$$

where blkdiag denotes a block diagonal matrix.

Recall that the Krylov subspace in (3.1) is constructed to approximately solve the Newton equation and obtain the update direction $\Delta\mathbf{x}_i$. This is equivalent to building a rank-$r$ approximation $\nabla^2 q_i(\mathbf{x}_i) \approx \mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top$ and computing the next iterate by

$$\mathbf{x}_{i+1} = \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^\top \mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top(\mathbf{x} - \mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(\mathbf{x} - \mathbf{x}_i). \tag{3.2}$$

Here, the columns of $\mathbf{V}_i \in \mathbb{R}^{n\times r}$ form an orthonormal basis for the Krylov subspace, and $\mathbf{T}_i \in \mathbb{R}^{r\times r}$. The precise structure of $\mathbf{T}_i$ depends on the Krylov subspace method. For instance, if we use the conjugate gradient (CG) method, then $\mathbf{T}_i$ is tridiagonal [31, Section 3.1]. Since $\nabla f(\mathbf{x}_i) \in \operatorname{row}(\mathbf{J}) = \operatorname{col}(\mathbf{J}^\top(\mathbf{H}_i + \beta_i\mathbf{I})\mathbf{J})$ for $\beta_i > 0$ and the Krylov subspace always contains $\nabla f(\mathbf{x}_i)$, the column space of $\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top$ always contains $\nabla f(\mathbf{x}_i)$. This means that the quadratic function (3.2) is bounded from below [1] and admits a minimum. The iterate $\mathbf{x}_{i+1}$ is the minimum-norm solution of (3.2) given by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i, \qquad \text{where} \qquad \Delta\mathbf{x}_i = -\mathbf{V}_i\mathbf{T}_i^{-1}\mathbf{V}_i^\top\nabla f(\mathbf{x}_i). \tag{3.3}$$

Next, we state and prove some lemmas, which will be used to prove the main theorem.

LEMMA 3.2 (Update Direction). *The update $\Delta\mathbf{x}_i$ generated by the iterative scheme* (3.3) *satisfies*

$$\Delta\mathbf{x}_i \in \operatorname{row}(\mathbf{J}), \tag{3.4}$$
$$\Delta\mathbf{x}_i^\top \nabla^2 q_i(\mathbf{x}_i)\Delta\mathbf{x}_i = \Delta\mathbf{x}_i^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i. \tag{3.5}$$

*Here,* (3.4) *means that the update direction is in the row space of the linear model.*

*Proof.* By construction, the Krylov subspace (2.3) is a subspace of $\operatorname{row}(\mathbf{J})$, and by (3.3) we have $\Delta\mathbf{x}_i \in \operatorname{col}(\mathbf{V}_i)$. Thus we have $\Delta\mathbf{x}_i \in \operatorname{col}(\mathbf{V}_i) \subseteq \operatorname{row}(\mathbf{J})$, which proves (3.4).

To prove (3.5) we note that $\Delta\mathbf{x}_i \in \operatorname{col}(\mathbf{V}_i)$, and hence its projection onto the orthogonal complement of $\operatorname{col}(\mathbf{V}_i)$ vanishes. To be more precise, consider the full representation of the Hessian of (2.1) obtained by performing $n$ iterations of the Krylov subspace method

$$\nabla^2 q_i(\mathbf{x}_i) = \mathbf{J}^\top(\mathbf{H}_i + \beta_i\mathbf{I})\mathbf{J} = \tilde{\mathbf{V}}_i\tilde{\mathbf{T}}_i\tilde{\mathbf{V}}_i^\top.$$

Here, $\tilde{\mathbf{V}}_i \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, and its first $r$ columns satisfy $\tilde{\mathbf{V}}_i(:, 1:r) = \mathbf{V}_i$, $\tilde{\mathbf{T}}_i \in \mathbb{R}^{n\times n}$, and its $r$th leading principal submatrix satisfies $\tilde{\mathbf{T}}_i(1:r, 1:r) = \mathbf{T}_i$. The structure of $\tilde{\mathbf{V}}_i$ depends on the Krylov subspace method. For instance, $\tilde{\mathbf{V}}_i$ is tridiagonal if we use the CG method [31]. We have

$$\begin{aligned}\Delta\mathbf{x}_i^\top\nabla^2 q_i(\mathbf{x}_i)\Delta\mathbf{x}_i &= \Delta\mathbf{x}_i^\top\tilde{\mathbf{V}}_i\tilde{\mathbf{T}}_i\tilde{\mathbf{V}}_i^\top\Delta\mathbf{x}_i \\ &= \begin{bmatrix}\Delta\mathbf{x}_i^\top\mathbf{V}_i & \mathbf{0}\end{bmatrix}\tilde{\mathbf{V}}_i\begin{bmatrix}\mathbf{V}_i^\top\Delta\mathbf{x}_i \\ \mathbf{0}\end{bmatrix} = \Delta\mathbf{x}_i^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i\,.\end{aligned}$$

Here, in the second step we used $\Delta\mathbf{x}_i \in \operatorname{col}(\mathbf{V}_i) \perp \operatorname{col}(\tilde{\mathbf{V}}_i(:, r+1:n))$. This proves (3.5).
□

LEMMA 3.3 (Descent Direction). *The update $\Delta\mathbf{x}_i$ generated by (3.3) satisfies the descent condition*

$$\nabla f(\mathbf{x}_i)^\top \Delta\mathbf{x}_i \le -\Delta\mathbf{x}_i^\top \mathbf{J}^\top(\mathbf{H}_i + \beta_i\mathbf{I})\mathbf{J}\Delta\mathbf{x}_i\,. \tag{3.6}$$

*Proof.* Since $\mathbf{x}_{i+1}$ is a solution to (3.2), for any $t \in (0,1)$, we have

$$\frac{1}{2}\Delta\mathbf{x}_i^\top \mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i + \nabla f(\mathbf{x}_i)^\top\Delta\mathbf{x}_i \le \frac{1}{2}(t\Delta\mathbf{x}_i)^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top(t\Delta\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(t\Delta\mathbf{x}_i).$$

By rearranging the terms, we have

$$\begin{aligned}\frac{(1-t^2)}{2}\Delta\mathbf{x}_i^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i + (1-t)\nabla f(\mathbf{x}_i)^\top\Delta\mathbf{x}_i &\le 0\\ \frac{(1+t)}{2}\Delta\mathbf{x}_i^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i + \nabla f(\mathbf{x}_i)^\top\Delta\mathbf{x}_i &\le 0\\ \nabla f(\mathbf{x}_i)^\top\Delta\mathbf{x}_i &\le -\frac{(1+t)}{2}\Delta\mathbf{x}_i^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i\,.\end{aligned}$$

Letting $t \to 1^-$, we obtain

$$\nabla f(\mathbf{x}_i)^\top\Delta\mathbf{x}_i \le -\Delta\mathbf{x}_i^\top\mathbf{V}_i\mathbf{T}_i\mathbf{V}_i^\top\Delta\mathbf{x}_i\,. \tag{3.7}$$

Combining (3.5) and (3.7), we obtain (3.6). ∎

In the following lemma, we will make use of the fact that $\nabla f$ is Lipschitz continuous. This is because the gradient of the log-sum-exp function is the softmax function, which is Lipschitz continuous [10, 24].

LEMMA 3.4 (Armijo Line-Search Condition). *Let $\lambda_{\min}$ be the smallest nonzero eigenvalue of $\mathbf{J}^\top\mathbf{J}$ and $L$ be the Lipschitz constant for $\nabla f$. For a line-search parameter $\gamma \in (0,1)$ and*

$$\beta_i \ge \frac{L}{2\lambda_{\min}(1-\gamma)}, \tag{3.8}$$

*the following Armijo line-search condition holds:*

$$f(\mathbf{x}_{i+1}) \le f(\mathbf{x}_i) + \gamma\nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i). \tag{3.9}$$

In (3.8), the constant $L$ is the global Lipschitz constant for $\nabla f$. In practice, as long as $\beta_i$ satisfies (3.8) with a local Lipschitz constant of $\nabla f$ around $\mathbf{x}_i$, the Armijo condition (3.9) will hold. Recall that a larger $\beta_i$ corresponds to a smaller step size in a nonlinear line-search arc. The line-search scheme of LSEMINK adaptively chooses the value of $\beta_i$ so that the Armijo condition is satisfied. If the Armijo condition is not satisfied, we then increase $\beta_i$ (decrease the step size); see lines 7–11 of Algorithm 1. Theorem 3.4 guarantees that the Armijo condition must hold for a large enough $\beta_i$. Therefore, the line-search scheme must stop at some point and identify a $\beta_i$ that satisfies the Armijo condition. Moreover, we decrease $\beta_i$ (increase the step size) if the Armijo condition is satisfied before the line search; see lines 16–20 of Algorithm 1. This allows LSEMINK to flexibly take a larger step and make more progress.

*Proof of Lemma 3.4.* First, note that

$$\|\mathbf{J}(\mathbf{x}_{i+1} - \mathbf{x}_i)\|^2_{\mathbf{H}_i+\beta_i\mathbf{I}} \ge \beta_i\|\mathbf{J}(\mathbf{x}_{i+1} - \mathbf{x}_i)\|_2^2 \ge \beta_i\lambda_{\min}\|(\mathbf{x}_{i+1} - \mathbf{x}_i)\|_2^2. \tag{3.10}$$

Here, in the second step we used that $(\mathbf{x}_{i+1} - \mathbf{x}_i) \in \text{row}(\mathbf{J}) = \text{row}(\mathbf{J}^\top\mathbf{J})$ (Lemma 3.2), $\text{row}(\mathbf{J}^\top\mathbf{J})^\perp = \text{null}(\mathbf{J}^\top\mathbf{J})$, and $\lambda_{\min}$ is the smallest nonzero eigenvalue of $\mathbf{J}^\top\mathbf{J}$. Next, we have

$$\begin{aligned}
f(\mathbf{x}_{i+1}) &\le f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i) + \frac{L}{2}\|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2^2 \\
&\le f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i) + \beta_i\lambda_{\min}(1-\gamma)\|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2^2 \\
&\le f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i) + (1-\gamma)\|\mathbf{J}(\mathbf{x}_{i+1} - \mathbf{x}_i)\|_{\mathbf{H}_i+\beta_i\mathbf{I}}^2 \\
&\le f(\mathbf{x}_i) + \nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i) - (1-\gamma)\nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i) \\
&= f(\mathbf{x}_i) + \gamma\nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i).
\end{aligned}$$

Here, the first, second, third, and fourth steps use the Lipschitz continuity of $\nabla f$, (3.8), (3.10), and Lemma 3.3, respectively. ◻

LEMMA 3.5 (Stationary Point). *The iterative scheme (3.3) generates a fixed point $\mathbf{x}_*$ if and only if $\mathbf{x}_*$ is a stationary point.*

*Proof.* "$\Leftarrow$": Substituting $\nabla f(\mathbf{x}_*) = \mathbf{0}$ into (3.3), we obtain $\Delta\mathbf{x}_* = 0$. Hence, $\mathbf{x}_*$ is a fixed point.
"$\Rightarrow$": Let $\mathbf{v} = \mathbf{x} - \mathbf{x}_*$ for an arbitrary $\mathbf{x}$. Since $\mathbf{x}_*$ is a fixed point to (3.2), we have, for any $t \in \mathbb{R}$,

$$\begin{aligned}
&\frac{1}{2}(t\mathbf{v})^\top\mathbf{V}_*\mathbf{T}_*\mathbf{V}_*^\top(t\mathbf{v}) + \nabla f(\mathbf{x}_*)^\top(t\mathbf{v}) \\
&\qquad \ge \frac{1}{2}(\mathbf{x}_* - \mathbf{x}_*)^\top\mathbf{V}_*\mathbf{T}_*\mathbf{V}_*^\top(\mathbf{x}_* - \mathbf{x}_*) + \nabla f(\mathbf{x}_*)^\top(\mathbf{x}_* - \mathbf{x}_*).
\end{aligned}$$

Simplifying this, we obtain

$$\begin{aligned}
\frac{t^2}{2}\mathbf{v}^\top\mathbf{V}_*\mathbf{T}_*\mathbf{V}_*^\top\mathbf{v} + t\nabla f(\mathbf{x}_*)^\top\mathbf{v} \ge \mathbf{0}, \\
\nabla f(\mathbf{x}_*)^\top\mathbf{v} \ge -\frac{t}{2}\mathbf{v}^\top\mathbf{V}_*\mathbf{T}_*\mathbf{V}_*^\top\mathbf{v}.
\end{aligned}$$

Taking $t \to 0$, we obtain $\nabla f(\mathbf{x}_*)^\top\mathbf{v} \ge 0$ for any $\mathbf{v}$. This implies $\nabla f(\mathbf{x}_*)$ is a zero vector, that is, $\mathbf{x}_*$ is a stationary point. ◻

Now, we are ready to prove the main theorem.

*Proof of Theorem 3.1.* The sequence $\{f(\mathbf{x}_i)\}_i$ is decreasing because the update directions are descent directions (Lemma 3.3) and the Armijo line-search scheme guarantees sufficient descent at each step (Lemma 3.4). By the continuity of $f$, it is closed [2, Proposition 1.1.2]. Since $f$ is closed and attains its infimum in $\mathbb{R}$, the decreasing sequence $\{f(\mathbf{x}_i)\}_i$ converges to a limit.

By the sufficient descent condition (3.9), the convergence of $\{f(\mathbf{x}_i)\}_i$, and $\gamma > 0$, the term

$$\nabla f(\mathbf{x}_i)^\top(\mathbf{x}_{i+1} - \mathbf{x}_i)$$

converges to zero. Hence, by (3.6),

$$\Delta\mathbf{x}_i^\top\mathbf{J}^\top(\mathbf{H}_i + \beta_i\mathbf{I})\mathbf{J}\Delta\mathbf{x}_i$$

converges to zero. Since $(\mathbf{H}_i + \beta_i\mathbf{I})$ is positive definite and $\Delta\mathbf{x}_i \in \text{row}(\mathbf{J})$ (Lemma 3.2), $\Delta\mathbf{x}_i$ converges to the zero vector.

This implies that $\mathbf{x}_i$ converges to a fixed point of (3.3). By Lemma 3.5, $\mathbf{x}_i$ converges to a stationary point. By the convexity of $f$, $\mathbf{x}_i$ converges to a global minimum. ◻

## 4. Numerical experiments.

We perform two numerical experiments for minimizing the log-sum-exp function for a linear model. We compare the performance of the proposed LSEMINK method with three commonly applied line-search iterative methods and three disciplined convex programming (DCP) solvers; see Section 4.1. In Section 4.2, we consider multinomial logistic regression (MLR) arising in image classification. In Section 4.3, we experiment with a log-sum-exp minimization problem arising in geometric programming. The experimental results show that LSEMINK has much better initial convergence and is more robust and scalable than the comparing methods.

### 4.1. Benchmark methods.

We compare the proposed LSEMINK method with three common line-search iterative schemes and three DCP solvers for machine learning and geometric programming applications. Firstly, we implement a standard Newton-CG (NCG) algorithm with a backtracking Armijo line search. Secondly, we compare with an $L^2$-natural gradient descent (NGD) method [40, 43] that approximately solves

$$\min_{\mathbf{x}} \frac{1}{2}\nabla f(\mathbf{x}_i)^\top(\mathbf{x}-\mathbf{x}_i) + \frac{\lambda_i}{2}\sum_{k=1}^{N} w^{(k)}\|\mathbf{J}^{(k)}(\mathbf{x}-\mathbf{x}_i)\|_2^2,$$

using the CG method to obtain the next iterate, where $\lambda_i$ controls the step size and is determined by a backtracking Armijo line-search scheme and the last term is a proximal term acting on the row space of the linear model. This scheme bears similarity to LSEMINK as the proximal term has the same effect as the shift in the Hessian of LSEMINK. However, it does not use the Hessian and only approximates curvature information using the linear model. Thirdly, to demonstrate the effectiveness of the Hessian modification in LSEMINK, we compare with a standard modified Newton–Krylov (SMNK) scheme, which approximately solves (1.2) with $\mathbf{M}_i = \mathbf{I}$ using Lanczos tridiagonalization, which has the same iterates as the CG method up to rounding errors but allows computations for the update direction to be re-used during line search. For LSEMINK, the Newton equation (2.2) is approximately solved by CG. We note that an update direction has to be re-computed for each attempted value of $\beta_i$ during line search. In other words, unlike SMNK, the update direction computation cannot be re-used. However, our experimental results show that LSEMINK is still efficient in terms of computational cost thanks to the effectiveness of the modified Hessian. In each experiment, we use the same maximum number of iterations and tolerance for the CG and Lanczos schemes across different line-search iterative methods.

In addition, we apply CVX [15], a DCP package, paired with three different backend solvers (SPDT3 [47], SeDuMi [45], and MOSEK [33]). The best precision for CVX is used in the experiments; see [15] for detailed information.

*Cost measurement.* We measure the computational costs for different line-search iterative methods in terms of work units. In particular, a work unit represents a matrix-vector product with the linear models or their transpose. This is because these computations are usually the most expensive steps. For instance, in the MLR experiments of Section 4.2, the $\mathbf{J}^{(k)}$ matrices of the linear models contain the propagated high-dimensional features of all the training data. Note that the number of work units in one iteration can differ across different line-search iterative methods since a different number of CG/Lanczos iterations or line-search updates can be performed. In addition to work units, we also compare computational costs for all methods in total runtime.

### 4.2. Experiment 1: image classification.

Perhaps the most prominent example of log-sum-exp minimization is multinomial logistic regression (MLR) in supervised classification. Here, we experiment on an MLR problem for the classification of MNIST [27] and CIFAR-10 [25] image datasets. The MNIST dataset consists of 60,000 $28 \times 28$ hand-written images for

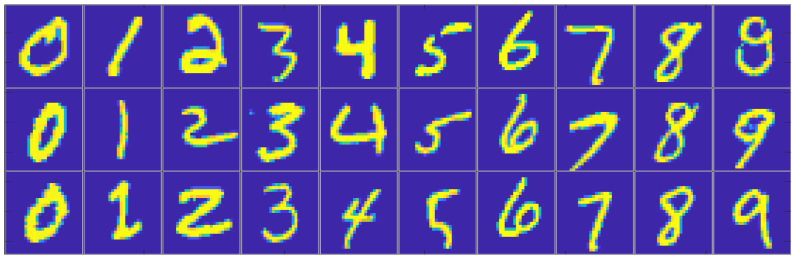

FIG. 4.1. *Example images from the MNIST data set.*

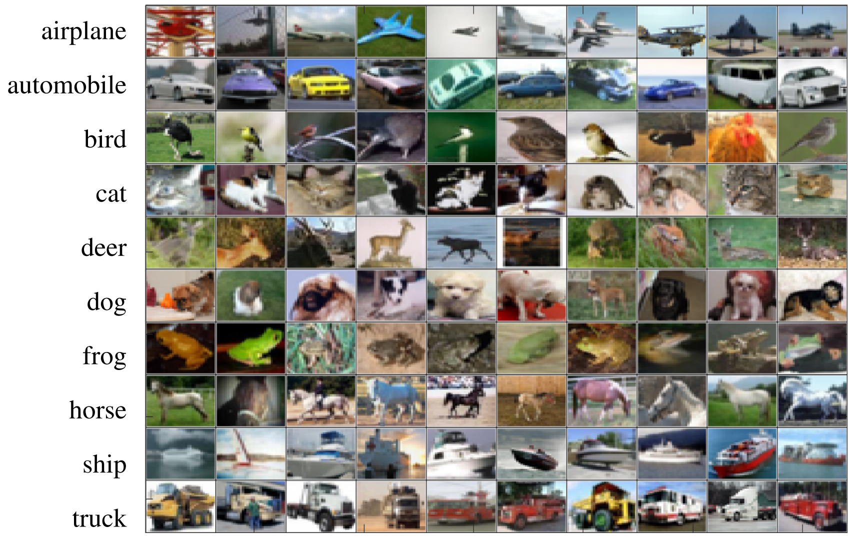

FIG. 4.2. *Example images for the CIFAR-10 dataset.*

digits from 0 to 9. The CIFAR-10 consists of 60,000 $32 \times 32$ color images equally distributed for the following ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Example images for the two datasets are shown in Figure 4.1 and Figure 4.2, respectively.

*Problem description.* Let $n_f$ be the number of features, $n_c$ be the number of classes, and $\Delta_{n_c}$ be the probability simplex in $\mathbb{R}^{n_c}$. We use $\{\mathbf{y}^{(k)}, \mathbf{c}^{(k)}\}_{k=1}^{N} \subset \mathbb{R}^{n_f} \times \Delta_{n_c}$ to denote the dataset, where $\mathbf{y}^{(k)}$ and $\mathbf{c}^{(k)}$ are the input feature and target output label, respectively. In our experiments, we consider two feature extractors that enhance the features $\mathbf{y}^{(k)}$ by propagating it into a higher-dimensional space $\mathbb{R}^{n_p}$. The first feature extractor is the random feature model (RFM) [18, 44]. It applies a nonlinear transformation given by

$$\mathbf{a}_{\mathrm{RFM}}(\mathbf{y}^{(k)}) = \sigma(\mathbf{Z}\mathbf{y}^{(k)} + \mathbf{b}),$$

where $\sigma$ is the element-wise ReLU activation function, $\mathbf{Z} \in \mathbb{R}^{n_p \times n_f}$, and $\mathbf{b} \in \mathbb{R}^{n_p}$ are randomly generated. The second feature extractor is performed by propagating the features through the hidden layers of a pre-trained AlexNet [26]. In particular, the AlexNet was pre-trained on the ImageNet dataset [6], which is similar to the CIFAR-10 dataset, using MATLAB's deep neural networks toolbox. This procedure is also known as transfer learning.

These feature extractors can empirically enhance the generalization of the model, i.e., the ability to classify unseen data correctly.

The goal of the supervised classification problem is to train a softmax classifier

$$s(\mathbf{X}, \mathbf{a}(\mathbf{y}^{(k)})) = \frac{\exp(\mathbf{X}\mathbf{a}(\mathbf{y}^{(k)}))}{\mathbf{1}_{n_c}\mathbf{1}_{n_c}^\top \exp(\mathbf{X}\mathbf{a}(\mathbf{y}^{(k)}))} \tag{4.1}$$

such that $s(\mathbf{X}, \mathbf{a}(\mathbf{y}^{(k)})) \approx \mathbf{c}^{(k)}$. Here, $\mathbf{X}$ are model parameters, the $\exp$ and division are applied element-wise, $\mathbf{1}_{n_c}$ is an $n_c$-dimensional vector of all ones, and $\mathbf{a} : \mathbb{R}^{n_f} \to \mathbb{R}^{n_p}$ is a feature extractor. To this end, we first consider the sample average approximation (SAA) [22, 23, 36] of an MLR problem formulated as

$$\begin{aligned}\min_{\mathbf{X}\in\mathbb{R}^{n_c\times n_p}} F(\mathbf{X}) &= -\frac{1}{N}\sum_{k=1}^{N} {\mathbf{c}^{(k)}}^\top \log\left(s(\mathbf{X}, \mathbf{a}(\mathbf{y}^{(k)}))\right) \\ &= \frac{1}{N}\sum_{k=1}^{N}\left[({\mathbf{c}^{(k)}}^\top \mathbf{1}_{n_c})\log\left(\mathbf{1}_{n_c}^\top \exp(\mathbf{X}\mathbf{a}(\mathbf{y}^{(k)}))\right) - {\mathbf{c}^{(k)}}^\top \mathbf{X}\mathbf{a}(\mathbf{y}^{(k)})\right] \\ &= \frac{1}{N}\sum_{k=1}^{N}\left[\log\left(\mathbf{1}_{n_c}^\top \exp(\mathbf{X}\mathbf{a}(\mathbf{y}^{(k)}))\right) - {\mathbf{c}^{(k)}}^\top \mathbf{X}\mathbf{a}(\mathbf{y}^{(k)})\right],\end{aligned}$$

where the $\log$ operation is applied element-wise, and we use the fact that ${\mathbf{c}^{(k)}}^\top \mathbf{1}_{n_c} = 1$ since $\mathbf{c}^{(k)} \in \Delta_{n_c}$. The feature extractor is assumed to be fixed since the focus is on the log-sum-exp minimization problem. We vectorize the variable $\mathbf{x} = \mathrm{vec}(\mathbf{X})$ so that the MLR problem becomes

$$\min_{\mathbf{x}\in\mathbb{R}^{n_c n_p}} f(\mathbf{x}) = \frac{1}{N}\sum_{k=1}^{N}\left[\log\left(\mathbf{1}_{n_c}^\top \exp(\mathbf{J}^{(k)}\mathbf{x})\right) - {\mathbf{c}^{(k)}}^\top \mathbf{J}^{(k)}\mathbf{x}\right],$$

which is of the form of (1.1) and where $\mathbf{J}^{(k)} = \mathbf{a}(\mathbf{y}^{(k)})^\top \otimes \mathbf{I}_{n_c}$.

*Experimental results.* In the MLR experiments, the line-search iterative solvers stop when the norm of gradient is below $10^{-14}$ or after 3,000 work units. We stop the CG and Lanczos scheme when the norm of the relative residual drops below $10^{-3}$ or after 20 iterations.

We first perform a small-scale experiment in which only $N = 100$ training data are used and a random feature model with dimension $m = 1{,}000$ is applied. Since under this setup the data can be fit perfectly to achieve a zero training error, the model predictions (4.1) are standard basis vectors, and the Hessian is a zero matrix at an optimum. In this situation, the second-order approximation is unbounded from below, and the robustness of the solvers can be tested. The results are reported in Table 4.1 and Figure 4.3. For the MNIST experiment, the standard Newton-CG scheme fails to converge near the end where the Hessian vanishes. For this, we report the values for the last iterate before the Hessian vanishes. The natural gradient descent method has the slowest convergence and has yet to converge at the end. Both the standard modified Newton–Krylov method and LSEMINK achieve the stopping criteria under the specified work units. In particular, LSEMINK has superior convergence where the objective function value is up to five orders of magnitude smaller than the second-best method during optimization. It also has better initial convergence and the fastest time-to-solution. This demonstrates the effectiveness of LSEMINK and the efficacy of its modified Hessian over the standard one. SeDuMi, particularly SDPT3, can achieve very accurate results, but their runtime is about 15 times more than the LSEMINK. MOSEK fails to obtain a solution.

**ETNA**
**Kent State University and**
**Johann Radon Institute (RICAM)**

TABLE 4.1
*Results on small-scale MLR experiments described in Section 4.2 in which the propagated random features have dimension $m = 1{,}000$ and $N = 100$ training data are used. The final objective function value, gradient norm, and total runtime are reported. The best results are highlighted in bold. Some results are not shown because the corresponding scheme fails to return a solution. The tests are run on an Apple Macbook Pro with a 10-core M1 Max CPU and 32 GB of memory, and the software platform is MATLAB R2022a.*

| Dataset | | NCG | NGD | SMNK | LSEMINK | SeDuMi | SDPT3 | MOSEK |
|---|---|---|---|---|---|---|---|---|
| MNIST | $f$ | 1.50e-15 | 1.54e-02 | 1.41e-15 | 8.37e-16 | 6.65e-15 | **0.00e+00** | – |
| | $\|\nabla f\|_2$ | 1.16e-14 | 1.33e-01 | 2.68e-15 | 7.05e-15 | 2.05e-14 | **5.14e-140** | – |
| | Time | 2.23s | 2.58s | 3.03s | **1.70s** | 37.88s | 28.51s | – |
| CIFAR-10 | $f$ | 1.31e-15 | 1.27e-02 | 4.26e-15 | 8.77e-16 | 7.93e-15 | **0.00e+00** | – |
| | $\|\nabla f\|_2$ | 6.16e-15 | 7.43e-02 | 6.58e-15 | 7.33e-15 | 2.11e-14 | **1.65e-212** | – |
| | Time | 1.95s | 2.60s | 3.02s | **1.69s** | 31.60s | 36.33s | – |

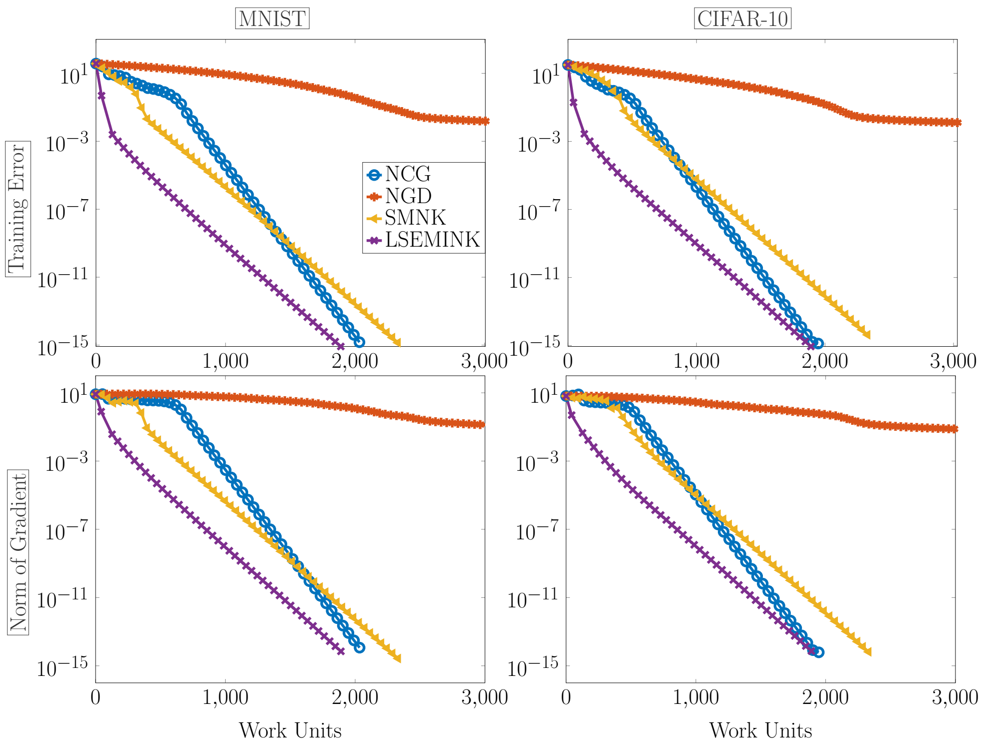

FIG. 4.3. *Experimental results on small-scale MLR experiments in which the propagated random features have dimensions $n_p = 1{,}000$ and $N = 100$ training data are used.*

We then experiment with $n = 50{,}000$ training data and 10,000 validation data. For the MNIST dataset, we use an RFM to propagate the features to an $m = 1{,}000$-dimensional space. For the CIFAR-10 dataset, features with dimension $m = 9{,}216$ are extracted from the *pool5* layer of a pre-trained AlexNet. Here, different feature extractors are used for the two datasets because a better validation accuracy can be achieved. In Figure 4.4, the results for an MLR problem are illustrated. In Figure 4.5, we report the performance for an MLR problem with a Tikhonov regularization term $\frac{\alpha}{2}\|\mathbf{x}\|_2^2$, where $\alpha = 10^{-3}$. Using our state-of-the-art laptop, the CVX solvers cannot complete the experiments within thirty minutes, while the line-search methods finish in thirty seconds. Hence, we focus on the latter methods in this test. The figures show that the $L^2$-natural gradient descent method is the slowest. The standard

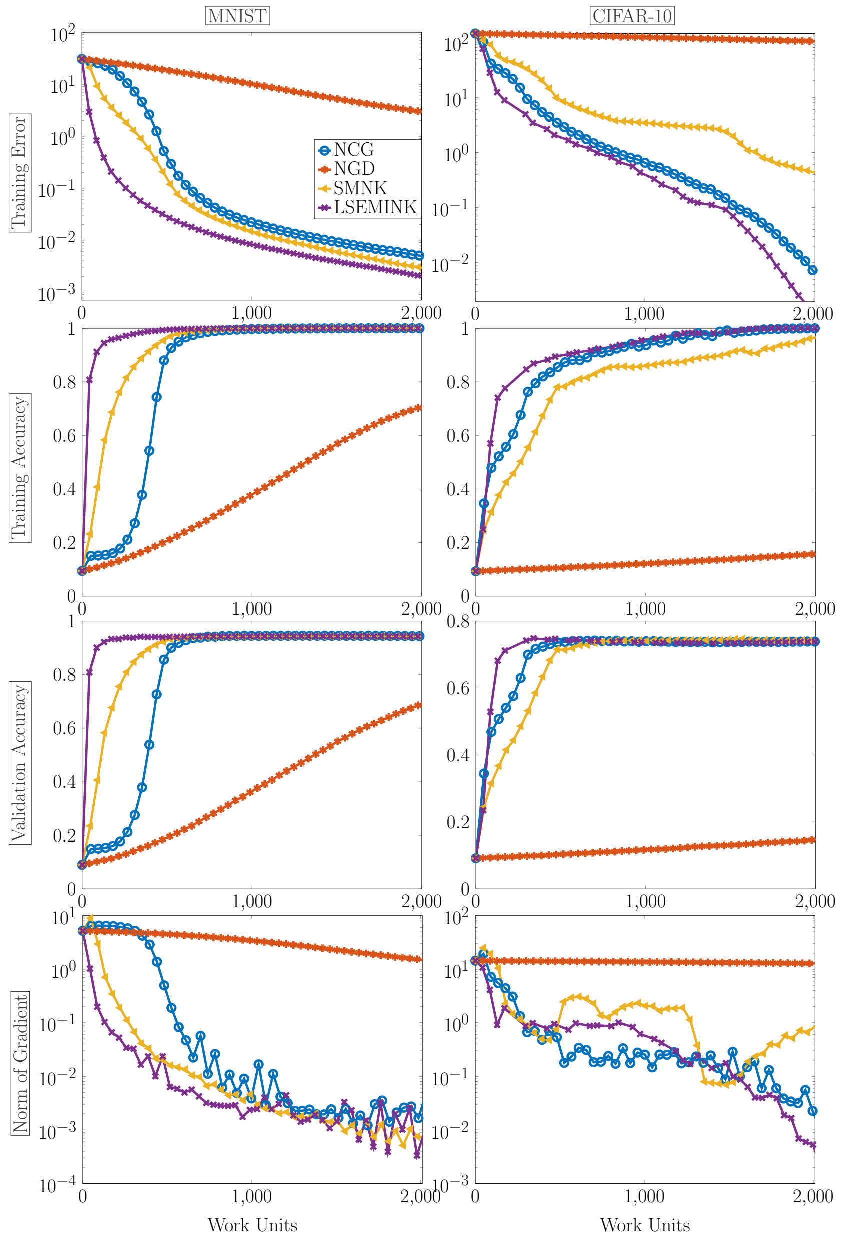

FIG. 4.4. *Experimental results on MLR without regularization. The $x$-axes report the number of work units. Here, $N = 50{,}000$ training data and $10{,}000$ validation data are used.*

Newton-CG and standard modified Newton–Krylov methods underperform in the MNIST and CIFAR-10 experiments, respectively. In contrast, LSEMINK is very competitive on both datasets. Specifically, it has good initial convergence where the objective function value is up to an order of magnitude smaller than the second-best scheme in the first few iterations. Moreover, its results are comparable with the other methods in terms of final training error, training accuracy, validation accuracy, and norm of the gradient.

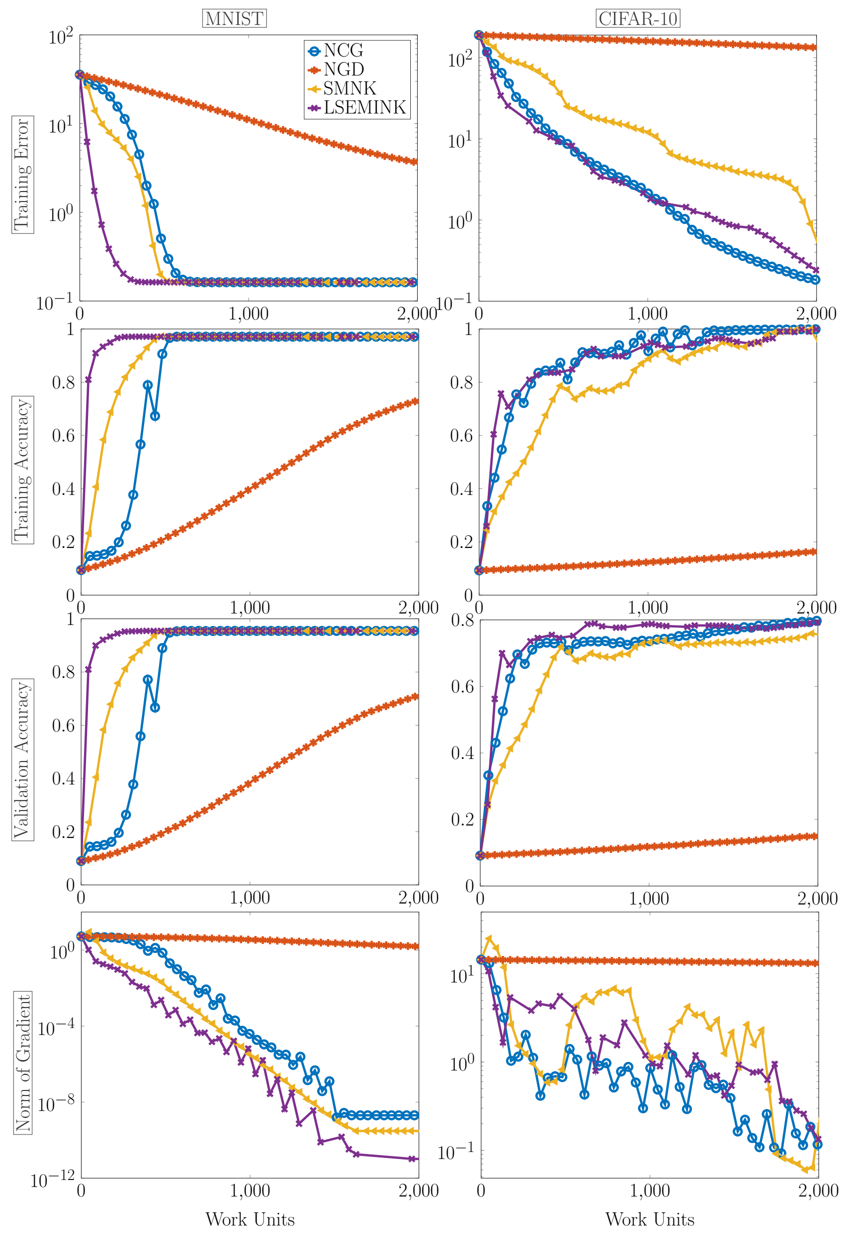

FIG. 4.5. *Experimental results on MLR with a Tikhonov regularization* $\frac{\alpha}{2}\|\mathbf{x}\|_2^2$*, with* $\alpha = 10^{-3}$*. The* $x$*-axes report the number of work units. Here,* $N = 50{,}000$ *training data and* $10{,}000$ *validation data are used.*

**4.3. Experiment 2: geometric programming.** We consider a log-sum-exp minimization problem which commonly arises in geometric programming [46, 51, 52] and is used to test optimization algorithms [21, 41]. In particular, it is formulated as

$$\min_{\mathbf{x}} \eta \log\left(\mathbf{1}_m^\top \exp((\mathbf{J}\mathbf{x} + \mathbf{b})/\eta)\right),$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{J} \in \mathbb{R}^{m\times n}$, and $\eta$ controls the smoothness of the problem. In particular, when $\eta \to 0$, the objective function converges to the point-wise maximum function $\max(\mathbf{J}\mathbf{x} + \mathbf{b})$ and its Hessian vanishes.

TABLE 4.2
*Results on geometric programming experiments described in Section 4.3. The final objective function value, gradient norm, and total runtime are reported. The best results are highlighted in bold. Some results are not shown because the corresponding scheme fails to return a solution. The tests are run on an Apple Macbook Pro with a 10-core M1 Max CPU and 32 GB of memory, and the software platform is MATLAB R2022a.*

| $\eta$ | | NCG | NGD | SMNK | LSEMINK | SeDuMi | SDPT3 | MOSEK |
|---|---|---|---|---|---|---|---|---|
| 1e-5 | $f$ | – | 7.48e+00 | 2.47e+00 | 1.44e+00 | **9.45e-01** | **9.45e-01** | – |
| | $\lVert\nabla f\rVert_2$ | – | 4.68e+00 | 5.96e+00 | 5.65e+00 | 5.76e-03 | **2.52e-06** | – |
| | Time | – | 0.44s | 1.43s | **0.38s** | 2.18s | 8.40s | – |
| 1e-3 | $f$ | – | 7.37e+00 | **9.48e-01** | **9.48e-01** | – | **9.48e-01** | – |
| | $\lVert\nabla f\rVert_2$ | – | 4.68e+00 | **3.80e-13** | 7.50e-11 | – | 2.51e-06 | – |
| | Time | – | 0.40s | 0.80s | **0.27s** | – | 17.85s | – |
| 1e-1 | $f$ | **1.24e+00** | 2.43e+00 | **1.24e+00** | **1.24e+00** | – | – | – |
| | $\lVert\nabla f\rVert_2$ | 2.72e-15 | 1.88e+00 | **2.38e-15** | 3.65e-15 | – | – | – |
| | Time | 0.09s | 0.35s | **0.02s** | **0.02s** | – | – | – |

We follow the experimental setups in [21, 41], which use $m = 100$, $n = 20$, and generate the entries of $\mathbf{J}$ and $\mathbf{b}$ randomly. We perform the experiments with small values of $\eta$ to test the robustness of the methods. In particular, we test with $\eta = 10^{-5}$, $10^{-3}$, and $10^{-1}$, respectively. We stop the line-search iterative schemes after $10{,}000$ work units. The CG and Lanczos schemes stop when the relative residual drops below $10^{-3}$ or after 20 iterations.

The experimental results are shown in Table 4.2 and Figure 4.6. We see that the experiments are very challenging as the standard Newton-CG method and all the CVX solvers cannot return a solution in some or all the experiments. In particular, the standard Newton-CG method breaks in the first iteration in two of the experiments. This is because the quadratic approximation is unbounded from below. Both SeDuMi and SDPT3 algorithms fail in some of the experiments. MOSEK fails in all the experiments. When the CVX solvers succeed in returning a solution, they have significantly longer runtime (up to 60 times slower) compared to the line-search methods. Similar to the previous experiments, the $L^2$-natural gradient descent method has the slowest convergence and has yet to converge after the specified work units. The standard modified Newton–Krylov method and LSEMINK are robust in the experiments and can return accurate solutions for $\eta = 10^{-3}$ and $10^{-1}$. This indicates the effectiveness of the Hessian modification in handling challenging optimization problems. Moreover, LSEMINK converges faster than the comparing standard modified Newton–Krylov method in the early stage. This indicates the effectiveness of the proposed Hessian modification over the standard one. However, when $\eta = 10^{-3}$ and $10^{-5}$, LSEMINK and all comparing methods cannot return a solution with the desired gradient norm. This is because the convergence of gradient-based methods like LSEMINK requires the differentiability of the objective function. However, for a small $\eta$, the objective function is close to being non-differentiable. Moreover, this also implies that the gradient is close to being discontinuous, which can be a reason of the oscillating gradient norm in Figure 4.6 when $\eta = 10^{-5}$.

## 5. Conclusion.

We present LSEMINK, a modified Newton–Krylov algorithm tailored for optimizing the log-sum-exp function for a linear model. The novelty of our approach is incorporating a Hessian shift in the row space of the linear model. This does not change the minimizers and renders the quadratic approximation to be bounded from below and the overall scheme to provably converge to a global minimum under standard assumptions. Since the update direction is computed using Krylov subspace methods which only require matrix-vector products with the linear model, LSEMINK is applicable to large-scale problems. Numerical experiments on image classification and geometric programming illustrate that LSEMINK has significantly faster initial convergence than standard Newton–Krylov methods, which is particularly attractive in applications like machine learning, considerably reduces the time-to-solution, and is more scalable compared to DCP solvers and natural gradient descent

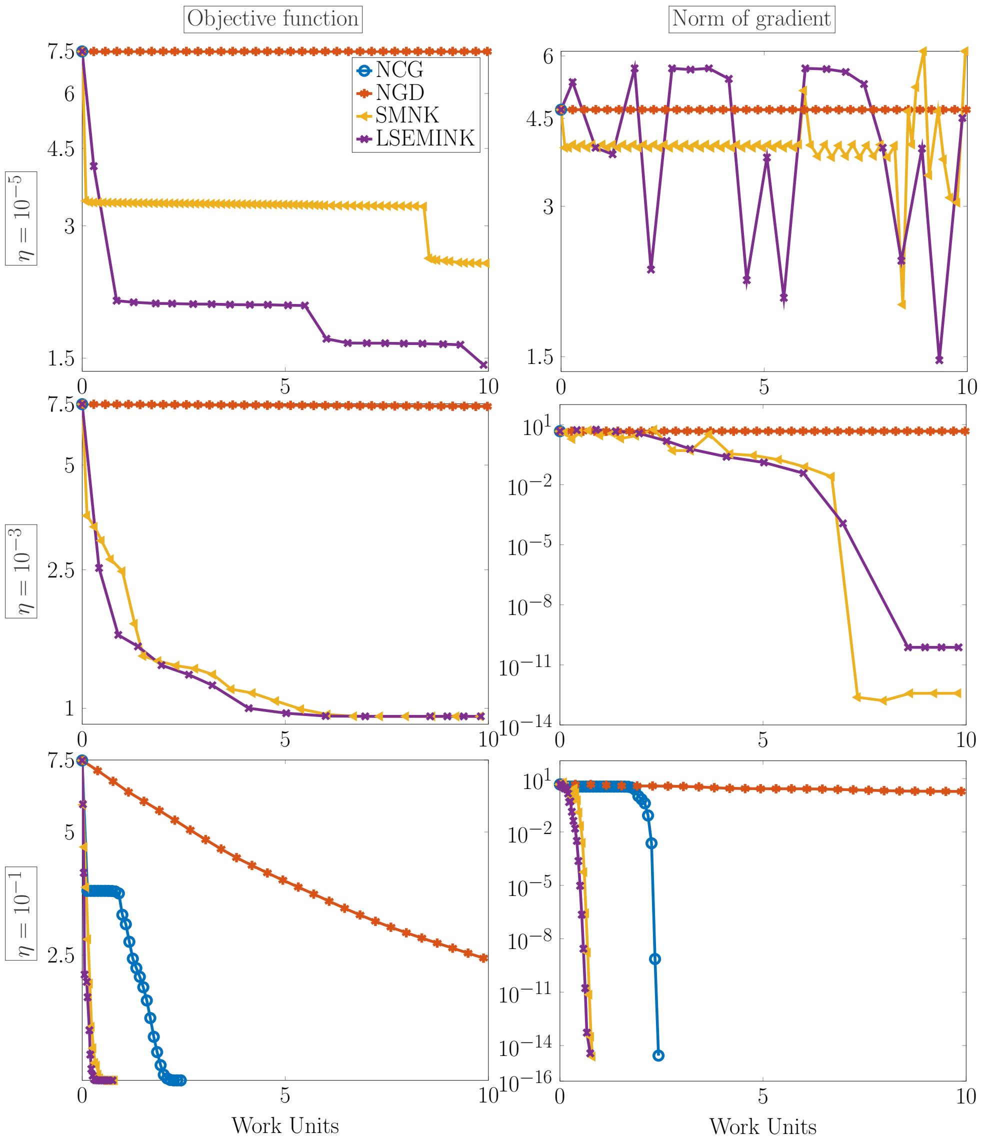

FIG. 4.6. *Experimental results on geometric programming. The $x$-axes report the number of work units (in thousands). The results for the standard Newton-CG scheme are not shown because it fails in the first iteration.*

methods. Also, LSEMINK is more robust to ill-conditioning arising from the nonsmoothness of the problem. We provide a MATLAB implementation at https://github.com/KelvinKan/LSEMINK.

**Acknowledgements.** This work was supported in part by NSF awards DMS 1751636, DMS 2038118, DMS-2208294, AFOSR grant FA9550-20-1-0372, and US DOE Office of Advanced Scientific Computing Research Field Work Proposal 20-023231. The authors would like to thank Samy Wu Fung for sharing the code for propagating the features of the CIFAR-10 dataset with AlexNet.

## REFERENCES

[1] A. Beck, *Introduction to Nonlinear Optimization*, SIAM, Philadelphia, 2014.
[2] D. Bertsekas, *Convex Optimization Theory*, Athena Scientific, Nashua, 2009.
[3] S. P. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, 2004.
[4] D. Calvetti, G. H. Golub, and L. Reichel, *Estimation of the L-curve via Lanczos bidiagonalization*, BIT Numer. Math., 39 (1999), pp. 603–619.
[5] J. Chung, J. G. Nagy, and D. P. O'Leary, *A weighted GCV method for Lanczos hybrid regularization*, Electron. Trans. Numer. Anal., 28 (2007/08), pp. 149–167. https://etna.ricam.oeaw.ac.at/vol.28.2007-2008/pp149-167.dir/pp149-167.pdf
[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *Imagenet: a large-scale hierarchical image database*, in 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE Conference Proceedings, Los Alamitos, 2009, pp. 248–255.
[7] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization.*, J. Mach. Learn. Res., 12 (2011), pp. 2121–2159.
[8] J. C. Dunn, *Newton's method and the Goldstein step-length rule for constrained minimization problems*, SIAM J. Control Optim., 18 (1980), pp. 659–674.
[9] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of Inverse Problems*, Kluwer, Dordrecht, 1996.
[10] B. Gao and L. Pavel, *On the properties of the softmax function with application in game theory and reinforcement learning*, Preprint on arXiv, 2017. https://arxiv.org/abs/1704.00805
[11] A. Ghodousian, A. N. Azad, and H. Amiri, *Log-sum-exp optimization problem subjected to Lukasiewicz fuzzy relational inequalities*, Preprint on arXiv, 2022. https://arxiv.org/abs/2206.09716
[12] P. E. Gill and W. Murray, *Newton-type methods for unconstrained and linearly constrained optimization*, Math. Programming, 7 (1974), pp. 311–350.
[13] G. H. Golub, M. Heath, and G. Wahba, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215–223.
[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, 2016.
[15] M. Grant, S. Boyd, and Y. Ye, *CVX: MATLAB software for disciplined convex programming*, 2008. https://cvxr.com/cvx
[16] J. Greenstadt, *On the relative efficiencies of gradient methods*, Math. Comp., 21 (1967), pp. 360–367.
[17] P. C. Hansen, *Rank-Deficient and Discrete Ill-Posed Problems*, SIAM, Philadelphia, 1998.
[18] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, *Extreme learning machine: theory and applications*, Neurocomputing, 70 (2006), pp. 489–501.
[19] K. Kan, J. G. Nagy, and L. Ruthotto, *Avoiding the double descent phenomenon of random feature models using hybrid regularization*, Preprint on arXiv, 2020. https://arxiv.org/abs/2012.06667
[20] K. Kan, S. W. Fung, and L. Ruthotto, *PNKH-B: A projected Newton-Krylov method for large-scale bound-constrained optimization*, SIAM J. Sci. Comput., 43 (2021), pp. S704–S726.
[21] D. Kim and J. A. Fessler, *Adaptive restart of the optimized gradient method for convex optimization*, J. Optim. Theory Appl., 178 (2018), pp. 240–263.
[22] S. Kim, R. Pasupathy, and S. G. Henderson, *A guide to sample average approximation*, in Handbook of Simulation Optimization, M. C. Fu, ed., Springer, New York, 2015, pp. 207–243.
[23] A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello, *The sample average approximation method for stochastic discrete optimization*, SIAM J. Optim., 12 (2002), pp. 479–502.
[24] W. Kong, W. Krichene, N. Mayoraz, S. Rendle, and L. Zhang, *Rankmax: an adaptive projection alternative to the softmax function*, in Advances in Neural Information Processing Systems, 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds., NIPS Proceedings, 2020, pp. 633–643.
[25] A. Krizhevsky, *Learning multiple layers of features from tiny images*, Tech. Report., University of Toronto, Toronto, 2009.
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, Commun. ACM, 60 (2017), pp. 84–90.
[27] Y. LeCun, *The MNIST database of handwritten digits*, Web Resource, 1998. https://yann.lecun.com/exdb/mnist/
[28] J. D. Lee, Y. Sun, and M. A. Saunders, *Proximal Newton-type methods for minimizing composite functions*, SIAM J. Optim., 24 (2014), pp. 1420–1443.
[29] K. Levenberg, *A method for the solution of certain non-linear problems in least squares*, Quart. Appl. Math., 2 (1944), pp. 164–168.
[30] D. W. Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, J. Soc. Indust. Appl. Math., 11 (1963), pp. 431–441.
[31] G. Meurant and Z. Strakoš, *The Lanczos and conjugate gradient algorithms in finite precision arithmetic*, Acta Numer., 15 (2006), pp. 471–542.
[32] J. J. Moré and D. C. Sorensen, *On the use of directions of negative curvature in a modified Newton*

*method*, Math. Programming, 16 (1979), pp. 1–20.
[33] MOSEK ApS, *MOSEK optimization toolbox for MATLAB*, User's Guide and Reference Manual, Version, 4, 2019. https://docs.mosek.com/
[34] ———, *MOSEK modeling cookbook*, 2020. https://docs.mosek.com/
[35] S. G. Nash, *Newton-type minimization via the Lanczos method*, SIAM J. Numer. Anal., 21 (1984), pp. 770–788.
[36] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, *Robust stochastic approximation approach to stochastic programming*, SIAM J. Optim., 19 (2009), pp. 1574–1609.
[37] E. Newman, L. Ruthotto, J. Hart, and B. van Bloemen Waanders, *Train like a (Var)Pro: efficient training of neural networks with variable projection*, SIAM J. Math. Data Sci., 3 (2021), pp. 1041–1066.
[38] F. Nielsen and K. Sun, *Guaranteed bounds on the Kullback–Leibler divergence of univariate mixtures*, IEEE Signal Process. Letters, 23 (2016), pp. 1543–1546.
[39] J. Nocedal and S. Wright, *Numerical Optimization*, Springer, New York, 2006.
[40] L. Nurbekyan, W. Lei, and Y. Yang, *Efficient natural gradient descent methods for large-scale optimization problems*, Preprint on arXiv, 2022. https://arxiv.org/abs/2202.06236
[41] B. O'Donoghue and E. Candès, *Adaptive restart for accelerated gradient schemes*, Found. Comput. Math., 15 (2015), pp. 715–732.
[42] N. Parikh and S. Boyd, *Proximal algorithms*, Found. Trends Optimization, 1 (2014), pp. 127–239.
[43] R. Pascanu and Y. Bengio, *Revisiting natural gradient for deep networks*, Preprint on arXiv, 2013. https://arxiv.org/abs/1301.3584
[44] A. Rahimi and B. Recht, *Random features for large-scale kernel machines*, in Advances in Neural Information Processing Systems, 20, J. Platt, D. Koller, Y. Singer, and S. Roweis, eds., NIPS Proceedings, 2007.
[45] J. F. Sturm, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optim. Methods Softw., 11/12 (1999), pp. 625–653.
[46] C.-L. Tseng, Y. Zhan, Q. P. Zheng, and M. Kumar, *A MILP formulation for generalized geometric programming using piecewise-linear approximations*, European J. Oper. Res., 245 (2015), pp. 360–370.
[47] R. H. Tütüncü, K.-C. Toh, and M. J. Todd, *Solving semidefinite-quadratic-linear programs using SDPT3*, Math. Programming, 95, Ser. B (2003), pp. 189–217.
[48] A. Vidal, S. Wu Fung, L. Tenorio, S. Osher, and L. Nurbekyan, *Taming hyperparameter tuning in continuous normalizing flows using the JKO scheme*, Preprint on arXiv, 2022. https://arxiv.org/abs/2211.16757
[49] C. R. Vogel, *Computational Methods for Inverse Problems*, SIAM, Philadelphia, 2002.
[50] S. Wu Fung, S. Tyrväinen, L. Ruthotto, and E. Haber, *ADMM-softmax: An ADMM approach for multinomial logistic regression*, Electron. Trans. Numer. Anal., 52 (2020), pp. 214–229. https://etna.ricam.oeaw.ac.at/vol.52.2020/pp214-229.dir/pp214-229.pdf
[51] X. Xi, J. Xu, and Y. Lou, *Log-sum-exp optimization based on continuous piecewise linearization techniques*, in 2020 IEEE 16th International Conference on Control & Automation (ICCA), IEEE Conference Proceeings, Los Alamitos, 2020, pp. 600–605.
[52] Y. Zhan, Q. P. Zheng, C.-L. Tseng, and E. L. Pasiliao, *An accelerated extended cutting plane approach with piecewise linear approximations for signomial geometric programming*, J. Global Optim., 70 (2018), pp. 579–599.