ISSN 1346-5597

NII Technical Report

Greville's Method for Preconditioning Least Squares Problems

Xiaoke Cui, Ken Hayami

NII-2008-008E
Aug 2008

Greville's Method for Preconditioning Least Squares Problems

Xiaoke Cui [*, 1)]   Ken Hayami [*, †, 2)]

Abstract

In this paper, we construct a preconditioner for least squares problems $\min \|b - Ax\|_2$, where A can be matrices with any shape or rank. The preconditioner itself is a sparse approximation to the Moore-Penrose inverse of the coefficient matrix A. For this preconditioner, we provide theoretical analysis to show that under our assumption, the problem preconditioned by this preconditioner is equivalent to the original problem, and the GMRES method can determine a solution to the preconditioned problem before breakdown happens.

Keywords: Least Squares Problem, Preconditioning, Moore-Penrose Inverse, Greville Algorithm, GMRES

1 Introduction

Consider a least squares problem,

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$.

Assume A is large and sparse, then iterative methods are preferred to use to solve 1.1. In [6], Hayami proposed we can use GMRES [12] to solve least squares problems by using some preconditioners. If we have a preconditioner $B \in \mathbb{R}^{n\times m}$ and we precondition 1.1 from the left, we can transform problem 1.1 to

$$\min_{x\in\mathbb{R}^n} \|Bb - BAx\|_2. \tag{1.2}$$

[*]Department of Informatics, School of Multidisciplinary Sciences The Graduate University for Advanced Studies (Sokendai)

[†]Principles of Informatics Research Division, National Institute of Informatics

[1]Email: xkcui@nii.ac.jp

[2]Email: hayami@nii.ac.jp

On the other hand, we can also precondition problem 1.1 from the right and transform the problem 1.1 to

$$\min_{y\in\mathbb{R}^m} \|b - ABy\|_2. \tag{1.3}$$

Since generally A is rectangular and not necessarily full rank, after preconditioned, the problem might not be equivalent to the original problem 1.1. For this issue, please refer to [6].

In this paper, we use the idea, Approximate Inverse(AINV) Preconditioners [11] which were originally developed for solving large sparse linear systems of the form,

$$Ax = b. \tag{1.4}$$

In this paper, since A is a general matrix, we construct an matrix $M \in \mathbb{R}^{n\times m}$, which is an approximation to the Moore-Penrose inverse [10] of A, and use M to precondition least squares problem 1.1.

The main contribution of this paper is to give out a new way to precondition general least squares problems from the perspective of approximate Moore-Penrose inverse. Our method also the includes RIF preconditioner[1] as a specific case. Hence, from our analysis, we give a better insight to the RIF preconditioner. We also give a theoretical analysis on the equivalence between the preconditioned problem and the original problem, and discuss the possibility of breakdown when using GMRES to solve the preconditioned system.

The rest of the paper is organized as follows. In Section 2, we first introduce some theoretical results about the generalized inverse of rank-one update and the original Greville's method. Based on the Greville's method, we give a global algorithm to construct a preconditioner M which is an approximate generalized inverse of A in Section 3. In Section 4, we rewrite the global method into vector-wise form, and show that for full column rank matrix A, our algorithm is equivalent to the RIF preconditioning algorithm[1]. In Section 6 and Section 7, we prove that under certain assumption, using our preconditioner M, the preconditioned problem is equivalent to the original problem, and the GMRES method can determine a solution to the preconditioned problem before breakdown happens. In Section 8, we consider some details when we implement our algorithms. Numerical results are presented in Section 9. We conclude the whole paper in Section 10. We start with introduce Greville's Method.

2 Greville's Method

Given a rectangular matrix $A \in \mathbb{R}^{m\times n}$, rank(A) = r $\leq$ min{m, n}. Assume the Moore-Penrose inverse of A is known, we are interested in how to compute the Moore-Penrose inverse of

$$A + cd^T, \quad c \in \mathbb{R}^m, \quad d \in \mathbb{R}^n, \tag{2.5}$$

which is a rank-one update of A. In [10], the following six logical possibilities are considered

1. $c \notin \mathcal{R}(A)$, $d \notin \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c$ arbitrary,
2. $c \in \mathcal{R}(A)$, $d \notin \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c = 0$,
3. $c \in \mathcal{R}(A)$, d arbitrary and $1 + d^T A^\dagger c \neq 0$,
4. $c \notin \mathcal{R}(A)$, $d \in \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c = 0$,
5. c arbitrary, $d \in \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c \neq 0$,
6. $c \in \mathcal{R}(A)$, $d \in \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c = 0$.

Here $\mathcal{R}(A)$ denotes the range space of A. For each possibility, an expression for the Moore-Penrose inverse of the rank one update of A is given by the following theorem[10].

Theorem 2.1 *For $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^m$, $d \in \mathbb{R}^n$, let $k = A^\dagger c$, $h = d^T A^\dagger$, $u = (I - AA^\dagger)c$, $v = d^T(I - A^\dagger A)$, and $\beta = 1 + d^T A^\dagger c$. Notice that,*

$$c \in \mathcal{R}(A) \quad \Leftrightarrow \quad u = 0 \tag{2.6}$$
$$d \in \mathcal{R}(A^T) \quad \Leftrightarrow \quad v = 0. \tag{2.7}$$

Then, the generalized inverse of $A + cd^T$ is given as follows.

1. *If $u \neq 0$ and $v \neq 0$, then $(A + cd^T)^\dagger = A^\dagger - ku^\dagger - v^\dagger h + \beta v^\dagger u^\dagger$.*
2. *If $u = 0$ and $v \neq 0$, and $\beta = 0$, then $(A + cd^T)^\dagger = A^\dagger - kk^\dagger A^\dagger - v^\dagger h$.*
3. *If $u = 0$ and $\beta \neq 0$, then $(A + cd^T)^\dagger = A^\dagger + \frac{1}{\beta} v^T k^T A^\dagger - \frac{\bar\beta}{\sigma_1} p_1 q_1^T$, where $p_1 = -\left(\frac{\|k\|_2^2}{\beta} v^T + k\right)$, $q_1^T = -\left(\frac{\|v\|_2^2}{\beta} k^T A^\dagger + h\right)$.*
4. *If $u \neq 0$, $v = 0$ and $\beta = 0$, then $(A + cd^T)^\dagger = A^\dagger - A^\dagger h^\dagger h - ku^\dagger$.*
5. *If $v = 0$ and $\beta \neq 0$, then $(A + cd^T)^\dagger = A^\dagger + \frac{1}{\beta} A^\dagger h^T u^T - \frac{\bar\beta}{\sigma_2} p_2 q_2^T$, where $p_2 = -\left(\frac{\|u\|_2}{\beta} A^\dagger h^T + k\right)$, $q_2^T = -\left(\frac{\|h\|_2^2}{\beta} u^T + h\right)$, and $\sigma_2 = \|h\|_2^2 \|u\|_2^2 + |\beta|^2$.*
6. *If $u = 0$, $v = 0$ and $\beta = 0$, then $(A + cd^T)^\dagger = A^\dagger - kk^\dagger A^\dagger - A^\dagger h^\dagger h + (k^\dagger A^\dagger h^\dagger)kh$.*

To utilize the above theorem, let

$$A = \sum_{i=1}^{n} a_i e_i^T, \tag{2.8}$$

where a_i is the ith column of A. Further let, $A_i = [a_1, \ldots, a_i, 0, \ldots, 0]$. Hence we have

$$A_i = \sum_{k=1}^{i} a_i e_i^T, \quad i = 1, \ldots, n, \tag{2.9}$$

and if we denote $A_0 = 0_{m\times n}$,

$$A_i = A_{i-1} + a_i e_i^T, \quad i = 1, \dots, n. \tag{2.10}$$

Thus every A_i, $i = 1, \dots, n$ is a rank-one update of A_{i-1}. Noticing that $A_0^\dagger = 0_{n\times m}$, we can utilize Theorem 2.1 to compute the Moore-Penrose inverse of A step by step and have $A^\dagger = A_n^\dagger$ in the end.

According to Equation (2.6), we are especially interested in Case 1 and Case 2. Substitute c with a_i and d with e_i, we can rewrite Equation 2.6 as following,

$$a_i \notin \mathcal{R}(A_{i-1}) \quad \Rightarrow \quad u = (I - A_{i-1}A_{i-1}^\dagger)a_i \neq 0, \qquad v = e_i^T(I - A_{i-1}^\dagger A_{i-1}) \neq 0 \tag{2.11}$$

$$a_i \in \mathcal{R}(A_{i-1}) \quad \Rightarrow \quad u = (I - A_{i-1}A_{i-1}^\dagger)a_i = 0, \qquad \beta = 1 + e_i^T A_{i-1}^\dagger a_i = 1. \tag{2.12}$$

Equation (2.12) associates with the columns which are linear combinations of some of the previous columns.

Then from Theorem 2.1, denoting $A_0 = 0_{m\times n}$, we obtain a method to compte $A_i^\dagger$ based on $A_{i-1}^\dagger$ as

$$A_i^\dagger = \begin{cases} A_{i-1}^\dagger + (e_i - A_{i-1}^\dagger a_i)((I - A_{i-1}A_{i-1}^\dagger)a_i)^\dagger & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ A_{i-1}^\dagger + \frac{1}{\sigma_i}(e_i - A_{i-1}^\dagger a_i)(A_{i-1}^\dagger a_i)^T A_{i-1}^\dagger & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}, \tag{2.13}$$

where $\sigma_i = 1 + \|k_i\|_2^2$. This method was proposed by Greville in the 1960s[5].

3 Global Algorithm for General Matrices

In this section, we will construct our preconditioning algorithm according to the Greville's method of section 2. First of all, we notice that the different part between case $a_i \notin \mathcal{R}(A_{i-1})$ and case $a_i \in \mathcal{R}(A_{i-1})$ lies in the second term. If we define f_i and v_i as

$$k_i = A_{i-1}^\dagger a_i, \tag{3.1}$$

$$u_i = a_i - A_{i-1}k_i = (I - A_{i-1}A_{i-1}^\dagger)a_i, \tag{3.2}$$

$$\sigma_i = 1 + \|k_i\|_2^2, \tag{3.3}$$

$$\tag{3.4}$$

$$f_i = \begin{cases} \|u_i\|_2^2 & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ \sigma_i & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}, \tag{3.5}$$

$$v_i = \begin{cases} u_i & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ (A_{i-1}^\dagger)^T k_i & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}, \tag{3.6}$$

we can express $A_i^\dagger$ in a unified form for general matrices, as

$$A_i^\dagger = A_{i-1}^\dagger + \frac{1}{f_i}(e_i - k_i)v_i^T, \tag{3.7}$$

and we have

$$A^\dagger = \sum_{i=1}^{n} \frac{1}{f_i}(e_i - k_i)v_i^T. \tag{3.8}$$

If we denote

$$\begin{aligned} K &= [k_1, \dots, k_n], & (3.9)\\ V &= [v_1, \dots, v_n], & (3.10)\\ F &= \begin{bmatrix} f_1 & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & f_n \end{bmatrix}, & (3.11) \end{aligned}$$

we obtain a matrix factorization of $A^\dagger$ as follows.

Theorem 3.1 *Let $A \in \mathbb{R}^{m\times n}$ and $\operatorname{rank}(A) \le \min\{m, n\}$. Using the above notations, the Moore-Penrose inverse of A has the following factorization*

$$A^\dagger = (I - K)F^{-1}V^T. \tag{3.12}$$

Here I is the identity matrix of order n, K is a strict upper triangular matrix, F is a diagonal matrix, whose diagonal elements are all positive.

If A is full column rank, then

$$\begin{aligned} V &= A(I - K) & (3.13)\\ A^\dagger &= (I - K)F^{-1}(I - K)^T A^T. & (3.14) \end{aligned}$$

PROOF. Denote $\bar{A}_i = [a_1, \dots, a_i]$, then since

$$\begin{aligned} k_i &= A_{i-1}^\dagger a_i & (3.15)\\ &= [a_1, \dots, a_{i-1}, 0, \dots, 0]^\dagger a_i & (3.16)\\ &= \left[\bar{A}_{i-1}, 0, \dots, 0\right]^\dagger a_i & (3.17)\\ &= \begin{bmatrix} \bar{A}_{i-1}^\dagger \\ 0 \end{bmatrix} a_i & (3.18)\\ &= \begin{bmatrix} k_{i,1}, \\ \vdots \\ k_{i,i-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, & (3.19) \end{aligned}$$

$K = [k_1, \ldots, k_n]$ is a strictly upper triangular matrix.

Since $u_i = 0 \Leftrightarrow a_i \in \mathcal{R}(A_{i-1})$,

$$f_i \quad = \quad \begin{cases} \|u_i\|_2^2 & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ \sigma_i & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}. \tag{3.20}$$

Thus $f_i (i = 1, \ldots, n)$, are always positive, which implies that F is a diagonal matrix with positive diagonal elements.

If A is a full rank matrix, we have

$$\begin{aligned} V &= [u_1, \ldots, u_n] && (3.21)\\ &= \Big[(I - A_0A_0^\dagger)a_1, \ \ldots, \ (I - A_{n-1}A_{n-1}^\dagger)a_n\Big] && (3.22)\\ &= [a_1 - A_0k_1, \ \ldots, \ a_n - A_{n-1}k_n] && (3.23)\\ &= A - [A_0k_1, \ \ldots, \ A_{n-1}k_n] && (3.24)\\ &= A - [A_1k_1, \ \ldots, \ A_nk_n] && (3.25)\\ &= A(I - K). && (3.26) \end{aligned}$$

The second from the bottom equality follows from the fact that K is a strictly upper triangular matrix. Now, when $A^\dagger$ is full rank, can be decomposed as follows,

$$A^\dagger = (I - K)F^{-1}V^T = (I - K)F^{-1}(I - K)^T A^T. \qquad \Box \tag{3.27}$$

Remark 1 *From the about proof, it is easy to see that when A is a full column rank matrix, $(I - K)F^{-1}(I - K)^T$ is a LDL^T Decomposition of $(A^TA)^{-1}$.*

Based on Greville's method, we obtain a simple algorithm. We only want to construct a sparse approximation to the Moore-Penrose inverse of A, hence, we perform some numerical droppings in the middle of the algorithm to maintain the sparsity of the preconditioner. We call the following algorithm the **Global** Greville Preconditioning algorithm, since it forms or updates the whole matrix at a time rather than column by column.

Algorithm 1 *Global Greville Preconditioning algorithm*

```
1. set $M_0 = 0$
2. for $i = 1 : n$
3.     $k_i = M_{i-1}a_i$
4.     $u_i = a_i - A_{i-1}k_i$
```

```
5.      if $u_i \neq 0$
6.          $f_i = \|u_i\|_2^2$
7.          $v_i = u_i$
8.      else
9.          $f_i = 1 + \|k_i\|_2^2$
10.         $v_i = M_{i-1}^T k_i$
11.     end if
12.     $M_i = M_{i-1} + \frac{1}{f_i}(e_i - k_i)v_i^T$
13.     perform numerical droppings to $M_i^\dagger$
14. end for
15. Get $M_n \approx A^\dagger$.
```

Remark 2 *In Algorithm 1, actually, we do not need to store k_i, v_i, f_i, $i = 1, \ldots, n$, because we form the $M_i^\dagger$ explicitly.*

Remark 3 *In Algorithm 1, we need to update M_i in every step, but actually we do not need to update the whole matrix, since only the first $i-1$ rows of M_{i-1} could be nonzero. Hence, to compute M_i, we need to update the first $i-1$ rows of M_{i-1}, and then add one new nonzero row to be the ith row.*

Remark 4 *We can also perform numerical droppings to k_i. Thus, if k_i is sparse, when we update M_{i-1}, we only need to update the rows which correspond to the nonzero elements in k_i. Hence, the rank-one update will be very cheap.*

4 Vector-wise Algorithm for General Matrices

If we want to construct the matrix K, F and V without forming M_i explicitly, we can use a vector-wise version of the above algorithm. In Algorithm 1, the column vectors of K are constructed one column at a step, and we compute v_i to compute the diagonal element of F. Hence, it is possible to rewrite Algorithm 1 into a vector-wise form.

Since u_i can be computed from $a_i - A_{i-1}k_i$, which does not refer to M_{i-1} explicitly, to vectorize Algorithm 1, we only need to form k_i and $v_i = M_{i-1}^T k_i$ when linear dependence happens, without using M_{i-1} explicitly.

Consider the numerical droppings are not used. Since we already know that

$$\begin{aligned}
A^\dagger &= (I-K)F^{-1}V^T. \\
&= (I - [\,k_1 \quad \dots \quad k_n\,]) \begin{bmatrix} f_1^{-1} & & \\ & \ddots & \\ & & f_n^{-1} \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_n^T \end{bmatrix} \\
&= \sum_{i=1}^{n} (e_i - k_i) \frac{1}{f_i} v_i^T,
\end{aligned}$$

for any integer p, it is easy to see that

$$A_p^\dagger = \sum_{i=1}^{p} (e_i - k_i) \frac{1}{f_i} v_i^T. \tag{4.1}$$

Therefore, we have

$$\begin{aligned}
v_i &= (A_{i-1}^\dagger)^T k_i \qquad (4.2) \\
&= (\sum_{p=1}^{i-1} (e_p - k_p) \frac{1}{f_p} v_p^T)^T k_i \qquad (4.3) \\
&= \sum_{p=1}^{i-1} \frac{1}{f_p} v_p (e_p - k_p)^T k_i \qquad (4.4)
\end{aligned}$$

and

$$\begin{aligned}
k_i &= A_{i-1}^\dagger a_i \qquad (4.5) \\
&= \sum_{p=1}^{i-1} (e_p - k_p) \frac{1}{f_p} v_p^T a_i \qquad (4.6) \\
&= \sum_{p=1}^{i-2} (e_p - k_p) \frac{1}{f_p} v_p^T a_i + (e_{i-1} - k_{i-1}) \frac{1}{f_{i-1}} v_{i-1}^T a_i \qquad (4.7) \\
&= A_{i-2}^\dagger a_i + (e_{i-1} - k_{i-1}) \frac{1}{f_{i-1}} v_{i-1}^T a_i \qquad (4.8)
\end{aligned}$$

To make this more clear, from the last column of K, the requirement relationship can be shown as

$$\begin{tikzcd}
 & & & k_n = A_{n-1}^\dagger a_n & & & \\
 & A_{n-2}^\dagger a_n \arrow[urr] & & & & k_{n-1} = A_{n-2}^\dagger a_{n-1} \arrow[ull] & \\
A_{n-3}^\dagger a_n \arrow[ur] & & k_{n-2} = A_{n-3}^\dagger a_{n-2} \arrow[ul] & & A_{n-3}^\dagger a_{n-1} \arrow[ur] & & k_{n-2} = A_{n-3}^\dagger a_{n-2} \arrow[ul]
\end{tikzcd}$$

$$\cdots\cdots.$$

In other words, we need to compute every $A_i^\dagger a_k,\ k = i+1,\dots,n$. Denote $A_i^\dagger a_j,\ j > i$ as $k_{i,j}$. In this sense, $k_i = k_{i-1,i}$. In the algorithm, $k_{i,j},\ j > i$ will be stored in the jth column of K, if $j = i+1$, $k_{i,j} = k_j$, and it will not be changed any more. If $j > i+1$, $k_{i,j}$ will be updated to $k_{i+1,j}$ and still stored in the same position.

Based on the above discussion, and add the numerical dropping strategy, we can write the following algorithm. In the algorithm, we omit the first subscript of $k_{i,j}$.

Algorithm 2 *Vector-wise Greville Preconditioning Algorithm*

```
 1. set $K = 0_{n\times n}$
 2. for $i = 1 : n$
 3.     $u = a_i - A_{i-1}k_i$
 4.     if $u \neq 0$
 5.         $f_i = \|u\|_2^2$
 6.         $v_i = u$
 7.     else
 8.         $f_i = \|k_i\|_2^2 + 1$
 9.         $v_i = (A_{i-1}^\dagger)^T k_i = \sum_{p=1}^{i-1} \frac{1}{f_p} v_p (e_p - k_p)^T k_i$
10.     end if
11.     for $j = i+1,\dots,n$
12.         $k_j = k_j + \frac{v_i^T a_j}{f_i}(e_i - k_i)$
13.         perform numerical droppings on $k_j$
14.     end for
15. end for
16. $K = [k_1,\dots,k_n]$, $F = Diag\{f_1,\dots,f_n\}$, $V = [v_1,\dots,v_n]$.
```

If we consider the expression of v_i, we can rewrite Algorithm 2 as the follows.

Algorithm 3 *Vector-wise Greville Preconditioning Algorithm*

```
 1. set $K = zeros(n,n)$
 2. for $i = 1 : n$
 3.     $u = a_i - A_{i-1}k_i$
 4.     if $u \neq 0$
 5.         $f_i = \|u\|_2^2$
```

```
6.          $v_i = u$
7.          for $j = i + 1, \ldots, n$
8.              $k_j = k_j + \frac{v_i^T a_j}{f_i}(e_i - k_i)$
9.              perform numerical droppings on $k_j$
10.         end for
11.     else
12.         $f_i = \|k_i\|_2^2 + 1$
13.         $v_i = (A_{i-1}^\dagger)^T k_i = \sum_{p=1}^{i-1} \frac{1}{f_p} v_p (e_p - k_p)^T k_i$
14.         for $j = i + 1, \ldots, n$
15.             $k_j = k_j + \frac{k_i^T k_j}{f_i}(e_i - k_i)$
16.             perform numerical droppings on $k_j$
17.         end for
18.     end if
19. end for
20. $K = [k_1, \ldots, k_n]$, $F = Diag\{f_1, \ldots, f_n\}$, $V = [v_1, \ldots, v_n]$.
```

Remark 5 *For the full column rank case, we already showed that $V = A(I - K)$. Hence, we do not need to store matrix V in this case. However, it does not mean that for the general case, we need to store the whole matrix V. In fact, we only need to store the vectors of V which correspond to the columns $a_i \in \mathcal{R}(A_{i-1})$. Hence, if the rank deficiency is small, the extra storage is small.*

5 Greville Preconditioning Algorithm and RIF Preconditioner

In this section, we especially take a look at the full column rank case. When A is full column rank, both Algorithm 2 and Algorithm 3 can be simplified as follows.

Algorithm 4 *Vector-wise Greville Preconditioning Algorithm for Full Column Rank Matrices*

```
1. set $K = 0_{n\times n}$
2. for $i = 1 : n$
3.     $u_i = a_i - A_{i-1} k_i$
4.     $f_i = \|u_i\|_2^2$
5.     for $j = i + 1, \ldots, n$
6.         $k_j = k_j + \frac{u_i^T a_j}{f_i}(e_i - k_i)$
7.         perform numerical droppings on $k_j$
```

```
8.      end for
9.  end for
10. $K = [k_1, \ldots, k_n]$, $F = Diag\{f_1, \ldots, f_n\}$.
```

In Algorithm 4,

$$\begin{aligned}
u &= a_i - A_{i-1}k_i \\
&= [a_1, \ldots, a_i, 0, \ldots, 0] \begin{bmatrix} -k_{i,1} \\ \vdots \\ -k_{i,i-1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
&= A_i(e_i - k_i) \\
&= A(e_i - k_i).
\end{aligned}$$

If we denote $e_i - k_i$ as z_i, then $u_i = Az_i$.

The Line 6 in the Algorithm 4, can also be rewritten as

$$\begin{aligned}
k_j &= k_j + \frac{u_i^T a_j}{\|u_i\|_2^2}(e_i - k_i) \\
e_j - k_j &= e_j - k_j - \frac{u_i^T a_j}{\|u_i\|_2^2}(e_i - k_i) \\
z_j &= z_j - \frac{u_i^T a_j}{\|u_i\|_2^2} z_i.
\end{aligned}$$

Denote $d_i = \|u_i\|_2^2$ and $\theta = \dfrac{u_i^T a_j}{d_i}$. Then combining all the new notations, we can rewrite the algorithm as follows.

Algorithm 5

```
1. set $Z = I_{n \times n}$
2. for $i = 1 : n$
3.     $u_i = A_i z_i$
4.     $d_i = (u_i, u_i)$
```

```
5.      for $j = i+1, \dots, n$
6.          $\theta = \frac{(u_i, a_j)}{d_i}$
7.          $z_j = z_j - \theta z_i$
8.      end for
9.  end for
10. $Z = [z_1, \dots, z_n]$, $D = Diag\{d_1, \dots, d_n\}$.
```

Remark 6 *Since $z_i = e_i - k_i$, we have $Z = I - K$. Denoting $D = \mathrm{Diag}\{d_1, \dots, d_n\}$, the factorization of $A^\dagger$ in Theorem 3.1 can be rewritten as*

$$A^\dagger = ZD^{-1}Z^TA^T \tag{5.1}$$

Thus, we can see that Algorithm 5 is exactly the same as the RIF preconditioner, which was proposed based on a A^TA-orthogonalization procedure, by Benzi and Tůma [1].

Theorem 5.1 *For full column rank matrix $A \in \mathbb{R}^{m\times n}$, $\mathrm{rank}(A) = n$, the Greville Preconditioning Algorithms we proposed in this paper are equivalent to the Robust Incomplete Factorization Preconditioning Algorithm proposed in [1].*

6 Equivalence Condition

Consider solving the least squares problem (1.1) by transforming it into the left preconditioned form,

$$\min_{x\in\mathbb{R}^n} \|Mb - MAx\|_2, \tag{6.1}$$

where $A \in \mathbb{R}^{m\times n}$, $M \in \mathbb{R}^{n\times m}$, and b is a right-hand-side vector $b \in \mathbb{R}^m$.

When preconditioning a least squares problem, one important issue is whether the solution of the preconditioned problem is the solution of the original problem. For square nonsingular linear systems, the condition for this equivalence is that the preconditioner M should be nonsingular. Since we are dealing with general rectangular matrices, we need some other conditions to ensure that the preconditioned problem (6.1) is equivalent to the original least squares problem (1.1).

First note the following [6], where $\mathcal{R}(X)$ denotes the range space of matrix X.

Lemma 6.1
$$\|b - Ax^T\|_2 = \min_{x\in\mathbb{R}^n} \|b - Ax\|_2$$
and
$$\|Mb - MAx^T\|_2 = \min_{x\in\mathbb{R}^n} \|Mb - MAx\|_2$$
are equivalent for all $b \in \mathbb{R}^m$, if and only if $\mathcal{R}(A) = \mathcal{R}(M^T MA)$.

If we perform any of Algorithm 1, Algorithm 2 and Algorithm 3 completely and exactly, we will finally have an exact Moore-Penrose inverse of A, i.e. $M = A^\dagger$. However, we need to performe some numerical droppings to control the sparsity of the preconditioner M. Assume the dropping threshold is τ. After dropping the elements in k_i which are smaller than τ, k_i becomes $\tilde{k}_i$. This results in u_i becoming $\tilde{u}_i$, and v_i becoming $\tilde{v}_i$. Hence, the norm of u_i may not be an accurate way to detect if $a_i \in \mathcal{R}(A_{i-1})$ or $a_i \notin \mathcal{R}(A_{i-1})$. We will come back to how to detect the linear dependence later. After droppings, K becomes $\tilde{K}$, F becomes $\tilde{F}$, V becomes $\tilde{V}$, and we have
$$A^\dagger \approx M = (I - \tilde{K})\tilde{F}^{-1}\tilde{V}^T. \tag{6.2}$$

To analyze the equivalence between the original problem (1.1) and the preconditioned problem (6.1), where M is from any of our algorithms, we first consider the simple case, in which A is a full column rank matrix. After numerical droppings, we have,
$$A^\dagger \approx M = (I - \tilde{K})\tilde{F}\tilde{U}^T, \tag{6.3}$$
where $\tilde{U}$ is
$$\tilde{U} = A(I - \tilde{K}). \tag{6.4}$$
Notice that $\tilde{K}$ is a strictly upper triangular matrix and $\tilde{F}$ is a diagonal matrix with positive elements. Hence, we can denote
$$M = CA^T, \tag{6.5}$$
where C is an nonsingular matrix. According to the discussion in [6], we have the following result.

Theorem 6.1 *If $A \in \mathbb{R}^{m\times n}$, and A is full column rank, by Algorithm 1, Algorithm 2 or Algorithm 3 with numerical droppings, we can construct a preconditioner M. With this preconditioner M, the preconditioned least squares problem and the original least squares problem are equivalent and GMRES can determine a least squares solution to the preconditioned problem before breakdown.*

For the general case, we still have $\tilde{K}$ and $\tilde{F}$ nonsingular. However, the expression for $\tilde{V}$ is not straightforward. To simplify the problem, we **assume that there is no zero column in A, and our algorithm can detect all the linear independence correctly**. Hence, if we denote $\{a_{i_1}, a_{i_2}, \ldots, a_{i_r}\}$ to be the maximum linear independent columns set of A, with $1 = i_1 < i_2 < \ldots < i_r$ and there is no other maximum linear independent columns set in $\{a_1, \ldots, a_{i_r}\}$, so that this maximum linear independent columns set is uniquely defined. Then we will have $u_{i_1}, \ldots, u_{i_r}$ such that $\|u_{i_j}\|_2 \neq 0,\ j = 1, \ldots, r$, and $v_{i_j} = u_{i_j},\ j = 1, \ldots, r$. For every u_{i_j}, the following relation still holds.

$$\begin{aligned} u_{i_j} &= a_{i_j} - A_{i_j - 1}\tilde{k}_{i_j} \\ &= A(I - \tilde{K})e_{i_j} \end{aligned}$$

Rewriting the 12th line of Algorithm 1 with numerical droppings, we have

$$M_i = M_{i-1} + \frac{1}{f_i}(e_i - \tilde{k}_i)\tilde{v}_i^T. \tag{6.6}$$

From the above equation, we can see that every row of M_i is a linear combination of the vectors $\tilde{v}_k^T,\ 1 \le k \le i$. For example, the kth row of M_i is a linear combination of $\tilde{v}_1^T, \tilde{v}_2^T, \ldots, \tilde{v}_i^T$. Then we have

$$\mathcal{R}(M_i^T) = \text{span}\{\tilde{v}_1, \ldots, \tilde{v}_i\}. \tag{6.7}$$

Now consider the columns $a_i,\ 1 = i_1 < i \le i_2 \ge 2$. According to our assumption, all the columns $a_i, 1 < i < i_2$ are linearly dependent on a_1.

$$\begin{aligned} \tilde{v}_2 &= M_1^T\tilde{k}_2 \\ &\in \mathcal{R}(M_1^T) \\ &= \text{span}\{\tilde{v}_1\} \\ &= \text{span}\{\tilde{u}_1\}. \end{aligned}$$

From the above relationship,

$$\begin{aligned} \mathcal{R}(M_2^T) &= \text{span}\{\tilde{v}_1, \tilde{v}_2\} \\ &= \text{span}\{\tilde{u_1}\} \\ &= \mathcal{R}(M_1^T), \end{aligned}$$

and similarly,

$$\begin{aligned} \tilde{v}_3 &= M_2^T\tilde{k}_3 \\ &\in \mathcal{R}(M_2^T) \\ &= \mathcal{R}(M_1^T) \\ &= \text{span}\{\tilde{u}_1\}. \end{aligned}$$

Then, in the same way, we have

$$\mathcal{R}(M_{i_2-1}^T) = \text{span}\{\tilde{u}_1\}. \tag{6.8}$$

Hence, generally,

$$\begin{aligned} \mathcal{R}(M_i^T) &= \text{span}\{\tilde{v}_1, \ldots, \tilde{v}_i\} && (6.9) \\ &= \text{span}\{\tilde{u}_{i_1}, \ldots, \tilde{u}_{i_t}\}, && (6.10) \end{aligned}$$

where $i_1, \ldots, i_t \in \{i_1, \ldots, i_r\}$, where $i_t \leq i$ and $i_{\max\{t+1,r\}} \geq i$. Noticing the fact that,

$$\tilde{u}_i = A(I - \tilde{K})e_i, \tag{6.11}$$

we have the following theorem.

Theorem 6.2 *Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, and* $\text{rank}(A) = r$, *and assume that the linear independence is detected correctly by Algorithm 1, Algorithm 2 or Algorithm 3. Then, we have the following relationships, where M is the approximate Moore-Penrose inverse constructed by any of these algorithms,*

$$\begin{aligned} \mathcal{R}(M^T) &= \mathcal{R}(\tilde{V}) & (6.12)\\ &= \text{span}\{\tilde{\text{u}}_{\text{i}_1}, \ldots, \tilde{\text{u}}_{\text{i}_\text{r}}\} & (6.13)\\ &= \mathcal{R}(A). & (6.14) \end{aligned}$$

Combine the above theorem and

Theorem 6.3 *[6] For all $b \in \mathbb{R}^m$, the equation $BAx = Bb$ has a solution, and the solution attains $\min\limits_{x\in\mathbb{R}^n} \|b - Ax\|_2$, if and only if $\mathcal{R}(A) = \mathcal{R}(B^T)$.*

we have the following

Theorem 6.4 *For all $b \in \mathbb{R}^m$, M is constructed by Algorithm 1, Algorithm 2, or Algorithm 3, assume that all the linear independence is detected correctly by any of these algorithms, and used as a left preconditioner, the least squares problem (6.1) is equivalent to the original least squares problem (1.1).*

Remark 7 *About our assumption, we assume that our algorithms can detect all the linear independency in the columns of A. Hence, we allow such mistakes that a linear dependent column is judged as a linear independent column. An extreme case is that we judge all the columns of A as linear independent, in this way, we obtain the RIF preconditioner.*

7 Breakdown Free Condition

In this section we assume without losing generality that the first r columns of A are linear independent. Hence,

$$\mathcal{R}(A) = \text{span}\{a_1, \ldots, a_r\}, \tag{7.1}$$

where $\text{rank}(A) = r$, and $a_i, (i = 1, \ldots, r)$ is the ith column of A. The reason is that we can incorporate a column pivoting in Algorithm 1 easily. With the same assumption as in Theorem 6.2, every time when a linear dependence is detected, we can pivot the current column to the end of the matrix A, and after we have the least squares solution to the pivoted A, we can permute the solution to get the solution to the original problem.

Then we have,

$$a_i \in \mathcal{R}(A_r), \quad i = r+1, \ldots, n. \tag{7.2}$$

In this case, after performing Algorithm 1 with numerical dropping, matrix $\tilde{V}$ can be written in the form

$$\tilde{V} = [\tilde{u}_1, \ldots, \tilde{u}_r, \tilde{v}_{r+1}, \ldots, \tilde{v}_n]. \tag{7.3}$$

If we denote $[\tilde{u}_1, \ldots, \tilde{u}_r]$ as $\tilde{U}_r$, then

$$\tilde{U}_r = A(I - \tilde{K})I_r, \quad I_r = \begin{bmatrix} I_{r\times r} \\ 0 \end{bmatrix}. \tag{7.4}$$

From Equation (6.14)-(6.17), there exists a matrix $\tilde{H}$ such that

$$\begin{aligned} [\tilde{v}_{r+1}, \ldots, \tilde{v}_n] &= \tilde{U}_r \tilde{H} && (7.5) \\ &= A(I - \tilde{K})I_r H, && (7.6) \end{aligned}$$

where $\tilde{H}$ is an appropriate matrix, the size of $\tilde{H}$ is $r \times (n-r)$. It could be singular or nonsingular. Then the whole $\tilde{V}$ is given by

$$\begin{aligned} \tilde{V} &= [\tilde{u}_1, \ldots, \tilde{u}_r, \tilde{v}_{r+1}, \ldots, \tilde{v}_n] && (7.7) \\ &= [\tilde{U}_r, \tilde{U}_r \tilde{H}] && (7.8) \\ &= \tilde{U}_r \,[\, I_{r\times r} \quad \tilde{H} \,] && (7.9) \\ &= A(I - \tilde{K}) \begin{bmatrix} I_{r\times r} \\ 0 \end{bmatrix} [\, I_{r\times r} \quad \tilde{H} \,] && (7.10) \\ &= A(I - \tilde{K}) \begin{bmatrix} I_{r\times r} & \tilde{H} \\ 0 & 0 \end{bmatrix}. && (7.11) \end{aligned}$$

Hence,

$$M = (I - \tilde{K})\tilde{F}^{-1} \begin{bmatrix} I_{r\times r} & 0 \\ \tilde{H}^T & 0 \end{bmatrix} (I - \tilde{K})^T A^T. \tag{7.12}$$

From the above equation, we can also see the difference between the full column rank case and the rank deficient case lies in

$$\begin{bmatrix} I_{r\times r} & 0 \\ \tilde{H}_{r\times n-r} & 0 \end{bmatrix}, \tag{7.13}$$

which should be an identity matrix when A is full column rank.

If there is no numerical dropping, M will be the Moore-Penrose inverse of A,

$$A^\dagger = (I - K)F^{-1} \begin{bmatrix} I_{r\times r} & 0 \\ H^T & 0 \end{bmatrix} (I - K)^T A^T. \tag{7.14}$$

Comparing Equation (7.12) and Equation (7.14), we can have the following theorem.

Theorem 7.1 *Let $A \in \mathbb{R}^{m\times n}$, and* $\operatorname{rank}(A) = r$*. Assume that all the linear independence is detected by Algorithm 1, Algorithm 2 or Algorithm 3. Then the following relationships hold, where M denotes the approximate Moore-Penrose inverse constructed by any of these algorithms,*

$$\begin{aligned} \mathcal{R}(M) &= \mathcal{R}(A^\dagger) && (7.15)\\ &= \mathcal{R}(A^T) && (7.16)\end{aligned}$$

Based on Theorem 6.2 and Theorem 7.1, we have the following theorem which ensures that the GMRES method can determine a solution to the preconditioned problem $MAx = Mb$ before breakdown happens for any $b \in \mathbb{R}^m$.

Theorem 7.2 *Let $A \in \mathbb{R}^{m\times n}$, and* $\operatorname{rank}(A) = r$*. Assume that all the linear independence is detected by Algorithm 1, Algorithm 2 or Algorithm 3. Then using the preconditioner M which is constructed by Algorithm 1, Algorithm 2 or Algorithm 3, GMRES can determine a least squares solution to*

$$\min_{x\in\mathbb{R}^n} \|MAx - Mb\|_2 \tag{7.17}$$

before breakdown happens for all $b \in \mathbb{R}^m$.

Proof. According to Theorem 2.1 in [2] by Brown and Walker, we only need to prove

$$\mathcal{N}(MA) = \mathcal{N}(A^T M^T), \tag{7.18}$$

which is equivalent to

$$\mathcal{R}(MA) = \mathcal{R}(A^T M^T). \tag{7.19}$$

Using the result from Theorem 7.1, there exists a nonsingular matrix T, such that $A = M^T T$. Hence,

$$\begin{aligned} \mathcal{R}(MA) &= \mathcal{R}(MM^T T) && (7.20)\\ &= \mathcal{R}(MM^T) && (7.21)\\ &= \mathcal{R}(M). && (7.22)\end{aligned}$$

On the other hand,

$$\begin{aligned} \mathcal{R}(A^T M^T) &= \mathcal{R}(A^T A T^{-1}) && (7.23)\\ &= \mathcal{R}(A^T A) && (7.24)\\ &= \mathcal{R}(A^T). && (7.25)\end{aligned}$$

The proof is completed using Theorem 7.1. $\square$

Remark 8 *From the proof of the above theorem, we have $\mathcal{R}(MA) = \mathcal{R}(M)$, which means that the preconditioned least squares problem (6.1) is a consistent problem.*

Thus, with Theorem 6.4 and Theorem 7.2, we finally obtain our main result for our Greville preconditioner M.

Theorem 7.3 *Let $A \in \mathbb{R}^{m\times n}$, and* $\mathrm{rank}(A) = r$. *Assume that all the linear independence is detected by Algorithm 1, Algorithm 2 or Algorithm 3, and that the preconditioner M is computed using one of these algorithms. Then, for all $b \in \mathbb{R}^m$, preconditioned GMRES determines a least squares solution of*

$$\min_{x\in\mathbb{R}^n} \|MAx - Mb\|_2 \tag{7.26}$$

before breakdown and this solution attains $\min\limits_{x\in\mathbb{R}^n} \|b - Ax\|_2$.

8 Implementation Consideration

8.1 Detect Linear Dependence

In Algorithm 1, Algorithm 2, and Algorithm 3, one important issue is how to judge the condition $\|u_i\| \neq 0$ in the "if" statement. Simply speaking, we can set up a tolerance τ in advance, and switch to "else" when $\|u_i\|_2 < \tau$. However is this good enough to help us detect the linear dependent columns of A when we perform numerical dropping? To address this issue, we first take a look at the RIF preconditioning algorithm.

The RIF preconditioner was developed for full rank matrices. However, numerical experiments showed it also works for rank deficient matrices. For this phenomenon, our equivalent Algorithm 4 can give a better insight into the RIF preconditioner. Since the only possibility for the RIF preconditioner to breakdown is when $f_i = 0$, which implies that u_i is a zero vector. From our algorithm, we know that

$$\begin{aligned} u_i &= a_i - A_{i-1}k_i & (8.1)\\ &= a_i - A_{i-1}A_{i-1}^{\dagger}a_i & (8.2)\\ &= (I - A_{i-1}A_{i-1}^{\dagger})a_i. & (8.3)\end{aligned}$$

It is clear that u_i is the projection of a_i onto $\mathcal{R}(A_{i-1})^{\perp}$. Hence in exact arithmetic $u_i = 0$ if and only if $a_i \in \mathcal{R}(A_{i-1})$. Our algorithm has an alternative when $a_i \in \mathcal{R}(A_{i-1})$ happens, i.e. when $u = 0$, our algorithm will turn into "else" case. However, this is not always necessary, because of the numerical droppings. With numerical droppings, the u_i is actually,

$$\begin{aligned} u_i &= a_i - A_{i-1}\tilde{k}_i && (8.4)\\ &= a_i - A_{i-1}M_{i-1}a_i && (8.5)\\ &\neq 0. && (8.6) \end{aligned}$$

Hence, even though $a_i \in \mathcal{R}(A_{i-1})$, since u_i will not be the exact projection of a_i onto $\mathcal{R}(A_{i-1})^{\perp}$, the RIF algorithm will not necessarily breakdown when linear dependence happens.

The RIF preconditioner does not take the rank deficient columns or nearly rank deficient columns into consideration. Hence, if we can capture the rank deficient columns, we might be able to have a better preconditioner. Assume the M we compute from any of our three algorithms can be viewed as an approximation to $A^{\dagger}$ with error matrix $E \in \mathbb{R}^{n\times m}$,

$$M = A^{\dagger} + E. \tag{8.7}$$

First note a theoretical result about the perturbation lower bound of the generalize inverse.

Theorem 8.1 *[13] If rank$(A+E) \neq$ rank(A), then*

$$\|(A+E)^{\dagger} - A^{\dagger}\|_2 \geq \frac{1}{\|E\|_2}. \tag{8.8}$$

By Theorem 8.1, if the rank of $M = A^{\dagger} + E$ from our algorithm is not equal to the rank of $A^{\dagger}$, (or A, since they have the same rank), by the above theorem, we have,

$$\begin{aligned} \|M^{\dagger} - (A^{\dagger})^{\dagger}\|_2 &\geq \frac{1}{\|E\|_2} && (8.9)\\ \Rightarrow \|M^{\dagger} - A\|_2 &\geq \frac{1}{\|E\|_2}. && (8.10) \end{aligned}$$

The above inequality says that, if we denote $M^{\dagger} = A + \Delta A$, then $\|\Delta A\|_2 \geq \frac{1}{\|E\|_2}$. Hence, when $\|E\|_2$ is small, which means M is a good approximation to $A^{\dagger}$, M can be an exact generalized inverse of another matrix which is far from A, and the smaller the $\|E\|_2$ is, the far $M^{\dagger}$ from A is. In this sense, if the rank of M is not the same as that of A, M may not be a good preconditioner.

Thus, it is important to maintain the rank of M to be the same of rank(A). Hence, when we perform our algorithm, we need to sparsify the preconditioner M, but at the same time we also want to capture the rank deficient columns as many as possible, and maintain the rank of M. To achieve this, apparently, it is very import to decide how to judge when the exact value $u_i = \|(I - A_{i-1}A_{i-1}^{\dagger})a_i\|_2$ is close to zero or not based on the computed value

$\tilde{u}_i = \|(I - A_{i-1}M_{i-1})a_i\|_2$.

Taking a closer look at $\tilde{u}_i$, we have

$$\begin{aligned} \tilde{u}_i &= a_i - A_{i-1}M_{i-1}a_i && (8.11)\\ &= a_i - A_{i-1}(A_{i-1}^{\dagger} + E_1)a_i && (8.12)\\ &= (a_i - A_{i-1}A_{i-1}^{\dagger}a_i) - A_{i-1}E_1a_i && (8.13)\\ &= u_i - A_{i-1}E_1a_i. && (8.14) \end{aligned}$$

When $a_i \in \mathcal{R}(A_{i-1})$, $u_i = a_i - A_{i-1}A_{i-1}^{\dagger}a_i = 0$. Then,

$$\begin{aligned} \tilde{u}_i &= -A_{i-1}E_1a_i && (8.15)\\ \|\tilde{u}_i\|_2 &\leq \|A_{i-1}\|_F\|a_i\|_2\|E_1\|_F, && (8.16) \end{aligned}$$

If we require E_1 to be small, we can use a tolerance τ_1. If

$$\|\tilde{u}_i\|_2 \leq \tau_1\|A_{i-1}\|_F\|a_i\|_2, \tag{8.17}$$

we suppose we detect a column a_i which is in the range space of A_{i-1}.

Another consideration is,

$$\begin{aligned} \tilde{u}_i &= a_i - A_{i-1}\tilde{k}_i && (8.18)\\ &= a_i - A_{i-1}(k_i + \varepsilon_2) && (8.19)\\ &= (a_i - A_{i-1}k_i) - A_{i-1}\varepsilon_2 && (8.20)\\ &= u_i - A_{i-1}\varepsilon_2. && (8.21) \end{aligned}$$

When $a_i \in \mathcal{R}(A_{i-1})$, $u_i = a_i - A_{i-1}k_i = 0$, then,

$$\begin{aligned} \tilde{u}_i &= -A_{i-1}\varepsilon_2 && (8.22)\\ \|\tilde{u}_i\|_2 &\leq \|A_{i-1}\|_F\|\varepsilon_2\|_2. && (8.23) \end{aligned}$$

If we require ε_2 to be small, we can use a tolerance τ_2. Hence, if

$$\|\tilde{u}_i\|_2 \leq \tau_2\|A_{i-1}\|_F, \tag{8.24}$$

we judge that we have detected a column a_i which is in the range space of A_{i-1}.

8.2 When $m < n$

So far we assume $A \in \mathbb{R}^{m\times n}$, and discussed the left-preconditioning. When $m \geq n$, it is better to perform a left-preconditioning since the size of the preconditioned problem will be smaller. When $m \leq n$, a right-preconditioning will be better. In this subsection we will show that all the results for left-preconditioning can be extended to the right-preconditioning case.

We would like to remark that it is more preferable to perform Algorithm 1, Algorithm 2 or Algorithm 3 to A^T rather than A when $m < n$, based on the following three reasons. By doing so, we construct $\hat{M}$, an approximate generalized inverse of A^T, hence, we can use $\hat{M}^T$ as the preconditioner to the original least squares problem.

1. By taking a look at the Algorithm 1, Algorithm 2 or Algorithm 3, we can find out that we construct the approximate generalized inverse row by row. Hence, we perform a loop which goes through all the columns of A once. When $m \geq n$, this loop is relatively short, however, when $m < n$, this loop could become very long, and the preconditioning will be more time-consuming.

2. Another reason is that, linear dependence will always happen in this case even though matrix A is full row rank. If $m << n$, then when we perform the precondition algorithm on A, a lot of linear dependence should be detected. This fact makes it more difficult to capture the rank deficiency of A, and may result in a bad preconditioner.

3. Even though our algorithms can detect the linear dependence accurately, if we look at the algorithms, for a certain column a_i of A, it is more expensive to deal with than when a_i is independent of the space spanned by the previous columns.

First we consider full row rank case, in which we perform our algorithm on A^T. According to Theorem 3.1, there is a nonsingular matrix C such that $\hat{M} = C(A^T)^T$. Our preconditioner $M = \hat{M}^T$ would be $A^T C^T$, and if we use it as a right preconditioner, combine Lemma 8.1 [6], it is easy to obtain Theorem 8.2.

Lemma 8.1 $\min\limits_{x\in\mathbb{R}^n} \|b - Ax\|_2 = \min\limits_{z\in\mathbb{R}^m} \|b - ABz\|_2$ *holds for all* $b \in \mathbb{R}^m$ *if and only if* $\mathcal{R}(A) = \mathcal{R}(AB)$.

Theorem 8.2 *Let* $A \in \mathbb{R}^{m\times n}$ *and* A *is full row rank, by Algorithm 1, Algorithm 2 or Algorithm 3 with numerical droppings, we can construct an preconditioner* M*. With this preconditioner* M*, the preconditioned least squares problem and the original least squares problem are equivalent and GMRES can determine an least squares solution to the preconditioned problem before breakdown.*

For general case, by using Theorem 6.2, we have

$$\mathcal{R}(\hat{M}^T) = \mathcal{R}(A^T), \tag{8.25}$$

which is saying that,

$$\mathcal{R}(M) = \mathcal{R}(A^T), \tag{8.26}$$

where $M = \hat{M}^T$.

Theorem 8.3 *Let $A \in \mathbb{R}^{m\times n}$, $m \le n$. M is constructed by Algorithm 1, Algorithm 2, or Algorithm 3. Assume that all the linear independence is detected by any of these algorithms, and M is used as a right preconditioner, the least squares problem*

$$\min_{z\in\mathbb{R}^m} \|b - AMz\|_2 \tag{8.27}$$

is equivalent to the original least squares problem (1.1), for all $b \in \mathbb{R}^m$.

And by using Theorem 7.1, we have

$$\mathcal{R}(\hat{M}) = \mathcal{R}(A), \tag{8.28}$$

which is saying that

$$\mathcal{R}(M^T) = \mathcal{R}(A). \tag{8.29}$$

Hence if we use M as a right preconditioner, we can have the following theorem for $m < n$ case.

Theorem 8.4 *Let $A \in \mathbb{R}^{m\times n}$, $m \le n$, and* rank(A) = r. *Assume that all the linear independence is detected by Algorithm 1, Algorithm 2 or Algorithm 3. M is constructed by using one of these algorithms, used as a right preconditioner. Then for all $b \in \mathbb{R}^m$, preconditioned GMRES can determine a least squares solution to*

$$\min_{z\in\mathbb{R}^m} \|AMz - b\|_2 \tag{8.30}$$

before breakdown happens, and this solution attains $\min_{z\in\mathbb{R}^m} \|b - AMz\|_2 = \min_{x\in\mathbb{R}^n} \|b - Ax\|_2$.

9 Numerical Examples

In this section, we use a matrix lp_cycle from the Florida University Sparse Matrices Collection, where zero rows are omitted. Detailed information is given in Table 1.

Table 1: Information on the matrix

Name	m	n	rank	nnz	rank deficiency
A	3371	1890	1875	21234	15

The condition number of A, which is given by $\frac{\sigma_1(A)}{\sigma_n(A)}$, is 1.46×10^7. We construct the preconditioner M and perform the BA-GMRES[6] which is given below.

Algorithm 6 *BA- GMRES*

```
1.  Choose $x_0$
2.  $\tilde{r}_0 = B(b - Ax_0)$
3.  $v_1 = \tilde{r}_0 / \|\tilde{r}_0\|_2$
4.  for $i = 1, 2, \ldots, k$
5.      $w_i = BAv_i$
6.      for $j = 1, 2, \ldots, i$
7.          $h_{j,i} = (w_i, v_j)$
8.          $w_i = w_i - h_{j,i} v_j$
9.      end for
10.     $h_{i+1,i} = \|w_i\|_2$
11.     $v_{i+1} = w_i / h_{i+1,i}$
12.     Find $y_i \in \mathbb{R}^i$ which minimizes $\|\tilde{r}_i\|_2 = \| \|\tilde{r}_0\|_2 e_i - \bar{H}_i y\|_2$
13.     $x_i = x_0 + [v_1, \ldots, v_i] y_i$
14.     $r_i = b - Ax_i$
15.     if $\|A^T r_i\|_2 < \varepsilon$ stop
16. end for
17. $x_0 = x_k$
18. Go to 2.
```

The BA-GMRES is a method that solving least squares problems with GMRES by preconditioning the original problem with a suitable preconditioner B.

In the following example, the right hand side vector b is constructed artificially so that the true solution is all ones vector. In this section, we use

$$\|\tilde{u}_i\|_2 \leq 10^{-6} \|A_{i-1}\|_F \|a_i\|_2, \tag{9.1}$$

the criterion to judge if we need to switch to the "else" case. When the switching tolerance is zero, it implies that we are constructing RIF preconditioners. The stopping rule for GMRES is

$$\|A^T(b - Ax)\|_2 \leq 10^{-8} \cdot \|A^T b\|_2. \tag{9.2}$$

In this example matrix A, we know that the rank deficient columns are,

$$\begin{array}{ccccc} 182 & 184 & 216 & 237 & 253 \\ 717 & 754 & 961 & 1221 & 1239 \\ 1260 & 1261 & 1278 & 1640 & 1859, \end{array} \tag{9.3}$$

15 columns in all. In the following example, we can see that our preconditioning algorithm can detect most of them precisely.

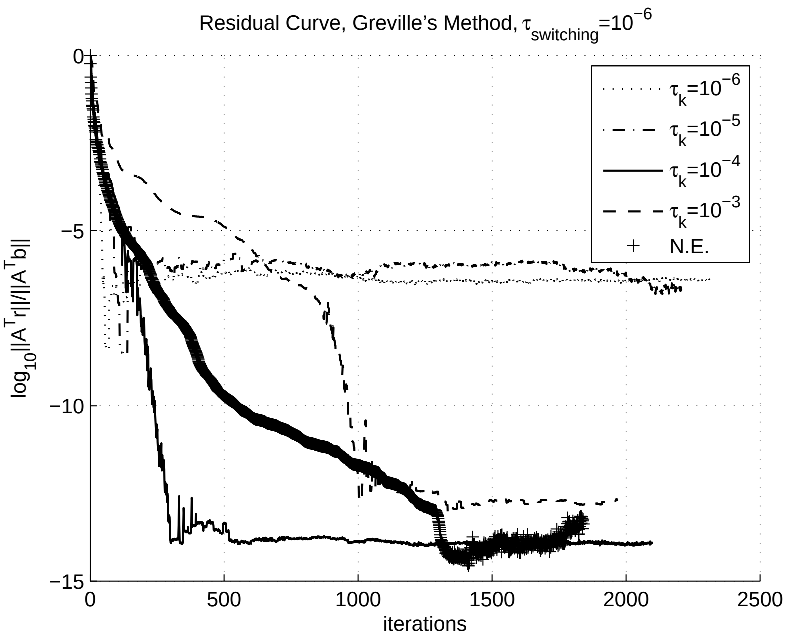
Table 2: Numerical Results

τ_k	nnz in $K,\ F,\ V$	rank(V)	deficiency detected	ITS	Pre. T	Its. T	Tot. T
10^{-1}	156082	NaN	182th	+	3.14	RIF	
	244633	1875	$-1260, -1261, -1278$	1676	7.88	49.20	57.07
10^{-2}	282707	NaN	182th	+	4.00	RIF	
	542066	1875	$-1260, -1261, -1278$	1376	14.52	35.40	49.73
10^{-3}	584959	NaN	182th	+	4.55	RIF	
	1103814	1875	$-1260, -1261, -1278$	905	24.86	18.00	42.86
10^{-4}	742283	NaN	182th	+	4.98	RIF	
	1875272	1875	$-1239, -1278$	204	40.41	2.37	42.78
10^{-5}	76220	NaN	182th	+	5.03	RIF	
	2449916	1873	$+1537, +1545$	111	54.16	1.19	55.34
10^{-6}	788238	NaN	182th	+	5.14	RIF	
	2932859	1873	$+1537, +1545$	56	65.58	0.76	66.34
N.E.				374	0.00	5.94	

In Table 2, we compared the RIF preconditioners and the preconditioners constructed by Algorithm 3. In the last row, we have the result computed by using the GMRES to solving the normal equation $A^T Ax = A^T b$. The rows with "RIF" in column "Its. T" are results of the RIF method. For this problem, the RIF preconditioning algorithm broke down for all the dropping tolerance τ_k. The column "nnz in K, F, V" gives the numbers of nonzero elements in K, F, V, i.e. it is $\text{nnz}(K) + \text{nnz}(F) + \text{nnz}(V)$. The column "rank($V$)" gives the rank of V. According to our analysis before, K and F are always nonsingular. The column "deficiency detected" gives the linear dependent columns detected by the RIF method or our algorithm. -1260 means the 1260th column, which is a rank deficient column is missed by our algorithm. $+1537$ means the 1537th column, which is a linear independent column, is recognized as a rank deficient column by mistake. For RIF precondition, we have "182th" in the table, which means that RIF broke down at 182th column, which is the first linear dependent column in A. For our algorithm, we successfully avoided the breakdown. In this column, we gave the number of linear dependent columns we detected by our algorithm, and gave the specific column numbers in the next rows. For other columns, "ITS" means iteration numbers, "Pre. T" means preconditioning time, "Its. T" means iteration time, and "Tot. T" means total CPU time.

In Table, 2, we found that when $\tau_k = 10^{-1},\ 10^{-2},\ 10^{-3}$, our algorithm detected 12 linear dependent columns, and when $\tau_k = 10^{-4}$, our algorithm detected 13 linear dependent columns. And all these linear dependent columns we detected are correct. Hence, our assumption is satisfied. When $\tau_k = 10^{-5},\ 10^{-6}$, our algorithm gave 17 linear dependent columns, in which 15 of them are correct and 2 of them are wrong. Hence, in this two cases, our algorithm did not detected all the linear independent columns in A, our assumption is not satisfied, which implies the preconditioned problem is not equivalent to the original problem. However, from Table 2, we can see that the GMRES still converged to a good enough approximate solution. From the following figure, we can have a better insight into this situation.

Figure 1: Convergence Curve for Numerical Example 1



From Figure 1 we can see, when the dropping tolerance $\tau_k = 10^{-5}$, 10^{-6}, $\dfrac{\|A^Tr\|_2}{\|A^Tb\|_2}$ reached 10^{-9} first and then went back to level 10^{-6}, and then maintained at the level. This phenomenon illustrates our assumption very well.

10 Conclusion

In this paper, we proposed a new preconditioner for least squares problems. And we showed that when matrix A is full rank, our preconditioning method is the same as the RIF preconditioner[1], and when A is rank deficient, our preconditioners still work while the RIF preconditioner may break down. We proved that under certain assumption, using our preconditioners, the preconditioned problems are equivalent to the original problems. And also under the same assumption, we showed that the GMRES method can determine a solution to the preconditioned problem before breakdown happens. And in the numerical experiment part, our numerical results confirmed our theories.

References

[1] M. Benzi and M. Tůma, A robust preconditioner with low memory requirements for large sparse least squares problems, *SIAM J. Comput.*, Vol. 25, pp. 499-512, 2003.

[2] P. N. Brown and H. F. Walker, GMRES On (nearly) singular system, *SIAM J. Matrix Anal. Appl.,* Vol. 18, pp. 37-51, 1997.

[3] I. S. Duff, R. G. Grimes and J. G. Lewis, Sparse matrix test problems, *ACM Trans. Math. Software*, Vol. 15, pp. 1-14, 1989.

[4] W. J. Duncan, Some devices for the solution of large sets of simultaneous equations (with an appendix on the reciprocation of partitioned matrices), *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, Seventh Series, 35, pp. 660, 1944.

[5] T. N. E. Greville, Some applications of the pseudoinverse of a matrix, *SIAM Review*, Vol. 2, pp. 15-22, 1960.

[6] K. Hayami, J-F Yin, and T. Ito, GMRES methods for least squares problem, *National Institute of Informatics Technical Report*, NII-2007-09E, 2007.

[7] K. S. Riedel, A Sherman Morrison Woodbury Identity for rank Augmenting Matrices With Application to Centering, *SIAM J. Mat. Anal.*, Vol. 12, No. 1, pp. 80-95, 1991.

[8] M. S. Bartlett, An inverse matrix matrix adjustment arising in discriminant analysis, *Annals of Mathematical Statistics*, Vol. 22, p107, 1951.

[9] J. A. Fill and D. E. Fishkind, The Moore-Penrose Generalized Inverse for Sums of Matrices, *SIAM J. Matrix Anal. Appl.*, Vol. 21, pp. 629–635, 1999.

[10] S. L. Campbell and Carl. D. Meyer, Jr, Generalized Inverses of Linear Transformations, *Pitamn Publishing Limited*, 1979.

[11] Y. Saad, *Iterative Methods for Sparse Linear Systems (2nd edition)*, SIAM, Philadelphia, 2003.

[12] Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual method for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.*, Vol. 7, pp. 856-869, 1986.

[13] P.-Å. Wedin, Perturation Theory for Pseudo-inverse, *Bit*, Vol. 13, pp. 217-232, 1973.