ISSN 1346-5597

National Institute of Informatics

NII Technical Report

Jacobian-Free Three-Level Trust Region Method for Nonlinear Least Squares Problems

Wei Xu, Ning Zheng, and Ken Hayami

NII-2014-003E
Sep. 2014

URL: http://www.nii.ac.jp/ Tokyo 101-8430, JAPAN +81-3-4212-2000 FAX: +81-3-3556-1916

Jacobian-Free Three-Level Trust Region Method for Nonlinear Least Squares Problems*

Wei Xu[†] Ning Zheng[‡] and Ken Hayami[§]

September 17, 2014

Abstract

Nonlinear least squares (NLS) problems arise in many applications. The common solvers require to compute and store the corresponding Jacobian matrix explicitly, which is too expensive for large problems. In this paper, we propose an effective Jacobian free method especially for large NLS problems because of the novel combination of using automatic differentiation for $J(\mathbf{x})\mathbf{v}$ and $J^T(\mathbf{x})\mathbf{v}$ along with the preconditioning ideas that do not require forming the Jacobian matrix $J(\mathbf{x})$ explicitly. Together they yield a new and effective three-level iterative approach. In the outer level, the dogleg/trust region method is employed to solve the NLS problem. At each iteration of the dogleg method, we adopt the iterative linear least squares (LLS) solvers, CGLS or BA-GMRES method, to solve the LLS problem generated at each step of the dogleg method as the middle iteration. In order to accelerate the convergence of the iterative LLS solver, we propose an inner iteration preconditioner based on the weighted Jacobi method. Compared to the common dogleg solver and truncated Newton method, our proposed three level method need not compute the gradient or Jacobian matrix explicitly, and is efficient in computational complexity and memory storage. Furthermore, our method does not rely on the sparsity or structure pattern of the Jacobian, gradient or Hessian matrix. Thus, it can be applied to solve any large general NLS problem. Numerical experiments show that our proposed method is much superior to the common trust region method and truncated Newton method.

Key words: nonlinear least squares problem, Newton's method, automatic differentiation, CGLS method, BA-GMRES method, weighted Jacobi method, trust region method.

*This work was supported by the Natural Science Foundation of China (Project No: 11101310), the MOU Grant of the National Institute of Informatics, Japan, and the Grant-in-Aid for Scientific Research (C) of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

[†]Department of Mathematics, Tongji University, 1239 Siping Road, Shanghai, 200092, People's Republic of China (wdxu@tongji.edu.cn).

[‡]Department of Informatics, School of Multidisciplinary Sciences, The Graduate University for Advanced Studies (Sokendai), 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan/Department of Mathematics, Tongji University, 1239 Siping Road, Shanghai, 200092, People's Republic of China (nzheng@nii.ac.jp).

[§]National Institute of Informatics/Department of Informatics, School of Multidisciplinary Sciences, The Graduate University for Advanced Studies (Sokendai), 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan (hayami@nii.ac.jp).

1 Introduction

Nonlinear least squares (NLS) problems arise in numerous areas of applications, such as imaging, tomography, geophysics and economics. The general form of the NLS problem can be expressed as

$$\min_{\mathbf{x}\in\mathbb{R}^{\mathbf{n}}} \frac{1}{2}\|\mathbf{F}(\mathbf{x})\|_2^2, \tag{1.1}$$

where $\mathbf{F}(\mathbf{x}) \equiv (F_1(\mathbf{x}), \cdots, F_m(\mathbf{x}))^T$ is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$ (usually, $m \geq n$) and twice differentiable. Denote the objective function in (1.1) as $f(\mathbf{x})$, i.e., $f(\mathbf{x}) \equiv \frac{1}{2}\|\mathbf{F}(\mathbf{x})\|_2^2$. Then, the solution of (1.1) is equivalent to the solution of the following nonlinear equations,

$$\nabla f(\mathbf{x}) = J^T(\mathbf{x})\mathbf{F}(\mathbf{x}) = \mathbf{0},$$

where $\nabla f(\mathbf{x})$ is the gradient of $f(\mathbf{x})$ and $J(\mathbf{x})$ is the Jacobian matrix of $\mathbf{F}(\mathbf{x})$. Thus, the corresponding iterative process of Newton's method can be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k,$$

where λ_k is a parameter for the length of the step and $\mathbf{d}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k)$ and $\nabla^2 f(\mathbf{x}_k)$ is the second derivatives of function $f(\mathbf{x})$, i.e., the Hessian matrix of $f(\mathbf{x})$, which is given by

$$H(\mathbf{x}) \equiv \nabla^2 f(\mathbf{x}) = J^T(\mathbf{x})J(\mathbf{x}) + \sum_{i=1}^{m} F_i(\mathbf{x})\nabla^2 F_i(\mathbf{x}). \tag{1.2}$$

In the case where $\mathbf{x}$ is close to the solution, the Hessian matrix $H(\mathbf{x})$ can be approximated by the first term, thus avoiding a rather lengthy calculation in the second term. This approximation leads to the Gauss-Newton (G-N) method [7] for the NLS problem. In this method, the following linear least squares problem is required to be solved at each iteration of the G-N method,

$$\min_{\mathbf{d}_k} \|J(\mathbf{x}_k)\mathbf{d}_k + \mathbf{F}(\mathbf{x}_k)\|_2. \tag{1.3}$$

Another well known method to solve the nonlinear least squares problem (1.1) is the Levenberg-Marquardt (L-M) method [7] which approximates the Hessian matrix $H(\mathbf{x})$ in (1.2) by $J^T(\mathbf{x})J(\mathbf{x}) + \mu I$, where μ is called the damping parameter updated at each iteration. Correspondingly, an expanded linear least squares problem, rather than (1.3), is required to be solved, that is

$$\min_{\mathbf{d}_k} \left\| \begin{bmatrix} J(\mathbf{x}_k) \\ \sqrt{\mu} I \end{bmatrix} \mathbf{d}_k + \begin{bmatrix} \mathbf{F}(\mathbf{x}) \\ \mathbf{0} \end{bmatrix} \right\|_2. \tag{1.4}$$

After the direction vector $\mathbf{d}_k$ is obtained from (1.3) or (1.4), some optimization schemes, such as line search [7], can be employed to determine the parameter λ_k at each iteration. Although both G-N and L-M methods avoid calculating the second term in (1.2), they still require to calculate the Jacobian matrix $J(\mathbf{x})$ for each Newton iteration, which is expensive in computational time and storage.

In fact, Jacobian-free Newton's methods are popular in solving nonlinear equations in many real applications [6, 16, 29]. In solving nonlinear equations when $m = n$, a linear system $J(\mathbf{x}_k)\mathbf{d}_k = -\mathbf{F}(\mathbf{x}_k)$, rather than an LS problem, has to be solved at each iteration. Thus, some iterative methods, such as the GMRES [12, 23], and IDR methods[25] can be employed to solve the linear system without

forming the Jacobian matrix explicitly while the Jacobian-vector multiplication can be estimated by the finite difference as

$$J(\mathbf{x})\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{x}+\epsilon\mathbf{v}) - \mathbf{F}(\mathbf{x})}{\epsilon}.$$

In order to accelerate the convergence of the iterative method, some preconditioner construction techniques were also proposed [16, 8, 10, 3, 28]. However, all these techniques require some part of the Jacobian matrix or the structure of the Jacobian matrix in advance. Thus, these Jacobin-free methods are usually problem dependent. On the other hand, although the LS problem (1.3) is equivalent to the normal equation $J_k^T J_k \mathbf{d}_k = -J_k^T \mathbf{F}(\mathbf{x}_k)$, the finite difference method can only estimate $J(\mathbf{x})\mathbf{v}$, rather than $J^T(\mathbf{x})\mathbf{v}$. Thus, these Jacobian-free techniques cannot be applied to solve the LS problem (1.3) or (1.4) without forming $J(\mathbf{x})$ itself.

The truncated Newton method is another category of iterative methods for solving optimization problem [19]. In the Newton's framework, it employs the iterative methods, say CG and GMRES methods, to solve the linear system for each Newton iteration with the first and second derivatives of the objective function. Usually, it assumes that the gradient of the objective function is available. Then, finite differencing is employed to estimate Hessian-vector multiplication so that the linear system can be solved without forming the Hessian matrix. In order to accelerate the convergence of the iterative method, some preconditioner construction techniques were also proposed in [18]. Unfortunately, just like the Jacobian-free method [6, 16] for solving nonlinear equations, most of these techniques still require some or the whole part of the Hessian matrix itself. A diagonal scaling preconditioner, which is the only effective preconditioner without forming the Hessian matrix itself in [18], can be constructed through the BFGS formula in the quasi-Newton method. In 1989, Dixon and Price combined the truncated Newton method with AD and proposed a Hessian truncated Newton method solving optimization problems with sparse Hessian matrices without the requirement of the gradient[9]. In fact, they just replaced the finite difference method with AD to calculate the Hessian-vector multiplication. No preconditioner construction is mentioned in [9]. Thus, the truncated Newton method converges quite slow in some cases without preconditioners.

The automatic differentiation(AD) was first proposed in the 1970's [2, 11] for computing the derivatives in machine precision via the chain rule. There are two modes to compute the derivatives, the forward mode and the reverse mode. The forward mode can compute the product $J(\mathbf{x})\mathbf{v}$ with any given vector $\mathbf{v}$ in the same cost as one function evaluation while the reverse mode estimates the product $\mathbf{w}^T J$. There are quite a few AD packages available on [1] supporting various programming languages such as C/C++, Fortran, Matlab and so on. In this paper, we adopt both modes to construct the preconditioner and solve the LS problem (1.3) or (1.4) without forming $J(\mathbf{x})$ explicitly.

As known, it is not necessary to treat the nonlinear LS problem as a general optimization problem, since computing a Hessian matrix is quite expensive. In this paper, we first adopt the dogleg/trust-region method to solve the NLS problem. Then, the popular LS iteration solvers, CGLS [5, 13, 24] and BA-GMRES [17] methods, can be applied to solve (1.3) and (1.4) without forming $J(\mathbf{x})$ explicitly. Thus, the automatic differentiation (AD), instead of the finite difference method, is employed to calculate $J(\mathbf{x})\mathbf{v}$ and $J^T(\mathbf{x})\mathbf{v}$. Compared to the finite difference method, AD can estimate both $J(\mathbf{x})\mathbf{v}$ and $J^T(\mathbf{x})\mathbf{v}$ in machine precision with almost the same cost as one function evaluation. In order to accelerate the convergence of these solvers, we propose an inner iteration preconditioner based on a weighted Jacobi method. In our proposed three-level trust-region method, all the required information is the objective function $\mathbf{F}(\mathbf{x})$ itself in (1.1). There is no request for the sparsity of the Jacobian matrix of $\mathbf{F}(\mathbf{x})$ or the gradient of $\frac{1}{2}\|\mathbf{F}(\mathbf{x})\|_2^2$. In other words, our method can solve the NLS problems

no matter whether the Jacobian matrix is dense, sparse or structured as long as the Jacobian matrix is full rank. In all these three-level iterations, only the Jacobian matrix-vector multiplication $J(\mathbf{x})\mathbf{v}$ and $J^T(\mathbf{x})\mathbf{v}$ are required, which can be computed by AD through the forward mode and reverse mode, respectively. Thus, the Jacobian matrix $J(\mathbf{x})$ never needs to be stored explicitly and the structure of the Jacobian matrix is never needed. Compared to other preconditioner techniques for sequence of matrices [8, 10, 3], our inner iteration preconditioner is cheap and does not rely on the sparsity or structure of $J(\mathbf{x})$. Thus, theoretically, our proposed three-level method can be applied to any nonlinear LS problems as long as its Jacobian matrix is full rank. As a special case, when $m = n$, our method is also applicable for solving nonlinear equations. Although we only focus on the trust region method in this paper, our technique is applicable to the L-M method as well, since the expanded LS problem (1.4) can be treated as a special case of (1.3).

The rest of the paper is organized as follows. In Section 2, we give the algorithm of the Jacobian free trust region method. The method is described under the generic dogleg/trust-region framework. Since the dominant part of the dogleg method is to solve LS problems, we only focus on how to solve the LS problem (1.3) via the iterative method. Then, the inner-iteration preconditioner construction is discussed in Section 3. After that, we apply our proposed method to solve some NLS problems from the optimization test problem set CUTEr [4] and two data fitting problems in Section 4. Compared to the common dogleg method implementation from immoptibox [20], Matlab built-in NLS solver lsqnonlin and the truncated Newton method with diagonal scaling preconditioner [19], our proposed method requires much less storage and computational time, especially when the problem size is large. Finally, we conclude with some remarks in Section 5.

2 Jacobian free Trust Region Method

One of the popular methods for solving the NLS problems is the dogleg/trust-region method [7, 21]. It solves a subproblem at each iteration to generate a trial step $\mathbf{s}$ as

$$\min_{\mathbf{s}} \left\{ \mathbf{s}^T J_k^T \mathbf{F}(\mathbf{x}_k) + \frac{1}{2}\mathbf{s}^T J_k^T J_k \mathbf{s},\ \|\mathbf{s}\|_2 \le \Delta_k,\ \mathbf{s} \in P_k \right\}, \tag{2.5}$$

where $J_k \equiv J(\mathbf{x}_k)$, $\Delta_k > 0$ is the trust region radius and P_k is the 'dogleg' piecewise linear path connecting $\mathbf{x}_k$ to the Cauchy point (i.e., the minimizer of the quadratic function (2.5) along the negative gradient direction $-J_k^T\mathbf{F}(\mathbf{x}_k)$). In other words, the solution of the subproblem (2.5) can be in two directions. One is the solution of the LS problem (1.3), denoted as $\mathbf{d}_k^{gn}$, a step for the Gauss-Newton method. The other direction is the steepest decent direction, given by

$$\mathbf{d}_k^{sd} = -J_k^T \mathbf{F}(\mathbf{x}_k).$$

Then, a parameter α is evaluated to determine how far the step is on the steepest decent direction, that is

$$\alpha = \frac{\|J_k^T \mathbf{F}(\mathbf{x}_k)\|_2}{\|J_k J_k^T \mathbf{F}(\mathbf{x}_k)\|_2}. \tag{2.6}$$

Now, we have two candidates for the step to take from the current point $\mathbf{x}_k$: $\mathbf{a} = \alpha \mathbf{d}_k^{sd}$ and $\mathbf{b} = \mathbf{d}_k^{gn}$. Then, the trust region radius Δ is used to choose the step $\mathbf{d}_k$ between $\mathbf{a}$ and $\mathbf{b}$. In order to make a proper length on the step, a ratio ρ is calculated to control the radius Δ of the trust region as

$$\rho = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{d})}{L(\mathbf{0}) - L(\mathbf{d})},$$

where $f(\mathbf{x}) = 0.5\|\mathbf{F}(\mathbf{x})\|_2^2$ and $L(\mathbf{d}) = 0.5\|\mathbf{F}(\mathbf{x}_k) + J_k\mathbf{d}\|^2$. When the ratio ρ is large, it indicates that the linear model is good. We can increase Δ and thereby take a longer step, which is close to the direction $\mathbf{d}_k^{gn}$. If ρ is small, maybe even negative, Δ is reduced, indicating smaller steps, which is close to the steepest decent direction $\mathbf{d}_k^{sd}$. Thus, the algorithm for the dogleg/trust region method can be summarized as follows.

Algorithm 1 *Dogleg/trust region method for solving NLS problems.*

1). Given $\mathbf{x}_0$, Δ_0, ϵ, k_{max} and $nostop = \mathbf{true}$;
2). $k = 0$, $\mathbf{x} = \mathbf{x}_0$, $\Delta = \Delta_0$, $\mathbf{g} = -J_0^T\mathbf{F}(\mathbf{x})$;
3). **While** $k < k_{max}$ and $nostop$
4). $\quad$ $k = k + 1$, Compute α as (2.6);
5). $\quad$ $\mathbf{d}^{sd} = \alpha\mathbf{g}$;
6). $\quad$ Solve LS problem $J_k\mathbf{d}^{gn} = -\mathbf{F}(\mathbf{x}_k)$;
7). $\quad$ **if** $\|\mathbf{d}^{gn}\| \leq \Delta$
8). $\quad\quad$ $\mathbf{d} = \mathbf{d}^{gn}$;
9). $\quad$ **elseif** $\|\alpha\mathbf{d}^{sd}\| \geq \Delta$
10). $\quad\quad$ $\mathbf{d} = (\Delta/\|\mathbf{d}^{sd}\|)\mathbf{d}^{sd}$;
11). $\quad$ **else**
12). $\quad\quad$ $\mathbf{d} = \alpha\mathbf{d}^{sd} + \beta(\mathbf{d}^{gn} - \alpha\mathbf{d}^{sd})$, where β is chosen to satisfy $\|\mathbf{d}\| = \Delta$;
13). $\quad$ **end**;
14). $\quad$ **if** $\|\mathbf{d}\| \leq \epsilon(\|\mathbf{x}\|)$;
15). $\quad\quad$ $nonstop = \mathbf{false}$;
16). $\quad$ **else**
17). $\quad\quad$ $\mathbf{x}_{new} = \mathbf{x} + \mathbf{d}$;
18). $\quad\quad$ $\rho = (f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{d}))/(L(\mathbf{0}) - L(\mathbf{d}))$;
19). $\quad\quad$ **if** $\rho > 0$
20). $\quad\quad\quad$ $\mathbf{x} = \mathbf{x}_{new}$, $\mathbf{g} = -J_k^T\mathbf{F}(\mathbf{x})$;
21). $\quad\quad$ **end**
22). $\quad\quad$ **if** $\rho > 0.75$
23). $\quad\quad\quad$ $\Delta = \max\{\Delta, 3\|\mathbf{d}\|\}$;
24). $\quad\quad$ **elseif**
25). $\quad\quad\quad$ $\Delta = \Delta/2$;
26). $\quad\quad$ **end**
27). $\quad$ **end**
28). **end**

As shown, the major part in Algorithm 1 is to solve the LS problem in line 6, which dominated the computational complexity of the dogleg method. For the solution of the LS problem, we adopt the two typical iterative methods, CGLS [13, 5] and BA-GMRES [14, 17] methods. Thus, the preconditioned CGLS (PCGLS) method and BA-GMRES methods to solve the LS problem (1.3) can be described as follows.

Algorithm 2 *PCGLS method for solving the LS problem (1.3).*

1). Let $\mathbf{d}_0$ be the initial solution of the LS problem (1.3);
2). Compute $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}_k) - J(\mathbf{x}_k)\mathbf{d}_0$;
3). $\mathbf{s}_0 = J_k^T\mathbf{r}_0$, $\mathbf{z}_0 = P^{-1}\mathbf{s}_0$, $\mathbf{p}_0 = \mathbf{z}_0$, $\gamma_0 = (\mathbf{s}_0,\ \mathbf{z}_0)$;
4). **for** $j = 1, 2, \cdots$ do

5). $\mathbf{q}_j = J_k \mathbf{p}_j$;
6). $\alpha_j = \gamma_0/(\mathbf{q}_j,\ \mathbf{q}_j)$;
7). $\mathbf{d}_{j+1} = \mathbf{d}_j + \alpha_j \mathbf{p}_j$;
8). $\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j \mathbf{q}_j$;
9). $\mathbf{s}_{j+1} = J_k^T \mathbf{r}_{j+1}$;
10). **if** $\|\mathbf{s}_{j+1}\| < \epsilon \|\mathbf{s}_0\|$, then stop;
11). Compute $\mathbf{z}_{j+1} = P^{-1}\mathbf{s}_{j+1}$;
12). $\gamma_{j+1} = (\mathbf{s}_{j+1},\ \mathbf{z}_{j+1})$;
13). $\beta_j = \gamma_{j+1}/\gamma_j$;
14). $\mathbf{p}_{j+1} = \mathbf{z}_{j+1} + \beta_j \mathbf{p}_j$;
15). **end**

The lines 3 and 11 in Algorithm 2 require to compute the product of preconditioner P^{-1} with a vector $J_k^T \mathbf{r}_j$ so that the CGLS method can converge fast. Some preconditioner construction technique for sequence of matrices [3, 8, 10] and the incomplete LU factorization [12] may be used to determine the preconditioner in line 3 and 11 in Algorithm 2. However, since the Jacobian matrix is updated at each iteration of Algorithm 1, all these techniques require the information of some part of the Jacobian matrix or the sparsity pattern. Unfortunately, all these information are not available if we do not form the Jacobian matrix explicitly. Thus, we will employ inner-iterations to construct the preconditioner implicitly as in [17]. The idea behind the inner-iteration is like this. The preconditioner P is constructed as a close approximation of the parameter matrix $J_k^T J_k$ so that P^{-1} is an approximation of $(J_k^T J_k)^{-1}$. It implies that the product $P^{-1} J_k^T \mathbf{F}(\mathbf{x}_k)$ is an approximate solution of the normal equation, $J_k^T J_k \mathbf{z} = J_k^T \mathbf{F}(\mathbf{x}_k)$. In other words, since the explicit form of P is not necessary in Algorithm 2, but $P^{-1}(J_k^T \mathbf{F}(\mathbf{x}_k))$ only, we can employ a simple iterative method to solve the normal equation approximately in a fixed number of iterations. The returned solution from the simple iterative method can be treated as $P^{-1}(J_k^T \mathbf{F}(\mathbf{x}_k))$ in the lines 3 and 11. The same idea is also applicable to the BA-GMRES method [14, 17].

Algorithm 3 *BA-GMRES iteration scheme to solve the LS problem (1.3).*

1). Let $\mathbf{d}_0$ be the initial solution of the LS problem (1.3);
2). $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}_0) - J_k \mathbf{d}_0$;
3). $\mathbf{s}_0 = J_k^T \mathbf{r}_0$, $\mathbf{z}_0 = P^{-1}\mathbf{s}_0$;
4). $\beta = \|\mathbf{z}_0\|_2$, $\mathbf{v}_1 = \mathbf{z}_0/\beta$;
5). **for** $j = 1, 2, \cdots$ m
6). $\mathbf{u}_j = J_k \mathbf{v}_j,\ \mathbf{z}_j = P^{-1} J_k^T \mathbf{u}_j$;
7). **for** $i = 1, 2 \cdots, j$
8). $h_{i,j} = (\mathbf{z}_j,\ \mathbf{v}_i)$;
9). $\mathbf{z}_j = \mathbf{z}_j - h_{i,j}\mathbf{v}_i$
10). **end**
11). $h_{j+1,j} = \|\mathbf{w}_j\|_2$, $\mathbf{v}_{j+1} = \mathbf{z}_j/h_{j+1,j}$;
12). Find $\mathbf{y}_j \in \mathbf{R}^j$ that minimizes $\|\beta \mathbf{e}_1 - \bar{H}\mathbf{y}_j\| = \|J_k^T \mathbf{r}_j\|_2$
13). $\mathbf{d}_j = \mathbf{d}_0 + [\mathbf{v}_1,\ \mathbf{v}_2, \cdots, \mathbf{v}_j]\mathbf{y}_j$;
14). **end**
15). Return $\mathbf{d}_m$;

In fact, there are two preconditioner construction techniques proposed in this paper. One is the diagonal scaling while the other is the inner preconditioner based on the weighted Jacobi method. The

preconditioner P^{-1} should approximate $(J_k^T J_k)^{-1}$. Thus, in the first approach, we take P^{-1} as an approximation of the diagonal of $(J_k^T J_k)^{-1}$. As shown in [26], the diagonal scaling preconditioner is nearly optimal in some circumstances. Based on this approximated diagonal, we can further solve the normal equation $J_k^T J_k \mathbf{z}_0 = J_k^T \mathbf{r}_0$ approximately, by the weighted Jacobi method with matrix-vector multiplications $J_k^T \mathbf{w}$ and $J_k \mathbf{v}$, instead of forming P^{-1} explicitly. Thus, it leads to the inner-iteration preconditioner in Section 3.

3 Preconditioner Construction

In this section, we propose two preconditioner strategies to accelerate the convergence of the preconditioned CGLS and BA-GMRES method in the middle iteration. As we mentioned in Section 2, the preconditioner vector product $P^{-1} J_k^T \mathbf{r}_j$ can be defined as an approximate solution, $\mathbf{z}$, of the normal equation,

$$J_k^T J_k \mathbf{z} = J_k^T \mathbf{r}_j. \tag{3.7}$$

The first approach is to construct P as the estimated diagonal of $J_k^T J_k$ explicitly. Since J_k is not calculated explicitly, we adopt the approximation of J_k in [27] to estimate the diagonal of $J_k^T J_k$. Then, based on this estimated diagonal, we can apply the weighted Jacobi method [15] to solve (3.7) approximately to obtain the product $P^{-1} J_k^T \mathbf{F}(\mathbf{x}_k)$. For the sake of analysis, we can also construct the explicit form of the preconditioner P based on the weighted Jacobi method. Its property is also presented at the end of this section.

3.1 Diagonal Scaling Preconditioner

The diagonal scaling technique is commonly used as preconditioners in solving nonlinear minimization problems. Theoretical evidence indicates that the diagonal scaling technique is an effective way to accelerate the convergence of the iterative method in solving normal equations [18, 26]. In this subsection, a sample scaling strategy based on the quasi-Newton method for the NLS problem [27] is described. In [27], a BFGS rank one update formula was proposed to approximate the Jacobian matrix J_{k+1} as

$$B_{k+1} = B_k + \frac{(\mathbf{y}_k - B_k \mathbf{d}_k)\mathbf{d}_k^T}{\mathbf{d}_k^T \mathbf{d}_k},$$

where $\mathbf{d}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \mathbf{F}(\mathbf{x}_{k+1}) - \mathbf{F}(\mathbf{x}_k)$. Then, the product of $J_{k+1}^T J_{k+1}$ can be estimated by

$$\begin{aligned} B_{k+1}^T B_{k+1} &= \left(B_k + \frac{(\mathbf{y}_k - B_k \mathbf{d}_k)\mathbf{d}_k^T}{\mathbf{d}_k^T \mathbf{d}_k} \right)^T \left(B_k + \frac{(\mathbf{y}_k - B_k \mathbf{d}_k)\mathbf{d}_k^T}{\mathbf{d}_k^T \mathbf{d}_k} \right) \\ &= B_k^T B_k + \frac{B_k^T \mathbf{y}_k \mathbf{d}_k^T + \mathbf{g}_k \mathbf{d}_k^T + \mathbf{d}_k \mathbf{y}_k^T B_k + \mathbf{d}_k \mathbf{g}_k^T}{\mathbf{d}_k^T \mathbf{d}_k} \\ &\quad + \frac{\mathbf{y}_k^T \mathbf{y}_k - \mathbf{d}_k^T B_k^T \mathbf{y}_k - \mathbf{y}_k^T B_k \mathbf{d}_k - \mathbf{d}_k^T \mathbf{g}_k}{(\mathbf{d}_k^T \mathbf{d}_k)^2} \mathbf{d}_k \mathbf{d}_k^T, \end{aligned} \tag{3.8}$$

since $B_k^T B_k \mathbf{d}_k = -\mathbf{g}_k$ in the secant algorithm in [27]. Since B_k is an approximation of J_k, the product $B_k^T \mathbf{y}_k$ can be estimated by $J_k^T \mathbf{y}_k$. Then, the diagonal of the preconditioner P_{k+1} for the $(k+1)$th

outer iteration can be estimated by the diagonal of (3.8), that is

$$\mathrm{diag}(P_{k+1}) = \mathrm{diag}(P_k) + \frac{2}{\mathbf{d}_k^T\mathbf{d}_k}(\mathbf{d}_k.*\mathbf{u}_k + \mathbf{g}_k.*\mathbf{d}_k) + \frac{\mathbf{y}_k^T\mathbf{y}_k - 2\mathbf{d}_k^T\mathbf{u}_k - \mathbf{d}_k^T\mathbf{g}_k}{\mathbf{d}_k^T\mathbf{d}_k}\mathbf{d}_k.*\mathbf{d}_k, \qquad (3.9)$$

where $\mathbf{u}_k = J_k^T\mathbf{y}_k$, '.∗' represents the entry-wise multiplication of two vectors, and $P_0 = \mathrm{diag}(1, 1, \cdots, 1)$. The main advantage of this preconditioner construction is its low cost. It only requires one extra function evaluation at each outer iteration. However, the diagonal entries on P_{k+1} cannot be guaranteed to be positive in (3.9) since $B_k^T\mathbf{y}_k$ is only an approximation of $J_k^T\mathbf{y}_k$. If some diagonal entries are nonpositive, we will force them to be 1 to guarantee the positiveness so that the diagonal scaling matrix P_{k+1} can also be a proper preconditioner for the CGLS method. As for the BA-GRMES method, the preconditioner is constructed as $P_{k+1}^{-1}J_k^T$ so that the BA-GMRES method returns the solution of the LS problem (1.3).

3.2 Weighted Jacobi Inner Iteration

Given the linear system

$$J_k^T J_k\mathbf{d}_k = -J_k^T\mathbf{F}(\mathbf{x}_k),$$

its weighted Jacobi iteration [15] to solve this normal equation can be rewritten as

$$\mathbf{d}_k^{(j+1)} = \mathbf{d}_k^{(j)} + \omega D^{-1}(-J_k^T\mathbf{F}(\mathbf{x}_k) - J_k^T J_k\mathbf{d}_k^{(j)}), \qquad (3.10)$$

where D can be any positive diagonal matrix and ω is the weight. As shown in (3.10), the cost for each weighted Jacobi iteration is two function evaluations to calculate $J_k^T J_k\mathbf{d}_k^{(j)}$. Thus, in our inner iteration preconditioner construction, we just run the weighted Jacobi method a few steps, say one or two iterations to get an approximate solution for the normal equation. The parameter matrix $J_k^T J_k$ can be split into

$$J_k^T J_k = \underbrace{\frac{1}{\omega}D}_{M} - \underbrace{\left(\frac{1}{\omega}D - J_k^T J_k\right)}_{N} = M - N.$$

Thus, the iteration matrix for the weighted Jacobi method is

$$H = M^{-1}N = \omega D^{-1}\left(\frac{1}{\omega}D - J_k^T J_k\right) = I - \omega D^{-1}J_k^T J_k.$$

Thus, we have

$$\begin{aligned} H &= I - \omega D^{-1}J_k^T J_k = D^{-\frac{1}{2}}(I - \omega D^{-\frac{1}{2}}J_k^T J_k D^{-\frac{1}{2}})D^{\frac{1}{2}} \\ &= D^{-\frac{1}{2}}V(I - \omega\Sigma^T\Sigma)V^T D^{\frac{1}{2}}, \end{aligned}$$

where the singular value decomposition of $J_k D^{-\frac{1}{2}}$ is $J_k D^{-\frac{1}{2}} = U\Sigma V^T$. Assume that J_k is full column rank, then the spectrum of H is

$$\rho(H) = \max_i(|1 - \omega\sigma_i^2|), \quad i = 1, 2, ..., n,$$

where $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n > 0$ are the singular values of $J_k D^{-\frac{1}{2}}$. Set $\rho(H) < 1$, then

$$|1 - \omega\sigma_i^2| < 1, \quad i = 1, 2, ..., n \quad \Rightarrow \quad 0 < \omega < \frac{2}{\sigma_1^2}.$$

Suppose that the weighted Jacobi method runs l steps for the preconditioner in the inner iteration with initial value $\mathbf{d}_k^{(0)} = \mathbf{0}$. Then, we have

$$\mathbf{d}_k^{(l)} = \omega \sum_{j=0}^{l-1} \mathbf{h}_k^{(j)},$$

where

$$\mathbf{h}_k^{(j)} = D^{-1}(-J_k^T \mathbf{F}(\mathbf{x}_k) - J_k^T J_k \mathbf{d}_k^{(j)}) = D^{-1} J_k^T \mathbf{r}_k^{(j)}.$$

Note that

$$\begin{aligned} \mathbf{r}_k^{(j)} &= -\mathbf{F}(\mathbf{x}_k) - J_k \mathbf{d}_k^{(j)} = \mathbf{r}_k^{(j-1)} - \omega J_k \mathbf{h}_k^{(j-1)} \\ &= (I - \omega J_k D^{-1} J_k^T)\mathbf{r}_k^{(j-1)} = -(I - \omega J_k D^{-1} J_k^T)^j \mathbf{F}(\mathbf{x}_k), \end{aligned}$$

thus

$$\mathbf{h}_k^{(j)} = -(I - \omega D^{-1} J_k^T J_k)^j D^{-1} J_k^T \mathbf{F}(\mathbf{x}_k).$$

Let $B^{(l)}$ be

$$B^{(l)} = \omega \sum_{j=0}^{l-1} (I - \omega D^{-1} J_k^T J_k)^j D^{-1} J_k^T,$$

then $\mathbf{d}_k^{(l)} = -B^{(l)} \mathbf{F}(\mathbf{x}_k)$, and

$$C^{(l)} = \omega \sum_{j=0}^{l-1} (I - \omega D^{-1} J_k^T J_k)^j D^{-1}.$$

Note that $B^{(l)}$ can act as B in BA-GMRES of Algorithm 3 for $\mathcal{R}(B^{(l)}) = \mathcal{R}(J_k^T)$. Since D is an approximation of the diagonal of $J_k^T J_k$ and each diagonal entry of D is positive, then $C^{(l)}$ can be rewritten as

$$\begin{aligned} C^{(l)} &= \omega D^{-\frac{1}{2}} [\sum_{j=0}^{l-1} (I - \omega D^{-\frac{1}{2}} J_k^T J_k D^{-\frac{1}{2}})^j] D^{-\frac{1}{2}} \\ &= \omega D^{-\frac{1}{2}} V [\sum_{j=0}^{l-1} (I - \omega \Sigma^T \Sigma)^j] V^T D^{-\frac{1}{2}} \\ &= D^{-\frac{1}{2}} V \mathrm{diag}(\tilde{\sigma}_1, \cdots, \tilde{\sigma}_n)(D^{-\frac{1}{2}} V)^T, \end{aligned}$$

where $\tilde{\sigma}_i = \frac{1-(1-\omega\sigma_j^2)^l}{\sigma_j}$, $i = 1, 2, ..., n$. Thus, we obtain the following theorem.

Theorem 3.1

$$\begin{aligned} \omega < 0 &\implies C^{(l)} \textit{ is negative definite;} \\ 0 < \omega < \frac{2}{\sigma_1^2} &\implies C^{(l)} \textit{ is positive definite;} \\ \omega \geq \frac{2}{\sigma_1^2} &\implies \begin{cases} C^{(l)} \textit{ is negative definite when } l \textit{ is even,} \\ C^{(l)} \textit{ is not definite when } l \textit{ is odd.} \end{cases} \end{aligned}$$

As for the L-M method, the outer iteration is similar to Algorithm 1. Its dominant part is to solve an expanded LS problem (1.4), instead of (1.3). For the expanded LS problem (1.4), the corresponding normal equation is

$$(J_k^T J_k + \mu I)\mathbf{z} = -J_k^T \mathbf{F}(\mathbf{x}_k).$$

Thus, the iteration matrix for the weighted Jacobi method can be expressed as

$$H \equiv M^{-1}N = I - \omega\mu D^{-1} - \omega D^{-1} J_k^T J_k.$$

With similar analysis as in Theorem 3.1, we can show that the spectrum of H is less than 1 when $0 < \omega < \frac{2}{\mu+\sigma_1^2}$. The corresponding preconditioning matrix for the weighted Jacobi method is

$$C^{(l)} = \omega \sum_{i=0}^{l-1} (I - \omega\mu D^{-1} - \omega D^{-1} J_k^T J_k)^i.$$

We can also show that $C^{(l)}$ is symmetric positive definite when $0 < \omega < 1/(\mu + \sigma_1^2)$, which implies that matrix $C^{(l)}$ is a suitable preconditioner for the CGLS method when solving the expanded LS problem (1.4) when $0 < \omega < 1/(\mu + \sigma_1^2)$. In other words, the weighted Jacobi method can also be applied in the inner-iteration as an implicit preconditioner for the L-M method.

4 Numerical Experiments

In this section, we compare our proposed methods with the common dogleg implementation in immoptibox [20], MATLAB built-in NLS solver lsqnonlin, which is implemented by the dogleg/trust region method and the truncated Newton method with diagonal scaling preconidtioner [19]. All experiments are carried on a machine with Intel Core Duo 3.16GHz CPU, 8GB RAM and 500GB hard driver. The machine is running MATLAB 2013b under Windows 7 Professional.

Here, two sets of problems are taken for the experiment. The first set of problems is from the Constrained and Unconstrained Testing Environment (CUTEr) [4]. We choose four unconstrained NLS problems from this set. The size of all these problems varies from 2000 to 15000. The other problem set is two data fitting problems. The performance of nine methods is compared on all these testing problems. We first run the dogleg method (DL) in immoptibox to solve these problems, where the Jacobian matrix J_k is evaluated by the AD at each iteration and the LS problem is solved by the 'backslash' built in Matlab. Then, the CGLS method is embedded into the dogleg method without a preconiditioner, i.e., dogleg with CGLS method (DL-CG), to solve the LS problem (1.3). After that, we try our proposed Jacobian-free dogleg method with different preconditioner techniques. In these methods, we take the dogleg framework for the outer iteration, preconditioned CGLS and BA-GMRES methods for the middle iteration with different preconditioner strategies. Thus, these methods include dogleg with diagonal scaling preconditioner strategy, i.e., DL-CG-DS and DL-BA-DS, and with the weighted Jacobi method as the inner preconditioner, i.e., DL-CG-R(1) and DL-BA-R(i), where the number i in the parenthesis represents how many steps in the weighted Jacobi matrix are taken for the inner weighted Jacobi method. Usually, we set i to be 1 or 2, that is only one or two steps of weighted Jacobi method are run for the inner preconditioner. Due to Theorem 3.1, $C^{(l)}$ is only positive definite for $\omega > 1/\sigma_1^2$ when l is odd. Thus, based on our determination of ω, $C^{(1)}$ is positive definite so that it is suitable as a preconditioner for the CGLS method. Finally, we solve the NLS problem by the function lsqnonlin in the optimization toolbox in Matlab

(MATLAB) and the truncated Newton method with preconditioner proposed in [19] (TNewton).

Then, we record the computational times in seconds (time), the number of function evaluations (fun), the number of iterations for the dogleg method (out) and the total number of iterations for the preconditioned CGLS and BA-GMRES method solving LS problems (mid) as measurements for the performance and effectiveness of preconditioners. In all these methods, we set the stop tolerance for the outer iteration, the dogleg/trust region framework, as 10^{-6} and the maximum number of outer iterations as 100. As for the middle iteration, we set the stop tolerance for the PCGLS or BA-GMRES method as 10^{-8} and the maximum number of iterations as 300. In the inner iteration, we only run i steps of the weighted Jacobi method for preconditioning. As analyzed in the previous section, the parameter ω in the weighted Jacobi method should be less than $2/\lambda_{max}(D^{-1}J_k^T J_k)$. Thus, we estimate $\lambda_{max}(D^{-1}J_k^T J_k)$ by running three steps of the power method, i.e., $\tilde{\lambda}_{max}(D^{-1}J_k^T J_k)$, then ω is determined as $2/(\tilde{\lambda}_{max}(D^{-1}J_k^T J_k) + 0.05)$.

In the first set of problems, we choose four functions from CUTEr. The following are the descriptions of function $\mathbf{F}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$ for all these four problems.

1. Penalty function I (Penalty I)
   (a) n is a variable, $m = n + 1$
   (b) $$\begin{aligned} y_i &= \sqrt{\alpha}(x_i - 1), \qquad 1 \leq i \leq n \\ y_{n+1} &= \sum_{j=1}^n x_j^2 - 0.25, \qquad \text{where } \alpha = 10^{-5}. \end{aligned}$$
2. Variable dimensioned function (VDF)
   (a) n is variable, $m = n + 2$
   (b) $$\begin{aligned} y_i &= x_i - 1, \qquad 1 \leq i \leq n \\ y_{n+1} &= \sum_{j=1}^n j(x_j - 1), \\ y_{n+2} &= \left(\sum_{j=1}^n j(x_j - 1)^2\right)^2. \end{aligned}$$
3. Brown almost linear function (BALF)
   (a) n is variable, $m = n$
   (b) $$\begin{aligned} y_i &= x_i + \sum_{j=1}^n x_j - (n + 1) \qquad 1 \leq i < n \\ y_n &= (\prod_{j=1}^n x_j^2) - 1. \end{aligned}$$
4. Linear function-full rank (LFFK)
   (a) n is variable, $m = 1.25n$
   (b) $$\begin{aligned} y_i &= x_i - \frac{2}{m}\left(\sum_{j=1}^n x_j\right) - 1, \qquad 1 \leq i \leq n \\ y_i &= \left(\sum_{j=1}^n x_j\right) - 1, \qquad n < i \leq m. \end{aligned}$$

The first two problems, Penalty I and VDF, are two typical unconstrained NLS problems while the third problem, BALF, is a nonlinear equation problem. The fourth problem, LFFK, can be converted into a linear LS problem. We choose these four problems with variable sizes to test the performance of our Jacobian-free methods on different category problems. Next, we show how all these nine methods perform on these four problems. Table 1 and 2 record the computational times (time) in seconds,

number of function evaluations(fun), outer iteration numbers(out) and total middle-iteration numbers(mid) of all nine methods on solving the first four NLS problems.

The results in Table 1 to 2 show that our Jacobian free method is far more efficient than the common implementation of dogleg method in immoptibox and Matlab built-in NLS solver lsqnonlin. The main computational cost in these two methods is dominated by estimating the Jacobian matrix and solving LS problems. Thus, the Jacobian-free methods can be a good alternative to reduce computational times and storage requirement. As for the VDF problem, only the TNewton, DL-CG-R(1) and DL methods manage to solve it. Other dogleg methods cannot return the NLS solution while the Matlab built-in solver, lsqnonlin fails within the maximum number of function valuations, which is defined as $100 \times n$ by default, either, when the problem size is larger than 6000. Thus, the inner preconditioner can also increase the accuracy and robustness of the Jacobian-free method.

It also shows that for large problems, say larger than 8000, our methods can return solutions in seconds while the common dogleg methods requires hours and the truncated Newton method solves the problems in minutes. In the whole computation, we only require the information of the objective function $\mathbf{F}(\mathbf{x})$, which means that our proposed method do not rely on any sparse format of Jacobian, gradient and Hessian matrix. Therefore, all these methods can be applied to general NLS problems. In other words, our method can solve a large variety of NLS problems.

For these selected CUTEr problems, the DL-CG method works pretty well. It only requires only a few iterations to solve each LS problem in the dogleg framework. Thus, our preconditioned Jacobian free method does not have too many advantages on these problems. However, our proposed methods have much better performance than the common dogleg implementations and truncated Newton method.

The second problem set includes two data fitting problems. The data fitting problem can be generally described as follows. Given data points $(x_1,\ y_1), \cdots, (x_m,\ y_m)$, which satisfies

$$y_i = \Psi(t_i) + \epsilon_i, \qquad \text{for } i = 1, 2, \cdots, m,$$

where Ψ is the background function and $\{\epsilon_i\}$ is measured as 'noise'. Our goal is to find an approximation to $\Psi(\mathbf{t})$ in the domain $[a,\ b]$ spanned by the data abscissas. Thus, a fitting model can be defined as $M(\mathbf{x}, \mathbf{t})$ with arguments $\mathbf{t} = (t_1, t_2, \cdots, t_m)$ and parameters $\mathbf{x} = (x_1, \cdots, x_n)$. We try to find $\mathbf{x}^*$ such that $M(\mathbf{x}^*,\ \mathbf{t})$ is the best approximation to $\Psi(\mathbf{t})$. Usually, the number of parameters n is smaller than the number of data points, m, i.e., $n < m$.

The first data fitting problem we consider is as follows.

$$M(\mathbf{x}, t) = x_1 e^{\frac{x_2}{t_i + x_3}} + e^{x_{\min\{i,n\}}},$$

where $i = 1, 2, \cdots, m$ and n is the number of entries in $\mathbf{x}$. We assume that there exists an $\mathbf{x}^\sharp$ so that

$$y_i = M(\mathbf{x}^\sharp, t_i) + \epsilon_i, \qquad i = 1, 2, \cdots, m, \tag{4.11}$$

where the $\{\epsilon_i\}$ are errors on the data ordinates, assumed to behave like 'white-noise'. Then, for any choice of $\mathbf{x}$, we can compute the residuals, $\mathbf{F} = (f_1, f_2, \cdots, f_m)^T$,

$$f_i(\mathbf{x}) = y_i - M(\mathbf{x}, t_i).$$

	$n = 2000$				$n = 4000$				$n = 6000$			
Method	time (s)	fun	out	mid	time (s)	fun	out	mid	time (s)	fun	out	mid
Penalty I												
DL	30.112	44000	14	-	495.18	100000	16	-	1634.39	150000	16	-
DL-CG	0.115	145	14	19	0.124	177	16	24	0.13	177	16	24
TNewton	1.62	2372	12	246	5.34	2489	12	264	5.16	1028	12	105
DL-CG-DS	0.124	232	14	51	0.127	179	16	33	0.152	259	16	65
DL-BA-DS	0.122	165	14	30	0.135	203	16	45	0.148	219	16	45
DL-CG-R(1)	0.128	299	14	42	0.13	258	16	45	0.142	324	16	65
DL-BA-R(1)	0.125	267	14	41	0.131	313	16	50	0.139	322	16	55
DL-BA-R(2)	0.136	385	14	44	0.191	434	16	55	0.156	441	16	60
Matlab	17.35	-	-	-	0.108	-	-	-	155.86	-	-	-
VDF												
DL	103.12	50000	25	-	846.97	104000	25	-	2901.37	162000	25	-
DL-CG	F	F	F	F	F	F	F	F	F	F	F	F
TNewton	2.91	2471	21	257	4.175	1699	21	177	15.76	2369	21	245
DL-CG-DS	F	F	F	F	F	F	F	F	F	F	F	F
DL-BA-DS	F	F	F	F	F	F	F	F	F	F	F	F
DL-CG-R(1)	1.27	908	25	204	1.66	994	25	235	1.77	1148	25	306
DL-BA-R(1)	F	F	F	F	F	F	F	F	F	F	F	
DL-BA-R(2)	F	F	F	F	F	F	F	F	F	F	F	F
Matlab	28.94	-	-	-	149.06	-	-	-	F	-	-	-
BALF												
DL	F	-	-	-	F	-	-	-	F	-	-	-
DL-CG	0.175	106	10	19	0.143	115	10	22	0.123	98	8	18
TNewton	0.98	511	9	57	1.46	438	9	45	2.53	346	8	51
DL-CG-DS	0.137	121	10	27	0.12	119	10	26	0.123	102	8	21
DL-BA-DS	0.134	109	10	21	0.117	109	10	21	0.117	94	8	17
DL-CG-R(1)	0.118	101	10	17	0.121	161	10	27	0.122	137	8	21
DL-BA-R(1)	0.1094	89	10	14	0.111	89	10	14	0.127	122	8	16
DL-BA-R(2)	0.114	139	10	15	0.126	221	10	27	0.125	168	8	18
Matlab	31.8	-	-	-	127.34	-	-	-	320.18	-	-	-
LFFK												
DL	36	10000	4	-	176.15	20000	4	-	583.38	30000	4	-
DL-CG	0.114	37	4	4	0.126	37	4	4	0.113	37	4	4
TNewton	0.385	41	4	4	0.217	41	4	4	0.451	41	4	4
DL-CG-DS	0.114	41	4	4	0.114	41	4	4	0.115	41	4	4
DL-BA-DS	0.113	41	4	4	0.108	41	4	4	0.112	41	4	4
DL-CG-R(1)	0.108	56	4	4	0.113	56	4	4	0.115	56	4	4
DL-BA-R(1)	0.108	56	4	4	0.112	65	4	4	0.117	65	4	4
DL-BA-R(2)	0.11	81	4	4	0.112	81	4	4	0.114	81	4	4
Matlab	18.344	-	-	-	146.36	-	-	-	414.76	-	-	-

Table 1: Performance of DL, DL-CG, TNewton, DL-CG-DS, DL-CG-DS, DL-CG-R(1), DL-BA-R(i) methods and Matlab NLS solver lsqnonlin on solving Penalty I, VDF, BALF and LFFK problems with the size varying from 2000 to 6000 where 'F' represents failure of returning the NLS solution.

	$n = 8000$				$n = 12000$				$n = 15000$			
Method	time (s)	fun	out	mid	time (s)	fun	out	mid	time (s)	fun	out	mid
Penalty I												
DL	407. 163	208000	17	-	12963.54	312000	17	-	O.M.	O.M.	O.M.	O.M.
DL-CG	0.125	165	17	26	0.139	165	17	22	0.71	252	17	40
TNewton	21.97	2405	12	241	21.03	1081	12	109	41.25	755	12	76
DL-CG-DS	0.147	253	17	62	0.142	247	17	44	1.23	439	17	135
DL-BA-DS	0.144	219	17	45	0.154	223	17	56	1.06	359	17	95
DL-CG-R(1)	0.143	318	17	62	0.156	312	17	48	1.18	434	17	105
DL-BA-R(1)	0.15	318	17	53	0.178	444	17	55	1.03	355	17	68
DL-BA-R(2)	0.167	464	17	56	0.191	486	17	48	1.15	527	17	67
Matlab	249.97	-	-	-	19879.39	-	-	-	O.M.	O.M.	O.M.	O.M.
VDF												
DL	7043.54	224000	28	-	23445.22	336000	28	-	O.M.	O.M.	O.M.	O.M.
DL-CG	F	F	F	F	F	F	F	F	F	F	F	F
TNewton	20.47	2411	21	251	38.95	2024	21	210	20.23	607	21	63
DL-CG-DS	F	F	F	F	F	F	F	F	F	F	F	F
DL-BA-DS	F	F	F	F	F	F	F	F	F	F	F	F
DL-CG-R(1)	2.25	1374	28	413	16.08	2408	28	818	5.27	2370	31	999
DL-BA-R(1)	F	F	F	F	F	F	F	F	F	F	F	F
DL-BA-R(2)	F	F	F	F	F	F	F	F	F	F	F	F
Matlab	F	-	-	-	F	-	-	-	O.M.	O.M.	O.M.	O.M.
BALF												
DL	F	-	-	-	F	-	-	-	O.M.	O.M.	O.M.	O.M.
DL-CG	0.122	112	10	21	0.146	112	9	21	0.75	113	9	23
TNewton	5.93	458	9	47	9.84	360	8	38	13.28	274	9	25
DL-CG-DS	0.13	119	10	26	0.134	115	9	19	0.65	95	9	17
DL-BA-DS	0.122	107	10	20	0.126	105	9	19	0.80	102	9	21
DL-CG-R(1)	0.125	159	10	26	0.13	155	9	24	0.84	137	9	21
DL-BA-R(1)	0.114	74	10	14	0.135	125	9	15	0.71	76	9	10
DL-BA-R(2)	0.117	94	10	11	0.146	148	9	12	0.81	98	9	10
Matlab	748.57	-	-	-	14507.28	-	-	-	O.M.	O.M.	O.M.	O.M.
LFFK												
DL	1376.31	40000	4	-	4542.97	60000	4	-	O.M.	O.M.	O.M.	O.M.
DL-CG	0.109	37	4	4	0.111	37	4	4	0.24	37	4	4
TNewton	0.88	41	4	4	1.74	41	4	4	2.50	41	4	4
DL-CG-DS	0.115	41	4	4	0.116	41	4	4	0.23	41	4	4
DL-BA-DS	0.11	41	4	4	0.113	41	4	4	0.25	41	4	4
DL-CG-R(1)	0.111	56	4	4	0.116	56	4	4	0.27	56	4	4
DL-BA-R(1)	0.112	65	4	4	0.117	65	4	4	0.30	65	4	4
DL-BA-R(2)	0.117	81	4	4	0.121	81	4	4	0.74	81	4	4
Matlab	475.36	-	-	-	45189.5	-	-	-	O.M.	O.M.	O.M.	O.M.

Table 2: Performance of DL, DL-CG, TNewton, DL-CG-DS, DL-CG-DS, DL-CG-R(1), DL-BA-R(i) methods and Matlab NLS solver lsqnonlin on solving Penalty I, VDF, BALF and LFFK problems with the size varying from 8000 to 15000 where 'F' represents failure of returning the NLS solution and 'O.M.' represents failure to solving the NLS problem due to running out of memory.

	$n = 2000$				$n = 4000$				$n = 6000$			
Method (n)	time (s)	fun	out	mid	time (s)	fun	out	mid	times(s)	fun	out	mid
DL	36.40	14000	6	-	272.68	28000	6	-	889.63	42000	6	-
DL-CG	1.62	568	6	218	1.64	595	6	287	2.01	616	6	278
TNewton	0.932	633	6	86	3.37	745	6	102	7.83	739	6	103
DL-CG-DS	1.26	448	6	201	1.21	428	6	196	1.42	437	6	203
DL-BA-DS	0.98	364	6	150	1.11	348	6	151	1.36	354	6	152
DL-CG-R(1)	1.25	475	6	172	1.30	467	6	178	1.53	475	6	202
DL-BA-R(1)	0.63	241	6	73	0.68	253	6	79	0.82	281	6	80
DL-BA-R(2)	1.07	410	6	66	1.01	366	6	68	1.34	393	6	71
Matlab	6.95	-	-	-	21.83	-	-	-	54.32	-	-	-

Table 3: Performance of DL, DL-CG, TNewton, DL-CG-DS, DL-CG-DS, DL-CG-R(1), DL-BA-R(i) methods and Matlab NLS solver lsqnonlin on solving data fitting problem with the size varying from 2000 to 6000.

	$n = 8000$				$n = 12000$				$n = 15000$			
Method	time (s)	fun	out	mid	time (s)	fun	out	mid	time (s)	fun	out	mid
DL	2095.86	56000	6	-	6878.98	84000	6	-	O.M.	O.M.	O.M.	O.M.
DL-CG	2.11	596	6	287	3.97	634	6	260	4.54	919	6	395
TNewton	16.75	1047	6	145	25.22	1023	6	147	F	F	F	F
DL-CG-DS	1.61	464	6	209	2.16	492	6	156	3.09	614	6	284
DL-BA-DS	1.61	358	6	156	1.97	360	6	157	2.79	440	6	197
DL-CG-R(1)	1.75	487	6	178	3.72	517	6	183	3.40	661	6	275
DL-BA-R(1)	1.00	285	6	82	1.28	283	6	76	1.65	293	6	95
DL-BA-R(2)	1.55	440	6	76	1.60	356	6	58	2.17	422	6	82
Matlab	98.67	-	-	-	16113.30	-	-	-	O.M.	O.M.	O.M.	O.M.

Table 4: Performance of DL, DL-CG, TNewton, DL-CG-DS, DL-CG-DS, DL-CG-R(1), DL-BA-R(i) methods and Matlab NLS solver lsqnonlin on solving data fitting problem with the size varying from 8000 to 15000 where 'F' represents failure of returning the NLS solution and 'O.M.' represents failure to solving the NLS problem due to running out of memory.

The least squares fit is to determine a minimizer $\mathbf{x}^*$, such that

$$\|\mathbf{F}(\mathbf{x}^*)\|_2 = \min \|\mathbf{F}(\mathbf{x})\|_2.$$

Here, we use these nine methods to solve such a problem with n varying from 2000 to 15000. The vector $\mathbf{x}^\sharp$ is chosen as a random vector with the corresponding size and generate the data points via (4.11). The initial guess, $\mathbf{x}_0$, is another random vector equally distributed on interval $[0, 1]$. The value of t_i is defined as $t_i = 5 + 45i,\ i = 1, \cdots, m$. In this problem, we assume that the number of observation data is 1.25 times of the unknowns, that is $m = 1.25n$. The performance of nine methods is recorded in Table 3 and 4. It shows that our Jacobian-free methods are much more efficient than the common dogleg method and Matlab built-in solver. Furthermore, when the problem size is 15000, the DL and MATLAB methods fail again due to running out of memory. It means that the estimated Jacobian matrix used up all available memory in Matlab. However, our Jacobian-free methods do not encounter any memory difficulties since no Jacobian matrix requires to be computed and stored explicitly. On the other hand, our preconditioner technique works well on this problem. The inner preconditioner with weighted Jacobi method can accelerate the convergence of the BA-

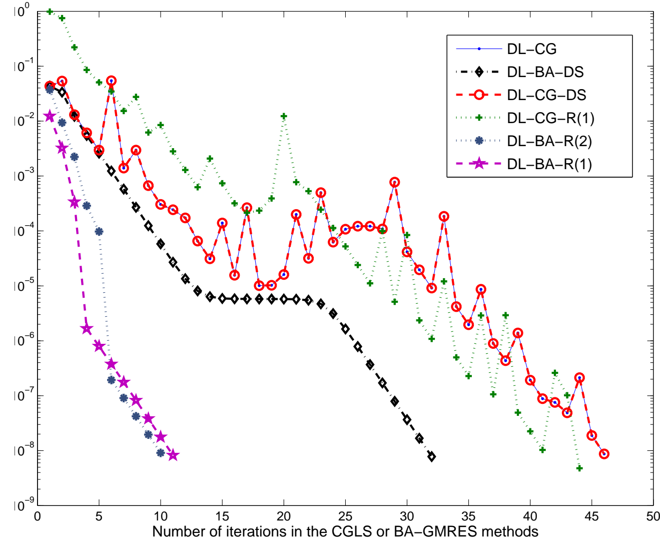

Figure 1: Residual of CGLS and BA-GMRES methods of DL-CG, DL-CG-DS, DL-CG-R(1) and DL-BA-R(i) at the first iteration of the outer level iteration for the data fitting problem.

GMRES method in the middle iteration significantly. As shown in Table 3 and 4, the DL-BA-R(i) methods make a significant savings on the function evaluations at each iteration of the BA-GMRES method. Another observation is that running one step of the weighted Jacobi method is enough for the preconditioning. Although two steps of the weighted Jacobi method can reduce the iterations for the BA-GMRES method a little bit further, the cost of each BA-GMRES iteration increases dynamically. Thus, DL-BA-R(2) requires less BA-GMRES iterations, but takes more computational time than DL-BA-R(1) does.

Figure 1 and 2 plot the residual of the DL-CG, DL-CG-DS, DL-BA-DS, DL-CG-R(1) and DL-BA-R(i) methods in the middle level iteration solving the LS problem at different steps of outer iteration for the data fitting problem. Figure 1 plots the first iteration of the outer level iteration while Figure 2 plots for the fifth iteration. These two figures illustrate that the inner preconditioner via the weighted Jacobi method provides the best performance in outer iterations with the CGLS and BA-GMRES method as the LS solver. The diagonal scaling preconditioner does not work well at the first outer iteration since it is far from the Jacobian matrix. However, its performance is improved at the fifth outer iteration due to its close approximation to the Jacobian matrix. Moreover, these two figures show that BA-GMRES method converges faster than the CGLS method, especially when matrix B is chosen as $B^{(l)}$ in Section 3.2 implicitly through the weighted Jacobi method in the BA-GMRES method.

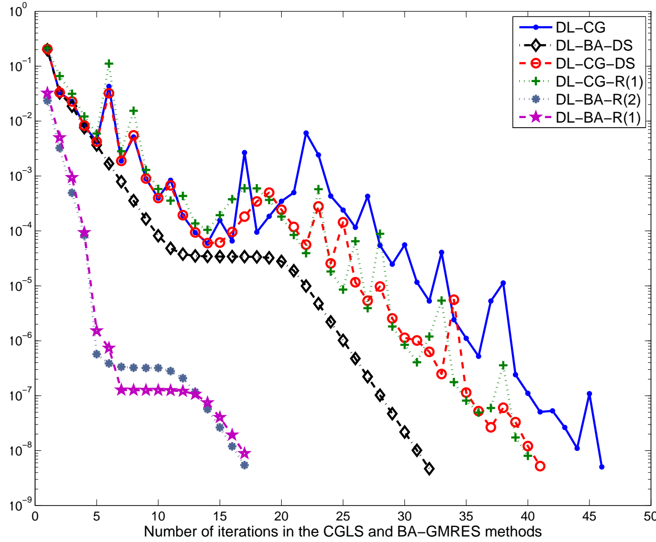

Figure 2: Residual of CGLS and BA-GMRES methods of DL-CG, DL-CG-DS, DL-CG-R(1) and DL-BA-R(i) at the fifth iteration of the outer level iteration for the data fitting problem.

The second data fitting problem is more complicated than the first one. Suppose that there is a set of data points $(t_i,\ y_i)$ on $t_i \in [0,\ 1]$, which satisfies

$$y_i = t_i^{S(\theta)} + \epsilon_i,$$

where $S(\theta)$ is a smooth curve of θ on the interval $\theta \in [0,\ \pi/2]$, but the closed form of $S(\theta)$ itself is unknown. Our goal is to calibrate the smooth curve, $S(\theta)$ on $[0,\ \pi/2]$ from the data set $(t_i,\ y_i)$, $i = 1, \cdots, m$. A commonly used technique for calibration is to select n knots from $[0,\ \pi/2]$, then build a spline curve through these selected knots as an approximation of $S(\theta)$. There are two advantages for this approach. First, due to n smaller than m, the complexity to rebuild the curve is small. Second, if we compute every S_i for each data point $(t_i,\ y_i)$, then the constructed curve from $\{S_i\}$, $i = 1, \cdots, m$, can be very bumpy, which means $\{S_i\}$ could be overfitted.

In this example, we assume that $S(\theta) = \sin\theta,\ \theta \in [0,\ \pi/2]$, and the interval $[0, \pi/2]$ is evenly divided into m subintervals. For each subinterval knots, we compute $S_i = \sin(\theta_i)$, $\theta_i = \pi i/2$ and $y_i = t_i^{S_i} + \epsilon_i$, where $t_i = i/m,\ i = 1, \cdots, m$ and ϵ_i is a random noise of $10^{-2} \cdot N(0, 1)$ and $N(0, 1)$ is the standard normal distribution. In order to calibrate this curve, we take $n+1$ knots on the interval $[0,\ \pi/2]$, $x_j,\ j = 0, \cdots, n$ and $x_j = j\pi/2n$. Then, we try to find the spline curve through all these $n+1$ knots, x_j, such that the curve is the solution of the following NLS problem,

$$\min_{z_j} \frac{1}{2} \sum_{i=1}^{m} \| t_i^{SP(x_j, z_j, t_i)} - y_i \|_2^2,$$

where $SP(x_j, z_j, t_i)$ represents the spline curve value with knots $\{x_j, z_j\}$ at point t_i. In this example, we consider the number of data points, m, varies from 2000 to 15000. Then, we only consider n evenly

	$m = 2000$				$m = 4000$				$m = 6000$			
Method (n)	time (s)	fun	out	mid	time (s)	fun	out	mid	times(s)	fun	out	mid
DL	45.10	1800	9		86.84	3200	8		169.22	4800	8	
DL-CG	24.87	746	9	334	27.07	804	8	377	30.32	771	8	344
TNewton	86.32	2897	9	310	84.31	3173	8	379	157.97	4672	8	431
DL-CG-DS	22.55	684	9	313	28.00	835	8	391	31.13	391	8	369
DL-BA-DS	18.74	554	9	248	20.50	607	8	277	24.46	617	8	282
DL-CG-R(1)	21.55	650	9	281	24.21	727	8	322	29.04	753	8	323
DL-BA-R(1)	17.21	518	9	209	17.45	521	8	213	21.10	541	8	223
DL-BA-R(2)	26.32	798	9	171	28.67	857	8	187	31.22	805	8	174
Matlab	41.98	-	-	-	94.53	-	-	-	220.44	-	-	-

Table 5: Performance of DL, DL-CG, TNewton, DL-CG-DS, DL-CG-DS, DL-CG-R(1), DL-BA-R(i) methods and Matlab NLS solver lsqnonlin on solving curve calibration problem with the size varying from 2000 to 6000.

	$m = 8000$				$m = 12000$				$m = 15000$			
Method	time (s)	fun	out	mid	time (s)	fun	out	mid	time (s)	fun	out	mid
DL	299.06	6400	8	-	903.81	12000	10	-	1192.86	13500	9	-
DL-CG	46.50	915	8	392	85.42	1072	10	441	230.00	2130	9	995
TNewton	267.09	5123	8	487	630.34	8893	10	763	932.89	12023	9	1101
DL-CG-DS	53.10	1041	8	494	106.17	1323	10	628	227.82	2103	9	1021
DL-BA-DS	35.00	695	8	321	67.94	843	10	388	121.69	1123	9	531
DL-CG-R(1)	44.52	879	8	375	84.12	1063	10	478	174.41	1616	9	760
DL-BA-R(1)	29.43	585	8	245	53.85	695	10	287	98.77	937	9	414
DL-BA-R(2)	48.48	957	8	212	86.23	1099	10	240	165.69	1553	9	357
Matlab	442.78	-	-	-	928.73	-	-	-	1379.88	-	-	-

Table 6: Performance of DL, DL-CG, TNewton, DL-CG-DS, DL-CG-DS, DL-CG-R(1), DL-BA-R(i) methods and Matlab NLS solver lsqnonlin on solving curve calibration problem with the size varying from 8000 to 15000.

distributed points on interval $[0,\ \pi/2]$ to build the spline curve, where $n = m/10$. Table 5 and 6 record all the performance results of these nine methods. The inner preconditioner works much better than the diagonal scaling one, especially for the BA-GMRES method. The DL-BA-R(2) requires much less iteration steps to solve the LS problems than DL-BA-R(1), but it requires much more function evaluations. Thus, from the efficiency point of view, DL-BA-R(1) is the most cost efficient method. Although TNewton method does not compute the Hessian matrix explicitly, the cost of computing a Hessian-vector product is much higher than computing a Jacobian-vector product. Thus, it still takes much more time than the Jacobian-free methods. In this curve calibration problem, the TNewton requires almost the same number of function evaluations as the DL method, which constructs the Jacobian matrix explicitly. Therefore, it does not have a big saving in computational times.

Figure 3 and 4 plot the convergence of all dogleg methods in solving the curve calibration problem with the first outer iteration and fifth outer iteration, respectively. Just as we observed in the previous data fitting problem, the dogleg with BA-GMRES method provides the fastest convergence, especially when we take the weighted Jacobi method as preconditioner.

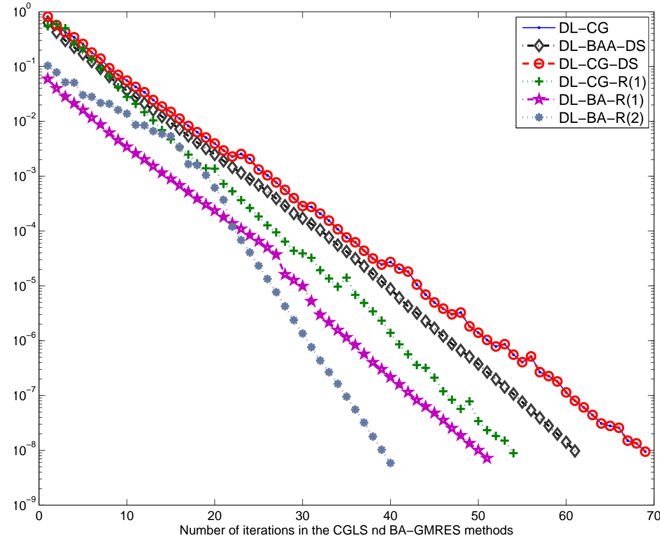

Figure 3: Residual of CGLS and BA-GMRES methods of DL-CG, DL-CG-DS, DL-CG-R(1) and DL-BA-R(i) at the first iteration of the outer level iteration for the curve calibration problem.

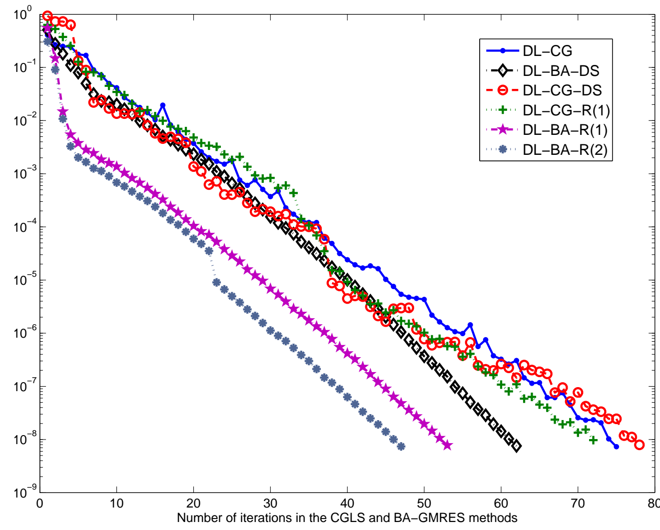

Figure 4: Residual of CGLS and BA-GMRES methods of DL-CG, DL-CG-DS, DL-CG-R(1) and DL-BA-R(i) at the fifth iteration of the outer level iteration for the curve calibration problem.

5 Conclusion

In this paper, we proposed an effective three-level Jacobian free method for solving general NLS problems based on the novel combination of AD and inner-iterative preconditioning ideas. In the outer level, we employ the dogleg/trust region framework for the NLS problem. At each iteration in the outer level, the main cost is to solve a linear LS (LLS) problem. Thus, we adopt the iterative linear LS solver, CGLS or BA-GMRES method, to build up a middle iteration. In order to accelerate the convergence of the iterative LLS solver, we propose an inner iteration preconditioner based on the weighted Jacobi method. Since the Jacobian matrix J_k is not computed and stored explicitly in each outer iteration, the diagonal of $J_k^T J_k$ can be approximated by the formula in [27]. Then, the weighted Jacobi method can be used as an inner preconditioner for the CGLS or BA-GMRES method. Also, the convergence of the weighted Jacobi method is analyzed. Finally, we compare our method with the common dogleg method implementations in Matlab optimization toolbox and immoptibox and the truncated Newton method with the diagonal scaling preconditioner. Numerical results show the superiority of our proposed methods. Due to no Jacobian matrix computed or stored explicitly, our methods can reduce both the computational times and memory requirement significantly. Furthermore, our methods do not rely on the sparsity or structure of Jacobian, gradient and Hessian matrix in the computation. Thus, our methods is applicable to solving any NLS problems. On the other hand, we note that our Jacobian-free method can also be extended to solve nonlinear systems as well.

Acknowledgement. We would like to thank Dr. Keiichi Morikuni for useful discussion and Prof. Thomas Coleman for valuable comments.

References

[1] *www.autodiff.org*, 2012.

[2] Cayuga Research Associates, LLC, "ADMAT-2.0 User's Guide", New York 2009.

[3] S. Bellavia, J. Gondzio and B. Morini, "A matrix-free preconditioner for sparse symmetric positive deffinite systems and least-squares problems", *SIAM J. Sci. Comput.* Vol. 35, A192-A211, 2013.

[4] I. Bongartz, A.R. Conn, N. I. M. Gould and Ph. L. Toint, "CUTE: Constrained and Unconstrained Testing Environment", *ACM Transactions on Mathematical Software*, Vol. 21:1, 123-160, 1995.

[5] *Ȧ*. Björck, "Numerical Methods for Least Squares Problems", *SIAM*, Philadelphia, PA, 1996.

[6] Y.Chen and C. Shen, "A Jacobian-free Newton-GMRES(m) with adaptive preconditioners and its application for power flow calculations", *IEEE Transactions on Power Systems*, Vol.21, 1096-1103, 2006.

[7] J.E. Dennis and R.B. Schnabel, "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", *SIAM*, Philadelphia, PA, 1996.

[8] J. Duintjer Tebbens and M. Tůma, " Efficient Preconditioning of Sequences of Nonsymmetric Linear Systems" , *SIAM Journal on Scientific Computing*, Vol.29, 1918-1941, 2007.

[9] L.C.W. Dixon and R.C. Price, “Truncated Newton method for sparse unconstrained optimization using automatic differentiation”, *J. Optimization Theory and Applications*, Vol.60, 261-275, 1989.

[10] J. Duintjer Tebbens and M. Tůma, “Preconditioner updates for solving sequences of linear systems in matrix-free environment”, *Numerical Linear Algebra and Applications*, vol. 17, 997-1019, 2010.

[11] A. Griewank and A. Walther, “Evaluating Derivatives : Principles, and Techniques of Algorithmic Differentiation”, 2nd Ed., *SIAM*, Philadelphia, PA, 2008.

[12] G.H. Golub and C.F. Van Loan, “Matrix Computations”, 3rd Ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[13] M.K. Hestenes aand E. Stiefel, “Methods of conjugate gradient for solving linear systems”, *J. Research Nat. Bur. Standrds*, Vol.49, 409-436, 1952.

[14] K. Hayami, J-F. Yin and T. Ito, “GMRES methods for least squares problems”, *SIAM J. Matrix and Anl. Appl.*, Vol. 31, 2400-2430, 2010.

[15] A. Imakura, T. Sakurai, K. Sumiyoshi and H. Matsufuru, “An auto-tuning technique of the weighted Jacobi-type iteration used for preconditioners of Krylov subspace methods”, *2012 IEEE 6th International Symposium on Embeded Multicore SoCs*, 183-190, 2012.

[16] D.A. Knoll and D.E. Keyes, “Jacobian-free Newton-Krylov methods: a survey of approaches and applications”, *J. Computational Physics*, Vol.193, 357-397, 2004.

[17] K. Morikuni and K. Hayami, “ Inner-iteratiion Krylov subspace methods for least squares problems”, *SIAM J. Matrix Anal. Appl.*, Vol.34, 1-22, 2013.

[18] S. G. Nash, “Preconditioning of truncated-newton methods”, *SIAM J. Sci. Stat. Comput.* Vol. 6, 599-616, 1985.

[19] S.G. Nash, “A surey of turncated Newton methods”, *J. Comput. App. Maths.* , Vol.124, 45-59, 2000.

[20] H.B. Nielsen, “immoptibox—A MATLAB TOOLBOX FOR OPTIMIZATION”, *IMM*, Technical University of Denmark, 2006. Available at http://www2.imm.dtu.dk/ hbn/immoptibox/

[21] K. Madsen and H.B. Nielsen, “Introduction to Optimization and Data Fitting”, *IMM*, Technical University of Denmark, 2010. Available at http://www2.imm.dtu.dk/pubdb/p.php?5938

[22] M.J.D. Powell, “A hybrid method for nonlinear equations”, *in Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz, ed., Gordon and Breach, London, 87-114, 1970.

[23] Y. Saad and M. Schultz, “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”, *SIMA J. Sci. Stat. Comput.*, Vol. 7, 856-869, 1986.

[24] Y. Saad, “Iterative methods for sparse linear systems”, 2nd ed., *SIAM*, Philadelphia, 2003.

[25] P. Sonneveld and M. B. Van Gijzen, “IDR(s): a family of simple and fast algorithms for solving large nonsymmetric systems of linear equations”, *SIAM J. SCI. COMPUT.*, vol. 31, pp. 1035 C1062, 2008.

[26] A. Van der Sluis, “Condition numbers and equilibration of matrices”, *Numer. Math.*, Vol.14, 14-23, 1969.

[27] W. Xu, T.F. Coleman and G. Liu, A secant method for nonlinear least squares minimization, *J. Comput. Optim. Appl.*, Vol. 51, 159-173, 2012.

[28] W. Xu and T. Coleman, Solving Nonlinear Equations with the Newton-Krylov Method Based on Automatic Differentiation, *Optimization Methods and Software*, Vol. 29, 88-101, 2014

[29] W. Xu and W. Li, Efficient preconditioners for Newton-GMRES method with application to power flow study, *IEEE Trans. Power Systems*, Vol. 28, 4173-4180, 2013.