Journal of Data Science 0 (0), 1–26
??? 2022
DOI: 10.6339/22-JDS1051
Data Science Reviews

Accelerating Fixed-Point Algorithms in Statistics and Data Science: A State-of-Art Review

Bohao Tang[1], Nicholas C. Henderson[2], and Ravi Varadhan[1,3,*]

[1]*Department of Biostatistics, Johns Hopkins University, Maryland, USA*
[2]*Department of Biostatistics, University of Michigan, Michigan, USA*
[3]*Quantitative Sciences Division, Sidney Kimmel Comprehensive Cancer Center, Johns Hopkins University, Maryland, USA*

Abstract

Fixed-point algorithms are popular in statistics and data science due to their simplicity, guaranteed convergence, and applicability to high-dimensional problems. Well-known examples include the expectation-maximization (EM) algorithm, majorization-minimization (MM), and gradient-based algorithms like gradient descent (GD) and proximal gradient descent. A characteristic weakness of these algorithms is their slow convergence. We discuss several state-of-art techniques for accelerating their convergence. We demonstrate and evaluate these techniques in terms of their efficiency and robustness in six distinct applications. Among the acceleration schemes, SQUAREM shows robust acceleration with a mean 18-fold speedup. DAAREM and restarted-Nesterov schemes also demonstrate consistently impressive accelerations. Thus, it is possible to accelerate the original fixed-point algorithm by using one of SQUAREM, DAAREM, or restarted-Nesterov acceleration schemes. We describe implementation details and software packages to facilitate the application of the acceleration schemes. We also discuss strategies for selecting a particular acceleration scheme for a given problem.

Keywords *convergence acceleration; EM; high dimensional models; MM; proximal gradient*

1 Introduction

Computational problems in science and mathematics are often solved using iterative algorithms, which produce a sequence of real-valued vectors converging to the solution of interest. Examples include solving systems of linear and nonlinear equations, numerical solutions of differential equations, approximation of integrals, and minimization of multivariate functions. Parameter estimation in many practical problems in statistics and data science can be ultimately reduced to a specific optimization problem often involving parameter constraints. To solve such optimization problems, various iterative algorithms have been developed including the expectation-maximization (EM) algorithm (Dempster et al., 1977), the majorization-minimization (MM) algorithm (Hunter and Lange, 2004), and gradient based methods like gradient descent (GD) and proximal gradient descent (Boyd et al., 2004). These methods are general and easy to use, and they can all be regarded as fixed-point iteration algorithms. A major appeal of these algorithms is their stability and their ability to readily handle high-dimensional problems which is a main reason for their surge in popularity for modern applications. However, a character-

*Corresponding author. Email: ravi.varadhan@jhu.edu.

Received March 7, 2022; Accepted May 25, 2022

istic weakness of these algorithms is their potential slow convergence, i.e., the vector sequence produced by the fixed-point iterative algorithm, $x_{n+1} = F(x_n)$, may converge very slowly (if it converges) to the solution x^*, severely limiting their effective use in solving real problems. Hence, it is desirable to have tools available that can accelerate the convergence of the sequence $\{x_n\}$. Please refer to Supplementary Material for a general, theoretical discussion of the rate of convergence of fixed-point iterations.

As highlighted in Varadhan and Roland (2008), an acceleration scheme should possess certain key properties in order to be an effective and practical tool for high-dimensional optimization problems. It should accelerate the convergence of the original iterative algorithm (fast local convergence); it should converge to the solution from any reasonable starting value (robust global convergence), provided, of course, that the base algorithm itself is convergent; it should have minimal storage/memory requirements (applicability to high-dimensional problems); and it should require minimal problem-specific tuning (off-the-shelf usability). The minimal storage requirement eliminates Newton-type algorithms which make use of full second-order information. In this paper, we examine several recently developed acceleration schemes that satisfy these listed requirements. The acceleration schemes discussed include SQUAREM (Varadhan and Roland, 2008), Anderson acceleration and DAAREM (Henderson and Varadhan, 2019), Quasi-Newton (Zhou et al., 2011), Nesterov acceleration with restarts (O'Donoghue and Candes, 2015), and Parabolic-EM (Berlinet and Roland, 2009).

The paper is organized as follows. First, in the next section, we describe several well-known fixed-point iterations and discuss their theoretical convergence properties. In Section 3, we introduce and describe the accelerating methods to be studied. In Section 5, we test the performance of each of these acceleration methods in a range of practical problems. In Section 4, implementation details and available R packages are described, and in last Section, we discuss the results and give strategies for choosing an acceleration scheme for a given problem at hand.

2 Popular Iterative Algorithms and Their Convergence

2.1 MM Algorithm

The MM in the MM algorithm stands for "Majorization-Minimization" or "Minorization-Maximization", depending on whether the particular optimization problem is a minimization or maximization problem. The MM algorithm actually describes a family of algorithms that are implemented by creating a surrogate function that majorizes (minorizes) the objective function of interest and optimizing this surrogate function in each iteration. A key feature of MM algorithms is that the objective function will increase (decrease)in every iteration. See Hunter and Lange (2004) for a general description of MM algorithms.

A function $g(x|x_k)$ is called a minorized version of the objective function f at x_k if it satisfies the following two conditions

$$\begin{aligned} &\forall x: \ g(x|x_k) \leqslant f(x) \\ &g(x_k|x_k) = f(x_k). \end{aligned}$$

Similarly, $g(x|x_k)$ will be called a majorized version of f at x_k if $-g(x|x_k)$ is a minorized version of $-f$. An MM maximization algorithm updates the current iterate x_k by maximizing the minorizing function $g(x|x_k)$. If we define F to be the argmax operator for $g(x|x_k)$, then we can

express the MM iteration as

$$x_{k+1} = \underset{x}{\operatorname{argmax}}\, g(x|x_k) =: F(x_k). \tag{1}$$

The fixed-point iteration (1) generates a sequence which is monotone with respect to the objective function f; that is, we are guaranteed to have $f(x_0) \leqslant f(x_1) \leqslant f(x_2) \leqslant \ldots$. This is due to the fact that $f(x_{k+1}) \geqslant g(x_{k+1}|x_k) \geqslant g(x_k|x_k) = f(x_k)$, and hence, one will get a strict increase in the objective function whenever $g(x_{k+1}|x_k) \neq g(x_k|x_k)$.

If x^* denotes an optimal point of f, then for x_k close to x^*, we have the following local approximation

$$x_{k+1} - x^* \approx dF(x^*)(x_k - x^*),$$

where $dF(x^*)$ is the Jacobian of F at x^*. It can be shown that $dF(x^*)$ is given by

$$dF(x^*) = I - [d^2 g(x^*|x^*)]^{-1} d^2 f(x^*), \tag{2}$$

where $d^2 f(x^*)$ and $d^2 g(x^*|x^*)$ denote the Hessian matrices of $f(x)$ and $g(x|x)$ respectively (with the derivatives in $d^2 g(x^*|x^*)$ being taken with respect to the first argument of $g(x|x)$). Therefore, an MM algorithm has linear convergence with a rate related to the largest eigenvalue of the Jacobian in (2), and the value of this Jacobian depends on both the objective function and choice of surrogate function. Globally, if the objective function f is strictly convex or concave, an MM algorithm will converge to the unique optimal point, assuming it exists. Otherwise, the MM algorithm will converge to one of the stationary points.

2.1.1 The EM Algorithm as a Special Case of MM

EM algorithms are used to find the value of a parameter vector x which maximizes a log-likelihood function $\ell(x) = \log p(\mathbf{Y}|x)$ of interest, where $\mathbf{Y}$ denotes the observed data vector and $p(\cdot|x)$ is the probability distribution for the observed data that is parameterized by x. To develop an EM algorithm for maximizing $\ell(x)$, one introduces a vector of unobserved latent data $\mathbf{U}$ and a probability distribution $p(\mathbf{Y}, \mathbf{U}|x)$ for $(\mathbf{Y}, \mathbf{U})$ which is also parameterized by x. Because $\ell(x)$ can be decomposed as $\log p(\mathbf{Y}|x) = \log p(\mathbf{Y}, \mathbf{U}|x) - \log p(\mathbf{U}|\mathbf{Y}, x)$ and $\log p(\mathbf{Y}|x)$ does not depend on $\mathbf{U}$, if we take the expectation of $\log p(\mathbf{Y}|x)$ with respect to the conditional distribution $[\mathbf{U}|\mathbf{Y}, x_k]$ where x_k is the current iterate of the EM algorithm, we obtain

$$\begin{aligned} \log p(\mathbf{Y}|x) &= \mathbb{E}_{\mathbf{U}|\mathbf{Y},x_k}\{\log p(\mathbf{Y}, \mathbf{U}|x)\} - \mathbb{E}_{\mathbf{U}|\mathbf{Y},x_k}\{\log p(\mathbf{U}|\mathbf{Y}, x)\} \\ &= Q(x|x_k) + H(x|x_k). \end{aligned} \tag{3}$$

In (3), $Q(x|x_k)$ is often referred to as the "Q-function", and computing it is referred to as the "E-step" of the EM algorithm. The term $H(x|x_k)$ is the cross entropy of the conditional distribution $[\mathbf{U}|\mathbf{Y}, x]$ relative to the conditional distribution $[\mathbf{U}|\mathbf{Y}, x_k]$.

After completing the "E-step", x_{k+1} is found by maximizing the Q-function $Q(x|x_k)$ with respect to x, namely,

$$x_{k+1} = \arg\max_x Q(x|x_k) =: F(x_k).$$

Computing x_{k+1} by maximizing $Q(x|x_k)$ is usually referred to as the "M-step" of an EM algorithm.

To see why the EM algorithm is a special case of the MM algorithm, note first that it directly follows from Jensen's inequality that

$$\begin{aligned} H(x|x_k) - H(x_k|x_k) &= \mathbb{E}_{\mathbf{U}|\mathbf{Y},x_k}\Big[\log\{p(\mathbf{U}|\mathbf{Y},\theta)/p(\mathbf{U}|\mathbf{Y},x_k)\}\Big] \\ &\leqslant \log\Big[\mathbb{E}_{\mathbf{U}|\mathbf{Y},x_k}\big\{p(\mathbf{U}|\mathbf{Y},\theta)/p(\mathbf{U}|\mathbf{Y},x_k)\big\}\Big] \\ &= 0. \end{aligned}$$

and hence $Q(x|x_k)+H(x_k|x_k) \leqslant \log p(\mathbf{Y}|x)$ for any value of x. In other words, $Q(x|x_k)+H(x_k|x_k)$ is a minorized version of the log-likelihood $\log p(\mathbf{Y}|x)$. Since $H(x_k|x_k)$ is a positive constant that does not depend on x, maximizing $Q(x|x_k) + H(x_k|x_k)$ is equivalent to maximizing the Q-function. Hence, we can regard the EM algorithm as an MM algorithm with minorization function $Q(x|x_k) + H(x_k|x_k)$.

2.2 Gradient Based Algorithms

2.2.1 Gradient Descent

Consider the following optimization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{4}$$

for a smooth function f that has all first order derivatives with $\nabla f(\mathbf{x}) = (\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_p})$ denoting the gradient of f at $\mathbf{x}$. Gradient descent is an iterative algorithm that always updates the current iterate $\mathbf{x}_k$ linearly in the direction where f decrease the fastest, namely, the negative gradient $-\nabla f(\mathbf{x}_k)$. In particular, for a given choice of step size or learning rate t_k, the gradient descent update of $\mathbf{x}_k$ is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - t_k \nabla f(\mathbf{x}_k). \tag{5}$$

Gradient descent may also be interpreted in the following way. At each step, we do not directly minimize the original function f, but instead, we minimize its first order approximation $f_k(\mathbf{x})$ around $\mathbf{x}_k$ which is given by

$$f_k(\mathbf{x}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2t_k}||\mathbf{x} - \mathbf{x}_k||^2, \tag{6}$$

where $||\cdot||$ is the Euclidean norm. One can directly check that the minimizer of the function f_k is equal to $\mathbf{x}_{k+1}$ in (5).

2.2.2 Proximal Gradient Descent

Optimization problem (4) is not general enough to handle optimization problems that have non-smooth terms. In such cases, one might consider the following generalization of (4)

$$\min_{\mathbf{x}} f(\mathbf{x}) + h(\mathbf{x}), \tag{7}$$

where, again, f is assumed to be smooth up to first order but h is instead a non-smooth function. As an example of (7), the objective function used in LASSO regression (Tibshirani, 1996) can be expressed as a sum of a smooth function and the non-smooth L_1 norm term.

Using the same reasoning used to obtain approximation (6), we can approximate the target $f(\mathbf{x}) + h(\mathbf{x})$ at each step by

$$\begin{aligned}(f+h)_k(\mathbf{x}) &= f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}-\mathbf{x}_k) + \frac{1}{2t_k}||\mathbf{x}-\mathbf{x}_k||^2 + h(\mathbf{x}) \\ &= \frac{1}{2t_k}||\mathbf{x}-\mathbf{x}_k + t_k\nabla f(\mathbf{x}_k)||^2 + h(\mathbf{x}) + Const, \end{aligned} \tag{8}$$

where $Const$ is a constant term that does not depend on $\mathbf{x}$. In step k, the proximal gradient descent update $\mathbf{x}_{k+1}$ is defined as the minimizer of the approximation $(f+h)_k(\mathbf{x})$ shown in (8). The minimizer $\mathbf{x}_{k+1}$ of (8) is typically expressed in terms of the proximal operator $\text{prox}_h(\cdot)$ of a function h which is defined as

$$\text{prox}_h(\mathbf{x}) = \arg\min_{\mathbf{z}} \left\{ \frac{1}{2}||\mathbf{z}-\mathbf{x}||^2 + h(\mathbf{z}) \right\}.$$

It follows from (8) that the proximal gradient descent update can be expressed in terms of the proximal operator of the function $t_k h$ as

$$\mathbf{x}_{k+1} = \text{prox}_{t_k h}(\mathbf{x}_k - t_k\nabla f(\mathbf{x}_k)) \tag{9}$$

The proximal gradient descent algorithm is most useful when the proximal operator has a closed form or is, at least, very easy to compute. For example, consider the case of LASSO regression where the non-smooth component $h(\mathbf{x})$ of the objective function is equal to the L_1 norm multiplied by a tuning parameter λ, i.e., $h(\mathbf{x}) = \lambda||\mathbf{x}||_1$ as L_1 norm. The proximal mapping of h can be expressed as

$$\text{prox}_h(\mathbf{x}) = \arg\min_{\mathbf{z}} \left\{ \frac{1}{2}||\mathbf{z}-\mathbf{x}||^2 + \lambda||\mathbf{z}||_1 \right\} = S_\lambda(\mathbf{x}),$$

where the jth component of $S_\lambda(\mathbf{x})$ is given by

$$[S_\lambda(\mathbf{x})]_j = \begin{cases} x_j - \lambda & \text{if } x_j > \lambda \\ 0 & \text{if } -\lambda \leqslant x_j \leqslant \lambda \\ x_j + \lambda & \text{if } x_j < -\lambda \end{cases}, \quad j = 1, \ldots, p, \tag{10}$$

where x_j is the jth component of $\mathbf{x}$.

As shown in Boyd et al. (2004), both gradient descent and proximal gradient descent are globally convergent with the same convergence rate in convex problems. Assuming f and h are both convex and ∇f is Lipschitz continuous with Lipschitz constant $L_f > 0$ (i.e., $\forall x, y : |\nabla f(x) - \nabla f(y)| \leqslant L_f|x-y|$), then, for some constant C, the following inequality holds when a constant step size $t_k = t_f < 1/L_f$ is used

$$f(\mathbf{x}_k) + h(\mathbf{x}_k) - f(\mathbf{x}_\infty) - h(\mathbf{x}_\infty) \leqslant C\frac{||\mathbf{x}_0 - \mathbf{x}_\infty||^2}{kt_f}, \tag{11}$$

where $\mathbf{x}_\infty$ is the optimal point and $\mathbf{x}_0$ the initial point. Therefore, to obtain a precision level which is within at least ε of the optimal value of the objective function, we will need $O(1/\epsilon)$ proximal gradient or gradient descent iterations.

3 Acceleration Techniques

3.1 Anderson Acceleration and DAAREM

Anderson acceleration (AA), also known as Anderson mixing, was originally introduced by D.G. Anderson in 1965 to accelerate the rate of convergence of fixed-point iterations in the context of integral equations (Anderson, 1965), and this acceleration technique has turned out to be useful in a range of other applications. Recent examples include computing the nearest correlation matrix (Higham and Strabić, 2016), reinforcement learning (Geist and Scherrer, 2018), EM acceleration (Henderson and Varadhan, 2019), and electronic structure computations (Fang and Saad, 2009).

The Anderson acceleration algorithm with order m applied to solving the fixed-point problem $f(x) = x$ is shown in Algorithm 1.

Algorithm 1: Anderson acceleration and DAAREM. In the description of the algorithm, $x_{k+1} = f(x_k)$ is the base fixed-point iteration, and m is the order of the acceleration scheme.

Initialize $x_0 \in X$
Set $x_1 = f(x_0)$
for $k = 1, 2, 3, \ldots$ **do**
  Set $m_k = \min\{k, m\}$
  Find the $\{\alpha_j^{k+1}\}$ to solve the following the minimization problem:

$$\min_{\sum_{j=k-m_k}^{k} \alpha_j^{k+1} = 1} \left\| \sum_{j=k-m_k}^{k} \alpha_j^{k+1} (f(x_j) - x_j) \right\|^2 + \lambda_k \left\| \boldsymbol{\alpha}_{-k}^{k+1} \right\|^2 \tag{12}$$

  Update $x_{k+1} = (1 - \beta_k) \sum_{j=k-m_k}^{k} \alpha_j^{k+1} x_j + \beta_k \sum_{j=k-m_k}^{k} \alpha_j^{k+1} f(x_j)$
  if *meets restart criteria* **then**
    Restart m_k from 1
  end
end

In Algorithm 1, $\boldsymbol{\alpha}_{-k}^{k+1}$ denotes the vector of length m_k containing the values α_j^{k+1} for $j = k - m_k, \ldots, k - 1$, and β_k is the relaxation factor used in Walker and Ni (2011) and Evans et al. (2020). The non-negative scalar $\lambda_k \geqslant 0$ is an optional damping factor used in Henderson and Varadhan (2019), and if $\lambda_k = 0$ for all k, then the update x_{k+1} in Algorithm 1 reduces to the more typical formulation of Anderson acceleration shown, in, for example, Walker and Ni (2011). Often, the minimization problem (12) in Algorithm 1 is stated as an equivalent unconstrained minimization problem with respect to m_k unconstrained parameters rather than the $m_k + 1$ constrained parameters $\{\alpha_j^{k+1}\}$. This formulation is used, for example, in Higham and Strabić (2016), and in Henderson and Varadhan (2019), where the unconstrained version of the minimization problem allows a more direct comparison with so-called multisecant quasi-Newton methods (Fang and Saad, 2009).

The convergence of Anderson acceleration for a general, nonlinear fixed-point iteration has been shown in Toth and Kelley (2015). A recent work (Evans et al., 2020) proved that Anderson acceleration can improve the convergence rate in a scenario with linear convergence but is not

guaranteed to improve the convergence rate in cases of quadratic convergence.

As shown in Algorithm 1, Anderson acceleration can be modified to include *restarts*, where the order m_k is sometimes reset to 1 and all previous memory are dropped. Different restart schema have been proposed for Anderson acceleration. Henderson and Varadhan (2019) implemented a direct, periodic restart scheme where the algorithm restarts whenever the m_k reaches the value m. Zhang et al. (2020) proposed an adaptive restart where the algorithm only restarts only when the algorithm shows signs of stagnation. In many practical examples, using restarts markedly improves the performance of Anderson acceleration and can reduce the occurrence of algorithm stagnation.

In all of the numerical experiments shown in Section 5, we use the version of Anderson acceleration described in Henderson and Varadhan (2019) which they refer to as the damped Anderson acceleration with restarts and epsilon monotonicity (DAAREM) algorithm. The first component which distinguishes DAAREM from many other implementations of Anderson acceleration is the addition of the damping terms $\lambda_k \geqslant 0$. This L_2 regularization term generates an update which is a compromise between a pure fixed-point update and a pure Anderson acceleration update. Having large values of λ_k in early iterations and allowing λ_k to decrease in later iterations allows the procedure to bridge the robustness of the original fixed-point iteration with the fast local convergence of Anderson acceleration. Another key component of DAAREM is the use of systematic restarts rather than adaptive restarts, which as mentioned before, is implemented by restarting whenever the value of m_k reaches m. Finally, DAAREM includes some degree of monotonicity control where the fixed-point iteration update is used if the proposed Anderson acceleration increases the objective function (in a minimization problem) by more than a small, pre-specified amount.

3.2 SQUAREM

SQUAREM (Varadhan and Roland, 2008) is a technique originally designed to accelerate EM algorithms, but it has also been shown to be useful in accelerating a range of other fixed-point iteration problems. SQUAREM has been acknowledged as a useful, general-purpose acceleration scheme by Lange and others: (Zhou et al., 2011),

> Unfortunately, most acceleration techniques are ill-suited to complicated models involving large number of parameters. The squared iterative methods (SQUAREM), recently proposed by Varadhan and Roland, constitute a notable exception.

SQUAREM was motivated by an interesting and highly original modification of the Barzilai-Borwein type spectral gradient algorithm for optimization (Raydan and Svaiter, 2002). SQUAREM readily scales to high-dimensional settings and is very simple to implement. Hence it has been used in numerous applications to accelerate convergence of underlying iterative algorithm. Examples include: large-scale genome-wide enrichment analysis (Zhu and Stephens, 2018); analysis of human movement (Raket et al., 2016); non-negative matrix factorization across multiple applications (Hobolth et al., 2020); analysis of differential expression in RNAseq data (Jin et al., 2015)); inferring and visualizing cancer mutation signatures (Shiraishi et al., 2015); and signal processing techniques using MM algorithms (Song et al., 2016). Convergence of SQUAREM was proved in Varadhan and Roland (2004) under certain restrictive assumptions. In Varadhan and Roland (2008), the global convergence was shown for the monotonic version of SQUAREM using the notion of Lyapunov function, which we describe in more detail in the supplementary

material. To date, there is no proof that provides an insight on the improved convergence rate from using SQUAREM.

Algorithm 2 describes the SQUAREM acceleration technique for finding a solution of the fixed-point problem $f(x) = x$.

Algorithm 2: SQUAREM. In the description of the algorithm, $x_{k+1} = f(x_k)$ is the base fixed-point iteration.

1 Initialize $x_0 \in X$
2 **for** $k = 1, 2, 3, \ldots$ **do**
3 Set $y_k = f(x_{k-1})$ and $z_k = f(y_k)$
4 Set $r = y_k - x_{k-1}$ and $v = z_k - y_k - r$
5 Compute step length $\alpha = \alpha(r, v)$
6 Update $x_k = x_{k-1} + 2\alpha r + \alpha^2 v$
7 Stabilize $x_k = f(x_k)$
8 **end**

There are different versions of SQUAREM which only differ according to how the step length in step 5 of Algorithm 2 is computed. The three main choices of the step length are: SqS1 which chooses $\alpha(r, v) = \frac{\langle r,v\rangle}{\langle v,v\rangle}$, SqS2 which chooses $\alpha(r, v) = \frac{\langle r,r\rangle}{\langle r,v\rangle}$, and SqS3 which chooses $\alpha(r, v) = -\frac{||r||}{||v||}$.

One can also relate SQUAREM to an order 1 Anderson acceleration update when the previous iterate has the form $x_k = f(x_{k-1})$. To see why this is the case, note that when $\lambda_k = 0$ and $x_k = f(x_{k-1})$ the solution of the minimization problem (12) in Algorithm 1 yields the following update

$$x_{k+1} = x_{k-1} + (\alpha + \beta_k)r + \alpha\beta_k v,$$

where $r = f(x_{k-1}) - x_k$, $v = f(f(x_{k-1})) - 2f(x_{k-1}) + x_{k-1}$ and $\alpha = \frac{\langle r,v\rangle}{\langle v,v\rangle}$. Therefore, a single SqS1-SQUAREM update of an iterate x_{k-1} is equivalent to the following procedure: define $\tilde{x}_k = f(x_{k-1})$ and find x_k by applying an order-1 Anderson acceleration update with $\beta_k = \alpha$ and where x_{k-1} and $\tilde{x}_k = f(x_{k-1})$ are considered to be the previous two iterates. Notice that α in SQUAREM does not necessarily belong to $(0, 1]$ and typically it can be much larger than 1, indicating that SQUAREM can be viewed as an over-relaxed version of order-1 Anderson acceleration where β_k is not restricted to the interval $(0, 1]$.

3.3 Parabolic-EM

Parabolic EM (Berlinet and Roland, 2009) is another extrapolation scheme designed to accelerate the EM algorithm. At each step, parabolic EM finds new iterate by extrapolating along a Bézier curve $M(t)$ controlled by the most recent three iterations x_{k-2}, x_{k-1}, x_k. Specifically, $M(t)$ is given by

$$\begin{aligned} M(t) &= (1-t)^2 x_{k-2} + 2t(1-t)x_{k-1} + t^2 x_k \\ &= x_{k-2} + 2t(x_{k-1} - x_{k-2}) + t^2(x_k - 2x_{k-1} + x_{k-2}). \end{aligned}$$

A direct calculation shows that, when recent iterations are obtained from the base EM iterations (i.e., $x_{k-1} = f(x_{k-2})$ and $x_k = f(x_{k-1})$ where f denotes the fixed-point iteration), all three forms

of the SQUAREM update $x_{new} = x_{k-2} + 2\alpha(f(x_{k-2}) - x_{k-2}) + \alpha^2(f \circ f(x_{k-2}) - 2f(x_{k-2}) + x_{k-2})$ lie on the curve $M(t)$.

Parabolic EM applies a line search to find t by increasing t from 1 and stopping once the likelihood decreases. If no values of t in the line search are found to increase the likelihood, the algorithm will restart using the original fixed point iteration. Parabolic EM has two subtypes called *arithmetic search* and *geometric search* version which differ only in the way they perform the line search across values of t. Given a step size $h > 0$, arithmetic search evaluates the likelihood at $M(t)$ for $t = 1 + h, 1 + 2h, \ldots$ until the likelihood function decreases at which point the line search stops. Similarly, given both a step size $h > 0$ and exponent $a > 1$, geometric search evaluates the likelihood at $M(t)$ for $t = 1+a, 1+a^2h, \ldots$ and stops whenever the likelihood function decreases. Algorithm 3 describes both the arithmetic and geometric search versions of parabolic EM.

Algorithm 3: Parabolic EM. In the description of the algorithm, $x_{k+1} = f(x_k)$ is the base fixed-point iteration.

Initialize $x_0 \in X$, $x_1 = f(x_0)$, $x_2 = f(x_1)$
for $k = 3, 4, 5, \ldots$ **do**
  $L_2 = \text{Likelihood}(x_{k-1})$
  $i = 0$, $t = 1 + a^i h$ (geometric) ; $i = 1$, $t = 1 + ih$ (arithmetic)
  $x_{new} = (1-t)^2 x_{k-3} + 2t(1-t)x_{k-2} + t^2 x_{k-1}$
  $L_{new} = \text{Likelihood}(x_{new})$
  if $L_{new} < L_2$ **then**
    $x_{k-2} = x_{k-1}$; $x_{k-1} = f(x_{k-2})$; $x_k = f(x_{k-1})$
  end
  else
    while $L_{new} \geqslant L_2$ **do**
      $x_{old} = x_{new}$; $L_2 = L_{new}$
      $i = i + 1$; $t = 1 + a^i h$ (geometric), $t = 1 + ih$ (arithmetic)
      $x_{new} = (1-t)^2 x_{k-3} + 2t(1-t)x_{k-2} + t^2 x_{k-1}$
      $L_{new} = \text{Likelihood}(x_{new})$
    end
    $x_k = f(f(x_{old}))$
  end
end

Note that parabolic EM can also be applied to a general fixed-point iteration as long as the fixed-point iteration has an associated loss function to minimize. In that case, one could directly implement Algorithm 3 by replacing the likelihood evaluations in Algorithm 3 with evaluations of the negative of the loss function of interest.

3.4 Quasi-Newton

Zhou et al. (2011) proposed a Quasi-Newton method that can be applied to accelerating fixed-point iterations. Consider a map $f : X \subset \mathbb{R}^d \to \mathbb{R}^d$ from which we want to find its fixed point x such that $f(x) = x$. This is equivalent as finding the root of function $g(x) = x - f(x)$. If f is assumed to be differentiable with Jacobian df, then Newton's method for finding the root of

$g(x)$ yields the following iteration

$$x_{k+1} = x_k - [I - df(x_k)]^{-1} g(x_k). \tag{13}$$

The goal in a quasi-Newton approach is to use an approximation of $df(x_k)$ in iteration (13) rather than the true $df(x_k)$. The secant method is a well-known root-finding algorithm for a function with scalar inputs that can also be thought of as a quasi-Newton algorithm.

Zhou et al. (2011) proposed Quasi-Newton method is based on the linear approximation $f \circ f(x_k) - f(x_k) \approx M(f(x_k) - x_k)$, where x_∞ denotes the fixed point of the iteration and $M = df(x_\infty)$ denotes the Jacobian of f at x_∞. If one sets $v_k = f \circ f(x_k) - f(x_k)$ and $u_k = f(x_k) - x_k$, then the secant requirement for a proposed approximate Jacobian M_k at iteration k would be that $M_k u_k = v_k$. For an improved approximation M_k, Zhou et al. (2011) require further that the following q secant conditions $M_k u_{k-j} = v_{k-j}$, $j = 1, \dots q-1$ hold. The matrix M_k with the smallest Frobenius norm among all matrices satisfying these q secant conditions is given by $M_k = V_k(U_k^T U_k)^{-1} U_k^T$, where U_k is the matrix with the q columns $\{u_{k-q+1}, \dots, u_k\}$ and V_k is the matrix with columns $\{u_{k-q+1}, \dots, u_k\}$. Using this approximate Jacobian in iteration (13) leads to the order q Quasi-Newton scheme described in Algorithm 4.

Algorithm 4: Quasi-Newton acceleration. In the description of the algorithm, x_k is a vector of length p, q is the order of the acceleration scheme, and $x_{k+1} = f(x_k)$ is the base fixed-point iteration.

```
Initialize β_0 ∈ X. Create an empty p × q matrix U
for i = 1, 2, ..., q + 1 do
    β_1 = f(β_0)
    if i > 1 then
        Add new column β_1 − β_0 to the right of matrix U
    end
    β_0 = β_1
end
Set β_2 = f(β_1)
Create matrix V = U
Remove the first column of V and add column β_2 − β_1 to the right of V
Set x_0 = β_0
for k = 1, 2, ... do
    Compute QN-updates x_k = f(x_{k−1}) − V(U^T U − U^T V)^{−1} U^T (x_{k−1} − f(x_{k−1}))
    Remove the leftmost columns of matrices U and V
    Add column f(x_k) − x_k to the right of U
    Add column f(f(x_k)) − f(x_k) to the right of V
    Check for convergence
end
```

3.5 Restarted Nesterov

Nesterov accelerated gradient descent (Nesterov, 2013; Tseng, 2009) is a popular technique for accelerating first order optimization methods. Using the same notation as in Section 2.2.2, Algorithm 5 outlines Nesterov acceleration applied to the composite optimization problem (7).

Algorithm 5: Nesterov accelerated proximal gradient descent. In the description of the algorithm, the objective function to be minimized is $f(\mathbf{x}) + h(\mathbf{x})$, where $f(\mathbf{x})$ is assumed to be a smooth function.

Initialize $\mathbf{x}_0 = \mathbf{x}_{-1}$ and $\theta_{-1} = 1$, $k = 0$
Find the Lipschitz constant L_f of f and set $t = \frac{1}{L_f}$
while *not converged* **do**
  $\theta_k = \frac{\sqrt{\theta_{k-1}^4 + 4\theta_{k-1}^2} - \theta_{k-1}^2}{2}$; $\alpha_k = \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}$
  $\mathbf{y}_k = \mathbf{x}_k + \alpha_k(\mathbf{x}_k - \mathbf{x}_{k-1})$
  $\mathbf{x}_{k+1} = \text{prox}_{th}(\mathbf{y}_k - t\nabla f(\mathbf{y}_k))$
  $k = k + 1$
end

It can be proved that Algorithm 5 has an error rate of $O(\frac{1}{k^2})$, where k is the iteration number. This is a substantial improvement over the $O(\frac{1}{k})$ error rate shown in Equation (11) for the fixed step length proximal gradient descent algorithm. Readers can consult Tseng (2009) for a proof of this result.

Unlike most implementations of gradient descent, Algorithm 5 does not guarantee or check for monotonicity of the objective function. In practice, if you trace the objective value when running Algorithm 5, it is often the case that you see ripples or bumps in the objective function across iterations, which reduces the efficiency of the algorithm. To address this, O'Donoghue and Candes (2015) introduce a heuristic adaptive restart technique to Nesterov acceleration that can dramatically improve the convergence rate. The basic idea is to reset θ_k to 1 whenever you see an increase of objective function $f(\mathbf{x}_k) + h(\mathbf{x}_k) > f(\mathbf{x}_{k-1}) + h(\mathbf{x}_{k-1})$. Setting $\theta_k = 1$ reduces the momentum term $\alpha_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k)$ to 0 and the accleration algorithm will degenerate to ordinary proximal gradient descent in the following step.

Notice that in Algorithm 5, the momentum coefficients α_k do not depend on the proximal gradient descent updates in any way. Therefore, it is possible to just replace the step $\mathbf{x}_{k+1} = \text{prox}_t(\mathbf{y}_k - t\nabla f(\mathbf{y}_k))$ with any fixed-point iteration $\mathbf{x}_{k+1} = F(\mathbf{y}_k)$ and obtain a Nesterov-like acceleration method for a general fixed-point iteration problem. In Section 5, we show that this strategy can accelerate convergence in problems where gradient descent is not the base fixed-point iteration. To the best of our knowledge, this is the first work to examine Nesterov acceleration for general fixed-point iteration problems.

4 Implementations

R packages are available to facilitate application of the acceleration schemes described above. For example, the `squarem` package (Du and Varadhan, 2020) implements the SQUAREM algorithm, and the `daarem` package (Henderson and Varadhan, 2020) implements the DAAREM algorithm described in Section 3.1. The `turboEM` package (Bobb and Varadhan, 2021) provides a unified API for SQUAREM, Parabolic-EM, and Quasi-Newton acceleration. Currently, `turboEM` does not implement DAAREM, although this should be available in the near future. In `turboEM`, when the objective function value for a proposed update increases the objective value by more than 0.1, we replace this update with one iteration of the base fixed-point iteration. This can dramatically increase the stability of the Quasi-Newton method and, in many cases, can improve

the ultimate convergence speed. Monotonicity control is also a default in the implementations of DAAREM, SQUAREM and Parabolic-EM. The use of restarts in Nesterov acceleration also plays a similar role to monotonicity control as the algorithm is restarted whenever a monotonicity violation occurs. It is worth mentioning that the implementation of the Quasi-Newton in `turboEM` includes the option of monotonicity control even though monotonicity control was not originally implemented in (Zhou et al., 2011).

Our package `AccelBenchmark` is available from Github which can be used to easily benchmark all of the methods described in this paper. We actually used it in all the experiments described in this paper.

There is an R package called `FixedPoint` which contains various acceleration methods for fixed-point problems, including Anderson acceleration and several vector extrapolation algorithms. A fundamental difference between that package and our software packages is that `FixedPoint` doesn't contain safeguards such as steplength and monotonicity controls, damping, and restarts. Consequently, the algorithms are less reliable for general purpose use.

5 Experiments

5.1 Settings for the Experiments

In all of the experiments in this paper we have used the default control parameters implemented in each of the acceleration packages (`turboEM` and `daarem`). We did not optimize the algorithmic settings for each problem. This is an important point because we would like to explore the performance of these methods when they are used directly off-the-shelf. Also, unless otherwise stated, convergence is defined as the first iteration where the norm of the parameter difference $||x_{k+1} - x_k||$ is less than 10^{-7}. We evaluated the performance of each acceleration algorithm in terms of the number of fixed-point iterations (*fpevals*) and the elapsed time in seconds (*elapsed*). We report the mean $\pm$ standard deviation of *fpevals* and *elapsed* across a certain number of simulated experiments. Some of the performance metrics have distributions with a heavy tail making the standard deviation bigger than the mean, which is a sign of instability of that algorithm. We also report the number of failures (*# failures*), where failure is defined as not achieving convergence within an allotted number of iterations, which varied for each problem. We also plotted the convergence trajectories of algorithms in terms of the objective function (e.g., log-likelihood). The panels of Figure 1 in the Supplementary Material display the objective function values versus fixed-point iteration for each of the five main acceleration methods. The *Loss* item in the figures are normalized by subtracting the minimum value of the objective function. For every experiment, we only plot results for the most difficult setting (e.g., $\nu = 25$ in the multivariate t distribution).

5.2 Multivariate t-Distribution

A d-dimensional Student-t distribution $T_\nu(\mu, \Sigma)$ with $\nu > 0$ degrees of freedom, location parameter $\mu \in \mathbb{R}^d$, and positive definite scatter matrix Σ has the density function:

$$p(x|\nu, \mu, \Sigma) = \frac{\Gamma(\frac{d+\nu}{2})}{\Gamma(\frac{\nu}{2})\nu^{\frac{d}{2}}\pi^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \frac{1}{(1 + \frac{1}{\nu}(x-\mu)^T\Sigma^{-1}(x-\mu))^{\frac{d+\nu}{2}}},$$

where $\Gamma(s)$ denotes the Gamma function $\Gamma(s) = \int_0^\infty t^{s-1}e^{-t}dt$. For observations $x_1, \ldots, x_n$ arising from a d-dimensional Student-t distribution, setting the derivative of the associated log likelihood

Algorithm 6: EM for estimating multivariate t-distribution parameters.

Initialize ν_0, μ_0, Σ_0.
for $k = 1, 2, \ldots$ **do**
  E-Step:
    $z_{i,k} = (x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k)$
    $\gamma_{i,k} = \frac{\nu_k + d}{\nu_k + z_{i,k}}$
  M-Step:
    $\mu_{k+1} = \frac{\sum_i \omega_i \gamma_{i,k} x_i}{\sum_i \omega_i \gamma_{i,k}}$; $\quad \Sigma_{k+1} = \sum_i \omega_i \gamma_{i,k} (x_i - \mu_{k+1})(x_i - \mu_{k+1})^T$
    $\nu_{k+1} =$ **zero** of $\phi\left(\frac{\nu}{2}\right) - \phi\left(\frac{\nu_k + d}{2}\right) + \sum_i \omega_i (\gamma_{i,k} - \log(\gamma_{i,k}) - 1)$
end

function with user-specified weights $\omega_1, \ldots, \omega_n$ to zero results in the following system of equations

$$0 = \sum_{i=1}^{n} \omega_i \frac{x_i - \mu}{\nu + (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)}, \tag{14}$$

$$I = (d + \nu) \sum_{i=1}^{n} \omega_i \frac{\Sigma^{-\frac{1}{2}} (x_i - \mu)(x_i - \mu)^T \Sigma^{-\frac{1}{2}}}{\nu + (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)}, \tag{15}$$

$$\begin{aligned} 0 =& \phi\left(\frac{\nu}{2}\right) - \phi\left(\frac{\nu + d}{2}\right) \\ &+ \sum_{i=1}^{n} \omega_i \left(\frac{\nu + d}{\nu + (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)} - \log\left(\frac{\nu + d}{\nu + (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)} \right) - 1 \right) \end{aligned} \tag{16}$$

where $\phi(x) = \frac{1}{\Gamma(x)} \frac{d\Gamma(x)}{dx} - \log(x)$ and where the weights ω_i are assumed to satisfy $\omega_i \geqslant 0$, $\sum_{i=1}^{n} \omega_i = 1$.

Here, we are interested in maximizing the weighted log-likelihood function under the assumption that all parameters, including the degrees of freedom ν, are unknown. Hasannasab et al. (2021) show that, under certain conditions, either a minimizer of the negative weighted log-likelihood exists, or the maximum likelihood estimator corresponds to the case $\nu \to \infty$, for which the Student-t distribution approaches the Gaussian distribution.

Because one can represent a $T_\nu(\mu, \Sigma)$ random variable as

$$\mu + \Sigma^{\frac{1}{2}} Z / \sqrt{Y} \sim T_\nu(\mu, \Sigma), \tag{17}$$

where $Z \sim N(0, I)$ and $Y \sim \Gamma(\nu/2, \nu/2)$, one can develop an EM algorithm for parameter estimation by augmenting the observed data $x_1, \ldots, x_n$ with latent data $y_1, \ldots, y_n$ where it is assumed that $y_i \sim \Gamma(\nu/2, \nu/2)$ independently and use the fact that the conditional distribution $x_i | y_i$ is a multivariate normal distribution with mean vector μ and covariance matrix Σ / y_i. This data augmentation leads to the conventional EM algorithm (Liu and Rubin, 1995) for solving equations (14)–(16). The conventional EM algorithm for the multivariate t-distribution is described in Algorithm 6.

Recently, Hasannasab et al. (2021) suggested a number of alternative algorithms for such problem that accelerated the naive EM algorithm. Among them, the Multivariate Myriad Filter (MMF) algorithm shows excellent overall performance. The MMF algorithm is the same as the

EM algorithm (Algorithm 6) except that the updates for Σ_k and ν_k are changed to

$$\Sigma_{k+1} = \sum_{i=1}^{n} \frac{\omega_i \gamma_{i,k}(x_i - \mu_{k+1})(x_i - \mu_{k+1})^T}{\sum_{i=1}^{n} \omega_i \gamma_{i,k}}$$

$$\nu_{k+1} = \textbf{zero of } \phi\left(\frac{\nu}{2}\right) - \phi\left(\frac{\nu + d}{2}\right) + \sum_{i=1}^{n} \omega_i \left(\frac{\nu_k + d}{\nu_k + z_{i,k+1}} - \log\left(\frac{\nu_k + d}{\nu_k + z_{i,k+1}}\right) - 1\right).$$

We conducted a simulation study to determine whether or not our black-box acceleration schemes can further accelerate the fast MMF algorithm. In this simulation study, we set $\mu = (0, 0)$ and $\Sigma = \text{diag}\{0.1, 1\}$, and we considered the three choices for the degrees of freedom ν from $\{3, 10, 25\}$. For each choice of ν, we simulated 1000 observations from the corresponding t-distribution and ran the MMF algorithm together with all acceleration methods to estimate the parameters μ, Σ, ν, and this procedure was repeated across 200 simulation runs. In each run, we initialized μ_0 to be the value of the sample mean, Σ_0 be the sample covariance matrix, and we initialized the degrees of freedom ν_0 by sampling with equal probability from the set $\{2, 3, 4\}$. The hyperparameters of the acceleration algorithms and the convergence criteria are set to their defaults. These default settings are discussed in detail in Section 4.

Results from this simulation study can be found in Table 1. From this table, we observe that SQUAREM, DAAREM and parabolic-EM (pEM) provide consistent acceleration when compared to the original MMF algorithm, and the factor of speedup from these methods increases as ν increases. Quasi-Newton and Nesterov acceleration also accelerated the MMF algorithm in some simulation settings, but the improvement over MMF was not as consistent as SQUAREM, DAAREM, and pEM.

It is interesting to note that the MMF algorithm already gives a huge speed advantage when compared to the original EM algorithm. For example, in the $\nu = 25$ case, the EM algorithm required, on average, 1235 fixed point iterations before convergence which is 9 times more than that of the MMF algorithm. Despite the fact MMF is much faster than EM, we can still further accelerate MMF using the methods described in this manuscript.

5.3 Poisson Mixtures

A finite mixture of Poisson distributions with C components has the following discrete probability distribution

$$f(y \mid \mathbf{p}, \boldsymbol{\lambda}) = \sum_{c=1}^{C} p_c f_{\mathcal{P}}(y \mid \lambda_c), \tag{18}$$

where $f_{\mathcal{P}}(y \mid \lambda_c) = e^{-\lambda_c}\lambda_c^y / y!$ denotes the probability distribution of y conditional on belonging to the c^{th} cluster of the mixture distribution. For observations $y_1, \ldots, y_n$, we can develop an EM algorithm for estimating the parameters in (18) by introducing latent variables $z_1, \ldots, z_n$ defined as $z_i = c$ if y_i belongs to the c^{th} cluster. This particular data augmentation scheme generates the following EM algorithm updates for the parameters of interest $(p_1, \ldots, p_C)$ and $(\lambda_1, \ldots, \lambda_C)$:

$$\hat{\pi}_{ic}^{(k)} = p_c^{(k)} \left(\lambda_c^{(k)}\right)^{y_i} e^{-\lambda_c^{(k)}} \Big/ \sum_{l=1}^{C} p_l^{(k)} \left(\lambda_l^{(k)}\right)^{y_i} e^{-\lambda_l^{(k)}}$$

$$p_c^{(k+1)} = \frac{1}{n} \sum_{i=1}^{n} \hat{\pi}_{ic}^{(k)}; \qquad \lambda_c^{(k+1)} = \sum_{i=1}^{n} \hat{\pi}_{ic}^{(k)} y_i \Big/ \sum_{i=1}^{n} \hat{\pi}_{ic}^{(k)}.$$

Table 1: Simulation results for the multivariate t-distribution from 200 independent runs. MMF represents the original MMF algorithm described above, and other columns show results for different accelerated version of it. If an algorithm failed to converge or if it converged to a negative log-likelihood more than 1% larger than that of the original MMF algorithm, then we called it a failure. As a measure of robustness, we also recorded the number of failures for each method.

Metric	MMF	SQUAREM	DAAREM	pEM	Quasi-Newton	Nesterov
fpevals ($\nu = 3$)	34.3 ± 2.94	15.2 ± 1.31	$\mathbf{14.4 \pm 1.45}$	21.1 ± 2.4	17.8 ± 17.9	29.5 ± 2.5
elapsed ($\nu = 3$)	1.12 ± 0.097	$\mathbf{0.398 \pm 0.035}$	0.501 ± 0.056	0.693 ± 0.088	0.678 ± 0.718	0.929 ± 0.085
# failures ($\nu = 3$)	0	0	0	0	0	0
fpevals ($\nu = 10$)	67.7 ± 12.7	20.4 ± 3.67	$\mathbf{16.0 \pm 1.42}$	22.5 ± 1.65	51.4 ± 39.3	50.2 ± 14.2
elapsed ($\nu = 10$)	2.20 ± 0.419	$\mathbf{0.54 \pm 0.103}$	0.561 ± 0.056	0.761 ± 0.063	1.38 ± 1.08	1.58 ± 0.451
# failures ($\nu = 10$)	0	0	0	0	0	0
fpevals ($\nu = 25$)	128 ± 42.1	22.9 ± 4.61	$\mathbf{17.3 \pm 3.37}$	24.9 ± 1.88	30.7 ± 12.8	77.8 ± 150
elapsed ($\nu = 25$)	4.11 ± 1.36	$\mathbf{0.61 \pm 0.127}$	0.615 ± 0.14	0.859 ± 0.08	1.22 ± 0.52	2.46 ± 4.81
# failures ($\nu = 25$)	0	0	0	0	0	1

We explored our acceleration methods using the real count data from Hasselblad (1966) which contains 1096 observations with each observation representing a day of survival. In this dataset, the observations range from 0 to 9 and the frequencies for these values are 162, 267, 271, 185, 111, 61, 27, 8, 3, 1 respectively, no censoring is presented. Using this dataset, we fit a finite mixture of Poisson distributions with 2 components so the parameters of interest are the mixture probability p_1 and the cluster-specific Poisson rates λ_1 and λ_2.

To study the performance of each acceleration procedure, we ran the original EM algorithm and all acceleration schemes 500 times. In each run, the mixture probability p_1 was drawn from a uniform distribution over $(0, 1)$, and the Poisson rates λ_c were independently drawn from a uniform distribution over $(0, 4)$. In the Quasi-Newton algorithm, the order was set to 2 since we only have 3 parameters in this problem. The maximum number of fixed-point iteration evaluations is set to 3000. Other hyperparameters of the acceleration algorithm and convergence criteria are set to their defaults, which will be discussed in more detail in Section 4. Results can be found in Table 2. In this experiment, all of the methods listed in Table 2 dramatically accelerated the original EM algorithm with an up to 11-fold reduction in execution time.

5.4 LASSO

The least absolute shrinkage and selection operator (LASSO) (Tibshirani, 1996) is a widely used technique for high dimensional inference due to its ability to perform simultaneous feature selection and coefficient estimation. However, the additional L_1 penalty in the LASSO objective function also makes it impossible to obtain a general, closed-form solution for the regression coefficient estimates, and therefore, iterative algorithms are needed for optimization. A simple but effective iterative algorithm is proximal gradient descent which uses the iteration (9) described

in Section 2.2.2.

In this experiment, we use the Madelon data (Guyon et al., 2004) to study the performance of difference acceleration methods applied to the proximal gradient descent algorithm. The Madelon data is artificially constructed to illustrate a particular difficulty for feature selection. It contains $n = 2600$ binary outcomes $y_1, \ldots, y_n$, and for each y_i, we have a predictor vector $\mathbf{x}_i$ of length $p = 500$. We use the logistic regression version of LASSO where the objective function $\ell(\boldsymbol{\beta})$ to be minimized is given by

$$\ell(\boldsymbol{\beta}) = f(\boldsymbol{\beta}) + h(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(\log\left(1 + e^{\mathbf{x}_i^T \boldsymbol{\beta}}\right) - y_i \mathbf{x}_i^T \boldsymbol{\beta} \right) + \lambda \sum_{p=1}^{p} |\beta_p|,$$

where λ is a parameter that controls the regularization level and where $h(\boldsymbol{\beta}) = \lambda \sum_{p=1}^{p} |\beta_p|$. Following (9)–(10) yields the following proximal gradient descent iteration

$$\begin{aligned} \boldsymbol{\beta}_{k+1} &= \operatorname{prox}_{t_k h}\left(\boldsymbol{\beta}_k - t_k \nabla f(\boldsymbol{\beta}_k)\right) \\ &= S_{\lambda t_k}\left(\boldsymbol{\beta}_k - t_k \mathbf{X}^T \{\mathbf{y} - \mu(\mathbf{X}\boldsymbol{\beta}_k)\}\right), \end{aligned} \tag{19}$$

where $\mathbf{X}$ is the $n \times p$ matrix whose ith row is $\mathbf{x}_i^T$ and where $\mu(\mathbf{X}\boldsymbol{\beta}_k)$ is the n-dimensional vector whose ith element is $1/\{1 + \exp(\mathbf{x}_i^T \boldsymbol{\beta}_k)\}$.

In each simulation run, we initialized each coefficient β_j independently and uniformly within $(-1, 1)$, and we then applied the proximal gradient descent algorithm (19) and each of the acceleration versions of it. The values of the tuning parameter λ where set to one of three values $\lambda \in \{0.1, 1, 10\}$. The maximum number of fixed-point iterations for each method was set to 20, 000. Other hyperparameters of the acceleration methods and convergence criteria were set to their defaults. For each choice of λ, we evaluated all of the methods using 200 independent simulation runs.

Results from this simulation study can be found in Table 3. On average, all of the methods provided consistent acceleration of the original proximal gradient descent algorithm. Among the different acceleration methods, DAAREM consistently gave the greatest acceleration with a more than 20 fold improvement in execution time across different choices of λ. The failure of gradient descent for smaller values of λ is due to slow convergence, i.e. it did not converge with a maximum number of iterations.

Table 2: Simulation results for estimating Poisson mixture parameters from 500 independent runs. Elapsed times are reporting in milliseconds. If an algorithm failed to converge or if it converged to a negative log-likelihood more than 1% larger than that of the original EM algorithm, then we called that run a failure. As a measure of robustness, we also recorded the number of failures for each acceleration method.

Metric	EM	SQUAREM	DAAREM	pEM	Quasi-Newton	Nesterov
fpevals	2238 ± 273	77 ± 19.1	$\mathbf{53 \pm 14.3}$	195 ± 101	131 ± 144	166 ± 23.5
elapsed	32.8 ± 6.34	$\mathbf{2.82 \pm 4.62}$	5.4 ± 4.28	5.49 ± 4.96	5.01 ± 5.82	3.82 ± 3.45
# failures	0	0	0	0	1	0

Table 3: Simulation results for estimating regression coefficients using LASSO logistic regression with 200 independent runs. *pGD* represents the original proximal gradient descent algorithm, and the other columns represent different acceleration methods. If an algorithm failed to converge or if it converged to a loss more than 1% larger than the optimal loss, we considered it to be a failure. As a measure of robustness, we also recorded the number of failures for each acceleration method.

Metric	pGD	SQUAREM	DAAREM	pEM	Quasi-Newton	Nesterov
fpevals ($\lambda = 10$)	11125 ± 1721	351 ± 53.2	$\mathbf{214 \pm 61.2}$	1894 ± 126	617 ± 170	404 ± 53.7
elapsed ($\lambda = 10$)	31.3 ± 4.85	0.782 ± 0.13	$\mathbf{0.711 \pm 0.22}$	5.47 ± 0.678	1.68 ± 0.49	1.23 ± 0.22
# failures ($\lambda = 10$)	0	0	0	0	0	0
fpevals ($\lambda = 1$)	19292 ± 720	1057 ± 104	$\mathbf{603 \pm 481}$	3094 ± 597	2947 ± 1874	676 ± 44.4
elapsed ($\lambda = 1$)	56.3 ± 5.84	2.46 ± 0.645	$\mathbf{2.01 \pm 1.69}$	9.09 ± 2.32	7.78 ± 4.88	2.17 ± 0.84
# failures ($\lambda = 1$)	54	0	10	0	1	0
fpevals ($\lambda = 0.1$)	20000+	1063 ± 156	$\mathbf{618 \pm 429}$	2603 ± 80.8	3112 ± 1741	658 ± 77.6
elapsed ($\lambda = 0.1$)	58 ± 2.46	2.52 ± 0.717	$\mathbf{1.96 \pm 1.34}$	7.55 ± 0.55	8.74 ± 4.85	2.08 ± 0.33
# failures ($\lambda = 0.1$)	200	0	0	0	0	0

5.5 Variational Inference in Bayesian Variable Selection

Bayesian variable selection methods for a regression model (e.g. George and McCulloch (1997), Chipman et al. (2001)) with binary outcomes often consider the following model:

$$\text{logit}\{p(y_i = 1|\mathbf{X}, \beta)\} = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j; \quad \beta_j = \gamma_j Z_j;$$
$$\gamma_j \sim \text{Bern}(\pi); \quad Z_j \sim \mathcal{N}(0, \sigma_\beta^2); \quad \theta = (\pi, \sigma_\beta^2) \sim p_\theta(\cdot),$$

where $\text{Bern}(p)$ denotes the Bernoulli distribution with success probability p and $p_\theta(\cdot)$ denotes the prior distribution for the vector of hyperparameters $\theta = (\pi, \sigma_\beta^2)$. The values of the variables γ_j drive the model selection as $\beta_j = 0$ whenever $\gamma_j = 0$. For simplicity, in this section we set $\beta_0 = 1$ as a known constant.

In Bayesian variable selection, calculation of the marginal posterior inclusion probabilities is a primary interest. The marginal posterior inclusion probability for variable j, defined as $PIP(j) = p(\gamma_j = 1|\mathbf{X}, y)$, can be expressed as

$$PIP(j) = \int p(\gamma_j = 1|\mathbf{X}, y, \theta)p(\theta|\mathbf{X}, y)d\theta, \tag{20}$$

where $\mathbf{X}$ is the $n \times p$ design matrix whose (i, j) element is X_{ij} and $y = (y_1, \ldots, y_n)$.

The quantity $p(\gamma_j = 1|\mathbf{X}, y, \theta)$ does not have a closed-form and is often calculated using Markov Chain Monte Carlo (MCMC) methods, for example Bottolo et al. (2010) and Clyde et al. (2011). However, in high-dimensional applications, MCMC can be very computationally

inefficient and often requires days or even weeks to run. To address this, Carbonetto et al. (2012) propose using variational inference to approximate $p(\gamma_j = 1|\mathbf{X}, y, \theta)$ and then approximate the integral in (20) with importance sampling.

To approximate $p(\gamma_j = 1|\mathbf{X}, y, \theta)$, one first approximates the posterior density $p(\beta, \gamma|\mathbf{X}, y, \theta)$ with a density function of the form $q(\beta, \gamma|\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{s}^2) = \prod_{j=1}^{p} q(\beta_j, \gamma_j|\alpha_j, \mu_j, s_j^2)$, where the components of this product are assumed to be a mixture of a normal density and a point mass at 0. Specifically,

$$q(\beta_j, \gamma_j|\alpha_j, \mu_j, s_j^2) = \begin{cases} \alpha_j N(\beta_j|\mu_j, s_j^2) & \text{if } \gamma_j = 1 \\ (1-\alpha_j)\delta_0(\beta_j) & \text{otherwise,} \end{cases} \tag{21}$$

where $N(\cdot|\mu, \sigma^2)$ stands for a normal density with mean μ and variance σ^2 and $\delta_0(\cdot)$ for a delta mass centered at 0. To find the best set of parameters, Carbonetto et al. (2012) derives an EM-type algorithm that can maximize the evidence lower bound (ELBO):

$$\begin{aligned} s_j^2 &= \frac{1}{(\mathbf{X}^T\hat{U}\mathbf{X})_{jj} + 1/\sigma_\beta^2} \\ \mu_j &= s_j^2\left((X^T\hat{y})_j - \sum_{h\neq j}(\mathbf{X}^T\hat{U}\mathbf{X})_{hj}\alpha_h\mu_h\right) \\ \frac{\alpha_j}{1-\alpha_j} &= \frac{\pi}{1-\pi} \times \frac{s_j}{\sigma_\beta} \times \exp\left(\frac{\mu_j^2}{2s_j^2}\right) \end{aligned} \tag{22}$$

where $(\mathbf{X}^T\mathbf{X})_{hj}$ denotes the (h, j) element of the matrix $\mathbf{X}^T\mathbf{X}$ and $(\mathbf{X}^T y)_j$ denotes the jth element of the vector $\mathbf{X}^T y$. Also, $\hat{U}$ and $\hat{y}$ are defined as $\hat{U} = \text{diag}\{\boldsymbol{u}\} - \boldsymbol{u}\boldsymbol{u}^T/\bar{\boldsymbol{u}}$; $\hat{y} = y - \frac{1}{2} - \boldsymbol{u}$, where $\boldsymbol{u} = (u_1, \ldots, u_n)$ and $\bar{\boldsymbol{u}} = \sum_{i=1}^{n} u_i$. The terms $u_i = \frac{1}{\eta_i}\big(\psi(\eta_i) - 1/2\big)$ are updated from

$$\eta_i^2 = \left(\sum_{j=1}^{p} X_{ij}\mathbb{E}[\beta_j]\right)^2 + \sum_{j=1}^{p} X_{ij}^2 \text{Var}[\beta_j],$$

where $\psi(x) = 1/(1+e^{-x})$. From the variational approximation (21), we have $\mathbb{E}[\beta_j] = \alpha_j\mu_j$ and $\text{Var}[\beta_j] = \alpha_j(s_j^2 + \mu_j^2) - (\alpha_j\mu_j)^2$.

In this experiment, we examined whether the acceleration schemes described above can effectively accelerate the coordinate descent algorithm for Bayesian variable selection in the case of logistic regression. For this simulation study, we use a setting similar to that described in Carbonetto et al. (2012). Specifically, we assume that $\text{logit}\{p(y_i = 1|\beta, \mathbf{X})\} = -1 - Z_{i1} + Z_{i2} + \sum_{j=1}^{p} X_{ij}\beta_j$, where Z_{i1} and Z_{i2} are independent standard normal distributions. In these simulations, $\beta = (\beta_1, \ldots, \beta_p)$ has length 2000, and only the first m components of β are assumed to be nonzero. The nonzero components of β were sampled independently from a $N(0, 0.25)$ distribution, and the remaining components were set to 0. The elements X_{ij} of $\mathbf{X}$ were drawn independently from a Binomial$(2, p_{ij})$, where p_{ij} is drawn from a uniform distribution over $(0.05, 0.5)$. Values of $m = 100$ and $m = 200$ were considered, and we performed 200 simulation runs for each setting of m. In each run, we set the sample size and number of covariates to $n = 200$ and $p = 2000$ respectively. We initialized the α_j by drawing independently from a uniform distribution over $(0, 1)$, the μ_j were initialized by drawing from a normal distribution with mean 0 and variance 0.1, and the s_j do not need to be initialized as they are first updated using equation (22). The model hyperparameters (π, σ_β^2) were set to $\pi = m/2000$ and $\sigma_\beta = 0.5$.

Table 4: Simulation results for Bayesian variable selection using 200 independent runs. *EM* represents the original algorithm with the various acceleration methods in other columns. If an algorithm failed to converge or if it converged to a loss more than 1% larger than the optimal loss, we called it a failure. We also recorded the number of failures for each method as a measure of robustness.

Metric	EM	SQUAREM	DAAREM	pEM	Quasi-Newton	Nesterov
fpevals ($m = 100$)	207 ± 11.7	115 ± 21.5	883 ± 230	158 ± 22	349 ± 205	$\mathbf{77.5 \pm 10.3}$
elapsed ($m = 100$)	4.71 ± 0.54	$\mathbf{1.3 \pm 0.29}$	20.4 ± 5.49	3.65 ± 0.64	5.06 ± 3.06	1.76 ± 0.31
# failures ($m = 100$)	0	0	150	0	0	4
fpevals ($m = 200$)	194 ± 7.42	102 ± 13.5	157 ± 37.3	132 ± 22.6	603 ± 310	$\mathbf{81.3 \pm 6.88}$
elapsed ($m = 200$)	4.19 ± 0.26	$\mathbf{1.1 \pm 0.15}$	3.6 ± 0.85	2.89 ± 0.47	8.29 ± 4.23	1.75 ± 0.21
# failures ($m = 200$)	0	0	0	0	51	0

The maximum number of fixed-point iterations was set to 1000. The hyperparameters of the acceleration algorithms and convergence criteria were set to their default values (see Section 4).

Results from this simulation study are shown in Table 4. We do not observe a huge acceleration in this experiment, but SQUAREM still provides a consistent speedup with a roughly 4 fold improvement for both settings of m. DAAREM, on the other hand, failed frequently partly due to it convergence to a solution with a slightly higher than the optimal value of the loss function.

5.6 Sinkhorn Scaling

Given a matrix A, the problem of re-scaling its rows and columns to form a doubly stochastic matrix $\Gamma = DAE$, where D and E are diagonal matrices, is called a matrix balancing problem. A more constrained version of the matrix balancing problem is to find diagonal scaling matrices D, E such that $\Gamma = DAE$ and that Γ has specified row and column sums, that is, $\Gamma\mathbf{1} = \mathbf{a}$; $\Gamma^T\mathbf{1} = \mathbf{b}$, where $\mathbf{1}$ is a vector whose entries are all equal to 1. This problem has a vast array of applications including, for example, ranking web pages (Knight, 2008), learning permutation matrices from data (Mena et al., 2018), solving optimal transport problems (Altschuler et al., 2017), etc.

A naive algorithm for the constrained matrix balancing problem is the Sinkhorn–Knopp algorithm (Sinkhorn and Knopp, 1967). The algorithm simply scales the matrix iteratively by rows and columns. Given an initialization $\mathbf{u}_0$, $\mathbf{v}_0$, the algorithm finds the next updates by

$$\mathbf{u}_{k+1} = \frac{\mathbf{a}}{A\mathbf{v}_k}; \quad \mathbf{v}_{k+1} = \frac{\mathbf{b}}{A^T\mathbf{u}_{k+1}}, \tag{23}$$

where the division of vectors in (23) is done element-wise and A is the matrix with entries A_{ij}.

In this numerical experiment, we tested the performance of the acceleration methods on the Sinkhorn-Knopp algorithm for a constrained matrix balancing problem where we used certain ill-conditioned matrices known as Marshall-Olkin and Hessenberg matrices (Parlett and Landis (1982)). A Marshall-Olkin matrix $\boldsymbol{M}_3$ is a 3×3 matrix with columns $(100, 100, 0)^T$,

Table 5: Experimental results for matrix scaling from 200 independent runs. *SK* represents the original Sinkhorn-Knopp algorithm, and the other columns are different accelerated versions of it. Elapsed time are reported in milliseconds. The number of failures (failure to converge) is also recorded to capture the robustness of each algorithm.

Metric	SK	SQUAREM	DAAREM	pEM	Quasi-Newton	Nesterov
fpevals ($\boldsymbol{M}_3$)	5764 ± 121	68 ± 10.1	29.4 ± 0.83	1243 ± 89.9	$\mathbf{27.5 \pm 10.8}$	218 ± 1.45
elapsed ($\boldsymbol{M}_3$)	185 ± 13.9	2.36 ± 0.69	3.88 ± 0.65	39.5 ± 3.91	$\mathbf{1.55 \pm 1.91}$	6.76 ± 4.51
# failures ($\boldsymbol{M}_3$)	0	0	0	0	0	0
fpevals ($\boldsymbol{H}_{10}$)	2671 ± 2.5	$\mathbf{146 \pm 53.6}$	3216 ± 5888	804 ± 17.1	7139 ± 1933	246 ± 10.1
elapsed ($\boldsymbol{H}_{10}$)	83.2 ± 4.26	$\mathbf{4.88 \pm 3.51}$	388 ± 721	29.3 ± 1.89	475 ± 148	8.14 ± 4.46
# failures ($\boldsymbol{H}_{10}$)	0	0	1	0	0	0
fpevals ($\boldsymbol{H}_{50}$)	50000+	4857 ± 10503	50000+	17853 ± 15.2	33307 ± 1910	$\mathbf{1336 \pm 735}$
elapsed ($\boldsymbol{H}_{50}$)	4700+	218 ± 516	7610+	1130 ± 37.2	3139 ± 168	$\mathbf{71.1 \pm 38.9}$
# failures ($\boldsymbol{H}_{50}$)	200	10	200	0	0	0

$(100, 10000, 1)^T$, $(0, 1, 100)^T$. We choose the order n Hessenberg matrix $\boldsymbol{H}_n$ to be the $n \times n$ matrix such that $\boldsymbol{H}_n(i, i) = 100$ for all i and $\boldsymbol{H}_n(i, j) = 1$, for all (i, j) such that $j > i - 1$, $j \neq i$. All of the other elements in $\boldsymbol{H}_n$ are equal to 0. For simplicity, $\mathbf{a}$ and $\mathbf{b}$ in iteration (23) will be set to all 1s.

We run such experiment for 200 independent times. In each run, we used Hessenberg matrices of order 10 and 50. We initialized parameter $\boldsymbol{v}$ as i.i.d draws from a uniform distribution between $[0.5, 2]$. Notice that $\boldsymbol{u}$ can be calculate from algorithm (23), therefore we do not treat it as part of the parameter vector in the acceleration algorithms. The maximum number of fixed-point iterations was set to 50,000. We measured the performance of the scaling algorithm by calculating the mean absolute differences (MAD) between row/column sums and 1. If the two MAD values were, at any time, both smaller than 10^{-10} we terminated the algorithm and regarded it to be converged. Other hyperparameters of the acceleration algorithms are set to their defaults (see Section 4).

Results from this numerical experiment can be found in Table 5. The results in Table 5 show that we gain substantial and consistent acceleration by using either Nesterov with restarts or SQUAREM.

It is also possible to use a different approach to matrix scaling, which uses a different fixed-point iteration (see Supplementary section 3). By using the intermediate scaled matrix $\text{diag}\{\boldsymbol{u}_k\}\ A\ \text{diag}\{\boldsymbol{v}_k\}$ as parameter vector rather than $\boldsymbol{u}$, $\boldsymbol{v}$. The acceleration schemes perform much better with this approach. Although the final output from this method is not guaranteed to be feasible, we confirmed that the relative difference is small. For example, in $\boldsymbol{H}_{50}$ case, the DAAREM algorithm, which originally failed, converged in a few hundreds of iterations. It also gave a reasonably accurate answer, the discrepancy from true result being smaller than 10^{-5}.

5.7 Manifold Embedding

Manifold learning is a useful approach for performing both dimension reduction and data visualization of high-dimensional data. However, many manifold learning approaches can potentially be influenced by the exploding distance in high-dimensional space, and make points with moderate distance in the original space become too crowded in the embedded low-dimensional space. This phenomenon is of called the "crowding problem" (Cook et al., 2007). To address this problem, Van der Maaten and Hinton (2008) extended the Stochastic Neighborhood Embedding (SNE) method by using a t-distributed neighborhood probability for points in the embedded space. The corresponding method is called t-SNE, and this method achieved great success in visualizing handwritten digits data and a variety of other higher-dimensional images and text data.

The main input into the t-SNE procedure, is a collection of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ from which one can compute nonnegative similarity scores P_{ij} between pairs $\mathbf{x}_i$ and $\mathbf{x}_j$ normalized so that $\sum_{ij} P_{ij} = 1$. Given the matrix P containing the values of P_{ij}, t-SNE seeks to find an embedding matrix $Y = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)^T$ with each $\mathbf{y}_k$ in a lower-dimensional space by minimizing the following KL divergence-based loss function with respect to Y

$$L(Y; P) = \sum_{ij} P_{ij} \log \frac{P_{ij}}{Q_{ij}} \tag{24}$$

$$Q_{ij} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}}{\sum_{ab} \left(1 + ||\mathbf{y}_a - \mathbf{y}_b||^2\right)^{-1}}. \tag{25}$$

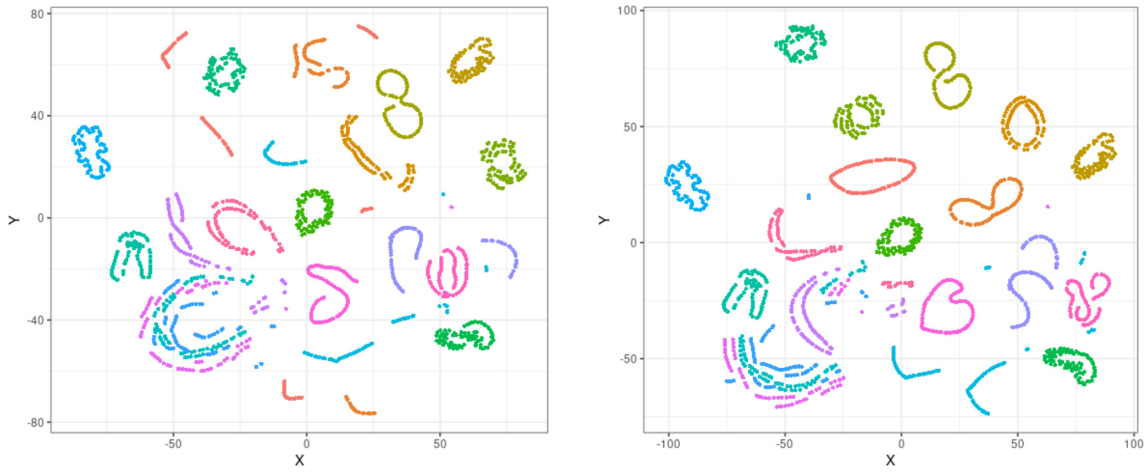
Yang et al. (2015) derived an alternative algorithm for solving a range of manifold embedding problems and claimed it can be more efficient than many existing algorithms. The algorithm proposed by Yang et al. (2015) is essentially an MM algorithm that iteratively majorizes the complex loss function of interest by a specially designed quadratic form and then minimizing it with a closed-form solution. In the context of t-SNE where we want to minimize (24), this algorithm results in the following updating scheme for a current embedding matrix Y_k

$$Y_{k+1} = \left(\mathcal{L}_{P \circ q} + \frac{\rho}{4} I\right)^{-1} \left(\mathcal{L}_{Q \circ q} Y_k + \frac{\rho}{4} Y_k\right), \tag{26}$$

where q is the matrix with elements $q_{ij} = \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}$, the operation $\circ$ denotes the Hadamard product, Q is the matrix whose elements are defined in (25), and ρ is a positive

Table 6: Experiment results for $t-$SNE from 50 independent runs. *MM* represents the original MM algorithm, and the other columns are different acceleration versions of it. *objval* is the final Kullback–Leibler divergence obtained by the acceleration method. Smaller values of *objval* correspond to better embeddings.

Metric	MM	SQUAREM	DAAREM	pEM	Quasi-Newton	Nesterov
fpevals	62.3 ± 24	162 ± 84.2	$\mathbf{50 \pm 10.6}$	100 ± 45.8	114 ± 57.1	127 ± 128
elapsed	11.8 ± 4.41	22.1 ± 11.4	$\mathbf{9.6 \pm 1.98}$	18.1 ± 8.4	17.9 ± 9.17	26.9 ± 27.2
objval	0.328 ± 0.022	$\mathbf{0.279 \pm 0.017}$	0.289 ± 0.024	0.309 ± 0.016	0.299 ± 0.023	0.347 ± 0.054



(a) MM (b) SQUAREM

Figure 1: Visualizing one run of experiments. (a) Embedding from the original MM algorithm with objective value 0.343. (b) Embedding from the SQUAREM algorithm with objective value 0.264. Different colors are used for different objects. We can see that SQUAREM, which obtained a lower value of the objective function, does provide better separation quality across different objects.

scalar. Also, $\mathcal{L}_M$ is the Laplacian of the matrix M which is defined as $\mathcal{L}_M = \Lambda - M$, where Λ is a diagonal matrix with diagonal entries $\Lambda_{ii} = \sum_j M_{ij}$.

In this experiment, we used the COIL20 data to study the MM algorithm (26) and its acceleration using the methods described in Section 3. The COIL20 dataset has 1440 images of 20 objects with resolution 128×128. Each object has 72 images which were taken by capturing an image at every 5 degrees along a 360 degree viewing circle. To evaluate the different acceleration methods, we ran each method 50 times using the MM algorithm (26) as the base iteration. In each of the 50 runs, we initialized the elements of the embedding matrix Y by sampling from a Normal distribution with mean zero and standard deviation 10^{-3}. The value of ρ in iteration (26) is found by using an initial value of 10^{-5} in each iteration and using a backtracking search to maintain monotonicity. Since the parameter values are not identified in embedding problems, we guided convergence using values of the objective function. Specifically, an algorithm is terminated whenever 1000 fixed point iterations have been reached or when the relative change of the objective function is less than 10^{-4}. All other hyperparameters were set to their default values (see Section 4).

Results from this numerical experiment can be found in Table 6. We can see that SQUAREM, DAAREM, parabolic-EM, and Quasi-Newton all achieved a substantially better objective value upon convergence when compared to the original MM algorithm. Moreover, DAAREM achieved this improved objective value with an even shorter computation time than the MM algorithm. Figure 1 visualizes and compares the embedding results between MM and SQUAREM, indicating that better values of the objective function achieved by SQUAREM do result in a better quality embedding.

6 Discussion

In Section 5, we tested all the acceleration schemes described in Section 3 on six different practical problems. Although no single method is guaranteed to always work, at least one of SQUAREM and DAAREM effectively accelerated the original algorithm in every setting of the six examples. Moreover, SQUAREM is particularly robust in providing improved performance in every scenario of our numerical experiments. Nesterov algorithm is most popular for accelerating proximal gradient. Here we have shown that SQUAREM and DAAREM are competitive when compared to Nesterov and hence deserving of greater attention.

We summarize the results from the six main simulation studies. SQUAREM effectively accelerated the original algorithm in 5 of the 6 problems, giving up to 78 fold improvement in elapsed time with a mean reduction of roughly 18 fold. In the t-SNE problem, SQUAREM did not accelerate the convergence of the original algorithm, but it did consistently converge to a better solution than the original MM algorithm. Nesterov with restarts accelerates convergence in 4 of the 6 problems, with a speedup of up to 68 fold and a mean of 16 fold improvement. DAAREM can also accelerate 4 of the 6 tasks, gaining a factor of up to 48 fold improvement with a mean of a 13 fold reduction in computation time. Quasi-Newton also accelerates 4 of the 6 problems with mean of a 13 fold improvement in convergence time. Lastly, *Parabolic-EM* accelerated convergence in 4 of the 6 tasks with a mean 4 fold improvement in convergence. Since the relative performance of the acceleration methods can vary across specific problems, we suggest trying SQUAREM, DAAREM, and Nesterov with restarts when acceleration is needed for a specific problem.

We refer the reader to the our `AccelBenchmark` Github package, which allows the user to easily identify the **best** acceleration scheme for their specific problem. To use `AccelBenchmark`, the user only needs to supply the data, fixed-point mapping function, and a loss function if one is available, and the package will then automatically benchmark the performance of the original algorithm and all acceleration methods. For details, one can consult the package vignette.

We would like to draw attention to the importance of monotonicity control implemented in all our acceleration algorithms. The base algorithms such as EM and MM are intrinsically monotone. Hence they have guaranteed global convergence. Acceleration schemes are based on extrapolation methods. They are fast in the neighborhood of the solution. However, they are non-monotone and are not globally convergent. Therefore, we implement safeguards such as controlling the steplength and ensuring monotonicity to ensure reliable convergence from any starting value (note that in the default settings monotonicity is relaxed as the solution is approached). Performance is not guaranteed without such safeguards. More importantly, adding the safeguards seldom degrades the speed of convergence, while providing a better guarantee of convergence that is essential for general purpose implementation.

It is worth noting that the fixed-point iterations used in some of the examples discussed here are already faster versions of the original fixed-point schemes. Specifically, the MMF procedure for the multivariate-t distribution is a faster version of the original EM algorithm; proximal gradient is a faster version of the subgradient method for the LASSO problem; and the MM algorithm for t-SNE is a faster procedure than gradient descent. Nevertheless, as we have demonstrated in our simulation studies, we can further accelerate these faster schemes using the described acceleration schemes and achieve more significant speedups in many cases. Therefore, the acceleration schemes listed here are worth trying even in problems where there is a relatively fast fixed-point iterative algorithm already available.

We have shown that acceleration methods can be very effective in a wide range of appli-

cations where fixed-point iteration algorithms are used. However, the use of these or similar acceleration methods in other contexts are still lacking in development. For example, adapting these methods to handle nondeterministic iterations such as stochastic gradient descent or Markov chain Monte Carlo procedures would be an interesting topic for future research. Adapting these acceleration techniques to infinite-dimensional parameter settings (e.g., solutions of integral equations (Atkinson, 1976)) would also be an interesting direction for future study. The main issue here is that the parameter dimension can change across iterations, which may require the introduction of operators like an inner product in a certain Hilbert space to compute the acceleration method updates. For example, the Picard iteration (Junkins et al., 2013) and gradient tree boosting (Friedman, 2001) involve update functions where the number of parameters increases across iterations.

Supplementary Material

1. We provide an R package `AccelBenchmark`, available from Github that can be used to reproduce the experiments in this paper. Please check the vignette and the `demo.R` file under vignette folder for reference.
2. We provide an additional `pdf` file that includes a) some basic properties of fixed point iterations; b) visualization of the convergence for all the experiments and c) additional analysis for the Sinkhorn scaling problem.

References

Altschuler J, Weed J, Rigollet P (2017). Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. arXiv preprint: https://arxiv.org/abs/1705.09634.

Anderson DG (1965). Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4): 547–560.

Atkinson KE (1976). *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*, volume 16. Society for Industrial and Applied Mathematics, Philadelphia.

Berlinet A, Roland C (2009). Parabolic acceleration of the EM algorithm. *Statistics and Computing*, 19(1): 35–47.

Bobb JF, Varadhan R (2021). turboEM: A Suite of Convergence Acceleration Schemes for EM, MM and Other Fixed-Point Algorithms. R package version 2021.1.

Bottolo L, Richardson S, et al. (2010). Evolutionary stochastic search for Bayesian model exploration. *Bayesian Analysis*, 5(3): 583–618.

Boyd S, Boyd SP, Vandenberghe L (2004). *Convex Optimization*. Cambridge University Press.

Carbonetto P, Stephens M, et al. (2012). Scalable variational inference for Bayesian variable selection in regression, and its accuracy in genetic association studies. *Bayesian Analysis*, 7(1): 73–108.

Chipman H, George EI, McCulloch RE, Clyde M, Foster DP, Stine RA (2001). The practical implementation of Bayesian model selection. *Lecture Notes-Monograph Series*, 65–134.

Clyde MA, Ghosh J, Littman ML (2011). Bayesian adaptive sampling for variable selection and model averaging. *Journal of Computational and Graphical Statistics*, 20(1): 80–101.

Cook J, Sutskever I, Mnih A, Hinton G (2007). Visualizing similarity data with a mixture of maps. In: *Artificial Intelligence and Statistics*, 67–74. PMLR.

Dempster AP, Laird NM, Rubin DB (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1): 1–22.

Du Y, Varadhan R (2020). SQUAREM: An R package for off-the-shelf acceleration of EM, MM and other EM-like monotone algorithms. *Journal of Statistical Software*, 92: 1–41.

Evans C, Pollock S, Rebholz LG, Xiao M (2020). A proof that Anderson acceleration improves the convergence rate in linearly converging fixed-point methods (but not in those converging quadratically). *SIAM Journal on Numerical Analysis*, 58(1): 788–810.

Fang Hr, Saad Y (2009). Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3): 197–221.

Friedman JH (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 1189–1232.

Geist M, Scherrer B (2018). Anderson acceleration for reinforcement learning. arXiv preprint: https://arxiv.org/abs/1809.09501.

George EI, McCulloch RE (1997). Approaches for Bayesian variable selection. *Statistica Sinica*, 339–373.

Guyon I, Gunn SR, Ben-Hur A, Dror G (2004). Result analysis of the NIPS 2003 feature selection challenge. In: *NIPS*, volume 4, 545–552.

Hasannasab M, Hertrich J, Laus F, Steidl G (2021). Alternatives to the EM algorithm for ML estimation of location, scatter matrix, and degree of freedom of the Student t distribution. *Numerical Algorithms*, 87(1): 77–118.

Hasselblad V (1966). Estimation of parameters for a mixture of normal distributions. *Technometrics*, 8(3): 431–444.

Henderson N, Varadhan R (2020). daarem: Damped Anderson Acceleration with Epsilon Monotonicity for Accelerating EM-Like Monotone Algorithms. R package version 0.5.

Henderson NC, Varadhan R (2019). Damped Anderson acceleration with restarts and monotonicity control for accelerating EM and EM-like algorithms. *Journal of Computational and Graphical Statistics*, 28(4): 834–846.

Higham NJ, Strabić N (2016). Anderson acceleration of the alternating projections method for computing the nearest correlation matrix. *Numerical Algorithms*, 72(4): 1021–1042.

Hobolth A, Guo Q, Kousholt A, Jensen JL (2020). A unifying framework and comparison of algorithms for non-negative matrix factorisation. *International Statistical Review*, 88(1): 29–53.

Hunter DR, Lange K (2004). A tutorial on MM algorithms. *The American Statistician*, 58(1): 30–37.

Jin Y, Tam OH, Paniagua E, Hammell M (2015). TEtranscripts: A package for including transposable elements in differential expression analysis of RNA-seq datasets. *Bioinformatics*, 31(22): 3593–3599.

Junkins JL, Bani Younes A, Woollands RM, Bai X (2013). Picard iteration, chebyshev polynomials and chebyshev-picard methods: Application in astrodynamics. *The Journal of the Astronautical Sciences*, 60(3): 623–653.

Knight PA (2008). The Sinkhorn–Knopp algorithm: Convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1): 261–275.

Liu C, Rubin DB (1995). ML estimation of the multivariate t distribution with unknown degrees of freedom. *Statistica Sinica*, 5: 19–39.

Mena G, Belanger D, Linderman S, Snoek J (2018). Learning latent permutations with gumbel-sinkhorn networks. arXiv preprint: https://arxiv.org/abs/1802.08665.

Nesterov Y (2013). Gradient methods for minimizing composite functions. *Mathematical Pro-*

gramming, 140(1): 125–161.

O'Donoghue B, Candes E (2015). Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3): 715–732.

Parlett B, Landis TL (1982). Methods for scaling to doubly stochastic form. *Linear Algebra and its Applications*, 48: 53–79.

Raket LL, Grimme B, Schöner G, Igel C, Markussen B (2016). Separating timing, movement conditions and individual differences in the analysis of human movement. *PLoS Computational Biology*, 12(9): e1005092.

Raydan M, Svaiter BF (2002). Relaxed steepest descent and cauchy-barzilai-borwein method. *Computational Optimization and Applications*, 155–167.

Shiraishi Y, Tremmel G, Miyano S, Stephens M (2015). A simple model-based approach to inferring and visualizing cancer mutation signatures. *PLoS genetics*, 11(12): e1005657.

Sinkhorn R, Knopp P (1967). Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2): 343–348.

Song J, Babu P, Palomar DP (2016). Sequence set design with good correlation properties via majorization-minimization. *IEEE Transactions on Signal Processing*, 64(11): 2866–2879.

Tibshirani R (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1): 267–288.

Toth A, Kelley CT (2015). Convergence analysis for Anderson acceleration. *SIAM Journal on Numerical Analysis*, 53(2): 805–819.

Tseng P (2009). On accelerated proximal gradient methods for convex-concave optimization. 2008, http://www.math.washington.edu/~tseng/papers/apgm.pdf.

Van der Maaten L, Hinton G (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11).

Varadhan R, Roland C (2004). Squared extrapolation methods (SQUAREM): A new class of simple and efficient numerical schemes for accelerating the convergence of the EM algorithm. *Department of Biostatistics Working Paper 63*, Johns Hopkins University. https://biostats.bepress.com/jhubiostat/paper63/.

Varadhan R, Roland C (2008). Simple and globally convergent methods for accelerating the convergence of any EM algorithm. *Scandinavian Journal of Statistics*, 35(2): 335–353.

Walker HF, Ni P (2011). Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4): 1715–1735.

Yang Z, Peltonen J, Kaski S (2015). Majorization-minimization for manifold embedding. In: *Artificial Intelligence and Statistics*, 1088–1097. PMLR.

Zhang J, O'Donoghue B, Boyd S (2020). Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4): 3170–3197.

Zhou H, Alexander D, Lange K (2011). A quasi-Newton acceleration for high-dimensional optimization algorithms. *Statistics and Computing*, 21(2): 261–273.

Zhu X, Stephens M (2018). Large-scale genome-wide enrichment analyses identify new trait-associated genes and pathways across 31 human phenotypes. *Nature Communications*, 9(1): 1–14.