Science & Technology Facilities Council

Preprint

RAL-P-2014-007

Preconditioning of linear least-squares problems by identifying basic variables

Arioli M and Duff IS

July 2014

Submitted for publication in SIAM Journal of Scientific Computing

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

ISSN 1361- 4762

Preconditioning of linear least-squares problems by identifying basic variables

Mario Arioli[1] and Iain S. Duff[2]

ABSTRACT

The preconditioning of linear least-squares problems is a hard task. The linear model underpinning least-squares problems, that is the overdetermined matrix defining it, does not have the properties of differential problems that make standard preconditioners effective. Incomplete Cholesky techniques applied to the normal equations do not produce a well conditioned problem. We attempt to remove the ill-conditioning by identifying a subset of rows and columns in the overdetermined matrix defining the linear model that identifies the "best" conditioned basic variables matrix. We then compute a symmetric quasi-definite linear system having a condition number depending solely on the geometry of the non-basic variables and that is independent of the original condition number. We illustrate the performance of our approach on some standard test problems and show it is competitive with other approaches.

Keywords: sparse LU factorization, sparse rectangular matrices, linear least squares, preconditioning, LSQR.

AMS(MOS) subject classifications: 65F05, 65F50

---

[1]mario.arioli@gmail.com, Berlin Mathematical School and TU-Berlin, Germany. This author was supported by the BMS and TU-Berlin as Guest Professor.
[2]iain.duff@stfc.ac.uk, R 18, RAL, Oxon, OX11 0QX, UK. The research of this author was supported in part by the EPSRC Grant EP/I013067/1.

Scientific Computing Department
R 18
Rutherford Appleton Laboratory
Oxon OX11 0QX

July 4, 2014

Contents

1 Introduction

Let $A \in \mathbb{R}^{m \times n}$, with $m \geq n$, be a full rank matrix and $b \in \mathbb{R}^m$, then the linear least-squares problem (LLSP)

$$\min_x ||b - Ax||_2^2 \tag{1.1}$$

has a unique solution $x^* \in \mathbb{R}^n$. The preconditioning of A when the LLSP is not related to differential problems is difficult. An incomplete Cholesky factorization on the normal equations

$$A^T Ax = A^T b, \tag{1.2}$$

is hard to make robust unless we allow fill-in in the Cholesky factor that is comparable with the fill-in of a full Cholesky factor, that is we are very close to using a direct method. Although many people have proposed ways for preconditioning the iterative solution of linear least squares problems [5, 8, 28, 30], the search for a robust preconditioner is still open and it is our intention to take a step in the direction of finding one.

In the following we will explore the use of the augmented system

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{1.3}$$

where r is the residual vector equal to $b - Ax$. In Section 2 we show that by partitioning the rows of A into basic and non-basic parts we can obtain an equivalent symmetric quasi-definite (SQD) linear system with a natural and effective preconditioner. We develop the theory supporting this in Section 3. Although our approach is independent of the conditioning of A we show that it is very much influenced by the partitioning of A and we discuss this in Section 4. In Section 5, we show some experiments with a sparse factorization routine that is able to identify a reasonable approximation to the best partition. We discuss the solution of the SQD system in Section 6. In Section 7, we present several experiments showing the performance of our approach on a range of test problems and compare this with using incomplete Cholesky on the normal equations and show it is competitive with the RIF preconditioner of [5]. We show, in Section 8, the importance of selecting a good basis matrix in the case of totally unimodular matrices although, in this case, we use a different technique for selecting the basis matrix. We present some conclusions in Section 9.

Hereafter, we will denote the condition number of A by:

$$\kappa(A) = \frac{\max_i(\sigma_i)}{\min_i(\sigma_i)},$$

where σ_i are the singular values of A.

Preconditioning least-squares matrices

A fascinating illustration of the dramatic effect of having a good preconditioner can be seen by looking at the (in)famous Vandermonde matrix, with first column of ones and succeeding columns increasing powers of the data points. If we consider a matrix with 20 columns and 10000 rows generated from points chosen randomly from the box (0,1), the condition number of this matrix is $\gtrsim 10^{16}$.

If we define a preconditioning matrix C by choosing uniformly spaced data points then the condition number of AC^{-1} is around 1×10^4. However, if the preconditioning matrix is formed using the Chebyshev points in the interval (0,1) then the condition number is around 5!

We show, in Figure 1.1, histograms for the distribution of these condition numbers over a large sample of matrices whose rows comprise different sets of random points.

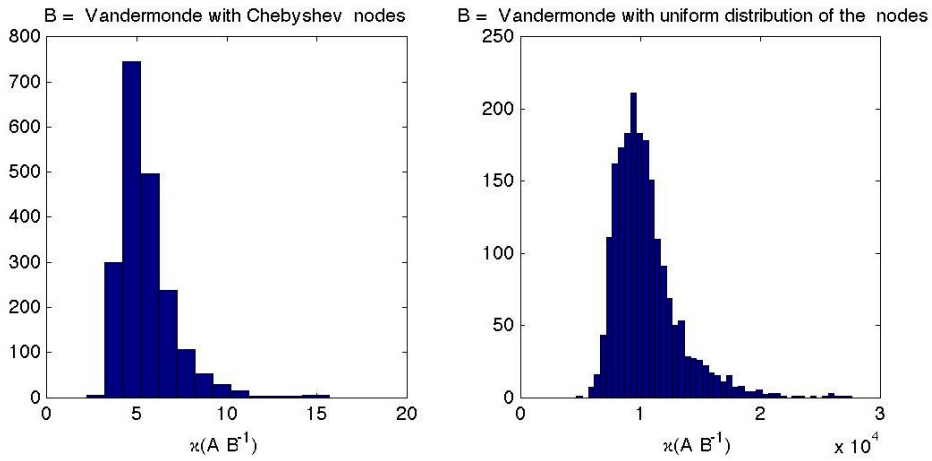

Figure 1.1: Histograms of condition numbers of Vandermonde and preconditioned Vandermonde matrices.

2 Expressing the least-squares problem as a symmetric quasi-definite system

Since the matrix A has full column rank, there exists an invertible matrix B and a permutation matrix P such that

$$PA = \begin{bmatrix} B \\ N \end{bmatrix}. \tag{2.1}$$

This corresponds to partitioning A into basic and non-basic parts.

We can rewrite the augmented system of equations (1.3) using the partitioning (2.1) as

$$\begin{bmatrix} I_n & 0 & B \\ 0 & I_{m-n} & N \\ B^T & N^T & 0 \end{bmatrix} \begin{bmatrix} r_B \\ r_N \\ x \end{bmatrix} = \begin{bmatrix} b_B \\ b_N \\ 0 \end{bmatrix}. \tag{2.2}$$

In passing, we note that setting the (1,1) block of the matrix in equation (2.2) equal to zero, the constraint preconditioner used by [13] is obtained. We observe that if we eliminate the first n-variables, we obtain the reduced system:

$$\begin{bmatrix} I_{m-n} & N \\ N^T & -B^T B \end{bmatrix} \begin{bmatrix} r_N \\ x \end{bmatrix} = \begin{bmatrix} b_N \\ -B^T b_B \end{bmatrix}. \tag{2.3}$$

This linear system has a matrix that is Symmetric Quasi-Definite (SQD) [20, 18, 39, 42], and by symmetrically scaling the system (2.3) by

$$M = \begin{bmatrix} I_{m-n} & 0 \\ 0 & B^{-T} \end{bmatrix} \tag{2.4}$$

we have

$$\begin{bmatrix} I_{m-n} & NB^{-1} \\ B^{-T}N^T & -I_n \end{bmatrix} \begin{bmatrix} r_N \\ Bx \end{bmatrix} = \begin{bmatrix} b_N \\ -b_B \end{bmatrix}. \tag{2.5}$$

As we will be interested in the conditioning of these systems and in the effect of using our preconditioner (2.4) we now study the eigenvalues and singular values of the relevant matrices.

3 Theory

In the following we will use the QR-decomposition of A:

$$A = QR, \qquad Q^TQ = I_n, \quad Q \in \mathbb{R}^{m\times n}, \quad \text{and} \quad R \in \mathbb{R}^n \quad \text{is upper triangular.}$$

We note that this is for theoretical purposes and we do not compute it explicitly in practice. Expressing the QR-decomposition in terms of the partitioning of A in Section 2 we get

$$\left.\begin{aligned} B &= Q_B R \\ N &= Q_N R \end{aligned}\right\}, \tag{3.1}$$

where $Q_B \in \mathbb{R}^{n\times n}$ and $Q_N \in \mathbb{R}^{(m-n)\times n}$ are the basic and non-basic partitions of Q, respectively.

From this, we see that the condition number of NB^{-1} is independent of R and so is independent of the singular values of A. In order to get a more precise expression for the conditioning of NB^{-1}, we will use the following version of the CS-decomposition that is a simplified version of the more general one given in [36]:

Theorem 3.1 *Let $Q \in \mathbb{R}^{m\times n}$ be a matrix such that $Q^TQ = I_n$. Let*

$$Q = \begin{bmatrix} Q_B \\ Q_N \end{bmatrix} \tag{3.2}$$

where Q_B is an $n \times n$ matrix and Q_N is an $(m-n)\times n$ matrix. Then there exist the matrices $U_B \in \mathbb{R}^{n\times n}$, $U_N^{(m-n)\times(m-n)}$, $V \in \mathbb{R}^{n\times n}$, all orthogonal such that

$$\begin{bmatrix} U_B & 0 \\ 0 & U_N \end{bmatrix} \begin{bmatrix} Q_B \\ Q_N \end{bmatrix} V^T = \begin{bmatrix} I_{n-j} & 0 \\ 0 & C \\ 0 & 0 \\ 0 & S \end{bmatrix} \begin{matrix} \} n-j \\ \} j \\ \} m-n-j \\ \} j \end{matrix} \tag{3.3}$$

where $C \in \mathbb{R}^{j\times j}$ and $S \in \mathbb{R}^{j\times j}$ are diagonal matrices $C = diag(c_i)$ and $S = diag(s_i)$ such that

$$c_i^2 + s_i^2 = 1.$$

Moreover, we have, without loss of generality, that $c_i, s_i > 0$ for all i.

Theorem 3.1 tells us that we can simultaneously compute the singular value decomposition of both Q_B and Q_N and that the singular values are the $\cos(\theta_i)$ and the $\sin(\theta_i)$ of the principal angles between the subspaces generated by the two matrices.

In particular, using Theorem 3.1 and (3.1), we can compute the singular value decomposition of NB^{-1}:

$$NB^{-1} = Q_N Q_B^{-1} = U_N^T \begin{bmatrix} 0 & 0 \\ 0 & SC^{-1} \end{bmatrix} U_B. \tag{3.4}$$

The eigenvalues of the matrix

$$\begin{bmatrix} I_{m-n} & NB^{-1} \\ B^{-T}N^T & -I_n \end{bmatrix} \tag{3.5}$$

are related to the singular values of the matrix NB^{-1}. From the expression (3.4), we have

$$\begin{bmatrix} U_N & 0 \\ 0 & U_B \end{bmatrix} \begin{bmatrix} I_{m-n} & NB^{-1} \\ B^{-T}N^T & -I_n \end{bmatrix} \begin{bmatrix} U_N^T & 0 \\ 0 & U_B^T \end{bmatrix} =$$

$$\begin{bmatrix} I_{m-n-j} & 0 & 0 & 0 \\ 0 & -I_{n-j} & 0 & 0 \\ 0 & 0 & I_j & SC^{-1} \\ 0 & 0 & SC^{-1} & -I_j \end{bmatrix}. \tag{3.6}$$

Therefore, the eigenvalues $\mu_i,\ i = 1, \ldots, m$ of the SQD matrix (3.5) are

$$\mu_i = \frac{1}{c_i} \quad i = 1, \ldots, j \quad , \quad \mu_i = \frac{-1}{c_i} \quad i = j+1, \ldots, 2j, \tag{3.7}$$

$$\mu_{2j+1} = 1 \quad \text{with multiplicity} \ \ m - n - j, \tag{3.8}$$

$$\mu_{2j+2} = -1 \quad \text{with multiplicity} \ \ n - j. \tag{3.9}$$

Since the largest singular value of NB^{-1} is equal to the norm $||NB^{-1}||_2$ then, from equations (3.4) and (3.7), the condition number for the preconditioned SQD system (2.5) is given by the expression

$$\sqrt{1 + ||NB^{-1}||_2^2}, \tag{3.10}$$

and we can easily obtain an estimate for this norm.

We can envisage the use of preconditioned Krylov methods such as `MINRES` [38] for the solution of the SQD system (2.5) or, alternatively, a version of the constrained preconditioned conjugate gradient algorithm [11, 12, 13].

Clearly our analysis shows the importance of the choice of the matrix B in our partitioning. We discuss this in detail in the next section.

Before doing so, we should point out that our preconditioning can be interpreted in various ways. In particular, special choices of constraint preconditioners such as those described in [11, 12, 13] applied to (1.3) will produce a preconditioned linear system mathematically equivalent to (2.5) on which the preconditioned conjugate gradient approach proposed in [12] is equivalent to applying the conjugate gradient method directly to (2.5). We refer to [4] for a full review of the possible alternative Krylov methods applicable to SQD systems. However, we will see in Section 6 how a specialized `CGLS` , `LSQR` and `LSMR` for SQD [4, 6, 7, 34, 39] halves the number of iterations of `MINRES` and `CG` applied to the SQD system.

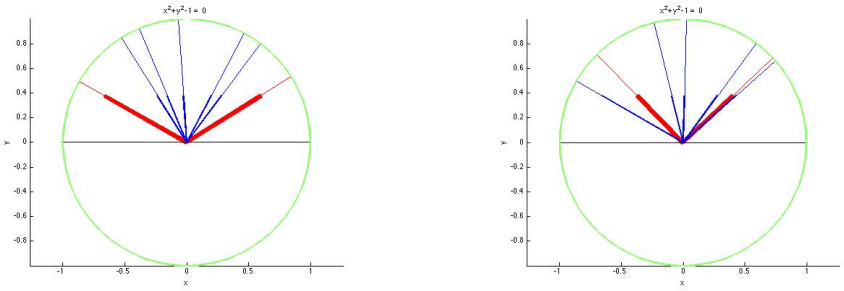

Figure 4.1: Vectors corresponding to rows of matrix. Optimal choice of basis in bold.

4 The choice of the basis B

From the results and the discussion of the previous sections, we can conclude that the best choice of the rows for forming the basic and non-basic partition B and N is independent of the condition number of A and is in reality a combinatorial problem, in the sense that we are choosing n rows from B that best span the space $\mathbb{R}^n$. We could assume that these rows should be close to orthogonal and this can be the case even when the matrix A is very badly conditioned. Taking into account the results of Theorem 3.1, the best choice of B is effectively the best choice of Q_B. This raises the question as to whether a Q_B exists that is close to being orthogonal. First, we point out that if Q_B is orthogonal then, from the orthogonality of Q, follows that $Q_N = 0$, and thus $N = 0$. Therefore, the only non-trivial possibility is to look for the best Q_B such that $||I_n - Q_B^T Q_B||_2 \ll 1$.

Unfortunately, the following example shows that this perfect Q_B may not exist even for an apparently nice matrix. We choose A to be the orthonormal matrix Q viz.

$$Q = \begin{bmatrix} I_{n-1} & 0 \\ 0 & v \end{bmatrix} \tag{4.1}$$

where $v \in \mathbb{R}^{m-n+1}$ and $v_i = 1/\sqrt{m-n+1}$, $\forall i$ so that $v^T v = 1$. For $m \to \infty$ we have

$$Q^T Q = I_n \quad \text{and} \quad \text{Rank}(Q_B) \to n-1.$$

It might be thought that one could devise a simple criterion for choosing the best rows of A for constructing the basis B. To see what this criterion might be, we generated random orthonormal matrices with 7 rows and 2 columns. For each matrix, we exhaustively checked all 21 choices for the 2×2 basis matrix to see which one gave the best conditioned matrix NB^{-1}. We then, using this a posteriori information, tried to understand what criterion could have been used in an a priori way to choose the best basis matrix.

In Figure 4.1 we show the vectors corresponding to the 7 rows of two matrices. As in the discussion in the previous subsection, we might think that it is best to choose the two most orthogonal rows. However, we see from the left-hand graph of Figure 4.1 that it is not the case for the matrix whose rows are represented in the figure. We notice that the best choice of rows for this matrix (shown in bold) are those forming an envelope for the other rows and so we could be tempted into thinking that this might be the best choice of basis. However, the right-hand graph in the figure shows that this is not always the case.

From these results and runs on many other random matrices, our conclusion is that there is no easy criterion to determine the best basis rows.

5 Selection of basis using LU factorization

Given the $m \times n$ matrix A, we look for ways of automatically finding a good set of basis vectors B through a pivoting strategy on A. A standard technique for choosing a pivot in sparse Gaussian elimination is to

choose an entry that minimizes the product of the number of entries in its row with the number in its column. This so-called Markowitz [31] count is a bound on the amount of fill-in that can occur at that step of the elimination. Of course, for stability reasons, the entry so chosen cannot be too small and so we only allow an entry to be eligible as a pivot if it satisfies a threshold pivoting criterion. The standard threshold pivoting strategy for sparse systems is to only choose a_{ij} as pivot if it satisfies

$$|a_{ij}| \geq u \max_k |a_{kj}|,$$

where the threshold parameter u is such that $0 < u \leq 1.0$. Note that values of u near zero allow greater freedom to choose pivots for good maintenance of sparsity while values nearer 1.0 should give a more stable and accurate factorization. Choosing $u = 1.0$ is just the partial pivoting algorithm commonly used for factorizing dense linear systems. For the solution of square sparse linear equations, a threshold value of 0.01 is often recommended and used, but we will see later that a higher value is beneficial when using the LU factorization to choose the basis for a rectangular system.

To identify the basis, we use the HSL [27] package `MC58` that performs a sparse LU factorization and has a wide range of options for sparsity pivoting although it does not keep the factors. When run on rectangular matrices, it will identify which rows and columns are in a nonsingular block of order r, the estimated rank of the matrix, where $r \leq \min\{m, n\}$. Note that this identification would not be possible if we used a sparse QR factorization [10] to find the basis matrix.

We experimented extensively with the range of pivoting options including a sparse threshold variant of rook pivoting where a_{ij} is only eligible to be chosen as a pivot if

$$|a_{ij}| \geq u * \max\{\max_k |a_{kj}|, \max_k |a_{ik}|\}.$$

We had thought that, because this approach is more stable than column threshold pivoting, it might be a better choice for choosing a basis. However, in our extensive experiments (not reported here) we found that the crucial parameter for getting a good basis was the threshold parameter. So much so, that we choose one of the cheapest options in `MC58` for selecting pivots. That is to choose the numerically eligible entry with the best Markowitz count from a search of the sparsest three columns of the reduced matrix.

In order to factorize the basis matrix that we have thus identified, we use `MA48` which is an HSL package for solving sets of sparse linear equations where the matrix is unsymmetric or rectangular. It also uses an LU factorization.

To summarize, we first use `MC58` with a high threshold value, u, on the matrix A to find the rows in B and then perform the factorization of B by `MA48` using threshold pivoting with a threshold value of 0.01. This two step approach has the advantage over earlier work by [33] because we can keep storage requirements low by avoiding the computation of the LU factors while using the high threshold to obtain the basis. We note that while the LUSOL code [19] has a range of pivoting options, it does not have this facility.

We now illustrate experimentally the effect of using this strategy on some standard test problems, popularized by Paige and Saunders that were obtained by Saunders from DSIR Wellington, NZ, and were included in the original Harwell-Boeing test set and in the Florida collection of Tim Davis. We show the main characteristics of these matrices in Table 5.1.

Although they are small, they are widely used in the literature and are challenging enough for our purpose. Indeed, the lack of realistic test matrices is a problem in this area although we do some runs on larger matrices in Section 7.3. Here we use the the Paige and Saunders matrices to comment on a few aspects of our preconditioner without overwhelming the reader with too much data.

We show the condition number of the scaled SQD matrix (3.5) for all the Paige-Saunders matrices and three levels of threshold in Table 5.2. We see that, in every case, the condition number decreases monotonically with increasing u. We will see the effect of this on the convergence of our iterative solution in Section 7.

id	m	n	nnz	$\kappa(A)$
well1033	1033	320	4732	1.6e+02
illc1033	1033	320	4719	1.8e+04
well1850	1850	712	8755	1.1e+02
illc1850	1850	712	8636	1.4e+04

Table 5.1: Dimensions, number of nonzeros and condition numbers for Paige-Saunders tests.

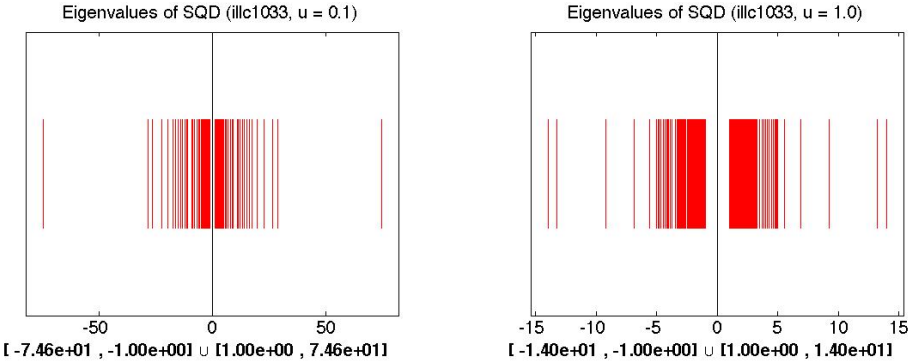

Figure 5.1: Spectra of preconditioned SQD system.

We illustrate the effect of our preconditioning and the dependence on the threshold parameter u in Figure 5.1. In this figure we show the eigenvalues of the SQD matrix (see equation (2.3)) for illc1033 using threshold values of 0.1 and 1.0 to determine the basis matrix B.

	$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$		
id	u=.1	u=.5	u=1.0
illc1033	7.5e+01	2.4e+01	1.4e+01
illc1850	2.0e+03	2.7e+01	2.1e+01
well1033	1.2e+02	2.4e+01	1.3e+01
well1850	8.3e+02	4.7e+01	2.4e+01

Table 5.2: Condition numbers of the preconditioned SQD matrices.

For matrices A with a particular structure, there can be other approaches for obtaining a good basis, and we illustrate this in the case of totally unimodular matrices in Section 8. In this case, we use an approach based on graphs rather than a matrix factorization to choose the basis matrix, but the importance of selecting a good basis and its effect on the conditioning of the SQD system is seen clearly in this case also.

6 Solution of the SQD system

The class of SQD matrices underpins the modelling of optimisation problems. In particular, it is useful and important in the formalization of regularized least-squares problems and hybrid-mixed finite-element methods. We can denote by

$$\mathcal{A} = \begin{bmatrix} I_{m-n} & H \\ H^T & -I_n \end{bmatrix}, \tag{6.1}$$

the matrix (3.5) that is in the class of SQD matrices. Arioli and Orban [4] give a full account of these regularized problems in a more general framework where I_{m-n} and I_n are substituted by general positive definite matrices, and introduce a formal Hilbert space setting, but we keep details to a minimum here for clarity. The matrix $\mathcal{D} = \mathcal{A}^2$ has a two by two block diagonal structure that is the key for implementing efficient Krylov methods, viz.

$$\mathcal{D} = \mathcal{A}^T\mathcal{A} = \mathcal{A}^2 = \begin{bmatrix} I_{m-n} + HH^T & 0 \\ 0 & I_n + H^TH \end{bmatrix}, \tag{6.2}$$

so that

$$\mathcal{A}^{-1} = \mathcal{D}^{-1}\mathcal{A} = \mathcal{A}\mathcal{D}^{-1}. \tag{6.3}$$

Similar relations are satisfied when we replace the diagonal blocks in $\mathcal{A}$ with M and $-N$, where both M and N are positive definite matrices [4]. We have seen in equation (3.7) that the spectrum of $\mathcal{A}$ is symmetric. Fischer [14, Theorem 6.9.9] has proved that Krylov space methods based on $\mathcal{A}$ such as `MINRES` [35] or `CG` display some form of stagnation at every other step (see also [16]) or exhibit oscillations in the energy norm of the error, respectively. In particular, `MINRES` and `CG` will, in exact arithmetic, require double the number of iterations of `LSQR` [37] or the `LSMR` algorithm introduced in [15].

We denote by `GLSMR` and `GLSQR` two algorithms based on the Golub-Kahan bidiagonalization algorithm [21] applied to H. They are generalizations of the `LSMR` and `LSQR` methods that solve

$$\begin{bmatrix} I_{m-n} & H \\ H^T & -I_n \end{bmatrix} \begin{bmatrix} r_N \\ z \end{bmatrix} = \begin{bmatrix} c \\ 0 \end{bmatrix}, \tag{6.4}$$

where, in our case, $H = NB^{-1}$ and the right-hand side c is obtained from r_N and Bx in (2.5) as follows

$$c = b_N - Hb_B$$

and

$$z = Bx - b_B.$$

We point out that the algorithms in exact arithmetic are equivalent to solving the equations

$$\big(H^TH + I_n\big)z = H^Tc, \tag{6.5}$$

by `MINRES` (`GLSMR`) or `CG` (`GLSQR`) methods, respectively.

By using recursive formulae, it is possible to compute iterative updates, at each step k, of both r_N and z in (6.4). Moreover, we can obtain both the solution and an estimate of the error measured in the energy norm, i.e. $||\cdot||_{A^TA}$. We remark that at each step k both `GLSQR` and `LSQR` locally minimise the error in the energy norm. Our stopping criterion uses this local property to obtain, by the use of d extra steps, a lower bound for the global energy norm error at step $k - d$. In our numerical experiments, we use a Matlab implementation of `GLSQR` given in [4, Algorithm 6.1]. We note that a viable alternative

algorithm is CGLS [6, 7] that can be specialised to solve (6.4) using (6.5) without explicitly forming the matrix $H^T H + I_n$. In exact arithmetic, GLSQR on (6.4) and CGLS applied to (6.5) will be equivalent. We emphasize that the aim of our current work is to demonstrate the performance of our preconditioner and not to compare this wide range of iterative solvers. Therefore, we decided to choose GLSQR in order to show how our preconditioner improves the convergence. The difference in the number of iterations between the different iterative solvers is marginal and is mainly due to the different stopping criteria used. We present our results for using GLSQR in Section 7. In these tests, we respect the error measures minimized in GLSQR and compute the relative errors in the $A^T A$-norm given by:

$$(x - x^{(k)})^T A^T A (x - x^{(k)}) / (x^T A^T A x). \tag{6.6}$$

In the GLSQR implementation we compute a lower bound for (6.6) to stop the iterative process with a given tolerance. This stopping criterion is based on the technique described in [4] where an extra d iterations are performed to compute the lower bound estimates for the quantities in equation (6.6).

We point out that a similar preconditioning to ours can be used in all the alternative iterative solvers we have referenced above. Moreover, for both GLSQR and CGLS we can take advantage of the fact that the smallest eigenvalue of (3.5) is equal to 1 in order to have upper bounds for the error in the $A^T A$-norm at each iteration using an extra d steps [4] .

Finally, we note that, with our preconditioner, we are able to compute the condition number of both $H^T H + I$ and of (3.5) using a few steps of a power method taking into account that the smallest singular value of (3.5) is one and the largest is given by (3.10). This will allow us to predict a priori the behaviour of the iterative solver and the quality of the preconditioner. We remark that this is not true for general preconditioners like incomplete Cholesky where the smallest singular value of the preconditioned matrix is unknown.

7 Numerical experiments

As we are want to entertain the possibility of having solutions to the overdetermined systems that yield a nonzero residual, we consider the generation of right-hand sides in Section 7.1. We then show the results from using our preconditioner and running GLSQR on the Paige-Saunders test matrices in Section 7.2 and on larger test matrices in Section 7.3. We discuss in Section 7.4 the solution of the systems using as a preconditioner an incomplete Cholesky factorization of the normal equations and compare our preconditioner with the well known RIF preconditioner [5] in Section 7.5.

7.1 Constructing right-hand sides

In most of the earlier work that we have referenced, for example [5], the right-hand side is computed by multiplying a given solution vector by A, that is the system is consistent. This does not reflect general linear least squares systems where the residual is often nonzero sometimes markedly so. In particular, when the right-hand side is obtained from experimental observations and is a realization of a stochastic variable, the residual, $r = b - Ax$, is also the realization of a stochastic variable with a standard deviation that is much larger than machine precision. In some cases, $||r||_2 \approx \mathcal{O}(10^{-2})$ with large oscillations in the entries.

We note that our method would act as a direct method for consistent equations and would converge in only one iteration and indeed would normally require only a few iterations if the residual was small. Thus we need to have right-hand sides that define inconsistent linear least-squares problems. We do this by using our knowledge of the matrices N and B. We compute the b in (1.1) by using (2.5) as follows:

- We fix $x_i = 1,\ i = 1, \ldots, n$ and $(r_N)_j = 1,\ j = 1, \ldots, m - n$;

- We compute $B * x$ and set

$$b = \begin{bmatrix} Bx - B^{-T}N^T r_N \\ r_N + Nx \end{bmatrix}.$$

We then compare our solution vector with the unit input vector noting that, depending on the condition number of B, we cannot guarantee that this should be small. However, for all our test problems, we assume that such an error will be in the worst case of order 10^{-5} and use this in setting the tolerance to stop the iterative method. For the small test matrices from Paige and Saunders, which come with a given inconsistent right-hand side, we compare our solution with that computed using the QR-method.

7.2 Runs on Paige-Saunders test problems

We give a summary of our results on the Paige-Saunders test problems in Table 7.1. We see that, as the theory predicts, the performance of using `LSQR` on the augmented system and using the `GLSQR` approach are very similar. Although mathematically equivalent, the computation is quite different so we would not expect to get identical results. We do note that, in all cases, `GLSQR` is marginally better in terms of iteration count and final residual. We also see that `MINRES` performs twice the number of iterations as `GLSQR` (we note that the stopping criterion used in `GLSQR` requires a few additional iterations in order to compute reliable estimates of the errors) with marginally worse final residual. We also note that `CGLS` performs similarly to `GLSQR` and the variations in number of iterations are insignificant.

Problem	Aug. System	SQD	
	`LSQR`	`MINRES`	`GLSQR`
`illc1033`	3.9e-13 (95)	6.7e-13 (182)	1.0e-13 (94)
`illc1850`	4.5e-14 (142)	6.2e-14 (270)	1.5e-14 (138)
`well1033`	1.2e-14 (105)	7.3e-15 (204)	4.8e-15 (105)
`well1850`	3.8e-15 (148)	1.5e-14 (285)	1.3e-15 (146)

Table 7.1: Actual errors $||x_{QR} - x^{(k)}||_2/||x_{QR}||_2$, with the number of iterations in parentheses. The matrix B is computed using a threshold of 1.0 and 5 steps are used in the computation for the stopping criterion of `GLSQR` .

To emphasize our point about the convergence of `GLSQR` wrt `MINRES` , we show, in Figure 7.1, convergence curves for `illc1033` with a threshold value of 0.5 used to determine the basis matrix B.

7.3 Runs on larger test problems

As we mentioned earlier, there is a dearth of large least-squares problems in the standard matrix test sets, but we have run our algorithm on a few larger problems and show the effect of our preconditioner on the condition number of the SQD system in Table 7.2. We chose these large problems from the Florida test set with the `Rav4.constraint` matrix coming from an LS-DYNA application and given to us by Cleve Ashcraft. We present our results from running `GLSQR` on these problems in Table 7.3. Although we have performed all the same runs with `GLSMR` as with `GLSQR`, we have not included the results of these since they were very similar to those from `GLSQR` and were essentially identical when the relative error in the solution for each method was calculated. The times in the tables were obtained on a 1.3 GHz Intel Core i5 laptop with 4GB of RAM.

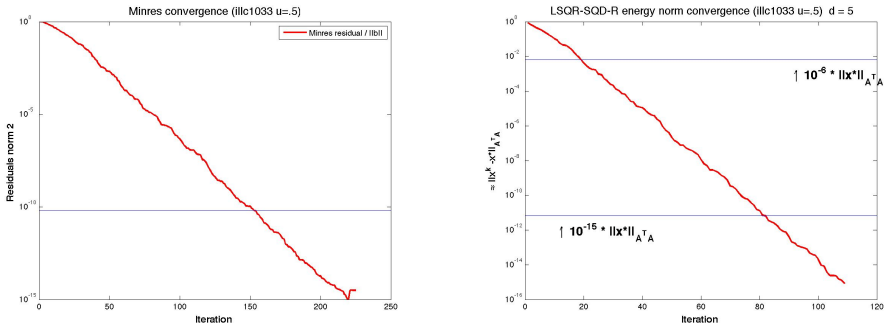

Figure 7.1: Convergence of `MINRES` and `GLSQR` on matrix `illc1033`

Name	m	n	entries	$\kappa(A)$	$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$	
Matrix					u=.1	u=1.0
`C.raw`	2850	1230	5687	8.3e+01	5.7e+1	5.7e+1
`Kemelmacher`	28452	9693	100875	2.5e+04	2.5e+4	7.1e+1
`lp_osa_07`	25067	1118	144812	6.8e+02	9.9e+2	2.5e+2
`lp_osa_30`	104374	4350	604488	1.4e+03	1.4e+3	3.6e+2
`lp_truss_d`	8806	1000	27836	784	174	166
`Rav4.constraint`	2880897	238324	558270	>3.8e+27	2.0e+2	6.9e+2
`mesh_deform`	234023	9393	853829	1.7e+03	7.1e+1	4.0e+1

Table 7.2: Condition numbers of the preconditioned SQD matrices for larger test matrices.

We see that the conditioning of the SQD system is markedly influenced by the threshold chosen in determining the basis and usually directly translates into a lower number of iterations for `GLSQR`. That this is not always the case is because the condition number does not tell the whole story for the convergence of Krylov based methods. It is the distribution of eigenvalues that is important. If the eigenvalues are uniformly distributed, the convergence of the Krylov based method can slow down and can behave as predicted by the Chebyshev method [40].

Tyrtyshnikov and his co-workers have implemented an iterative technique [22] based on an algorithm developed by [23] that tries to maximize the volume of the basis matrix. That is, they iterate to increase the modulus of the determinant of the matrix B. The idea of doing this was in [24] and is related to earlier work by Knuth [29]. We are very grateful to Dmitry Savostyanov for the results shown in Table 7.4, where he refined the basis that we obtained from a threshold of 1.0. We show that this can significantly improve the condition number and convergence.

However, the above problems are not used by earlier papers in this area, and so we also perform runs on problems from pig breeding that were obtained from Markus Hegland [25, 26] for experiments on a QR code at CERFACS [1]. We show the condition numbers and results from running `GLSQR` on these matrices in Tables 7.5 and 7.6, respectively. Again these show clearly the benefit of using our preconditioner and the effect of using a high threshold to determine the basis matrix.

Problem	thresh	Cond. number	GLSQR on SQD		
			Iterations	Rel. error	Time
C.raw	0.1	57.6	117	2.0e-05	0.8
	1.0	57.6	117	2.0e-05	0.6
Kemelmacher	0.1	24616.3	139	7.7e-03	1.7
	1.0	71.2	146	6.2e-05	2.1
lp_osa_07	0.1	994.4	56	2.3e-04	0.6
	1.0	246.0	43	1.6e-04	0.3
lp_osa_30	0.1	1457.5	50	1.1e-04	0.8
	1.0	358.4	34	1.9e-06	0.4
Rav4.constraint	0.1	196.6	356	1.3e-05	53.0
	1.0	69.9	185	3.4e-06	29.0
mesh_deform	0.1	71.2	230	2.3e-05	4.6
	1.0	39.9	152	1.7e-05	2.6

Table 7.3: Results of running GLSQR on preconditioned SQD system on larger systems. Rel error is $||x - x^{(k)}||_{A^TA}/||x||_{A^TA}$ ($u = 1$, $d = 5$).

Basis steps	Cond. number	GLSQR on SQD		
		Iterations	Rel. error	Time
0	71.20	146	7.7e-03	1.7
1229	36.19	71	1.1e-05	1.1
1778	37.13	71	1.3e-05	1.0

Table 7.4: Results of running GLSQR on preconditioned SQD system from Kemelmacher problem with threshold 1.0 using improved basis from Dmitry Savostyanov. Basis steps is number of iterations of algorithm to improve basis.

Name	m	n	entries	$\kappa(A)$	$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$	
Matrix					u=.1	u=1.0
PIGS_small1	3140	1988	8510	3.9e+05	125.4	33.7
PIGS_small2	6280	3976	25530	5.2e+05	115.3	30.4
PIGS_medium1	9397	6119	25013	4.3e+05	384.6	45.1
PIGS_medium2	18794	12238	75039	4.2e+05	317.7	38.5
PIGS_large1	28254	17264	75018	4.6e+05	334.9	93.2
PIGS_large2	56508	34258	225054	4.7e+05	686.7	66.3
PIGS_verylarge	174193	105882	463303	1.5e+06	592.1	193.6

Table 7.5: Condition numbers of the preconditioned SQD matrices for the pig-breeding matrices.

Problem	thresh	Cond. number	GLSQR on SQD		
			Iterations	Rel. error	Time
PIGS_small1	0.1	125.4	114	6.7e-06	0.6
	1.0	33.7	69	5.3e-07	0.4
PIGS_small2	0.1	115.3	173	5.8e-05	1.1
	1.0	30.4	77	6.4e-06	0.5
PIGS_medium1	0.1	384.6	288	4.6e-05	1.8
	1.0	45.1	107	3.7e-07	0.7
PIGS_medium2	0.1	317.7	376	7.2e-04	3.2
	1.0	38.5	108	6.2e-06	0.9
PIGS_large1	0.1	334.9	350	1.3e-04	3.4
	1.0	93.2	140	6.4e-06	1.4
PIGS_large2	0.1	686.7	441	5.2e-03	7.9
	1.0	66.3	153	5.9e-05	2.8
PIGS_verylarge	0.1	592.1	700	3.4e-04	34.0
	1.0	193.6	168	2.9e-05	8.0

Table 7.6: Results of running GLSQR on preconditioned SQD system from pig-breeding matrices. Rel error is $||x - x^{(k)}||_{A^TA}/||x||_{A^TA}$ ($u = 1$, $d = 5$).

7.4 Using incomplete Cholesky on the normal equations

We have run LSQR on the original augmented system (1.3) using the popular preconditioner generated by an incomplete Cholesky method on the normal equations matrix

$$A^T A,$$

using the code ichol supplied in Matlab. For the LP matrices and Kemelmacher, we used the ichol option of an incomplete Cholesky factor with no fill-in. For the Kemelmacher matrix we needed to use the modified incomplete Cholesky option. For both C.raw and mesh_deform, it was only when the drop tolerance value was close to zero that we were able to succeed in completing the incomplete factorization. In this case, the incomplete Cholesky factors were practically the true Cholesky factors of the normal equations matrix. The results in Table 7.7 show that when incomplete Cholesky is working it can do very well but that it has problems on larger problems and is not robust.

We did try to use ichol on the Rav4.constraint matrix but were unable to do so because of the enormous fill-in when forming the normal equations matrix $A^T A$. The construction of $A^T A$ for this test example required several minutes using Matlab on the same laptop as used for the runs in Section 7.3. Although better implementations of incomplete Cholesky can compute an incomplete factorization without forming the normal equations, the storage requirement for the incomplete factorization will be prohibitive. On Rav4.constraint, ichol requires essentially a full Cholesky factorization of the normal equations matrix. This matrix has 269,385,705 entries whereas the LU factorization using MA48 has only 642,577 entries; a factor of 419 less entries! The cost of using ichol as a preconditioner would be consequently that much more.

Problem	Its	$nnz(R)$	$nnz(L)$	$nnz(LU)$
Kemelmacher	5	3620709	72357	186838
lp_osa_07	20	56013	53584	1305
lp_osa_30	22	222970	220544	4539
C.raw	244	17710	17710	2454
mesh_deform	500	10205860	10205860	28405

Table 7.7: Number of nonzeros (nnz) in the full Cholesky factor R, in the factor L produced by ichol of Matlab, and in the LU-factorization of the basis matrix B.

7.5 Comparison with RIF

Perhaps the most referenced and one of the best general purpose preconditioners for least-squares preconditioning is the RIF preconditioner of [5]. We compare the storage for our approach with that for using RIF for the PIGS matrices in Table 7.8. For the RIF preconditioner, it is necessary to select a dropping parameter τ which could be hard to determine a priori. We show runs with several values for this parameter in Table 7.8. The higher values will normally decrease the storage required but will increase the number of iterations for the iterative solver (see Table 7.9), although the increase is certainly not monotonic. For example, for PIGS_verylarge, RIF with a τ value of 0.5 requires 210 iterations while our approach requires 168. However, sometimes RIF requires less iterations on the smaller problems in particular for small values of the drop tolerance τ but, in these cases, RIF requires more storage than our preconditioner. We see that our method is at least competitive and sometimes requires significantly less

storage than RIF. In all the cases where our method requires much more storage it converges much more quickly. We note that when using `GLSQR` to solve (2.5) we need to compute NB^{-1} times a vector and its transpose times a vector at each iteration, that is we do not need A. On the contrary, using the upper triangular matrix R given by RIF, we need to use A in order to compute AR^{-1} times a vector and its transpose times a vector. In Table 7.10, we compare the number of floating-point operations (additions and multiplications) for an iteration using each preconditioner on the `PIGS` matrices:

- complexity for NB^{-1} = number of nonzeros in N plus the number of nonzeros in the factor L and in the factor U produced by `MA48` with pivot threshold of 0.01;
- complexity for AR^{-1} = number of nonzeros in A plus the number of nonzeros in R.

These results show that even when the storage required for the LU factors is higher than that required for the R computed by RIF, the complexity is in favour of our method. In some cases, we require under 50% of the operations required by RIF and our runs using MATLAB require correspondingly half the time although we appreciate that the actual time will depend strongly on the implementation and the machine.

We also note that, in contrast to RIF, we will solve consistent equations in only one iteration and will normally require very few iterations when the residual at the solution is small. Finally, we did not directly compare our method with the other preconditioners discussed in [5] because we accepted from the results in [5] that RIF was the best preconditioner on the `PIGS` matrices.

Problem	SQD	RIF				
		$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$
PIGS_small1	5224	12200	9353	8633	5820	5723
PIGS_small2	16061	4307	4058	3979	3976	3976
PIGS_medium1	15392	37253	28422	26068	17795	17348
PIGS_medium2	47503	13265	12383	12241	12238	12238
PIGS_large1	42261	107015	82289	76097	50093	48324
PIGS_large2	131092	37647	35013	34530	34528	34528
PIGS_verylarge	258582	739982	499582	426833	272210	249333

Table 7.8: Comparison of storage with RIF.

Problem	Num.of iterations					
	SQD	$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$
PIGS_small1	69	57	76	85	80	93
PIGS_small2	77	288	371	349	329	308
PIGS_medium1	107	67	95	105	80	97
PIGS_medium2	108	310	463	466	379	349
PIGS_large1	140	70	103	110	79	105
PIGS_large2	153	340	521	493	401	366
PIGS_verylarge	168	193	172	205	229	210

Table 7.9: Comparison of iterations with RIF for an energy norm stopping criterion tolerance = 10^{-8}.

Problem	NB^{-1}	AR^{-1}				
		$\tau = 0.1$	$\tau = 0.2$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$
PIGS_small1	8680	20710	17863	17143	14330	14233
PIGS_small2	26098	29837	29588	29509	29506	29506
PIGS_medium1	25226	62266	53435	51081	42808	42361
PIGS_medium2	75790	88304	87422	87280	87277	87277
PIGS_large1	75231	182033	157307	151115	125111	123342
PIGS_large2	225694	262701	260067	259584	259582	259582
PIGS_verylarge	463785	1203285	962885	890136	735513	712636

Table 7.10: Comparison of complexity between NB^{-1} (SQD) and AR^{-1} (RIF) .

8 Totally unimodular matrices

We now discuss the selection of the basis matrix for a class of totally unimodular matrices. In this case we exploit a characteristic of the structure to avoid having to use a sparse LU factorization. For totally unimodular matrices, the determinant of every square submatrix is either 0, or ± 1. An important example of these are incidence matrices that arise in several fields including optimisation on networks and mixed finite-element methods [9]. Several algorithms using network programming techniques for the solution of Darcy's problems by finite-element methods are described in [2, 3].

The problem of identifying a suitable basis matrix B for this special class of incidence matrices of undirected graphs can be solved by simple heuristics based on the identification of spanning trees [32]. The incidence matrix A of a graph $\mathcal{G}$ with m edges and n vertices (we assume that $m > n$) is a totally unimodular matrix [32] with m rows and n columns. Its entries are $-1, 0, 1$ and in each row there are two nonzero entries corresponding to the two nodes identifying an edge. We can assume without loss of generality that one entry is equal to 1 and the other is equal to -1, that is we fix a direction for the edge. Each column j of A has a number of nonzero entries corresponding to the nodes connected to j by an edge in $\mathcal{G}$. These matrices have rank equal to $n - 1$ with

$$\mathrm{Ker}(A) = \{x : \text{such that } \; x = \rho e, \; \rho \in \mathbb{R}, \; e^T = (1, \ldots, 1)\}.$$

However, because of the elementary structure of the kernel, we easily reduce problem (1.1) to one of full rank by fixing one node arbitrarily (the "*root*") and removing the corresponding column. The resulting matrix is still totally unimodular. Denoting by $\widetilde{A}$ the matrix formed by the remaining $(n-1)$ columns, a heuristic that can be used to select the best basis matrix, $\widetilde{B}$, is to determine the rooted spanning tree that has the smallest height $\mathbf{h}$, where the tree height is the longest path from the root to a leaf node. A possible choice is to compute the shortest path tree [32, 41] and several low-complexity algorithms are available to do this (see [17] for a elegant presentation). Different choices of roots (thus different columns to remove) will result in different spanning trees and so different matrices $\widetilde{B}$. In each case the matrix $\widetilde{A}$ will be partitioned as $\begin{pmatrix} \widetilde{B} \\ \widetilde{N} \end{pmatrix}$.

The matrix $\widetilde{B} \in \mathbb{R}^{(n-1)\times(n-1)}$, representing the spanning tree, can be permuted to lower triangular form. The number of nonzeros (they are ± 1) in each row of $\widetilde{N}\widetilde{B}^{-1}$ is the length of the shortest circuit that can be formed using the edge corresponding to the row in $\widetilde{N}$ and edges in the tree. In the worst case, the number of nonzeros is twice the height of the spanning tree. Therefore, we have

$$\|\widetilde{N}\widetilde{B}^{-1}\|_\infty \leq 2\mathbf{h}.$$

Table 8.1 shows that the preconditioner does a good job of reducing the condition number and that choosing the root node to obtain a short bushy tree in general leads to a better preconditioned SQD system. In every case, the infinity norm of NB^{-1} is bounded by twice the tree height. As this is only an upper bound, a taller tree can give a lower norm as we can see in the runs on the geo matrices, but this is the only case on which we saw this happening.

Test problem class pref

(m,n)	Tree height		$\|\|NB^{-1}\|\|_\infty$		$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$		$\kappa(A)$
(m,n)	SPTs	SPTl	SPTs	SPTl	SPTs	SPTl	
(9975, 5000)	6	8	9.0	14.0	46.3	65.3	1597.1
(19965,10000)	6	9	10.0	16.0	56.8	141.2	2275.4

Test problem class smallw

(m,n)	Tree height		$\|\|NB^{-1}\|\|_\infty$		$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$		$\kappa(A)$
(m,n)	SPTs	SPTl	SPTs	SPTl	SPTs	SPTl	
(10499, 5000)	21	34	42.0	67.0	55.9	91.0	325.0
(20982,10000)	23	35	44.0	69.0	62.4	144.5	465.4

Test problem class kleinberg

(m,n)	Tree height		$\|\|NB^{-1}\|\|_\infty$		$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$		$\kappa(A)$
(m,n)	SPTs	SPTl	SPTs	SPTl	SPTs	SPTl	
(14906, 5000)	12	15	24.0	27.0	73.8	89.1	344.7
(29625,10000)	14	17	28.0	33.0	109.1	111.0	565.7

Test problem class geo

(m,n)	Tree height		$\|\|NB^{-1}\|\|_\infty$		$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$		$\kappa(A)$
(m,n)	SPTs	SPTl	SPTs	SPTl	SPTs	SPTl	
(11013, 5000)	83	156	86.0	67.0	31.6	23.3	956.8
(22384,10000)	173	338	136.0	173.0	33.2	52.7	833.9

Test problem class shar_te2-b1

(m,n)	Tree height		$\|\|NB^{-1}\|\|_\infty$		$\sqrt{1+\|\|NB^{-1}\|\|_2^2}$		$\kappa(A)$
(m,n)	SPTs	SPTl	SPTs	SPTl	SPTs	SPTl	
(17160, 286)	2	2	4.0	4.0	97.0	97.0	371.7

Table 8.1: SPTs: root giving tree of least height; SPTl: root for which we have the longest path to a node.

9 Conclusions

We have studied the almost paradoxical nature of conditioning and preconditioning in sparse linear least-squares problem, analysed carefully a particular form of preconditioning, and shown its efficacy on standard test problems. We have shown that a major aspect is to identify a suitable set of rows from the overdetermined system to constitute the basis matrix. One way that has proved successful over a range of problems has been to determine the basis matrix through a sparse threshold pivoting factorization of the least-squares matrix. We have also shown that, for particular matrix structures, other approaches can be used to get the basis matrix but the importance and influence of choosing a good basis matrix is still evident.

We showed that the use of a preconditioned SQD system is a competitive way of solving the least-squares problem. In this paper, we have assumed that the matrix A is full column rank. There are many cases in real applications where this is not so, and we plan to extend our approach to these cases, perhaps using versions of the algorithms of Savostyanov and his colleagues [22].

Acknowledgements

We would like to thank our colleagues at RAL, Tyrone Rees, John Reid, and Jennifer Scott for comments on an earlier draft of this paper, and Åke Björck and Mike Saunders for several helpful discussions. We also thank Mirek Tůma for his great help with using the RIF code.

References

[1] P. R. Amestoy, I. S. Duff, and C. Puglisi, *Multifrontal QR factorization in a multiprocessor environment*, Numerical Linear Algebra with Applications, 3 (1996), pp. 275–300.

[2] M. Arioli and G. Manzini, *Null space algorithm and spanning trees in solving Darcy's equation*, BIT Numerical Mathematics, 43 (2003), pp. 839–848.

[3] ——, *A network programming approach in solving Darcy's equations by mixed finite-element methods*, Electronic Transactions on Numerical Analysis, (2006), pp. 41–70.

[4] M. Arioli and D. Orban, *Iterative methods for symmetric quasi-definite linear systems part i: Theory*, Tech. Rep. Cahier du GERAD G-2013-32, GERAD, 2013.

[5] M. Benzi and M. Tuma, *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM Journal on Scientific Computing, 25 (2003), pp. 499–512.

[6] Å. Björck, *Conjugate gradient method for symmetric quasi-definite linear systems.* Private communication.

[7] ——, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia,, 1996.

[8] Å. Björk and J. Yuan, *Preconditioners for least squares problems by LU factorization*, Electron. Trans. Numer. Anal., 8 (1999), pp. 26–35.

[9] F. Brezzi and M. Fortin, *Mixed and Hybrid Finite Element Methods*, vol. 15, Springer-Verlag, Berlin, 1991.

[10] T. A. Davis, *Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse qr factorization*, Transactions of the ACM on Mathematical Software, 38 (2011), pp. xxx–yyy.

[11] H. S. Dollar, *Constraint-style preconditioners for regularized saddle-point problems*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 672–684.

[12] H. S. Dollar, N. I. M. Gould, W. H. A. Schilders, and A. J. Wathen, *Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 170–189.

[13] H. S. Dollar and A. J. Wathen, *Approximate factorization constraint preconditioners for saddle-point matrics*, SIAM Journal of Scientific Computing, 27 (2006), pp. 1555–1572.

[14] B. Fischer, *Polynomial Based Iteration Methods for Symmetric Linear Systems*, Society for Industrial and Applied Mathematics, Philadephia, PA, 2011.

[15] D. C.-L. Fong and M. A. Saunders, *LSMR: An iterative algorithm for sparse least-squares problems*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2950–2971.

[16] R. W. Freund, G. H. Golub, and N. M. Nachtigal, *Iterative solutions of linear systems*, ACTA-NUMERICA, (1991), pp. 1–44.

[17] G. Gallo and S. Pallottino, *Shortest path methods: a unifying approach*, Mathematical Programming Study, 26 (1986), pp. 38–64.

[18] A. George, K. Ikramov, and A. B. Kucherov, *Some properties of symmetric quasi-definite matrices*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1318–1323.

[19] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algprithm for large-scale constrained optimization*, SIAM Review, 47 (2005), pp. 99–131.

[20] P. E. Gill, M. A. Saunders, and J. R. Shinnerl, *On the stability of Cholesky factorization for symmetric quasidefinite systems*, SIAM Journal on Optimization, 17 (1996), pp. 35–46.

[21] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM Journal on Numerical Analysis, 2 (1965), pp. 205–224.

[22] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *How to find a good submatrix*, in Matrix Methods: Theory, Algorithms, Applications, V. Olshevsky and E. Tyrtyshnikov, eds., World Scientific, Hackensack, NY, 2010, pp. 247–256.

[23] S. A. Goreinov and E. E. Tyrtyshnikov, *Maximal-volume concept in approximation by low-rank matrices*, Contemporary Mathematics, 208 (2001), pp. 47–51.

[24] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, *A theory of pseudoskeleton approximations*, LINEAR ALGEBRA AND ITS APPLICATIONS, 261 (1997), pp. 1–21.

[25] M. Hegland, *On the computation of breeding values*, in Proceedings of CONPAR 90VAPP IV Joint International Conference on Vector and Parallel Processing, H. Burkhart, ed., vol. 457 of Lecture Notes in Comput. Sci, Berlin, 1990, SpringerVerlag, pp. 232–242.

[26] ——, *Description and use of animal breeding data for large least squares problems*, Tech. Rep. TR/PA/93/50, CERFACS, Toulouse, France, 1993.

[27] HSL, *HSL 2013: A collection of Fortran codes for large scale scientific computation*, 2013. `http://www.hsl.rl.ac.uk`.

[28] A. Jennings and M. Ajiz, *Incomplete methods for solving $a^t ax = b$*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 978–987.

[29] D. E. Knuth, *Semi-optimal bases for linear dependencies*, Linear and Multilinear Algebra, 17 (1985), pp. 1–4.

[30] N. Li and Y. Saad, *MIQR: A multilevel incomplete qr preconditioner for large sparse leastsquares problems*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 524–550.

[31] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255–269.

[32] K. G. Murthy, *Network Programming*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[33] A. R. L. Oliveira and D. C. Sorensen, *A new class of preconditioners for large-scale linear systems from interior point methods for linear programming*, Linear Algebra and its Applications, 394 (2005), pp. 1–24.

[34] C. C. Paige, *Bidiagonalization of matrices and solution of linear equations*, SIAM Journal on Numerical Analysis, 11 (1974), pp. 197–209.

[35] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM Journal on Numerical Analysis, 12 (1975), pp. 617–629.

[36] ——, *Toward a generalized singular value decomposition*, SIAM Journal on Numerical Analysis, 18 (1981), pp. 398–405.

[37] ——, *LSQR: An algorithm for sparse linear equations and sparse least squares*, Transactions of the ACM on Mathematical Software, 8 (1982), pp. 43–71.

[38] Y. Saad, *Iterative Methods for Sparse Linear Systems Second Edition*, Society for Industrial and Applied Mathematics, 2003.

[39] M. A. Saunders, *Solution of sparse rectangular systems using LSQR and CRAIG*, BIT, 35 (1995), pp. 588–604.

[40] D. S. Scott, *How to make the Lanczos algorithm converge slowly*, Math. Comp., 33 (1979), pp. 239–247.

[41] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.

[42] R. J. Vanderbei, *Symmetric quasi-definite matrices*, SIAM Journal on Optimization, 5 (1995), pp. 100–113.