Low-complexity Proximal Gauss-Newton Algorithm for Nonnegative Matrix Factorization

Kejun Huang
Department of CISE
University of Florida
kejun.huang@ufl.edu

Xiao Fu
School of EECS
Oregon State University
xiao.fu@oregonstate.edu

***Abstract*—In this paper we propose a quasi-Newton algorithm for the celebrated nonnegative matrix factorization (NMF) problem. The proposed algorithm falls into the general framework of Gauss-Newton and Levenberg-Marquardt methods. However, these methods were not able to handle constraints, which is present in NMF. One of the key contributions in this paper is to apply alternating direction method of multipliers (ADMM) to obtain the iterative update from this Gauss-Newton-like algorithm. Furthermore, we carefully study the structure of the Jacobian Gramian matrix given by the Gauss-Newton updates, and designed a way of exactly inverting the matrix with complexity $\mathcal{O}(mnk)$, which is a significant reduction compared to the naive implementation of complexity $\mathcal{O}((m+n)^3k^3)$. The resulting algorithm, which we call NLS-ADMM, enjoys fast convergence rate brought by the quasi-Newton algorithmic framework, while maintaining low per-iteration complexity similar to that of alternating algorithms. Numerical experiments on synthetic data confirms the efficiency of our proposed algorithm.**

***Index Terms*—nonnegative matrix factorization, Gauss-Newton, proximal algorithm, alternating direction method of multipliers, low-complexity**

I. Introduction

Nonnegative matrix factorization, or NMF for short, is a powerful data analysis tool for signal processing and machine learning [1]. First popularized by Lee and Seung [2] by showing its capability of "learning the parts of objects" in computer vision, NMF has found numerous applications by providing meaningful and interpretable results, including blind source separation [3], [4], hyperspectral unmixing [5], [6], topic modeling [7], [8], [9], and community detection [10], [11], to name just a few. In terms of model identifiability, NMF has been shown to be essentially unique under the "sufficiently scattered" condition [12], [13], which explained the wide applicability of NMF.

Computationally, NMF is shown to be NP-hard [14]. Most existing NMF algorithms start by formulating an optimization problem using the least-squares loss between the data matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ and the product of the two factor matrices $\boldsymbol{W} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{H} \in \mathbb{R}^{n\times k}$ with a target rank k as follows

$$\underset{\boldsymbol{W}\geq 0,\boldsymbol{H}\geq 0}{\text{minimize}} \quad \left\|\boldsymbol{X}-\boldsymbol{W}\boldsymbol{H}^\top\right\|_F^2. \tag{1}$$

Since (1) is non-convex, there is no algorithm that guarantees to optimally solve it in polynomial-time. The prevailing method is to update the two factors $\boldsymbol{W}$ and $\boldsymbol{H}$ in an alternating fashion. Fixing one of the factors, problem (1) with respect to the other factor becomes a nonnegative least-squares problem, which is convex, but does not admit a closed-form solution. Some well-known examples include:

- **Multiplicative update (MU).** Lee and Seung [15] proposed to iteratively minimize an auxilirary function that majorizes the nonnegative least-squares loss, which results in a low-complexity closed-form update;
- **Hierarchical alternating least squares (HALS)** updates the factors column by column [16];
- **Alternating projected gradient (APG)** goes one projected gradient step for each of the factor updates [17]; an accelerated version was proposed by taking extrapolations [18];
- **Alternating nonnegative least squares with active set method (ANLS-AS) or block principal pivoting (ANLS-BPP)**. To solve the nonnegative least-squares sub-problems exactly, classical methods like the active-set algorithm and block principal pivoting were applied in an alternating fashion [19], [20];
- **Alternating optimization with alternating direction method of multipliers (AO-ADMM)**. It turns out that nonnegative least-squares can be solved with essentially the same computational complexity as its unconstrained version using ADMM, even though it does not admit a closed-form solution; the resulting AO-ADMM algorithm is not only efficient but also flexible in incorporating other kinds of constraints/regularizations seamlessly [21].

For the plain NMF problem of minimizing (1), the literature shows that ANLS-BPP [20], APG with extrapolation [18], and AO-ADMM [21] give the best emprical performance.

A few attempts have been made on designing "all-at-once" algorithms to update the two factors simultaneously. The simplest one is perhaps the projected gradient algorithm for the entire problem, which was investigated in the original APG paper [17] and reported to work inferior than the alternating version. ADMM was also applied to the entire problem rather than the convex sub-problems [22], [23]; however, ADMM for non-convex problems is not even guaranteed to monotonically decrease the loss, and in practice do perform less stably than alternating updates.

In this paper, we propose to explore a Gauss-Newton-like algorithm for NMF, which also falls into the "all-at-once" update-rule category. To the best of our knowledge, this classical quasi-Newton algorithm has not been applied to NMF due to two reasons: on the one hand, there are in total $(m+n)k$ variables, and a second-order algorithm without exploiting problem structures would result in a per-iteration complexity of $\mathcal{O}((m+n)^3k^3)$, which is prohibitive for most practical problem sizes; on the other hand, the original Gauss-Newton algorithm was designed for *unconstrained* nonlinear least-squares, with the premise that unconstrained linear least-squares problems are easy to solve, whereas in NMF there are nonnnegativity constraints.

Contributions. We propose a Gauss-Newton-like algorithm called NLS-ADMM for NMF, which resolves the two aforementioned challenges that hinder the use of such algorithms for NMF. We start by writing out explicitly the highly structured Hessian-approximation

X. Fu is supported in part by the National Science Foundation under Project NSF ECCS-1608961, ECCS 1808159, III-1910118, and the Army Research Office (ARO) under Proejct ARO W911NF-19-1-0247.

given by the Gauss-Newton method, and show that it can be inverted with complexity $\mathcal{O}(mnk)$ by exploiting the structure. Notice that the complexity is significantly lower than directly inverting with complexity $\mathcal{O}((m+n)^3k^3)$, and is in fact in the same order as a single factor update. There is also no additional memory overhead in storing the Hessian-approximation, thanks again to the nice structures. The nonnegativity constraints are handled by the use of ADMM, so that all the problem structures exploited in solving the unconstrained problem can be seamlessly applied here, with the flexibility of incorporating not only nonnegativity but also a variety of constraints/regularizations with efficient projection/proximity operators.

Related works. The closest work is the use of Gauss-Newton and its proximal version, Levenberg-Marquardt, to the canonical polyadic decomposition (CPD) for tensors [24], [25]. It has been shown that the Hessian approximation exhibits similar structures to that of the matrix case [26]. However, because of the subtle differences, people have not been able to bring down the per-iteration complexity down to that of a single factor update like this paper. Furthermore, their ways of handling nonnegativity constraints are either squaring the variables or addition log-barriers to the objective function. However, both approaches destroy the structure of the Hessian approximation for efficient inversion, unless an indirect method like the conjugate gradient method is used [24]. In addition, these approaches are not amendable for other kinds of constraints/regularizations.

II. Proposed Algorithm

A. General framework: NLS-ADMM

Before we introduce the general algorithmic framework for our proposed algorithm, we briefly review the general idea of Gauss-Newton and Levenberg-Marquardt algorithm. Details can be found in many textbooks, e.g. [27].

Consider the (unconstrained) nonlinear least-squares problem

$$\underset{\boldsymbol{z}}{\text{minimize}} \quad \|\boldsymbol{f}(\boldsymbol{z})\|^2, \tag{2}$$

where $\boldsymbol{f}(\boldsymbol{z})$ is a vector of nonlinear functions with respect to the variable $\boldsymbol{z}$.

The Gauss-Newton algorithm is an iterative heuristic for solving (2). At iteration t when the update is at $\boldsymbol{z}_t$, we first take a linear approximation of $\boldsymbol{f}(\boldsymbol{z})$ at $\boldsymbol{z}_t$ as

$$\boldsymbol{f}(\boldsymbol{z}) \approx \boldsymbol{f}(\boldsymbol{z}_t) + \boldsymbol{J}_f(\boldsymbol{z}_t)(\boldsymbol{z} - \boldsymbol{z}_t),$$

where $\boldsymbol{J}_f(\boldsymbol{z}_t)$ is the Jacobian matrix of $\boldsymbol{f}$ at $\boldsymbol{z}_t$, which we will simplify its notation as $\boldsymbol{J}_t$ in the sequel, with its (p,q)-th entry defined as

$$[\boldsymbol{J}_t]_{p,q} = \left[\boldsymbol{J}_f(\boldsymbol{z}_t)\right]_{p,q} = \frac{\partial f_p}{\partial z_q}(\boldsymbol{z}_t).$$

The update rule of Gauss-Newton is therefore

$$\boldsymbol{z}_{t+1} = \arg\min_{\boldsymbol{z}} \|\boldsymbol{f}(\boldsymbol{z}_t) + \boldsymbol{J}_t(\boldsymbol{z} - \boldsymbol{z}_t)\|^2. \tag{3}$$

If $\boldsymbol{J}_t$ has full colum-rank, then the update (3) has a closed form $\boldsymbol{x}_{t+1} = \boldsymbol{z}_t - (\boldsymbol{J}_t^\top \boldsymbol{J}_t)^{-1}\boldsymbol{J}_t^\top \boldsymbol{f}(\boldsymbol{z}_t)$. In the vicinity of the solution, Gauss-Newton shows super-linear convergence rate.

There are some short-comings of Gauss-Newton. It is not a descent algorithm, so if the initialization is not close to the optimal solution, it may take some iterations to even start decreasing the loss function. Furthermore, if $\boldsymbol{J}_t$ does not have full column-rank, which indeed often happens in practice, the update rull (3) is not even well-defined. A remedy is to add a proximal term to (3), leading to the Levenberg-Marquardt algorithm

$$\boldsymbol{z}_{t+1} = \arg\min_{\boldsymbol{z}} \|\boldsymbol{f}(\boldsymbol{z}_t) + \boldsymbol{J}_t(\boldsymbol{z} - \boldsymbol{z}_t)\|^2 + \lambda_t\|\boldsymbol{z} - \boldsymbol{z}_t\|^2. \tag{4}$$

With the help of the proximal term, the update rule is always well defined $\boldsymbol{x}_{t+1} = \boldsymbol{z}_t - (\boldsymbol{J}_t^\top \boldsymbol{J}_t + \lambda_t \boldsymbol{I})^{-1}\boldsymbol{J}_t^\top \boldsymbol{f}(\boldsymbol{z}_t)$. Furthermore, one can adaptively change the value of λ_t to guarantee monotonic decrease of the loss function. One effective approach, which we adopt in this paper, is as follows: if the new update does not decrease the loss, then double λ_t and re-calculate the update; otherwise, accept the update and let $\lambda_{t+1} = \lambda_t/2$.

To incorporate constraints to the variable $\boldsymbol{z}$, say $\boldsymbol{z} \in \mathcal{Z}$, we propose to keep the general form of Levenberg-Marquardt update as

$$\boldsymbol{z}_{t+1} = \arg\min_{\boldsymbol{z}\in\mathcal{Z}} \|\boldsymbol{f}(\boldsymbol{z}_t) + \boldsymbol{J}_t(\boldsymbol{z} - \boldsymbol{z}_t)\|^2 + \lambda_t\|\boldsymbol{z} - \boldsymbol{z}_t\|^2. \tag{5}$$

We know that the constrained least-squares problem does not admit a closed-form solution in general. However, it can be efficiently calculated via the following ADMM iterates:

$$\begin{aligned} \widetilde{\boldsymbol{z}} &\leftarrow \boldsymbol{z}_t - \left(\boldsymbol{J}_t^\top \boldsymbol{J}_t + (\rho + \lambda_t)\boldsymbol{I}\right)^{-1}\left(\boldsymbol{J}_t^\top \boldsymbol{f}(\boldsymbol{z}_t) + \rho(\boldsymbol{z}_t - \boldsymbol{z} + \boldsymbol{u})\right) \\ \boldsymbol{z} &\leftarrow \text{Proj}_{\mathcal{Z}}(\widetilde{\boldsymbol{z}} + \boldsymbol{u}) \\ \boldsymbol{u} &\leftarrow \boldsymbol{u} + \widetilde{\boldsymbol{z}} - \boldsymbol{z} \end{aligned} \tag{6}$$

where $\widetilde{\boldsymbol{z}}$ is an auxiliary variable subject to the additional constraint $\widetilde{\boldsymbol{z}} = \boldsymbol{z}$, $\boldsymbol{u}$ is the scaled version of its Lagrange multiplier, ρ is some positive scalar, and we assume that the projection onto the set $\mathcal{Z}$ is easy to evaluate. For convex problems, ADMM always converges to a global solution, although the convergence rate may depend on the choice of ρ. Details on ADMM can be found in [28]. Once we find an accurate enough update of $\boldsymbol{z}_{t+1}$ per (5) via ADMM, all the other steps remain the same as the classical Levenberg-Marquardt algorithm.

B. Efficient inversion of the Jacobian Gramian

In the context of NMF, we rearrange the variables and nonlinear functions as follows

$$\boldsymbol{z} = \begin{bmatrix}\text{vec}(\boldsymbol{W}) \\ \text{vec}(\boldsymbol{H})\end{bmatrix}, \qquad \boldsymbol{f}(\boldsymbol{z}) = \text{vec}(\boldsymbol{W}\boldsymbol{H}^\top - \boldsymbol{X}).$$

As we can see from the key computational steps in (6), the computationally heaviest step is to calculate the inverse of the Jacobian Gramian. Since $\boldsymbol{z} \in \mathbb{R}^{(m+n)k}$, a naive implementation would require $\mathcal{O}((m+n)^3k^3)$ flops to compute the inverse, which is prohibitive in most practical cases.

In the supplementary material of [26], the Cramer-Rao bound for matrix factorization was derived. It turns out that replacing the ground-truth $\boldsymbol{W}$ and $\boldsymbol{H}$ with the current update $\boldsymbol{W}_t$ and $\boldsymbol{H}_t$ is exactly the Jacobian Gramian, which is highly structured. To further simplify notations, we drop the subscript t when deriving the inverses, and we have [26]

$$\begin{aligned} \boldsymbol{J} &= \begin{bmatrix}\boldsymbol{H} \otimes \boldsymbol{I}_m & \boldsymbol{C}_{nm}(\boldsymbol{W} \otimes \boldsymbol{I}_n)\end{bmatrix}, \\ \boldsymbol{J}^\top\boldsymbol{J} &= \begin{bmatrix}\boldsymbol{H}^\top\boldsymbol{H} \otimes \boldsymbol{I}_m & (\boldsymbol{I}_k \otimes \boldsymbol{W})\boldsymbol{C}_{kk}(\boldsymbol{I}_k \otimes \boldsymbol{H})^\top \\ (\boldsymbol{I}_k \otimes \boldsymbol{H})\boldsymbol{C}_{kk}(\boldsymbol{I}_k \otimes \boldsymbol{W})^\top & \boldsymbol{W}^\top\boldsymbol{W} \otimes \boldsymbol{I}_n\end{bmatrix}, \end{aligned}$$

where $\otimes$ denotes the Kronecker product and $\boldsymbol{C}_{nm}$ is a special permutation matrix called the commutation matrix, which has the property that $\boldsymbol{C}_{nm}\,\text{vec}(\boldsymbol{S}) = \text{vec}(\boldsymbol{S}^\top)$ for all $\boldsymbol{S} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{C}_{pm}(\boldsymbol{S} \otimes \boldsymbol{T}) = (\boldsymbol{T} \otimes \boldsymbol{S})\boldsymbol{C}_{qn}$ for all $\boldsymbol{T} \in \mathbb{R}^{p\times q}$. We will also make use of the property of the Kronecker product that $(\boldsymbol{S}\otimes\boldsymbol{T})^{-1} = \boldsymbol{S}^{-1}\otimes\boldsymbol{T}^{-1}$.

It was shown in [26] that $\boldsymbol{J}^\top\boldsymbol{J}$ is rank-deficient, which necessitates the use of Levenberg-Marquardt by adding a proximal term. Denote

$\gamma = \rho + \lambda_t$, our task here is to efficiently invert the matrix $\boldsymbol{J}^\top\boldsymbol{J} + \gamma\boldsymbol{I}$. Notice that $\gamma\boldsymbol{I} = \gamma\boldsymbol{I} \otimes \boldsymbol{I}$ with appropriate sizes, $(\boldsymbol{J}^\top\boldsymbol{J} + \gamma\boldsymbol{I})^{-1}$ is equal to

$$\begin{bmatrix} (\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k) \otimes \boldsymbol{I}_m & (\boldsymbol{I}_k \otimes \boldsymbol{W})\boldsymbol{C}_{kk}(\boldsymbol{I}_k \otimes \boldsymbol{H})^\top \\ (\boldsymbol{I}_k \otimes \boldsymbol{H})\boldsymbol{C}_{kk}(\boldsymbol{I}_k \otimes \boldsymbol{W})^\top & (\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k) \otimes \boldsymbol{I}_n \end{bmatrix}^{-1}. \quad (7)$$

We now invoke the block-wise inversion formula

$$\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \\ \boldsymbol{A}_2^\top & \boldsymbol{A}_3 \end{bmatrix}^{-1} = \quad (8)$$
$$\begin{bmatrix} (\boldsymbol{A}_1 - \boldsymbol{A}_2\boldsymbol{A}_3^{-1}\boldsymbol{A}_2^\top)^{-1} & -\boldsymbol{A}_1^{-1}\boldsymbol{A}_2(\boldsymbol{A}_3 - \boldsymbol{A}_2^\top\boldsymbol{A}_1^{-1}\boldsymbol{A}_2)^{-1} \\ -\boldsymbol{A}_3^{-1}\boldsymbol{A}_2^\top(\boldsymbol{A}_1 - \boldsymbol{A}_2\boldsymbol{A}_3^{-1}\boldsymbol{A}_2^\top)^{-1} & (\boldsymbol{A}_3 - \boldsymbol{A}_2^\top\boldsymbol{A}_1^{-1}\boldsymbol{A}_2)^{-1} \end{bmatrix}.$$

The upper-left block of (7) equals to

$$\Big((\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k) \otimes \boldsymbol{I}_m - (\boldsymbol{I}_k \otimes \boldsymbol{W})\left(\boldsymbol{H}^\top\boldsymbol{H} \otimes (\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I})^{-1}\right)(\boldsymbol{I}_k \otimes \boldsymbol{W})^\top\Big)^{-1}.$$

Then we invoke the matrix inversion lemma to further simplify the upper-left block of (7) as

$$(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{I}_m + \left((\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{W}\right) \times \quad (9a)$$
$$\left((\boldsymbol{H}^\top\boldsymbol{H})^{-1} \otimes (\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}) - (\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I})^{-1} \otimes \boldsymbol{W}^\top\boldsymbol{W}\right)^{-1} \quad (9b)$$
$$\times \left((\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{W}\right)^\top. \quad (9c)$$

As we can see, this matrix is highly structured with the use of Kronecker product and a small $k \times k$ matrix inversion $(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1}$, except for the matrix inversion in (9b), which is a $k^2 \times k^2$ matrix.

It turns out that (9b) can stil be efficiently calculated with complexity $\mathcal{O}(k^3)$ by using the eigen-decompositions of $\boldsymbol{W}^\top\boldsymbol{W}$ and $\boldsymbol{H}^\top\boldsymbol{H}$. Denote their eigen-decompositions as

$$\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{Q}_W\boldsymbol{\Lambda}_W\boldsymbol{Q}_W^\top, \quad \boldsymbol{H}^\top\boldsymbol{H} = \boldsymbol{Q}_H\boldsymbol{\Lambda}_H\boldsymbol{Q}_H^\top, \quad (10)$$

then (9b) equals to

$$(\boldsymbol{Q}_H \otimes \boldsymbol{Q}_W)\left(\boldsymbol{\Lambda}_H^{-1} \otimes (\boldsymbol{\Lambda}_W + \gamma\boldsymbol{I}) - (\boldsymbol{\Lambda}_H + \gamma\boldsymbol{I})^{-1} \otimes \boldsymbol{\Lambda}_W\right)^{-1}(\boldsymbol{Q}_H \otimes \boldsymbol{Q}_W)^\top.$$

The matrix inverse in the middle is a $k^2 \times k^2$ diagonal matrix, so the overall complexity is dominated by the eigen-decompostion of two $k \times k$ matrices (10) with $\mathcal{O}(k^3)$ flops. In fact forming these matrices takes $\mathcal{O}(mk^2)$ and $\mathcal{O}(nk^2)$ flops respectively, so that the computations of eigen-decompositions are amortized.

Due to symmetry, the lower-right block of (7) can be similarly calculated by inter-changing $\boldsymbol{W}$ and $\boldsymbol{H}$ in (9). Finally, according the block-wise inversion formula, the off-diagonal blocks are simply some additional matrix multiplications, again with nice Kronecker structures and small-sized matrix inversions $(\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I})^{-1}$ and $(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I})^{-1}$.

The reader may notice that we never counted the computational complexity of calculating the Kronecker product of two matrices. This is because the Kronecker product is never explicitly calculated in our implementation—in the sequel we will see that by using another useful property of the Kronecker product $(\boldsymbol{B}^\top \otimes \boldsymbol{A})\,\mathrm{vec}(\boldsymbol{S}) = \mathrm{vec}(\boldsymbol{ASB})$, we manage to reduce the per-iteration complexity of this Gauss-Newton-like method down to $\mathcal{O}(mnk)$.

C. NLS-ADMM for NMF

Now we apply the previous result to the context of NLS-ADMM for NMF, with key computations shown in (5). Let us first calculate $\boldsymbol{J}_t^\top\boldsymbol{f}(z_t)$, which equals to

$$\begin{aligned}\boldsymbol{J}_t^\top\boldsymbol{f}(z_t) &= \begin{bmatrix} \boldsymbol{H}_t^\top \otimes \boldsymbol{I}_m \\ (\boldsymbol{W}_t^\top \otimes \boldsymbol{I}_n)\boldsymbol{C}_{mn} \end{bmatrix}\mathrm{vec}(\boldsymbol{W}_t\boldsymbol{H}_t^\top - \boldsymbol{X}) \\ &= \begin{bmatrix} \mathrm{vec}(\boldsymbol{W}_t\boldsymbol{H}_t^\top\boldsymbol{H}_t - \boldsymbol{X}\boldsymbol{H}_t) \\ \mathrm{vec}(\boldsymbol{H}_t\boldsymbol{W}_t^\top\boldsymbol{W}_t - \boldsymbol{X}^\top\boldsymbol{W}_t) \end{bmatrix}.\end{aligned}$$

Partition the dual variable $\boldsymbol{u}$ in (6) as $[\ \mathrm{vec}(\boldsymbol{U}_W)^\top\ \mathrm{vec}(\boldsymbol{U}_H)^\top\ ]^\top$ with $\boldsymbol{U}_W \in \mathbb{R}^{m\times k}$ and $\boldsymbol{U}_H \in \mathbb{R}^{n\times k}$. Define

$$\begin{aligned}\widehat{\boldsymbol{W}} &= \boldsymbol{W}_t\boldsymbol{H}_t^\top\boldsymbol{H}_t - \boldsymbol{X}\boldsymbol{H}_t + \rho(\boldsymbol{W}_t - \boldsymbol{W} + \boldsymbol{U}_W), \\ \widehat{\boldsymbol{H}} &= \boldsymbol{H}_t\boldsymbol{W}_t^\top\boldsymbol{W}_t - \boldsymbol{X}^\top\boldsymbol{W}_t + \rho(\boldsymbol{H}_t - \boldsymbol{H} + \boldsymbol{U}_H),\end{aligned}$$

then

$$\boldsymbol{J}_t^\top\boldsymbol{f}(z_t) + \rho(z_t - z + \boldsymbol{u}) = \begin{bmatrix} \mathrm{vec}(\widehat{\boldsymbol{W}}) \\ \mathrm{vec}(\widehat{\boldsymbol{H}}) \end{bmatrix}.$$

Partition $(\boldsymbol{J}_t^\top\boldsymbol{J}_t + \gamma\boldsymbol{I})^{-1}$ into four blocks with appropriate sizes, then

$$\left(\boldsymbol{J}_t^\top\boldsymbol{J}_t + \gamma\boldsymbol{I}\right)^{-1}\left(\boldsymbol{J}_t^\top\boldsymbol{f}(z_t) + \rho(z_t - z + \boldsymbol{u})\right) = \begin{bmatrix} \boldsymbol{G}_1 & \boldsymbol{G}_2 \\ \boldsymbol{G}_2^\top & \boldsymbol{G}_3 \end{bmatrix}\begin{bmatrix} \mathrm{vec}(\widehat{\boldsymbol{W}}) \\ \mathrm{vec}(\widehat{\boldsymbol{H}}) \end{bmatrix}.$$

We start by calculating $\boldsymbol{G}_1\,\mathrm{vec}(\widehat{\boldsymbol{W}})$, where $\boldsymbol{G}_1$ is equal to (9). First

$$\begin{aligned}\left((\boldsymbol{H}_t^\top\boldsymbol{H}_t + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{I}_m\right)\mathrm{vec}(\widehat{\boldsymbol{W}}) &= \mathrm{vec}\left(\widehat{\boldsymbol{W}}(\boldsymbol{H}_t^\top\boldsymbol{H}_t + \gamma\boldsymbol{I}_k)^{-1}\right), \\ \left((\boldsymbol{H}_t^\top\boldsymbol{H}_t + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{W}_t\right)^\top\mathrm{vec}(\widehat{\boldsymbol{W}}) &= \mathrm{vec}\left(\boldsymbol{W}_t^\top\widehat{\boldsymbol{W}}(\boldsymbol{H}_t^\top\boldsymbol{H}_t + \gamma\boldsymbol{I}_k)^{-1}\right).\end{aligned}$$

Now let $\boldsymbol{K}_W = \boldsymbol{W}_t^\top\widehat{\boldsymbol{W}}(\boldsymbol{H}_t^\top\boldsymbol{H}_t + \gamma\boldsymbol{I}_k)^{-1}$, and define a $k \times k$ matrix $\boldsymbol{S}$ with its (i, j)-th value equal to

$$\boldsymbol{S}(i, j) = \left(\frac{\lambda_W(i) + \gamma}{\lambda_H(j)} - \frac{\lambda_W(i)}{\lambda_H(j) + \gamma}\right)^{-1},$$

where $\lambda_W(i)$ denotes the i-th eigenvalue of $\boldsymbol{W}^\top\boldsymbol{W}$ and $\lambda_H(j)$ denotes the j-th eigenvalue of $\boldsymbol{H}^\top\boldsymbol{H}$. Then

$$\begin{aligned}&\left((\boldsymbol{H}^\top\boldsymbol{H})^{-1} \otimes (\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}) - (\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I})^{-1} \otimes \boldsymbol{W}^\top\boldsymbol{W}\right)^{-1}\mathrm{vec}(\boldsymbol{K}_W) \\ &= \mathrm{vec}\left(\boldsymbol{Q}_W\left(\boldsymbol{S} * \boldsymbol{Q}_W^\top\boldsymbol{K}_W\boldsymbol{Q}_H\right)\boldsymbol{Q}_H^\top\right),\end{aligned}$$

where $*$ denotes matrix Hadamard product, i.e., element-wise multiplication. This part is then multiplied by $(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{W}$, and we obtain $\boldsymbol{G}_1\,\mathrm{vec}(\widehat{\boldsymbol{W}})$. Reverse the role of $\boldsymbol{W}$ and $\boldsymbol{H}$, $\boldsymbol{G}_3\,\mathrm{vec}(\widehat{\boldsymbol{H}})$ can be similarly calculated.

Finally, let $\overline{\boldsymbol{H}} = \boldsymbol{G}_3\,\mathrm{vec}(\widehat{\boldsymbol{H}})$, we have

$$\begin{aligned}&\boldsymbol{G}_2^\top\,\mathrm{vec}(\widehat{\boldsymbol{W}}) \\ &= -\left((\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k)^{-1} \otimes \boldsymbol{I}_n\right)(\boldsymbol{I}_k \otimes \boldsymbol{W})\boldsymbol{C}_{kk}(\boldsymbol{I}_k \otimes \boldsymbol{H})^\top\mathrm{vec}(\overline{\boldsymbol{H}}) \\ &= -\mathrm{vec}\left(\boldsymbol{W}(\overline{\boldsymbol{H}}^\top\boldsymbol{H})(\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k)^{-1}\right).\end{aligned}$$

Similarly, one can calculate $\boldsymbol{G}_2\,\mathrm{vec}(\widehat{\boldsymbol{H}})$ by reversing the role of $\boldsymbol{W}$ and $\boldsymbol{H}$, and

$$\left(\boldsymbol{J}_t^\top\boldsymbol{J}_t + \gamma\boldsymbol{I}\right)^{-1}\begin{bmatrix} \mathrm{vec}(\widehat{\boldsymbol{W}}) \\ \mathrm{vec}(\widehat{\boldsymbol{H}}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{G}_1\,\mathrm{vec}(\widehat{\boldsymbol{W}}) + \boldsymbol{G}_2\,\mathrm{vec}(\widehat{\boldsymbol{H}}) \\ \boldsymbol{G}_3\,\mathrm{vec}(\widehat{\boldsymbol{H}}) + \boldsymbol{G}_2^\top\,\mathrm{vec}(\widehat{\boldsymbol{W}}) \end{bmatrix}.$$

The overall algorithm is summarized in Algorithm 1.

Algorithm 1 NLS-ADMM for NMF

initialize $\boldsymbol{W}$ and $\boldsymbol{H}$
$\rho = 1$
repeat
  $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{Q}_W\boldsymbol{\Lambda}_W\boldsymbol{Q}_W^\top$
  $\boldsymbol{H}^\top\boldsymbol{H} = \boldsymbol{Q}_H\boldsymbol{\Lambda}_H\boldsymbol{Q}_H^\top$ ▷ eigen-decomposition
  $\boldsymbol{F}_W = \boldsymbol{X}\boldsymbol{H}, \boldsymbol{F}_H = \boldsymbol{X}^\top\boldsymbol{W}$
  $\widetilde{\boldsymbol{Z}}_W = \boldsymbol{W}, \widetilde{\boldsymbol{Z}}_H = \boldsymbol{H}, \boldsymbol{U}_W = \boldsymbol{U}_H = 0$ ▷ initialize ADMM
  repeat
    $\widehat{\boldsymbol{W}} \leftarrow \boldsymbol{W}\boldsymbol{H}^\top\boldsymbol{H} - \boldsymbol{F}_W + \rho(\boldsymbol{W} - \boldsymbol{Z}_W + \boldsymbol{U}_W)$
    $\widehat{\boldsymbol{H}} \leftarrow \boldsymbol{H}\boldsymbol{W}^\top\boldsymbol{W} - \boldsymbol{F}_H + \rho(\boldsymbol{H} - \boldsymbol{Z}_H + \boldsymbol{U}_H)$
    $\boldsymbol{K}_W \leftarrow \boldsymbol{W}^\top\widehat{\boldsymbol{W}}(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1}$
    $\boldsymbol{K}_H \leftarrow \boldsymbol{H}^\top\widehat{\boldsymbol{H}}(\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k)^{-1}$
    $\boldsymbol{K}_W \leftarrow \boldsymbol{Q}_W\Big(\boldsymbol{S} * \boldsymbol{Q}_W^\top\boldsymbol{K}_W\boldsymbol{Q}_H\Big)\boldsymbol{Q}_H^\top$
    $\boldsymbol{K}_H \leftarrow \boldsymbol{Q}_H\Big((1/\boldsymbol{S}) * \boldsymbol{Q}_H^\top\boldsymbol{K}_H\boldsymbol{Q}_W\Big)\boldsymbol{Q}_W^\top$
    $\overline{\boldsymbol{W}} \leftarrow \widehat{\boldsymbol{W}}(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1} + \boldsymbol{W}\boldsymbol{K}_W(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1}$
    $\overline{\boldsymbol{H}} \leftarrow \widehat{\boldsymbol{H}}(\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k)^{-1} + \boldsymbol{H}\boldsymbol{K}_H(\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k)^{-1}$
    $\widetilde{\boldsymbol{Z}}_W \leftarrow \boldsymbol{W} - \overline{\boldsymbol{W}} + \boldsymbol{H}(\overline{\boldsymbol{W}}^\top\boldsymbol{W})(\boldsymbol{H}^\top\boldsymbol{H} + \gamma\boldsymbol{I}_k)^{-1}$
    $\widetilde{\boldsymbol{Z}}_H \leftarrow \boldsymbol{H} - \overline{\boldsymbol{H}} + \boldsymbol{W}(\overline{\boldsymbol{H}}^\top\boldsymbol{H})(\boldsymbol{W}^\top\boldsymbol{W} + \gamma\boldsymbol{I}_k)^{-1}$
    $\boldsymbol{Z}_W \leftarrow [\widetilde{\boldsymbol{Z}}_W + \boldsymbol{U}_W]_+$
    $\boldsymbol{Z}_H \leftarrow [\widetilde{\boldsymbol{Z}}_H + \boldsymbol{U}_H]_+$
    $\boldsymbol{U}_W \leftarrow \boldsymbol{U}_W + \widetilde{\boldsymbol{Z}}_W - \boldsymbol{Z}_W$
    $\boldsymbol{U}_H \leftarrow \boldsymbol{U}_H + \widetilde{\boldsymbol{Z}}_H - \boldsymbol{Z}_H$
  until convergence
  if $\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}^\top\|_F^2 < \|\boldsymbol{X} - \boldsymbol{Z}_W\boldsymbol{Z}_H^\top\|_F^2$ **then**
    $\rho \leftarrow 2\rho$ ▷ re-calculate the update
  else
    $\boldsymbol{W} \leftarrow \boldsymbol{Z}_W$
    $\boldsymbol{H} \leftarrow \boldsymbol{Z}_H$ ▷ accept the update
    $\rho \leftarrow \rho/2$
  end if
until convergence

III. Numerical Experiments

We conclude the paper by conducting some experiment on synthetic data to showcase the fast convergence rate of the proposed NLS-ADMM for NMF. All experiments are conducted in MATLAB 2018b on an iMac Pro with 10 Xeon processers and 32GB memory. We compare with two baseline algorithms, AO-ADMM [21] and APG with extrapolation [18], both being reported to work the best in a lot of cases.

We let $m = 100$, $n = 150$, $k = 10$, and randomly generate the ground-truth factors $\boldsymbol{W}$ and $\boldsymbol{H}$ from uniform distribution between $[0, 1]$. Then the data matrix $\boldsymbol{X} = \boldsymbol{W}\boldsymbol{H}^\top$ is constructed. Although there is no noise, it is known that finding an exact factorization is relatively hard because the factors are not identifiable (since they are all dense) [12].

A typical convergence plot is shown in Figure 1. Indeed, even though we have greatly reduced the per-iteration complexity of NLS-ADMM, every iteration still takes longer time than that of AO-ADMM and APG; but thanks to the exploitation of second-order information in the loss function, NLS-ADMM is able to escape the swamp effect, which often happens in alternating-type algorithms. The result is not surprising as it resembles the cases people have observed in CPD when alternating optimization is compared with quasi-Newton methods—alternating least-squares (ALS) sometimes gets stucked at a non-critical point for a very long time, whereas Gauss-Newton or Levenberg-Marquardt is able to quickly converge

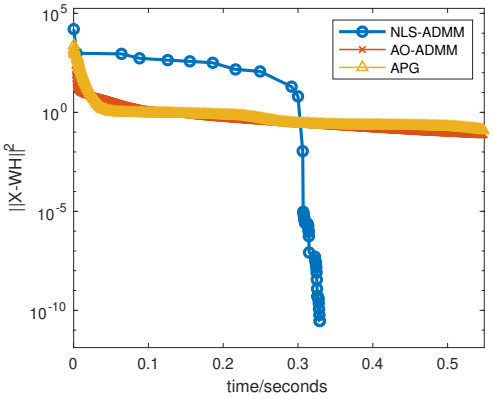

Fig. 1: One instance of convergence comparison on synthetic data.

TABLE I: Averaged performance of NMF algorithms

Algorithms	$\|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}^\top\|_F^2$	run time	iterations
NLS-ADMM	2.18×10^{-8}	1.1375	23.23
AO-ADMM	0.0108	7.4724	42019
APG	0.0099	10.9181	65035

to a global minimum with super-linear convegence rate [24]. It is interesting to see that by using ADMM to solve the constrained nonlinear least-squares problem, such good convegence property still exists.

The above experiment setting is randomly repeated 100 times and the averaged performances are shown in Table I. As we can see again, NLS-ADMM is able to obtain remarkably accurate solution with shortest amount of run time and very small number of iterations.

IV. Conclusion

This paper serves as the first attempt to apply the algorithmic framework of Gauss-Newton / Levenberg-Marquardt for the widely used nonnegative matrix factorization (NMF) problem. There are two notable contributions when applying this seemingly simple idea. The first resolves the challenge of how to properly handle the nonnegativity constraints on the latent factors, which is an important feature of NMF, but cannot be handled by the original Gauss-Newton framework. Our solution is to solve a constrained least-squares instead in each iteration, using alternating direction method of multipliers (ADMM). Another challenge is the high per-iteration complexity to naively solve a (constrained) least-squares problem due to the "all-at-once" update nature of Gauss-Newton. By carefully exploiting the structure of the Jacobian Grammian, we managed to bring down the per-iteration complexity to the same order of that of an alternating optimization algorithm such as AO-ADMM and APG. Numerical experiments on synthetic data shows the superior convergence rate compared to the state-of-the-arts.

For future work, we notice that it is still taking many ADMM iterations for each GN updates, especially at the beginning stage. An even faster algorithm to solve the constrained least-squares subproblem would greatly accelerate the proposed proximal Gauss-Newton algorithmic framework. It is also possible to extend the framework for constrained tensor factorization. However, to the best of our knowledge, there has not been a direct method of inverting the Jacobian Grammian matrix with complexity similar to an ALS step. A more in-depth study is required to make it work for tensors.

REFERENCES

[1] X. Fu, K. Huang, N. D. Sidiropoulos, and W.-K. Ma, "Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications," *IEEE Signal Processing Magazine*, vol. 36, pp. 59–80, 2019.

[2] D. D. Lee and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, no. 6755, p. 788, 1999.

[3] P. Smaragdis, C. Fevotte, G. J. Mysore, N. Mohammadiha, and M. Hoffman, "Static and dynamic source separation using nonnegative factorizations: A unified view," *IEEE Signal Processing Magazine*, vol. 31, no. 3, pp. 66–75, 2014.

[4] X. Fu, W.-K. Ma, K. Huang, and N. D. Sidiropoulos, "Blind separation of quasi-stationary sources: Exploiting convex geometry in covariance domain," *IEEE Transactions on Signal Processing*, vol. 63, no. 9, pp. 2306–2320, 2015.

[5] W.-K. Ma, J. M. Bioucas-Dias, T.-H. Chan, N. Gillis, P. Gader, A. J. Plaza, A. Ambikapathi, and C.-Y. Chi, "A signal processing perspective on hyperspectral unmixing: Insights from remote sensing," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 67–81, 2014.

[6] X. Fu, K. Huang, B. Yang, W.-K. Ma, and N. D. Sidiropoulos, "Robust volume minimization-based matrix factorization for remote sensing and document clustering," *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6254–6268, 2016.

[7] S. Arora, R. Ge, Y. Halpern, D. Mimno, A. Moitra, D. Sontag, Y. Wu, and M. Zhu, "A practical algorithm for topic modeling with provable guarantees," in *International Conference on Machine Learning*, 2013, pp. 280–288.

[8] K. Huang, X. Fu, and N. D. Sidiropoulos, "Anchor-free correlated topic modeling: Identifiability and algorithm," in *Advances in Neural Information Processing Systems*, 2016, pp. 1786–1794.

[9] K. Huang, X. Fu, and N. Sidiropoulos, "Learning hidden Markov models from pairwise co-occurrences with application to topic modeling," in *International Conference on Machine Learning*, 2018, pp. 2073–2082.

[10] J. Yang and J. Leskovec, "Overlapping community detection at scale: A nonnegative matrix factorization approach," in *ACM International Conference on Web Search and Data Mining*. ACM, 2013, pp. 587–596.

[11] K. Huang and X. Fu, "Detecting overlapping and correlated communities without pure nodes: Identifiability and algorithm," in *International Conference on Machine Learning*, 2019, pp. 2859–2868.

[12] K. Huang, N. D. Sidiropoulos, and A. Swami, "Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition," *IEEE Transactions on Signal Processing*, vol. 62, no. 1, pp. 211–224, 2013.

[13] X. Fu, K. Huang, and N. D. Sidiropoulos, "On identifiability of nonnegative matrix factorization," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 328–332, 2018.

[14] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1364–1377, 2009.

[15] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, 2001, pp. 556–562.

[16] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 3, pp. 708–721, 2009.

[17] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Computation*, vol. 19, no. 10, pp. 2756–2779, 2007.

[18] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on Imaging Sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[19] H. Kim and H. Park, "Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 2, pp. 713–730, 2008.

[20] J. Kim and H. Park, "Fast nonnegative matrix factorization: An active-set-like method and comparisons," *SIAM Journal on Scientific Computing*, vol. 33, no. 6, pp. 3261–3281, 2011.

[21] K. Huang, N. Sidiropoulos, and A. P. Liavas, "A flexible and efficient algorithmic framework for constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, vol. 64, no. 19, pp. 5052–5065, 2016.

[22] Y. Xu, W. Yin, Z. Wen, and Y. Zhang, "An alternating direction algorithm for matrix completion with nonnegative factors," *Frontiers of Mathematics in China*, vol. 7, no. 2, pp. 365–384, 2012.

[23] D. Hajinezhad, T.-H. Chang, X. Wang, Q. Shi, and M. Hong, "Nonnegative matrix factorization using ADMM: Algorithm and convergence analysis," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4742–4746.

[24] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 695–720, 2013.

[25] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Low complexity damped Gauss-Newton algorithms for CANDECOMP/PARAFAC," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 1, pp. 126–147, 2013.

[26] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582.

[27] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. Cambridge University Press, 2018.

[28] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.